[Software Capstone Design]

Software Capstone Design Final Report

Team Cloudy (구르미조)

Latency Reduction between Mobile Users and Edge Servers

■ Team Members: Jae-ho Jung (2011147106)
Myung-jong Kim (2011147026)
Samuel Shin (2012111008)

■ Professor: Seung-Jae Han
■ Teaching Assistant: Sung-woong Jung

Table of Contents

1. Introduction

1.1. Research Goal

1) Implement techniques to reduce workload offloading latency between mobile devices and edge servers
2) Define appropriate situations for different workload offloading techniques

1.2. Abstract

We propose a workload offloading Android application called *Cloudy*. Cloudy utilizes multiple techniques for latency reduction. For each technique, we present quantitative and qualitative evaluation of the reduction in latency as well as provide suggestions for appropriate situations to implement the technique. The techniques are:

1) Using operating-system-level virtualization over full virtualization
2) Using custom scheduling algorithms over naïve scheduling algorithms
3) Using edge servers that are single-hop distance away over cloud servers that are multi-hop distance away

We also illustrate a scenario in which our proposed techniques reduce workload completion time by 25.55%.

2. Background

2.1. Demand for Heavier Workloads in Mobile Devices

Mobile devices are multipurpose computers that allow users to perform a variety of tasks at any location, or even while on the move. However, to allow mobile devices to be mobile, they have low-capacity batteries and limited hardware (CPU, GPU, RAM, secondary storage space, etc.). Workloads that require heavier workloads can vary from 3D graphics rendering in mobile games[1] to mobile device data processing using deep neural networks[2]. While the idea of offloading workload into the cloud has existed for more than a decade[3], advancements in cloud computing technology have allowed tasks that require significant resources (power consumption, computing power) can be offloaded to servers in the cloud, reducing burden on the mobile devices. Therefore, it is important to reduce the time overhead in offloading workload to edge servers.

2.2. Unreliable Network Connectivity

There are situations where network connectivity is unreliable and power consumption is of utmost importance[4]. Examples of such situations are rescue operations in uninhabited areas such as mountains and patient diagnosis in rural areas. In such environments, multi-hop network connections will cause slow responses from servers or even data loss. In such situations, reducing the number of hops through the network will vastly increase reliability in workload offloading.

[1] Lu, Yao, Yao Liu, and Sujit Dey. "Cloud mobile 3D display gaming user experience modeling and optimization by asymmetric graphics rendering." *IEEE Journal of Selected Topics in Signal Processing* 9.3 (2015): 517-532.

[2] Clarke, Nathan L., and Steven M. Furnell. "Authenticating mobile phone users using keystroke analysis." *International Journal of Information Security* 6.1 (2007): 1-14.

[3] Li, Zhiyuan, Cheng Wang, and Rong Xu. "Computation offloading to save energy on handheld devices: a partition scheme." *Proceedings of the 2001 international conference on Compilers, architecture, and synthesis for embedded systems*. ACM, 2001.

[4] Jang, Minsung, et al. "Personal clouds: Sharing and integrating networked resources to enhance end user experiences." *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014.

2.3. Overhead Reduction Strategies

There are many ways to reduce time overhead in offloading workload to edge servers. The strategies that make the biggest improvements are:

1) Latency reduction between mobile devices and servers
2) Optimization of workload offloading algorithm
3) Optimization of mobile-to-server task conversion

3. Related Work

3.1. Computation Offloading as a Service

Shi's research allows mobile devices to make the decision on workload offloading by the COSMOS system[5]. COSMOS calculates the time overhead from creating a virtual machine (VM) on popular cloud service provides such as AWS, and the time overhead of completing the workload on both the mobile device and cloud VM to determine whether workload offloading for a single task will result in shorter time to receive the result of the workload. Shi uses Mantis[6], which is an automatic performance prediction for smartphone applications to extrapolate the time complexity of a mobile application.

