145/237D project specification: Cell Phone Data Center

Jennifer Switzer, Eric Siu, Subhash Ramesh,
Ruohan Hu, Emmanuel Zadorian

April 2021

1 Project Charter

Smartphones are a significant source of E-waste, with 150 million being discarded each year in the US alone [1]. Although the average smartphone has a theoretical 10 year lifespan, most are discarded within 2 years [2]. This is especially problematic because smartphones are difficult to recycle. Because of their specialized construction (and their classification as hazardous waste) they can't simply be thrown into the recycling [3]. Even when smartphones do make their way to an E-waste recycling facility, these are oftentimes unregulated, employing child labor and exposing workers to hazardous chemicals [6].

Recent work has suggested an alternative approach: Expanding the lifetime of discarded smartphones by reusing them for general computational tasks. Initial work has indicated the feasibility of this approach, but many systems challenges remain [5, 4]. For our 237D class project, we will attempt to address some of these challenges by designing and constructing our own cell phone data center. Our overall goal is to evaluate the feasibility of reusing smartphones as general-purpose cloud servers.

1.1 Project Overview

The data center consists of three main components: the **phone bank** itself, a collection of used cell phones; a **smartplug-enabled power strip**, which allows power to the phones to be toggled on and off remotely; and a **central management device**, which is responsible for managing power and distributing tasks among the phones. If time allows, we will also build a custom container for the hardware.

The management device will receive incoming job submissions, and will distribute them amongst the phones. The actual computations themselves will be performed on the phones. To evaluate our system, we will test it on a variety of representative cloud workloads.

1.2 Project Approach

Each smartphone will be modified to run Ubuntu Touch, an open-source mobile operating system. This ensures that we have a uniform operating system across the phones, and will also allow us to run arbitrary code more easily. We will use a Raspberry Pi as our centralized management device (for simplicity, we will refer to it as the "manager"). The manager communicates with the phones and smartplugs over WiFi.

Each phone will be connected to power via a smart plug, and the manager will maintain a table of phones and their associated smartplug. The phones periodically send heartbeat messages to the manager, so that it can keep track of which phones are active. This heartbeat message also includes information about the phone's battery level and CPU use. This information enables the manager to decide which phone(s) are able to accept new tasks, as well as monitor the health of the phones. For instance, we want our phones to stay around 80 percent charge capacity to operate optimally.

A user submits tasks to the Raspberry Pi as a link to a GitHub repo (for our MVP) or Docker container (for our reach goal). Included in this submission is a specification for how much CPU, RAM, and disk space should be allocated to the task. The manager will delegate this new task to the next phone with enough resources to complete it. Once completed, the result is returned to the user via the manager.

1.3 Project Objectives, Milestones, & Deliverables

Our **minimal viable product** is to build a proof-of-concept data center with three old phones, three smart plugs, and a Raspberry Pi. The Raspberry Pi should be able to communicate with the smart plugs and disposable phones via a pre-existing WiFi network. It should be able to toggle the smart plugs on and off, and should be able to submit simple jobs to the phones. Moreover, the phones should be able to send a heartbeat message to the Raspberry Pi to indicate their status in the cluster. As a proof of functionality, the MVP should include the ability for the phone to perform simple arithmetic operations by receiving a GitHub repository link and running the source code on that repository. Furthermore, it should allow for the following message types between components:

1. Heartbeat: Phone tells the manager that it's still alive/what its status is.
2. Job Submission: A cluster user submits a job to the manager.
3. Job Delegation: The manager assigns a job to a phone.
4. Job Acknowledge: Phone tells the manager that it has received the job and is capable of completing it.
5. Power on/off: The manager tells a specific smart plug to toggle on or off to control the power to its associated phone.

Over the quarter, we will iterate on this basic MVP in three ways. First, we will add functionality to the phones to perform more complicated, containerized compute tasks (these would be submitted as Docker images). Second, we will expand the functionality of the manager (Raspberry Pi) so that it can make more sophisticated decisions such as enforcing time constraints for the tasks. Overall, we will try to iterate on the code base to ensure that the cluster can run jobs by itself, with minimal supervision. Finally, if time allows we will build a physical container for the hardware, and will attempt to power it via solar panels.

The longer term goal of this project past this quarter would be to create a disposable phone data center larger than 3 phones that can perform general compute tasks. This long term goal would reduce the amount of cell phone waste by using the potential processing power from these throw-away phones. We also aim to submit our work to the MobiCom conference, which looks for research contributions in wireless networking and mobile computing.

1.4 Constraints, Risk, and Feasibility

The potential stumbling blocks are whether we can run arbitrary code on the Ubuntu Touch, and whether the Raspberry Pi will work well as a manager. If Docker doesn't work well with the Ubuntu Touch OS, we might have to constrain to specific types of programs we can run natively on the phones. Moreover, the processors on the phones can be using any type of ISA, but the Raspberry Pi is running an ARM chip. It's entirely possible that the binaries on the Raspberry Pi is incompatible with the phones, which is why we opted to research Docker containers. Moreover, we are hoping the Raspberry Pi OS contains the required functionality for us to connect and talk to all these devices, otherwise, we would have to install new OS's onto the Pi.

