Device Analyzer: Large-scale mobile data collection

Daniel T. Wagner, Andrew Rice and Alastair R. Beresford
University of Cambridge
Computer Laboratory
Cambridge, UK
dtw30@cam.ac.uk, acr31@cam.ac.uk, arb33@cam.ac.uk

ABSTRACT

We collected usage information from 12,500 Android devices in the wild over the course of nearly 2 years. Our dataset contains 53 billion data points from 894 types of devices running 687 versions of Android. Processing the collected data presents a number of challenges ranging from scalability to consistency and privacy considerations. We present our system architecture for collection and analysis of this highly-distributed dataset, discuss how our system can reliably collect time-series data in the presence of non-reliable timing information, and discuss issues and lessons learned that we believe apply to many other big data collection projects.

1. INTRODUCTION

In the Device Analyzer project we are building a dataset which captures real-world usage of Android smartphones. The intention is to share this dataset with industry and other researchers in order to better inform product development or research directions.

We have been collecting detailed usage information in the wild for nearly 2 years from 894 types of devices running 687 versions of Android. Over 12,500 users from 167 countries installed a copy of the software from the Android market and consented to their data being collected. In total, our dataset covers over 1,450 phone-years of usage, with days of inactivity removed. 10,320 of our participants contributed for at least one day, 3,680 users contributed more than one month of usage information and over and 820 participated for at least six months. The dataset contains a total of 53 billion data points.

Device Analyzer captures a time-series log of more than 200 different events. As much detail as possible is captured. For example, Device Analyzer not only records when a device connects to a wifi access point; it records all the details captured whenever a wifi scan occurs, including AP MAC address, SSID, signal strength, frequency and capabilities. By way of example, events recorded include changes to device settings (33 event types), installed applications (17), system characteristics (29), bluetooth devices (21), wifi networks (11), disk storage (6), charging characteristics (5), telephony (20), data usage (10), CPU and memory information for each running app and background process (11) and many more. A complete list of collected data is available on the project website.[1]

Processing the information collected by Device Analyzer and extracting higher-level insights from the large corpus of real-world usage data presents us with a number of challenges ranging from scalability to consistency and privacy considerations.

[1] http://deviceanalyzer.cl.cam.ac.uk/keyValuePairs.htm

In this paper we present our system architecture for collecting data from a large number of distributed sources that is resilient against failures of devices. There are six conceptual components to our system and we describe the operation of each and some of the problems which arise (Section 3). We explain how offline processing can recover wall clock time where traditional collection methods would fail in the presence of errors during collection or user interference (Section 4) and how our architecture accommodates this. We conclude (Section 5) with a brief discussion of some of the more general issues and lessons learned which we think might also apply to other data collection and analysis projects. We are interested in identifying productive areas of overlap.

2. BACKGROUND

Many previous projects have made use of volunteer contributions to large scientific projects. SETI@Home [1] is an early example; FoldIt [6] is a recent one in which researchers pose a computationally hard problem—protein folding—as a game which volunteers can play on the Internet. In SETI@Home and FoldIt, data collection is centralised, and processing is distributed. In Device Analyzer, data collection is decentralised, and data processing is (largely) centralised, leading to a few key differences. For example, there are no real privacy problems in SETI@Home, as data does not contain personal information; similarly, as data collection is centralised in SETI@Home, the project does not experience several difficulties we have experienced in determining an accurate estimate of time. There are similarities too: both SETI@Home and Device Analyzer run at the mercy of volunteer computers of dubious provenance, and both projects have experienced difficulties with centralised components. SETI@Home removed reliance on a database in favour of storing data in flat files for some parts of their design; we did so as well in Device Analyzer.

Previous projects have collected data from smartphones. Examples include the MIT Reality Mining dataset in which 100 Nokia 6600 mobile phones were given to undergraduates [4] and a recent study of application usage of 4,000 Android smartphones [2]. The Nokia Mobile Data Challenge [8] collected various information from 200 Nokia N95 phones over the course of a year. Girardello and Michahelles studied installation and removal of Android applications on 19,000 devices [5].

We believe that the very wide range of data collected, combined with the length of data collection, sets Device Analyzer apart from previous studies looking at usage of mobile devices. Most participants have given us permission to make their data available to other researchers. We intend to release the dataset in the near future. To the best of our knowledge, this is the largest, and most detailed, dataset on smartphone usage to be made publicly available.

