Renée: New Life for Old Phones

Jennifer Switzer, Eric Siu, Subhash Ramesh, Ruohan Hu, Emanoel Zadorian, and Ryan Kastner

***Abstract*—Discarded consumer electronics are a significant and growing source of hazardous waste. Repurposing smartphones has the potential to reduce the rate of disposal. However, reusing smartphones for general computational tasks is difficult due to the variety and domain-specificity of the mobile software stack, and the relatively low computational power of these devices. We present the design and proof-of-concept implementation of Renée, a smartphone-based cluster built from used phones that provides Function-as-a-Service (FaaS) capabilities. Our experience indicates that reusing decommissioned smartphones for general computational tasks is not only feasible, but that a local cluster built from these devices has the potential to provide faster response times than commercial FaaS, while reducing the yearly Global Warming Potential of the reused devices.**

I. Introduction

DESPITE their nominal 10-year lifespan, most phones are decommissioned within 2 years [1]. This represents not only a waste of computational power, but an acute environmental threat. Smartphones cannot be recycled by traditional means. Even when they do make their way to E-waste recycling facilities, these are often unregulated and hazardous [2]. Shorter lifespans also increase the carbon intensity of these devices, since as much as 84% of the greenhouse gases associated with smartphones is released during the manufacturing process [3].

We argue for an alternate approach: Extending the lifetime of discarded smartphones by reusing them for general computational tasks. Previous work has indicated the feasibility of this approach, but many challenges remain [4], [5]. Mobile operating systems tend to get in the way of long-term, unsupervised device deployments [6]—for instance, Android battery optimizations may kill background processes [7]—and smartphones boast less computational power than other consumer electronics [8]. Despite these challenges, we find it is possible to build a cluster of discarded smartphones that provides FaaS capabilities.

We approach this as a distributed systems' problem, and implement our smartphone server as a cluster of used phones, which we call *Renée*.[1] We replace the Android operating system with Ubuntu Touch, an open-source OS for mobile devices [9] that allows us to treat the phones as standard Linux machines. We treat the phones as unreliable nodes, and design our system to be robust to multi-phone failures. A central management device (we use a Raspberry Pi) provides a single point of entry for the outside user.

Renée is not meant to replace high-end cloud computing servers, but rather provide a cost-effective, low-latency solution for small-scale, local compute. Example use-cases for such a micro-datacenter include: edge computing; servers for small businesses; and computing in remote environments.

[1]Renée is French for reborn.

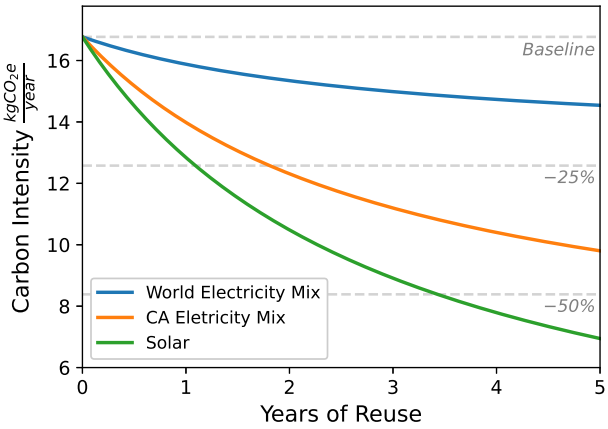

Fig. 1. **The carbon intensity of devices in our cluster decreases over time.** The rate of decline is dependent on the source of the electricity used to power the cluster. Curves are given for the world average (blue), California average (orange), and full solar (green). See **??** for more details.

We estimate that Renée has the potential to reduce the carbon intensity of reused phones by as much as 50% after 4 years of reuse (Figure 1).

II. Background

Prior work has shown that mobile phones have the potential to provide a cost- and energy-efficient alternative for high-performance computing (HPC). Rajovic et al. propose the use of mobile SoCs for HPC, and find that they are both sufficiently performant for many applications, and more energy-efficient than traditional HPC chips [5]. Shahrad and Wentzlaff propose a server built from decommissioned mobile phones [4]. They evaluate the total cost of ownership of such a system, and estimate it to be less than an equivalently-performant system built from high end components. Neither of these include an associated implementation.