While COSMOS has proven to be an excellent predictor for offloading for very demanding workloads, the setup time for an AWS VM takes up to 33 seconds (as seen in table 1). This system in unfeasible for situations where multiple workloads with mobile device runtimes of less than 33 seconds.

Instance	CPU cores	CPU clock speed (GHz)	Average setup time (second)
t2.nano	1	2.4	33.7
m4.large	2	2.3	33.6
c5.xlarge	4	3.0	34.7

Table 1) Characteristics of AWS EC2 on-demand instances

3.2. Content-Based Scheduling of VMs in the Cloud

Bazarbayev's research[7] took advantage of the characteristic in common operating systems (OSes) used in the cloud. Common VM disk image sizes can vary from 1 GB to tens of GBs, and content similarity between same OSes with different versions can be as high as 60%. By only transferring the differing sections of disk images, Bazarbayev's research was able to reduce traffic congestion within the data center, decreasing the amount of data transfer associated with deployment of VMs by about 70%.

While reducing traffic congestion within the data center can improve reliability and operating cost for cloud service providers, data centers have upwards of 25Gbps transfer speeds. As table 2 shows, the VM disk image transfer step one of the quickest steps in the VM creation process. Content-based scheduling optimization results in no discernable difference to the end user.

[5] 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.

[6] Kwon, Yongin, et al. "Mantis: Automatic performance prediction for smartphone applications." *Proceedings of the 2013 USENIX conference on Annual Technical Conference*. USENIX Association, 2013.

[7] 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.

OS	Image size (GB)	Estimated transfer time (second)
Ubuntu Server 12.04 i386	1.05	0.34
RHEL 6.2 x86_64	1.90	0.61
Windows Server 2008 x86_64	21.00	6.72

Table 2) Estimated VM image transfer time of different OSes in AWS EC2 t2.nano instance

3.3. Cyber Foraging in Resource-Constrained Environments

In specific cases such as language translation and face recognition by first responders operating in crisis environments, there are additional rationales for offloading workload to edge servers which are located in single-hop proximity. General-purpose workload offloading strategies include duplication, ignoring garbage disk blocks, communicating free memory information between VM and VM manager, and VM synthesis pipelining. Restricting the environment conditions allows for additional latency reduction techniques. Lewis's[8] work evaluates five different cloudlet provisioning strategies: optimized VM synthesis, application virtualization, cached VM, cloudlet push, on-demand VM provisioning.

	Optimized VM Synthesis	Application Virtualization	Cached VM	Cloudlet Push	On−Demand VM Provisioning
Cloudlet Content	Exact base VM	VM compatible with server code	Service (VM) repository	Repository of paired VMs (server code) and client apps	VM provisioning software, server code components
Mobile Device Content	Application overlays, client apps	Virtualized server code, client apps	Client apps	None	VM provisioning scripts, client apps
Payload	Application overlay	Virtualized service code	Service ID	Client apps	VM provisioning script
Advantages	Cloudlet can run any server code that can be installed on a base VM	Portability across OS distribution boundaries	Supports server code updates as long as service interface remains the same	Supports most client nodes with distribution at runtime	Server code can be assembled at runtime
Constraints	Requires exact base VM which limits distributions and patches	All server code dependencies have to be captured at packaging time	Cloudlet is provisioned with service VMs required by client apps	Cloudlet has a client app version that matches mobile client OS version	Cloudlet has all required server code components

Table 3) Qualitative comparison of cloudlet provisioning strategies[8]

[8] 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.

4. Implementation and Analysis

4.1. Simulations

We created sample workloads that can be run on the Android platform that use most of the system resource of a device. The workload applications are CalculateFibonacci, CalculateSha1, and EstimatePi. They take integer parameters which increases the complexity of the simulation.

4.1.1. CalculateFibonacci

CalculateFibonacci calculates the nth term in the Fibonacci sequence using recursion. We chose to use the naïve recursive implementation to maximize RAM usage. The integer parameter corresponds to the position of the Fibonacci term to calculate.