3. DATA COLLECTION AND PROCESSING

Our data collection and processing system can be viewed as six conceptual components: measurement; on-device processing; collection; server-side soft real-time analysis; archival storage; and server-side offline analysis. In this section we explain the operation of Device Analyzer with reference to these components and highlight some of the principles which might apply more generally to similar projects. We refer back to these components in our later discussion of design choices.

Measurement Data in Device Analyzer is measured by an application running on Google Android smartphone handsets. We distribute this as a free application on Google Play. The Device Analyzer application registers with the operating system to receive notifications when various events occur on the handset. A huge variety of information is available in this manner with notifications ranging from incoming or outgoing calls or texts and installation of new applications, to changes in volume settings. Some metrics such as the data counters on network interfaces are not available through a publish-subscribe interface and so these are polled at a 5 minute interval. Reliably measuring and recording this information across an open population of devices is a significant engineering effort: storing data using the platform-provided SQLite layer seemed appealing at first glance but having to work around issues with multi-threaded database operations on many devices and occasional data corruption on other handsets led us to simply storing data in flat files. Compressing these files (using gzip) requires care because some handsets have shipped with a compression library that occasionally (and silently) discards data by truncating files. These problems are specific to the Android platform but we expect any project running for an extended period of time with large numbers of data collection devices to be plagued by similar issues.

Data is stored as key-value pairs. Both values are plain-text and can contain (practically) arbitrarily long data. A single data point may contain as little information as the signal level of a WiFi access point or as much as the timestamps of all images in the device's photo library. The keys themselves are organized in a hierarchical structure to allow for prefix- matching during the analysis phase.

On-device processing In order to provide feedback and overview statistics about their device usage to the participant, the application processes data on the device itself. These statistics include the duration of phone calls, number of texts sent and received, historic battery level, and many more. In this stage we also remove direct personal identifiers and other sensitive information using a salted hash function (see Section 5).

Collection Building a dataset means that measured information must be collated at some central point. The Device Analyzer application batches measurements and attempts to periodically upload them to a server using HTTP over SSL. We add a strong checksum on every batch of data since we have seen transmission errors overcome the inbuilt checking in TCP/IP.

Due to the resource-limited nature of mobile phones we delay uploads until the phone is attached to a charger; users can further elect to upload only over WiFi connections. The application is designed to store data until it has been delivered (and receipt confirmed). If a preset maximum amount of data is stored we suspend data collection until the application was able to upload data.

Archival storage The principle task of the server process is to reliably receive and record the measured data from devices. We use a simple ARQ protocol with back-off to recover from transmission errors. Valid batches of measurements are appended to a flat file for the device in question. Duplicated data produced by repeated transmissions from the client are discarded at this point. New device files are started when the previous one reaches 10MB. Old files are compressed and moved to a permanent repository location.

Server-side soft real-time analysis Live statistics have proven to be extremely useful to the project. We provide information as to the current and overall number of participants to all users and the Device Analyzer website shows a dynamic map of the world showing uploads as they happen. We have made much use of these when presenting the projects to others as a recruitment strategy, but also as an indicator of overall system health when the map is blank. We currently compute these statistics as simple filters which are executed as incoming data arrives. Crucially, online processing does not interfere with the primay task of receiving device uploads, and data may be silently dropped in the presence of errors. We may increase the range of live information we provide to participants in the future as a way of better rewarding their participation.

Server-side offline processing During the offline phase we process all archived files of a given device in order and feed the data tuples to a directed graph of stateful processing plug-ins. Each plug-in exposes its state for other plug-ins to exploit. For example, the screen plug-in tracks "screen on" and "screen off" events in order report the state of the device's screen at any point in time to other plug-ins which list it as a dependency, e.g. when measuring data transferred while the screen was on. Prefix matching of keys allows us to quickly filter relevant data for a given plug-in.

Some of our work on the Device Analyzer dataset requires us to run simulations of device activity with a large number of varying parameters. We decided to implement these simulations as jobs for Apache Hadoop. We make use of the independence of measurements between devices: One job reads the output of the plug-in stage for one device only and uses the included data to run a simulation. The job outputs a set of results for each combination of parameters that it evaluated. Hadoop makes it very easy to aggregate these results across all devices based on parameter values for the individual simulation runs. Our simulations typically run between one minute and one hour per device, depending on the nature of the simulation. While Apache Hadoop was not designed to run these types of workloads on time-series data, we found it to be an easy-to-use framework that abstracts away much of the complexity normally associated with distributed computing.