Büsching et al. connect 6 Android phones over WiFi, and evaluate the cluster's performance via the LINPACK benchmark [10]. Their implementation targets parallel computing and does not include an associated cluster management system. They also do not explore applications beyond LINPACK.

An orthogonal line of work is the use of smartphones as unsupervised edge devices. Klugman et al. relay their experience deploying a smartphone monitoring system for monitoring the health of power grids [6]. They find that two factors limited the usefulness of the devices: the fact that the Android OS expects human input, and will sometimes stall without it, and the physical degradation of the phones, which experienced screen burn-in and battery swell. This experience motivated our decision to replace the Android OS with Ubuntu Touch, and to manage device power via smartplugs.

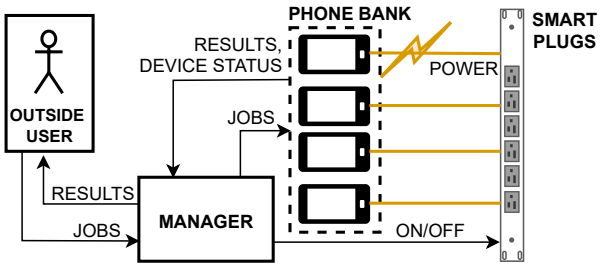

Fig. 2. **High-level system diagram.** An outside user should be able to treat our system as any other FaaS.

III. Renée

A high-level overview of the cluster is presented in Figure 2. It consists of three primary components:

1) A central **manager** that is responsible for managing power to and distributing tasks amongst the phones
2) The **phone bank** itself, a collection of used smartphones repurposed for our project
3) A collection of **smartplugs** (one for each phone), which allow the manager to control power to each device

The manager is a designated machine that provides a single point of entry to the cluster. It maintains a list of all currently active phones and their status, which includes battery level, storage use, and CPU utilization. The phones communicate this information to the manager via regular heartbeats.

Outside users submit jobs to the manager. Each submitted job consists of a zip file of the code to be executed, and metadata including the job's max runtime.

The manager then distributes the received tasks amongst the phones in ascending order of CPU utilization (e.g., the phone with the lowest utilization will be assigned a job first).

In addition to assigning jobs to devices, the manager also deals with phone-level and job-level failures.

Unreachable Phone: Phones send regular heartbeats to the manager. If the manager does not receive any heartbeats from a given phone for 2 seconds, it considers that phone to be inactive. No jobs will be assigned to the phone until a subsequent heartbeat is received from it.

Runaway Tasks: Every submitted job has a configured max run-time defined by the end user. The manager uses this property to periodically scan for jobs running past their configured max run-time. When this happens, the manager instructs the device to kill the job and reschedules it.

Phones listen for new task submissions from the manager. Upon receiving a submission, the phone checks the resource requests, and decides whether or not it is capable of running the job. It returns a job `ACCEPT/REJECT` message accordingly. Once the job is completed, the phone returns the result to the manager.

Power management is accomplished via network-connected smartplugs. When a phone's battery falls below 25% (as reported by the heartbeats), the manager sends an ON signal to the smartplug associated with the device; when the battery reaches 75%, an OFF signal is sent. This maintains the battery level of all phones between 20%-80%, avoiding low battery levels while minimizing charge time.

Renée's centralized design does present a single point of failure: if the manager dies, the entire cluster becomes unavailable. However, the cluster is robust to the failure of any number of phones. Given our choice of hardware, we believe this is a reasonable trade-off, since we expect the used phones to fail quite often, and the management device to fail rarely.

IV. Implementation

A. Hardware

Our management device is the Raspberry Pi 4 Model B with 8 GB of RAM, and we use the Wyze Indoor Smart Plug. Our development phones are LG Nexus 4's with 16 GB of disk space. They contain a Qualcomm Snapdragon S4 Pro APQ8064 SoC with a 1.5 GHz quad-core Krait CPU, and 2 GB of RAM. The Raspberry Pi communicates with the smart plugs via the If This Then That platform, and with the phones directly over WiFi via the local area network.