4.1.2. CalculateSha1

CalculateSha1 create a string of 100 random characters and calculates a SHA-1 hash value. We designed the algorithm to take short strings to calculate the hash value to reduce the effects of CPU optimization techniques. The integer parameter corresponds to the number of random strings to calculate the hash of.

4.1.3. EstimatePi

EstimatePi uses Monte Carlo simulation to estimate the value of π. Monte Carlo estimations are good real-world examples of resource-intensive and repetitive tasks that can be offloaded to servers. The integer parameter corresponds to the number of samples taken in the simulation.

4.2. Cloud Servers

Cloudy must delegate workloads to cloud servers. To build such cloud servers, we launched multiple web servers across the globe with Amazon Web Services (AWS) instances. We launched five t2.2xlarge instances. The servers were located in North Virginia, Oregon, London, Mumbai, and Seoul. All instances were running Ubuntu Server 16.04. All servers were running Apache, OpenSSH, and VirtualBox with Ubuntu 16.04.

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

4.2.1. Full virtualization

In most of the research done on workload offloading, the server handles job requests by creating a VM inside the cloud server using full virtualization software such as VMWare and VirtualBox. We tested job handling with VirtualBox, running Ubuntu 16.04. Each workload has a corresponding VM which can be provisioned in real time with the VM provisioning script payload. This method is our implementation of On-Demand VM Provisioning outlined previously in *table 3*. This method has the advantage of assembling server code at runtime, allowing for easier distribution of new job types.

Full virtualization allows higher level of access to virtualized hardware. This means that jobs can take advantage of hardware efficiency such as CPU L2 cache, resulting in some gains in speed when handling jobs.

4.2.2. Operating-system-level virtualization

Despite the efficiency benefits of full virtualization, the time overhead in provisioning the VM is very large. In our testing, the shortest time overhead in VM provisioning was 38608ms. On the other hand, the time for container provisioning in operating-system-level virtualization with Docker had a maximum of 15190ms and a minimum of 374ms. Only a small number of edge case jobs will require enough CPU-intensive computations to compensate the average of 32412ms in time overhead decrease using Docker. Our testing could not achieve a threshold in which the benefits of full virtualization compensated for the increased time overhead in VM provisioning.

Another advantage of operating-system-level virtualization is the modular nature of VM construction. While there is work being done to reduce overhead in VM scheduling for VMs with similar images (as mentioned in Section 3.2.), most work of this field focuses on VM image transfer, not VM construction. Therefore, even if two VMs have 99% similarity in the content, each VM image is treated separately. There is high time overhead in switching between the two VMs. Docker, on the other hand, is modular. Many manufacturers ship their base Docker images into the central server. Available images include Ubuntu Server 16.04 and Java 8. As long as the base VM are installed on a machine, only the remaining code has to be installed to run a particular job. Our custom scheduling algorithm that is discussed later takes advantage of this model.