The final stage of our analysis deals with generating human-readable statistics over the previously generated data. This may take the form of textual or graphical representations. We typically generate graphs and summative statistics using short ad-hoc scripts written in Python that parse the output of the plug-in or simulation stages to create graphical representations using matplotlib. In this final stage we typically use only a few megabytes of input data, down from several terabytes of raw data collected.

4. TIMESTAMPING MEASUREMENTS

Android exposes a number of different clocks to the programmer: *uptime* is the timespan since the device was last turned on; it does not count time while the device is off. The *real-time clock* continues counting time while the device is off and tracks UTC. *Wall-clock time* is a real-time clock with attached time zone information; it is used for displaying the local time to the user. External time sources can be used to set the local real-time clock to a known good value: *Network time* can be obtained by querying a server for a reference timestamp and *GPS time* is a highly accurate source of timing information available on many Android devices.

The most immediately appealing of these clocks is *wall-clock time*, as it represents time the way the user experiences it and al-

lows us capture diurnal patterns. However, both *real-time clock* and *wall-clock time* are subject to automated changes from the cellular network or a network time source, or manual changes by the user. An application using these clocks would record time in a non-linear fashion which may include overlaps if the time was set back. In particular, we have frequently observed devices reporting a date in the 1980s for a short period of time when they first start up after an operating system upgrade.

To avoid this phenomenon, we timestamp every measurement with the device's *uptime* in milliseconds. To know the user's local time, we additionally store the *wall-clock time* when Device Analyzer starts, and record all adjustments of the *wall-clock time* when they occur. Android helpfully provides notifications for this.

Notably, we observed thousands of jumps in the system-reported *uptime* across our dataset. These jumps may be a number of minutes to many tens of thousands of years either into the future or the past. The vast majority of jumps are transient and we can easily recover the correct *uptime*. However, a small fraction of jumps (roughly 1.5%) are not transient and continue increasing monotonically after the jump. As an additional safety measure we periodically log the *network time* obtained from our server. *GPS time* provides better accuracy, but requires the app to ask the user for GPS permissions, which we chose not to do.

We believe that constructing a reliable time source from different unreliable clocks is a challenge that many other distributed logging applications face. Indeed, Oliver found in 2010 that time synchronization "is the most problematic challenge to overcome in autonomous logging." [9]

We use a plug-in during the server-side offline processing stage to reconstruct a valid *wall-clock time* for any point in the dataset, based on the *uptime* measurements of every data point, the *wall-clock time* adjustment events we have collected, and the periodic *network time* measurements we collect. The reason for doing offline processing is that we can improve algorithms and re-compute timing information after the data was collected.

Figure 1 visualizes the relationship between *uptime* and wall-clock time. Yellow dots indicate *wall-clock time* reference points that are stored on application start and when adjustments are received. Line segments indicate the time for which the device was active without restarting ("session") and connect sequential time reference points or continue with gradient one until the device turns off. Ideally, both sources of time agree perfectly and for each second that *uptime* increases, we observe *wall-clock time* increase by one second as well. In reality minor corrections will be needed due to clock skew, which materialize in the graph as lines with gradients that are slightly different than 1.

We generate a global time frame per session by finding the best-fit line with gradient 1 and checking how much the best-fit without the furthest outlier differs. If removing the outlier makes the best-fit line move more than 30 seconds in *wall-clock time*, we remove the outlier and repeat the first step. Once the fit moves only little, we use it to generate *wall-clock time* references. Figure 1 shows an obvious case for one such outlier (marked red).

