237 Milestone report:
Cell phone data center team

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

May 22, 2021

1 Minimum Viable Product

As of 5/10, we have a fully functional minimum viable product. This includes the following functionality:

1. **Device registration.** Phones can initialize communication with the Raspberry Pi, and register themselves. The Raspberry Pi keeps a running list of all registered phones, for job delegation.
2. **Heartbeats.** Phones send a periodic heartbeat to the Raspberry Pi. This heartbeat includes information about the phone's current battery level, CPU use, and disk space.
3. **Job submission.** The Raspberry Pi receives a job, delegates it to one of the phones, and waits for the result of the job. Each job submission includes the following: A link to the GitHub repo that holds the script to be executed; the disk space needed for the job; and the cpu/memory that should be delegated to the job. The delegated phone then pulls the code from the repo, executes it, and returns the result to the phone.
4. **Power management.** The Raspberry Pi can toggle power to the phone via its associated smartplug.

The first three components are demonstrated in this video: https://youtu.be/TaK98wkG5LY. The power management functionality is demonstrated here: https://drive.google.com/drive/folders/1obKsECz7GGH-5G4juZTtiypXvUXnRsRw?usp=sharing.

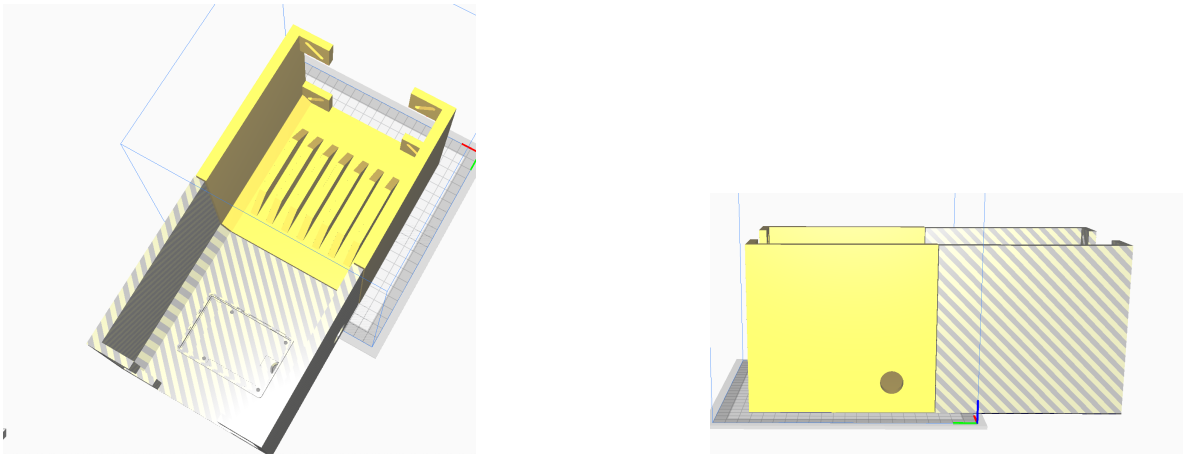
2 Next Steps for Full Prototype

After achieving our minimum viable product, we have started work on the additional tasks needed to achieve our full prototype, which includes a physical case, a full benchmarking suite, and clean documentation for reproducibility.

2.1 Building the Case

We used CAD software to design a case to hold the phones and the Raspberry Pi.The figure below is a rendering of how the case will look. We plan on mounting fans at each end of the case for optimal airflow. The case will also contain a power strip which will be glued to the inner wall of the chassis. Storing the power strip in this way allows us to support power strips of various shapes and sizes.

We have completed the design for the box, and will be printing it during Week 8. If time permits we will attempt to power the devices using solar panels mounted on the roof.



(a) Top down view.

(b) Side view.

Figure 1: CAD designs for the chassis. Phones are placed on their sides in the phone holder (on the yellow side), and the Raspberry Pi fits into a specially-cut depression (on the grey-striped side). There is space for a fan on both ends of the box. Not shown is a lid that fits on top.

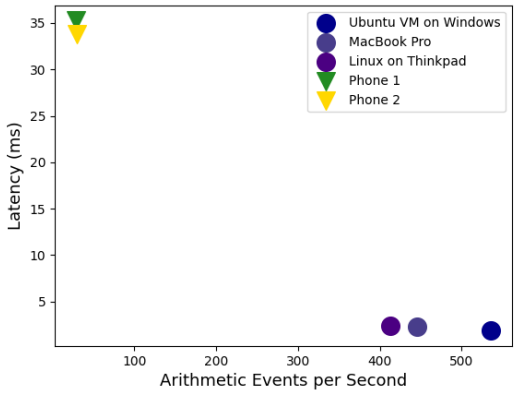


Figure 2: Results from benchmarking two of our phones (triangles) against our personal laptops (circles). The Nexus phones were significantly slower than our laptops, which is something that we need to take into account when exploring appropriate use-cases for the phones.