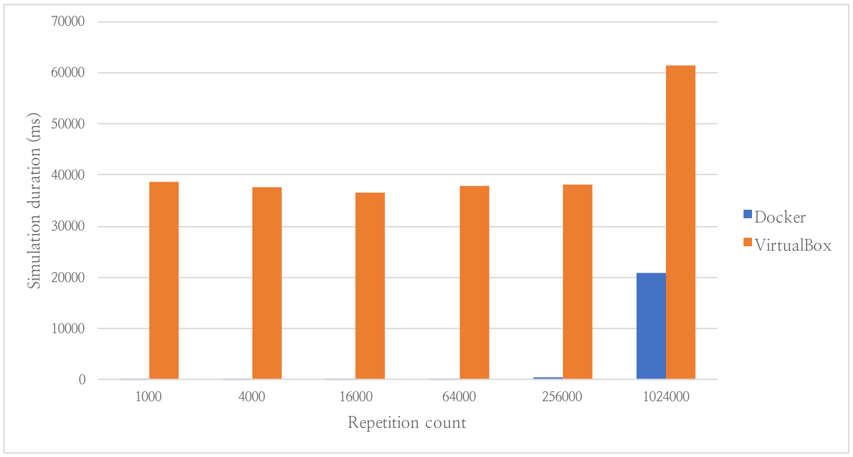

Chart 1) Simulation duration of CalculateFibonacci on Docker and VirtualBox

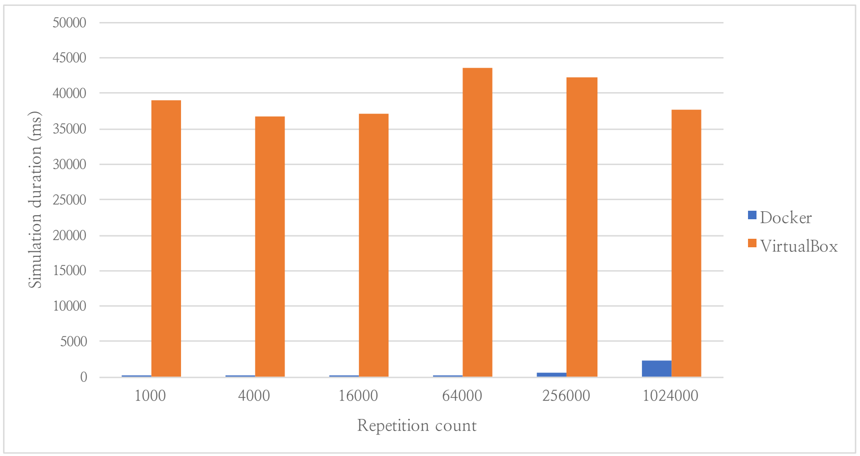

Chart 2) Simulation duration of CalculateSha1 on Docker and VirtualBox

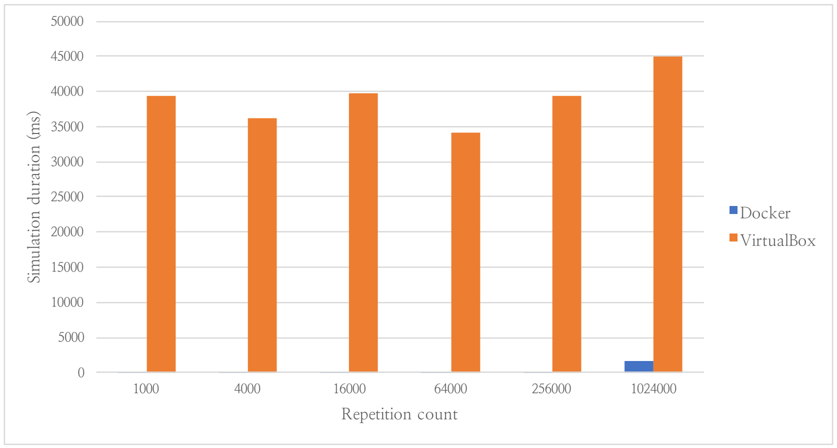

Chart 3) Simulation duration of EstimatePi on Docker and VirtualBox

4.3. Cloudlet Server

Cloudlet servers have the same OS specification and installed programs.

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

They are inside the school network, therefore there are only 2 hops from the client. In comparison, servers outside the subnet have hop counts in the range of 20 to 40. Hop count does not account for significant time overhead in communication. However, in crisis situations, ad-hoc networks may cause frequent packet drops. We simulated situations in which packets were dropped in between hops at different probabilities.