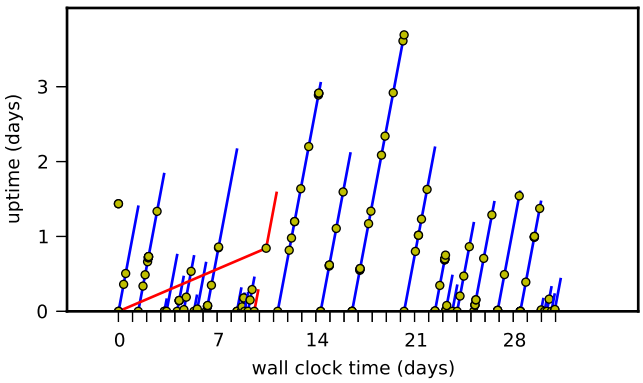

Figure 1: Relationship of device uptime against wall clock time. Yellow dots indicate time reference points. Red segments belong to an erroneous time after boot and need to be corrected.

5. DISCUSSION

In this section we discuss aspects of our work which we believe address issues that the majority of big data projects need to handle in one way or another. We hope that this discussion can help to identify projects with common goals and provide a stepping stone to share code, architectual ideas or resources.

Privacy Smartphones contain a lot of private data. As researchers we have a duty of care to our participants and aim to minimise any intrusion. Our approach is based on both the European legal framework, and the seven principles of *Privacy by Design*, an inclusive approach to data projection written by the Information Commissioner of Ontario, Canada. Our approach is compatible with an earlier set of recommendations made to researchers in ubiquitous computing [7].

Transparency, consent, and purpose Our data collection is both transparent and explicitly given by the participant: Device Analyzer is distributed as a stand-alone application on the Google Play store; it is never bundled with other software or pre-installed on devices. We require consent inside the application to activate data collection and remind the user of on-going data collection via a monthly notification. Data collection can be suspended at any time inside the application.

Security Communication between the Device Analyzer application and the server is secured using TLS and data is securely stored on a server hosted by our department.

Access and withdrawal Participants can access data collected from their smartphone. Due to space constraints the full archive is not available on the device itself. Instead, it can be downloaded from the project website. Participants can delete their data and withdraw from the study at any time from within the application.

Accountability We provide a "quick feedback" feature inside the application to allow participants to send feedback without revealing their email address. We also provide a working email address on the project website.

Proactive privacy Device Analyzer is designed with a focus on metadata rather than personal information. For example, we do not collect an audio recording of a phone call; we record time, duration and phone number of the call. Before any data is uploaded to our server, we preprocess and remove direct personal identifiers and other sensitive information using a hash function with a salt randomly chosen during installation.

Privacy by default Participants must explicitly opt-in to reveal particularly sensitive data items. For example, GSM cell tower identifiers are, by default, preprocessed by a salted hash function on the device itself, and any location data collected is, by default, only available for use by researchers at the University of Cambridge.

Data collection experiences Low-level *raw data* is stored whenever possible, rather than interpreting data on the device and storing the resulting high-level data. For example, we collect the number of bytes transmitted over a network interface as the OS reports it, including the 32bit wraparound. This allows us re-extract informa-

tion from the original raw data later if we encounter a bug in our interpretation code.

Resource use: Users are intimately familiar with their devices and are quick to uninstall applications that reduce battery lifetime or consume large amounts of disk space. However, rather than judging by absolute numbers, users evaluate apps against other apps that are already installed. The OS provides a list of applications that use lots of disk space or power. Where an application ranks on these lists depends therefore heavily on usage patterns.

We designed Device Analyzer to have a very low resource footprint, with a typical power drain reported as 2% of overall device power consumption. We also limit the amount of disk space Device Analyzer uses, suspending data collection if necessary.

Error handling: An early version of Device Analyzer prompted users to intervene when uploads failed. When a provisioning issue caused our server to be unavailable for several days, the majority of participants uninstalled the "broken" application; some left bad reviews on the Play store. Later versions do not prompt the user but rather try uploading more aggressively.

Trade-off of implementation vs computation complexity Our analysis framework has no notion of persistence of state between runs and earlier computations are repeated when a new archive file is added, even when that would not be strictly necessary.

Previous versions of the analysis framework persisted the internal state of plug-ins and would restore the state when new data arrived to avoid re-computing older data. While this technique meant that incoming data resulted in much reduced compute times, we found that state persistence and the necessary heuristics to determine which data needed to be re-computed quickly became difficult to manage. This is particularly true if we account for actions that influence earlier data, such as when the user changes the time on the device, invalidating previously computed results.

Re-computing all data allowed us to greatly simplify plug-ins and management code in exchange for increased compute times.