B. Mobile OS

We replace the phones' native Android with Ubuntu Touch's Nexus 4 distribution [9], which is built on version 3.4.0 of the Linux kernel. To set up the OS for development, we:

1) Reconfigure the file system to be writable.
2) Repartition the disk to allocate 6 GB to system folders (this leaves 10 GB for user data).
3) Install several common Python packages required by our benchmarking suite.

We otherwise leave the OS unaltered.

C. Manager

1) Initialization: At startup, the manager creates a local database for tracking the status of phones and jobs. It initializes a SocketIO server to listen for heartbeat messages from the phones, and job submissions from the end user.

2) Persistent State: The manager maintains the state of all phones in terms of CPU usage, active status, and cooperation. The **CPU usage** of a phone is defined as the average of the three most recent values reported via heartbeat.

Phones are considered **active** so long as they send regular heartbeats. If a phone fails to send a heartbeat, it is marked as inactive until the next heartbeat is received.

Lastly, **cooperation** (or lack thereof) is determine by the number of times a phone has failed to acknowledge a job, or failed to finish a job that it did acknowledge. Too many failures causes the phone to be marked as uncooperative, and automatically decommissioned. Decommissioned phones are disconnected from power, removed from the database, and flagged for later inspection (e.g. by a human operator).

3) Runtime: The manager maintains two main threads: one for accepting job submissions, and one for receiving heartbeats from the phones. When a job submission is received (via an HTTP POST request), the manager makes a local save of the job details (zip file and metadata), then forwards the job onto the least occupied active phone. If that phone fails to

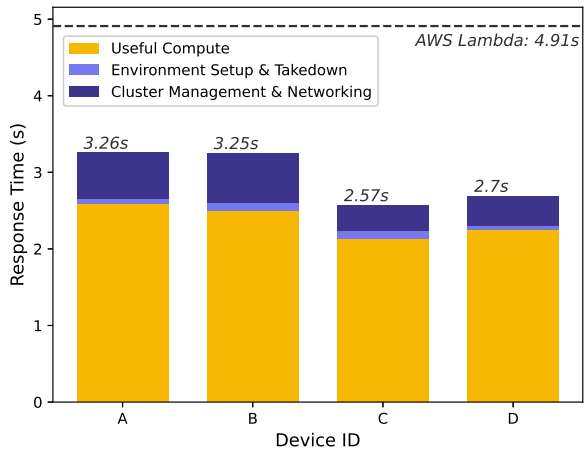

Fig. 3. **Cluster response time** for our `fib` benchmark submitted to Renée (bars) and AWS Lambda (dotted line). All results are mean over 10 iterations. Orange is useful work; blue and purple are overhead.

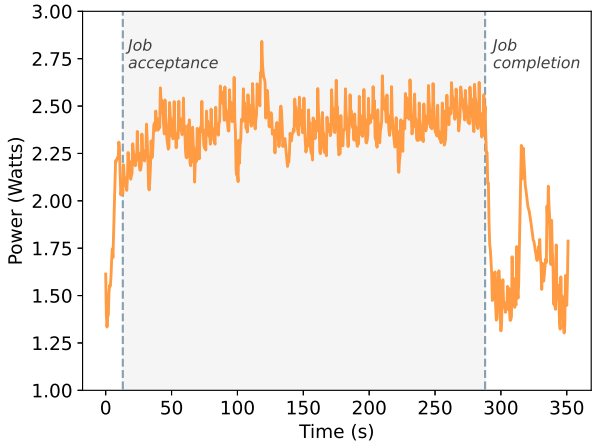

Fig. 4. **Power draw over time**, with events marked at the dotted lines. Data gathered via powerstat for a phone calculating the 40^{th} fibonacci number. Results were smoothed using a moving average with a window size of 4.5 seconds.

acknowledge the job, or if the job fails, another device is selected, and so on until a result is returned. The manager forwards completed results back to the end user.

D. Phone Client

The phone client consists of contains two threads: one for sending heartbeat messages, and one for listening for and executing job submissions. The heartbeat thread first sends an HTTP POST request with the phone's device ID and associated smartplug key. After that, heartbeat messages are sent every 300 ms. The second thread listens for SocketIO messages containing job submissions. After receiving a job, the phone returns a `task_acknowledgement`, and then starts execution. All submitted code contains a `main.sh` entrypoint within the zipped folder, which the device invokes.[2] On completion, the phone returns a succeed or fail message.