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

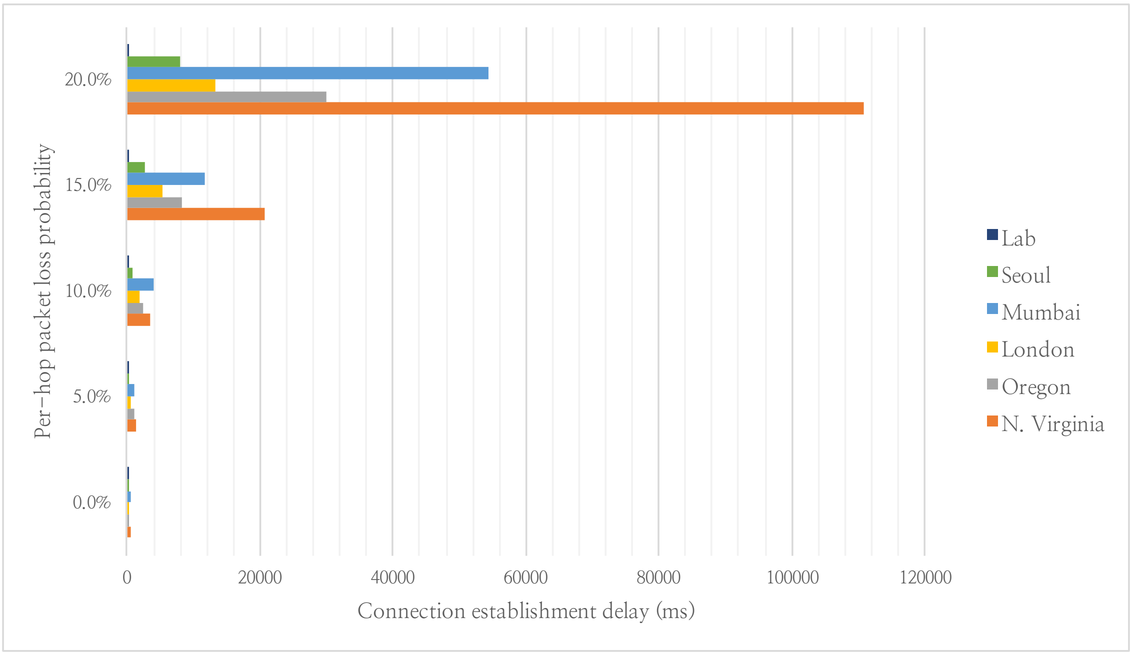

Chart 4) Simulated connection establishment delays between mobile device and servers

Increase in hop count corresponds to exponential increase in connection establishment time between mobile devices and servers. In a theoretical situation of 20.0% chance in packet loss for each hop, a server that is 33 hops away would result in 110743ms of delay. Cloudy switches to using the single two-hop server to delegate all its tasks if the estimated total completion time using multi-hop servers is longer.

4.4. Command and Control Application

Cloudy can communicate with and send jobs to servers. We implemented three scheduling algorithms to delegate jobs to the servers. Two of them are existing scheduling algorithms, and one of them is customized for Cloudy's specific use case.

4.4.1. Naïve Scheduling Algorithms

Of the many algorithms tested, the round robin and earliest deadline first implementation gave the best results. We compared our custom algorithm to the two algorithms.