Converting event streams to state models Device Analyzer logs events like "screen on" and "screen off" or "wifi access point x is visible". For most analyses we need to convert a device's event stream into a state model that changes in response to events as time progresses. As we assume independent devices, our plug-in system (Section 3) achieves this in a modular way. This allows us to measure durations that a certain condition was satisfied for, or to combine the state of multiple plug-ins, e.g. to capture visible wifi access points while the screen was on.

The general case involving dependent measurements is harder and it is unclear how the corresponding architecture might look like: One might imagine a system that steps through all data sources in sync with a global time frame, or an iterated solution that starts off assuming independent measurements and then converts towards a global solution by including changes of neighbouring devices as they propagate through the system.

Data provenance As discussed above, we created an ad-hoc solution to reduce the amount of re-computation needed when new data arrived. Our solution leveraged the dependency graph of plug-ins to re-compute results from the raw data. However, we found this to be insufficient as a holistic solution would need to capture the true provenance of each computed data item in the form of data that this item depended on. We would like to investigate generic provenance APIs which would allow us to capture this information more systematically [3]. This would allow us to easily determine when to recalculate results and also to provide better transparency to users of the derived data.

Collaboration with other researchers We are very much interested in collaborations with other researchers. We are currently evaluating a mechanism that allows interested parties to recruit a set of participants and gain access to their raw data stream, allowing them to re-identify the recruited individuals in the Device Analyzer dataset. We are also interested in questions that the research community would like to answer using our dataset. To that extent, we are preparing to publicly release the dataset and are collaborating with researchers to investigate questions of mutual interest. However, due to the nature of the collected raw data streams, extracting certain types of high-level information may require significant coding effort as the common-sense understanding of a problem may translate into non-trivial states and conditions that need to be understood when looking at the data in fine detail. We would appreciate any input that other researchers may have on this issue.

6. CONCLUSIONS

Device Analyzer is a large-scale study of over 12,500 smartphone users in the wild. We have described our collection and processing architecture in terms of easurement, on-device processing, collection, server-side soft real-time analysis, archival storage and server-side offline analysis. Processing timestamps shows a good example of the problems we faced in terms of information collection and recovering from faulty data. Our plug-in architecture allows us to easily integrate refinement steps such as this one into our processing system. Our experiences suggest some more general questions which might apply to other similar projects. We are interested in identifying these projects and investigating whether our infrastructure can be applied more generally.

7. ACKNOWLEDGEMENTS

We would like to thank Andy Hopper for his insight and support. We also want to thank all participants of our study without whom this work would not have been possible. A list of contributors who supplied their name is available on the project's website.[2] This work is supported by the University of Cambridge Computer Laboratory Premium Studentship scheme, a Google focussed research award and the EPSRC Standard Research Grant EP/P505445/1.

8. REFERENCES

[1] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: an experiment in public-resource computing. *Comms of the ACM*, 45(11), 2002.

[2] M. Böhmer, B. Hecht, J. Schöning, A. Krüger, and G. Bauer. Falling Asleep with Angry Birds, Facebook and KindleA Large Scale Study on Mobile Application Usage. *MobileHCI*, 2011.

[3] L. Carata, A. Rice, and A. Hopper. IPAPI: Designing an Improved Provenance API. *TaPP*, 2013.

[4] N. Eagle and A. S. Pentland. Reality mining: sensing complex social systems. *Personal and Ubiquitous Computing*, 10(4), 2005.

[5] A. Girardello and F. Michahelles. AppAware: which mobile applications are hot? *MobileHCI*, 2010.

[6] F. Khatib, S. Cooper, M. D. Tyka, K. Xu, I. Makedon, Z. Popovic, D. Baker, and F. Players. Algorithm discovery by protein folding game players. *PNAS*, 108(47), 2011.

[7] M. Langheinrich. Privacy by Design - Principles of Privacy-Aware Ubiquitous Systems. *UbiComp*, 2001.

[8] J. K. Laurila, J. Blom, O. Dousse, D. Gatica-perez, O. Bornet, and M. Miettinen. The Mobile Data Challenge : Big Data for Mobile Computing Research. *PerCom*, 2012.

[9] E. Oliver. The challenges in large-scale smartphone user studies. *HotPlanet*, 2010.

[2] `http://deviceanalyzer.cl.cam.ac.uk`