2.2 Benchmarking & Testing

Figure 2 shows the results of our initial CPU benchmarking with sysbench, a Linux benchmarking utility. The sysbench results demonstrate that the Nexus phones are about 30x slower than your typical laptop. Although the Nexus phones are not as quick as the laptops, newer phones are getting closer to laptop-like performance. As consumers continue to discard their phones, newer disposed phones contain hardware performance similar to laptops and desktops.

In addition to our sysbench results, we intend to produce two different suites of testing or benchmarking: BenchmarkSuite and TestSuite.

2.2.1 BenchmarkSuite

The BenchmarkSuite's purpose is to demonstrate the computational prowess of the phones, in other words, how well they perform their tasks. These tests will be similar to sysbench, in that we will be using them to determine the compute ability of the phones given specific tasks. We will be implementing BenchmarkSuite as a set of possible tasks to be sent to the phones. With our current implementation (e.g. tasks as Git repos with no containerization), this will involve writing several bash scripts with minimal dependencies, and hosting them on GitHub.

2.2.2 TestSuite

The TestSuite's purpose is to test the functionality of the data center. For example, we would want to test the API between the user and the data center, so that the user can send jobs and retrieve output from those jobs. Other functionalities will be tested such as: job assignment, phone decommissioning/health updates, and smart charging. The TestSuite will demonstrate that every part of the data center works correctly. We will be implementing TestSuite as a series of commands sent to the Raspberry Pi.

2.3 Documentation

Our project documentation is located on our Github Wiki pages. We've been updating it as we develop for our own internal use, and will be cleaning it up and publishing it (making it public) for the Web Presence assignment. The documentation includes instructions on how to set up and interface with your own phone data center. There are also instructions for setting up the phones with Ubuntu Touch OS, setting up the Raspberry Pi server, and registering and connecting to the smart plugs.

3 Stretch Goals

If we have time after achieving our full prototype, we plan to iterate further on the design, in order to make it more efficient and generalizable.

3.1 Docker

Currently, our phones execute the submitted compute jobs by downloading the job's GitHub repo and executing its *main.sh* file. As a result of this architecture, there is little isolation of the running job from the phone's environment, which not only introduces security risks for the phone and our platform, but can also make it harder to run certain types of jobs (which require a complex set of dependencies, etc.).

So, we want to see if we can use Docker to execute the jobs on the phones, since this will allow us to isolate the running job to its own container, which will not only be more secure than our current setup, but will also make it easier to setup the job's environment and dependencies on the developer's end. This will also simplify the managing the job's resources on the phone side, since Docker allows us to easily throttle the amount of CPU usable by the container, in addition to setting a max memory limit, etc.

3.2 Reduce battery usage

Currently, we're noticing that using the Ubuntu Touch OS appears to drain the device battery faster than Android. We want to explore if there are certain OS settings and processes we can modify, so that there is less power consumption on the phone end. Since we are re-purposing consumer phones to "server" phones that run compute jobs, we anticipate that we can disable many of the OS features meant for consumer-use, and thereby improve overall battery usage.

3.3 Job redundancy & checkpointing

Currently, our system does not support job redundancy. This will be a necessary feature for reliably running idempotent jobs on our unreliable infrastructure. An example of this functionality would be running & tracking a submitted job multiple times across more than one device simultaneously, and ensuring that at least one copy of the job succeeds.

Furthermore, another useful addition for our current system would be job checkpointing. Since we are essentially operating on unreliable hardware like smartphones, being able to checkpoint a job's progress over time would be very useful for resuming any interrupted jobs, such as those that failed due to a transient hardware issue, etc. Without this, executing long-running jobs in our system would not be practically possible.

To support this feature, however, we would need to have some sort of shared disk space between all the phones, so that a job running on one device can periodically save its state to this shared disk

space, and then read from this same disk space when resuming on another phone. As part of this we will therefore have to explore if it's possible to mount a networked file system connected to all the devices in our setup and managed via the Raspberry Pi. In this method, each job would be assigned its own folder in this shared file system, which it can use to checkpoint its progress over time.

4 Schedule

This section provides an overview of our schedule for the three remaining weeks of the quarter.

Week 8 All team members will work on the milestone report for the class. The mobile team will work on benchmarking the phones, creating unique identifiers for each phone, adding new logic such as checking exit codes, implementing time outs, and generating heartbeats information, as well as setting up docker. The server team will work on adding new logic such as identifying which phone to send the job to, notifying the phone to cancel the process, rescheduling a job if the job is timed out, as well as cutting the power to the phone if the phone misbehaves. The benchmarking team will work on 3D printing the case.

Week 9 All team members will work on whatever tasks are remaining from week 8. As team members are working on those tasks, they will make sure to provide good documentation including the newly added logic, and any type of issues encountered during implementation (and its corresponding solutions).

Week 10 All team members will work on wrapping up the project, and write and prepare for the final report and presentation.