4.4.2. Custom Scheduling Algorithm

We were able to tweak existing algorithms to be best fitted for our situation. To reduce the time required to download and install jobs from the central server, we minimized the number of unique job types each server handles. This logic takes full advantage of a characteristic of Docker (explained in *section 4.2.2.*). We also used the estimated completion time of each job and used the information to schedule jobs so that each server spends roughly the same amount of time to complete the assigned tasks.

```
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

We compared the performance of the three scheduling algorithms using different combinations of number of job types, servers, and jobs. For example, with 3 job types, 6 servers, and 30 jobs, the total time taken for all jobs to be completed was 100137ms for the round robin algorithm, and 79987ms for our custom algorithm. This is a 20.12% reduction in total time.

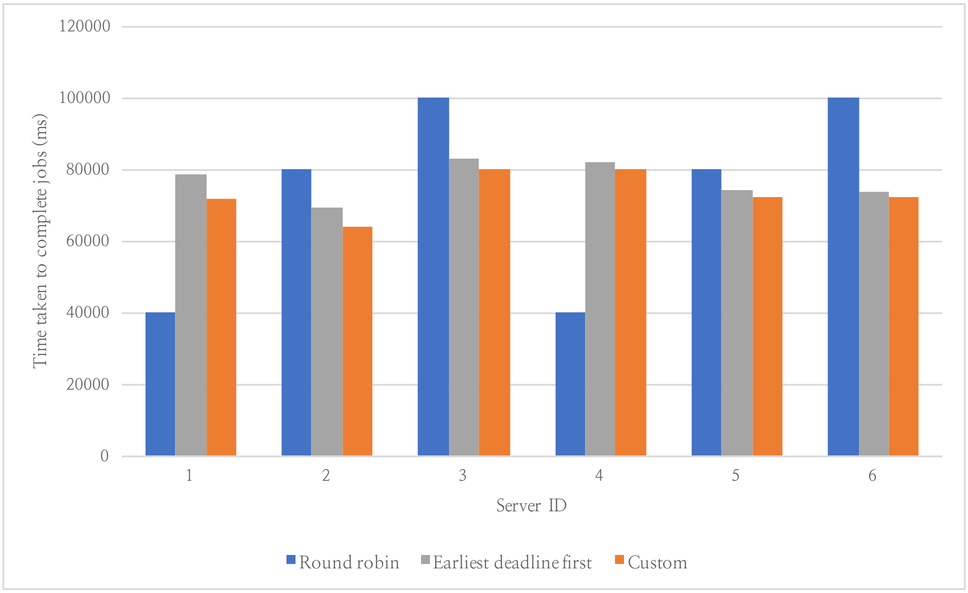

Chart 4) Performance comparison of naïve and custom scheduling algorithms with 3 job types, 6 servers, and 30 jobs

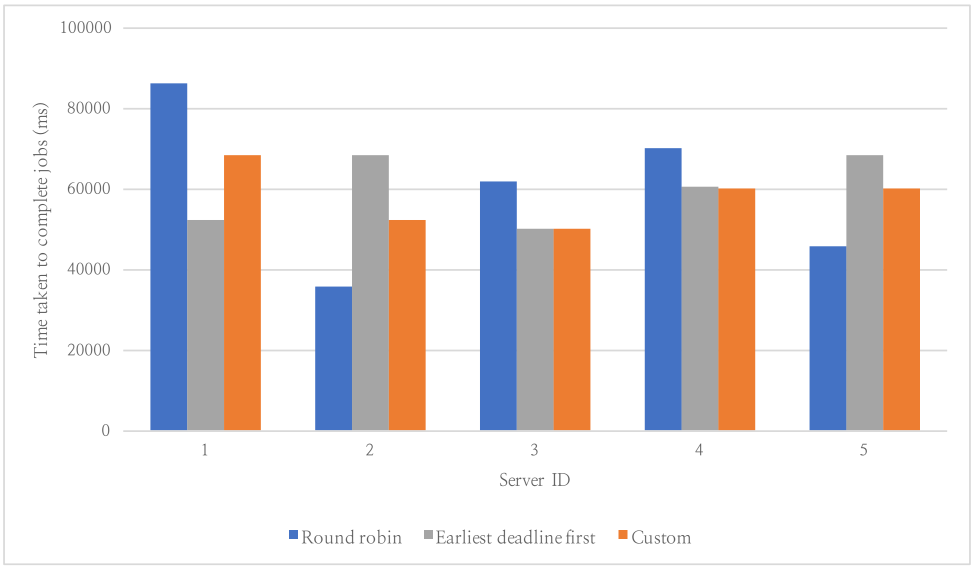

Chart 5) Performance comparison of naïve and custom scheduling algorithms with 4 job types, 5 servers, and 21 jobs

5. Conclusion

5.1. Analysis

5.1.1. Advantages

Cloudy is modular. Cloudy uses the Docker repository and version control system. Uploading a new simulation is completely isolated from the existing simulations, and downloading a new simulation is as simple as performing a single command in the server. It is also scalable, as our scheduling algorithm takes in the number of cloud servers dynamically as an input from the controller device, the scheduling algorithm can handle a range of different number of servers and simulations. It is adaptable because estimates which features will result in the least computation time. And as our tests have shown, it has better performance compared to existing solutions under low load.

5.1.2. Disadvantages

Cloudy requires initial setup of the cloud server, and all required components cannot be installed into a virtual disk image due to its dependency on the Docker repositories. Because the simulation codes are not identical between the mobile device version and the cloud server version, the programmer must write two separate versions of the simulation code. Lastly, because of the nature of Docker as sharing CPU resources, it is impossible to allocate a specific amount of CPU power to the server. This results in less predictable performance in workload offloading.

5.2. Results