Our cluster management code is available publicly here: https://github.com/jfswitzer/The_Renee_Project.

V. Evaluation

A. CPU Benchmarking

After replacing Android with Ubuntu Touch, we measure the individual performance of our Nexus 4 development phones across several benchmarks (Table I) and find that they are capable of performing many common computational tasks.

We further compare their performance against a modern (2020) laptop[3], and find that the phones are 9-16x slower than the laptop. This is to be expected, since the Nexus 4's have 8x less RAM than the laptop, and cost approximately 40x less.

B. Response Time

We define the response time of the datacenter to be the time elapsed between job submission to the manager, and the return of a completed result to the end user. We use this metric to compare Renée's responsiveness to that of AWS Lambda, a commercial FaaS provider.

1) Experimental Setup: We use the `fib` benchmark (described in Table I) as our test function.

To run the test function on Renée, we modify our cluster code slightly to fix which phone is assigned the job. We set up the cluster with one of our laptops acting as the end user. We use the bash `time` command to measure the total time elapsed between submitting the job to the manager, and the receipt of the result. We also benchmark the time elapsed for the phone to perform the computation alone, and for it to set up and take down the environment for the computation.[4]

We run the same experiment on AWS by copying our `fib` code into a Python Lambda function that we trigger via a REST API. We present the response time from CloudWatch.

2) Results: Figure 3 summarizes the results. For the `fib` job, Renée's response time is **1.5-1.9x faster** than AWS Lambda. The majority of this time is spent actually performing the computation. The overhead added by cluster management and environment setup/take down adds an additional 0.44-0.76 seconds.

C. Reliability & Recovery

Renée remains available as long as the manager and at least one phone are functional, although the capacity of the cluster scales with the number of phones. When a failure happens midway through a job, Renée recovers quickly, within 3 seconds for a single phone failure. In the case of no active phones, the submitted job is saved until a device comes online. If no phones are registered to the cluster, an error message is returned to the user within 0.6 seconds.

D. Energy Analysis

We measure the energy consumption of one of our development phones for a full cycle of start-up, registration, job

[2]This is similar to the format expected by AWS Lambda, which requires a zip file with a known entrypoint.

[3]A Lenovo ThinkPad X1 Carbon Gen 8 with an Intel(R) Core(TM) i7-10610U CPU @ 1.80 GHz and 16GB RAM.

[4]For our implementation, this means unzipping the received code, creating and deleting temporary folders, and packaging the result.

TABLE I
BENCHMARKING SUITE. INPUT SIZE INCLUDES CODE AND DATA.

	Description	Workload Type	Input Size	Output Size	Laptop Runtime	Phone Runtime	Slowdown
`fib`	Calculates 30th Fibonacci number	Math	190 bytes	3 bytes	0.199 s	2.277 s	**11.44x**
`mean`	Calculates location-based means on energy price dataset.	Data analytics	657MB	1.1 kB	15.35 s	247.31 s	**16.12x**
`resize`	Resizes and stretches the input image	Image processing	186 kB	1.0 MB	0.267 s	2.407 s	**9.01x**
`knn`	Trains a small knn classifier	Machine learning	28 kB	200 kB	0.685 s	10.475 s	**15.29x**

acceptance, and job completion. The results are shown in Figure 4. The phone draws an average of 1.6 W at rest, and 2.4 W while completing a job at 90%+ CPU utilization.

An average server utilization of 20% (AWS' reported utilization) implies a daily energy consumption of 152 kJ per phone. Taking the 2100 mAh battery capacity reported by LG for a new Nexus 4, this would require 5.4 charges or 16.4 hours of charge time to maintain. This also means that the phones are robust to battery capacity decline of up to 30% before the cluster has to lower its utilization rate.

VI. DISCUSSION

A. Global Warming Potential

Global Warming Potential (GWP) represents the greenhouse gas (GHG) emissions associated with a particular action, expressed in $kgCO_2e$ (kg of CO_2-equivalent). When applied to the lifetime of a smartphone, it represents the GHG emissions emitted as a result of that devices' production and activity. Yearly GWP ($kgCO_2e$/year) can be used as a proxy for carbon intensity. We can estimate the yearly GWP of our development phones when they are redeployed in our cluster.