The realistic goal is that we will be able to send simple arithmetic tasks using the Raspberry Pi to separate phones on the same local area network to perform tasks. If Docker does not work well with our architecture, then we will have to compromise with simpler arithmetic tasks instead of more complex tasks, like running a web scraper.

List of Risks:

1. Docker not working well in our environment
2. Other Ubuntu Touch-specific issues (for instance, we might not be able to install certain software packages)
3. Raspberry Pi OS does not have the required functionality to work as a manager
4. Hardware issues with Raspberry Pi / Disposable Phones

2 Group Management

On a high level, we've split our group into 3 different sub-teams (mobile OS, task management, and benchmarking), in order to effectively distribute the workload while minimizing the dependencies between the sub-teams. In the following sub-sections, we outline more details on how our group will work together to successfully finish this project.

2.1 Major roles

Jen is our integration manager. This way, she also has a copy of all of the hardware components required for our project, and is able to therefore ensure that our whole system works properly (which is important since our team is working in a remote, distributed fashion). Furthermore, Jen, Ruohan, Subhash, and Eric are developers, and Emmanuel is our benchmarking engineer. The specific responsibilities of each individual in our team is further outlined in the **Project Development** section.

2.2 Decision making

When making decisions (such as the design/spec for how certain components of our system will operate, etc.), we usually discuss these together in our weekly Zoom meeting and come to a consensus of what we will do and how we will do it. This way, we can effectively eliminate any dependencies that may exist between the sub-teams, while enabling the sub-teams to work independently on the agreed upon spec.

2.3 Communication

We use Slack for asynchronous, group communication, and we also meet via Zoom on Saturdays (and sometimes on Wednesdays as well).

2.4 Scheduling

When we convene for our weekly Zoom meeting, we first touch base on everyone's progress, to ensure that everyone is meeting the agreed upon milestone and deadlines, and towards the end, we establish the deliverable goals for the upcoming week. In terms of any schedule slips, we will use our Zoom meetings to discuss this, and in the event of any unforeseen schedule slips, we will use Slack to notify the team as soon as possible, so that we can work around the situation.

3 Project Development

This section provides an overview of our development subteams, the major hardware and software components of our system, and how we intend to test

and document our implementation.

3.1 Development roles

We have split our development roles into three subteams: Task distribution (Subhash and Eric); Mobile OS (Ruohan and Jen); and Benchmarking (Emmanuel).

The task distribution team is responsible for implementing the management software, which receives incoming jobs and assigns them to a phone or phones.

The mobile OS team is responsible for ensuring that the Ubuntu Touch operating system is sufficient to meet our needs, and for making any changes to the OS or our software as needed to ensure that everything is compatible.

Both the task distribution team and the mobile OS team are responsible for defining the interface between the manager and the phones.

The benchmarking team is responsible for identifying representative cloud workloads for use during the testing phase. They are also responsible for finding and/or writing a set of jobs for each workload type.

Finally, we have also named an integration manager (Jen), who is responsible for ensuring that all of the components of the system are compatible.

3.2 Hardware & Software

There are two major components to our implementation: The management software, which is implemented on a Raspberry Pi, and the worker software, which is implemented on the cell phones.

Hardware: We are using the following hardware: (1) For the manager, the Raspberry Pi 4 Model B with 8GB of RAM; (2) For the workers, three used Google Nexus 4 phones; (3) and for the smart plugs, the Wyze Indoor Smart Plug. We have distributed the hardware among the members of the varying teams as follows:

- **Task distribution** team members have a Raspberry Pi and a smart plug.
- **Mobile OS** team members have a Nexus 4 phone and a smart plug.
- The **integration manager** has a full set of Raspberry Pi, phone, and smart plug.

At the time of writing everyone has received their hardware.

Software Jobs are submitted to the Raspberry Pi as links to Github repos (for MVP) or Docker images (for reach goal). The Raspberry Pi management software, which is implemented in Arduino, is responsible for disseminating these links to the workers (the phones).

Since the phones are running a version of Ubuntu, we are able to implement the worker software in Python, and invoke it via a bash script. When one of the phones wishes to instantiate a job, it uses the link provided by the manager to pull the appropriate Git repo or Docker image, and then runs the code or image according to the job specifications provided.

The Raspberry Pi will communicate with the smart plugs via the If This, Then That (IFTTT) platform, and with the phones directly over WiFi. All communication is via the manager; the phones do not talk to each other, or to the smart plugs. For our initial implementation, we simply assume the presence of a WiFi network in the deployment area.