5.2.1. Virtualization Method

For non-specialized workloads that run on base VMs, running jobs on operating-system-level virtualization VMs are more suitable. Running different workloads that share same underlying structure (e.g. CPU architecture, OS) are also more suited for operating-system-level virtualization. Our tests show an average reduction in job completion time of 32412ms.

5.2.2. Scheduling Algorithm

We developed a custom scheduling algorithm to distribute workloads evenly across all available cloud servers. The custom scheduling algorithm also minimizes time overhead caused by downloading multiple workloads by minimizing the types of workloads handled by a single server. Our scheduling algorithm performs better than a naïve scheduling algorithm by 20.12%.

5.2.3. Hop Count Minimization

We developed a C&C Android application that detects network connectivity strength between the client and the servers and switch automatically to an edge server.

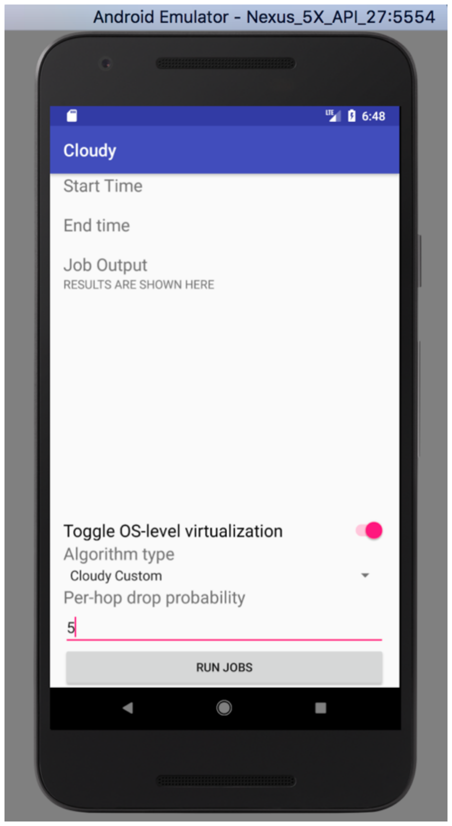

Fig 2) Screenshot of Cloudy C&C Android application

Real-world testing for a combination of tasks resulted in 225702ms completion time without enabling Cloudy's features. A theoretical completion time for 20% per-hop packet drop chance resulted in 265400ms completion time. With Cloudy's three features enabled, the tasks were completed in 168043ms. This is a 25.55% reduction in real-world and 36.68% reduction in 20% per-hop drop chance situation.

6. Reference

Lu, Yao, Yao Liu, and Sujit Dey. "Cloud mobile 3D display gaming user experience modeling and optimization by asymmetric graphics rendering." *IEEE Journal of Selected Topics in Signal Processing 9.3* (2015): 517-532.

Clarke, Nathan L., and Steven M. Furnell. "Authenticating mobile phone users using keystroke analysis." *International Journal of Information Security 6.1* (2007): 1-14.

Li, Zhiyuan, Cheng Wang, and Rong Xu. "Computation offloading to save energy on handheld devices: a partition scheme." *Proceedings of the 2001 international conference on Compilers, architecture, and synthesis for embedded systems.* ACM, 2001.

Jang, Minsung, et al. "Personal clouds: Sharing and integrating networked resources to enhance end user experiences." *INFOCOM, 2014 Proceedings IEEE.* IEEE, 2014.

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.

Kwon, Yongin, et al. "Mantis: Automatic performance prediction for smartphone applications." *Proceedings of the 2013 USENIX conference on Annual Technical Conference.* USENIX Association, 2013.

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.

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.