The general formula is as follows:

$$Carbon\ intensity\ =\ \frac{\mathbb{C}_M + C_{use}T}{T} \tag{1}$$

Where $\mathbb{C}_M$ is the carbon associated with manufacturing the device, in units of $kgCO_2e$, and C_{use} is the yearly carbon associated with use of the device, in units of $kgCO_2e$/year. T is the number of years that the device is in service.

We take the manufacturing carbon reported by [3] for the Samsung Z5 and scale it according to weight to get an estimated $\mathbb{C}_M$= 50.31 $kgCO_2e$ for the Nexus 4.

C_{use} is calculated as:

$$C_{use} = P * CI_{grid} \tag{2}$$

Where P is the device's yearly energy use (in kWh/year), and CI_{grid} is the carbon intensity of the electricity used, in units of $kgCO_2e$/kWh[5].

Electricity use is calculated by summing the mean power consumption of all cluster components, and dividing this number by four to get the amortized wattage of each phone.

At a 20% utilization, the per device energy usage is $P =$ 24.18 kWh/year. With a California energy mix, this works out to C_{use} = 6.12 $kgCO_2e$/year.

We find that four years of reuse has the potential to reduce device carbon intensity by as much as 50% (Figure 1).

[5]The carbon emitted per kWh varies across energy sources. For instance, 0.49 $kgCO_2e$/kWh for natural gas and 0.048 $kgCO_2e$/kWh for solar.

B. Limitations

Our approach relies on Ubuntu Touch to provide a desktop-like environment for development. While this is useful for prototyping, it is also limiting, since the Ubuntu Touch project only has support for 65 devices. In the future, it would be preferable to implement Renée on top of a modified Android OS, so that we can support more devices.

VII. CONCLUSION

Our experience shows that even 8 year old phones can support common computing tasks, and that a local cluster built from such devices can provide comparable performance to a commercial FaaS provider. This strategy has the potential to reduce harmful E-waste, and reduce the carbon intensity of the reused devices. As newer phones are released, and newer phones are thrown out, the potential of such a system will only increase.

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)*, 2014, pp. 57–62.

[2] I. Ilankoon, Y. Ghorbani, M. N. Chong, G. Herath, T. Moyo, and J. Petersen, "E-waste in the international context – a review of trade flows, regulations, hazards, waste management strategies and technologies for value recovery," *Waste Management*, vol. 82, pp. 258–275, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0956053X18306366

[3] M. Ercan, J. Malmodin, P. Bergmark, E. Kimfalk, and E. Nilsson, "Life cycle assessment of a smartphone," *Proceedings of the ICT for Sustainability, Amsterdam, The Netherlands*, pp. 29–31, 2016.

[4] M. Shahrad and D. Wentzlaff, "Towards deploying decommissioned mobile devices as cheap energy-efficient compute nodes," in *Proceedings of the 9th USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'17. USA: USENIX Association, 2017, p. 6.

[5] N. Rajovic, P. M. Carpenter, I. Gelado, N. Puzovic, A. Ramirez, and M. Valero, "Supercomputing with commodity cpus: Are mobile socs ready for hpc?" in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2013, pp. 1–12.

[6] N. Klugman, M. Clark, P. Pannuto, and P. Dutta, "Android resists liberation from its primary use case," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 849–851. [Online]. Available: https://doi.org/10.1145/3241539.3267726

[7] "Background optimizations," https://developer.android.com/topic/performance/background-optimization.

[8] "Android benchmarks - geekbench," https://browser.geekbench.com/android-benchmarks.

[9] "Ubuntu touch," https://ubuntu-touch.io/, 2021, accessed: 2021-04-12.

[10] F. Büsching, S. Schildt, and L. Wolf, "Droidcluster: Towards smartphone cluster computing–the streets are paved with potential computer clusters," in *2012 32nd International Conference on Distributed Computing Systems Workshops*. IEEE, 2012, pp. 114–117.