3.3 Testing

We will evaluate our system by running a variety of common cloud workloads on the data center. The specific categories of workloads for evaluation will be defined by our benchmarking team, but we expect they might include things like: Performing arithmetic operations; hosting a key-value store; or training machine learning models.

We will evaluate the ability of the data center to handle each of these workloads by measuring: (1) The average latency of the jobs; (2) The success rate of the jobs; and (3) The rate of battery depletion while running each type of workload. The latency is defined as the total time elapsed between the submission of the job and the return of the completed result to the user. A successful job is defined as a job that completed and returned a result within the specified time limit.

3.4 Documentation

We will be using GitHub's built-in Wiki feature to document our project as we go. During development, we will add pages to the Wiki as needed to document our design decisions, as well as the locations of important software entry points. We'll also use this as a place to store our class materials, including our presentations, report, and the final video. At the end of the quarter, we'll clean up and re-organize the Wiki.

4 Project Milestones and Schedule

4.1 Milestones

This section provides an overview of the project milestones and schedule. Larger milestones are indicated in **bold**. As the quarter progresses, those milestones and schedule are subject to change. We will also keep a live record of our goals on our wiki.

Week 4 Overall goal: Have heartbeat functionality implemented, determine if we can run docker on Ubuntu Touch, pick 3-4 example applications.

Mobile Team

- Determine if Docker will work on Ubuntu Touch (or another type of container)

- Implement the sending of heartbeats

Task Management Team

- Implement the receiving of heartbeats
- Connect to the smart plugs to turn them ON/OFF

Benchmarking Team

- Come up with a list of 3 example applications for initial testing.
- Begin research into representative cloud workloads, and existing benchmarking suites.

Week 5 Overall goal: Implement and test the job submission, job delegation, and job acknowledgement functionality. By the end of the week, perform **initial integration testing** and confirm that we are able to: toggle smartplugs ON/OFF from the Raspberry Pi; send heartbeat from phones to Raspberry Pi; submit job (as Git repo) from Raspberry Pi to phone.

Mobile Team

- Implement the receiving of jobs, including the logic to decide whether or not the phone is capable of accepting the job

Task Management Team

- Design and implement the algorithms to distribute jobs based on the state of each phone
- Implement the forwarding of jobs to the phones
- Implement the receiving of responses, and updating the state of each phone

Benchmarking Team

- Create a few example GitHub repos for integration testing.

Week 6 Overall goal: Continue to work on the implementation from week 5, and resolve any issues uncovered during the week 5 integration tests.

Mobile Team

- Fix any issues uncovered in integration testing
- Start to add functionality for job submission via Docker.

Task Management Team

- Fix any issues uncovered in integration testing
- Work on optimizing task distribution, including adding functionality for detecting and responding to faulty or slow phones.

Benchmarking Team

- Expand example GitHub repos to cover a representative sample of cloud workloads.

Week 7 Overall goal: Finish up the implementation, test the functionality thoroughly, and work on the milestone report together. **At this point, a minimum viable product should be available**.

Week 8 & Week 9 Overall goal: Resolve any issues, continue work on stretch goals. At the end of week 9, perform a **second round of integration tests**. At this point, if time allows, build the physical container.

Mobile Team

- Continue work on stretch goals, if applicable.
- Help benchmarking team with testing.

Task Management Team

- Continue work on optimizing task distribution.
- Help benchmarking team with testing.

Benchmarking Team

- Evaluate the final design on the identified workloads, according to the metrics described in Section 3.3.

Week 10 Overall goal: All team members will work on the **final documentation and presentation** together.

References

[1] M. Brannon, P. Graeter, D. Schwartz, and J. R. Santos. Reducing electronic waste through the development of an adaptable mobile device. In *2014 Systems and Information Engineering Design Symposium (SIEDS)*, pages 57–62, 2014.

[2] Roland Geyer and Vered Doctori Blass. The economics of cell phone reuse and recycling. *The International Journal of Advanced Manufacturing Technology*, 47(5-8):515–525, 2010.

[3] I.M.S.K. Ilankoon, Yousef Ghorbani, Meng Nan Chong, Gamini Herath, Thandazile Moyo, and Jochen Petersen. E-waste in the international context – a review of trade flows, regulations, hazards, waste management strategies and technologies for value recovery. *Waste Management*, 82:258–275, 2018.

[4] Noah Klugman, Meghan Clark, Pat Pannuto, and Prabal Dutta. Android resists liberation from its primary use case. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom '18, page 849–851, New York, NY, USA, 2018. Association for Computing Machinery.

[5] Mohammad Shahrad and David Wentzlaff. Towards deploying decommissioned mobile devices as cheap energy-efficient compute nodes. In *9th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 17)*, 2017.

[6] Manmohit Singh, Parteek Singh Thind, and Siby John. Health risk assessment of the workers exposed to the heavy metals in e-waste recycling sites of chandigarh and ludhiana, punjab, india. *Chemosphere*, 203:426–433, 2018.