Stateful Data Analytics Using Logs from the Edge Processed in The Cloud

Pervasive Computing Published by the IEEE Computer Society XXXX-XXXX

Department Head

T. Pasternak
Motorola Solutions Inc

D. Langerman
Motorola Solutions Inc

S. D. Tine
Motorola Solutions Inc

A. Faingold
Motorola Solutions Inc

Y. Munk
Motorola Solutions Inc

***Abstract*— We aim to collect and analyze logs from embedded devices for the purpose of generating behavioral usage statistics and insights. This paper describes a cloud-based platform dedicated to the collection and storage of logs from body-worn cameras used by public-safety officers and their analysis and visualization. Our contribution is two-fold. We achieve rapid introduction of new metrics by focusing in deployment in the cloud using industry-standard open-source tools. Secondly, we present an approach to elicit stateful behavioral information from the logs.**

⬛ **Understanding** how a product is really used by customers is of supreme importance to product management. However, hard data on product use is not always available for end-user devices. Using devices logs uploaded to the cloud we obtain usage statistics on the use of body-worn cameras which are used by public safety officers. These statistics can help product managers understand user needs. Since our solution is cloud-based, it is device-agnostic and does not require any firmware support apart from upload of the logs. The kind of data that we are looking for is platform-oriented data about the duration in which various hardware components are powered on and are residing in different modes of operation. One could take the approach of standardizing the device logs and requiring firmware component developers to add logging according to specific requirements. While this approach is helpful in ensuring required data is available in the log run it has the distinct disadvantage of having to go through a firmware development cycle for each new metric that is introduced. For that reason we take a server-side approach in which we focus on parsing the existing logs. Debug logs usually contain a vast amount of information so that the information of interest already exists in the logs, it only needs some pre-processing before it can be indexed and used to generate reports and dashboards. Our implementation used the open-source ELK stack (Elasticsearch Logstash, Kibana) and the said pre-processing is done in Logstash.

USES OF LOGS AND DATA ANALYTICS

Usage monitoring

Content providers use data analytics to determine the exposure of their content to consumers, providing ratings of shows and pricing of advertisements.

For application providers, data Analytics can be used by to monitor the use of an application and each of its features or user-interface elements, thereby indicating its relative value to users.

Usage statistics can be generated online by an application logging each user action into a server as it occurs. Such actions may be internally cached in case a server is not immediately available. For example, Visual Studio App Center Analytics from Microsoft provides a 10MB cache for this purpose [1].

In case of a device that is only sporadically connected, the data can be written to logs files on the device.

Usage monitoring frequently focuses on counting events per timeframe such as, for example, the number of clicks per day on a particular web link. Counting lends itself to parallel execution in the cloud due to the associativity and commutativity of addition [2].

Anomaly detection

Anomaly detection is the process of identification on non-typical observations which may indicate a fault in the system under observation.

A simple mechanism of anomaly detection is to collect success and error responses to requests and to display them on a dashboard. The number of errors of each type within a particular time frame (e.g. the last day) is displayed in a pie chart, providing decision makers an indication of the health of the system and of outstanding problems.

Usage Monitoring can also provide indirect anomaly detection. If a particular function within the system is suddenly triggered much more or much less frequently than usual, this may indicate a problem. For example, if no purchases are made through a particular web page it may be the case that the page is not reachable due to another problem in the system.

Prediction

Log data can be used to track performance metrics for the purpose of predicting trends and provisioning for the future. For example, by monitoring latency on server response times, scalability limitations can be identified and mitigated. [3]

Root-cause analysis and debugging

Root cause analysis is the process of analyzing observed behavior to determine the earliest event or events in a causal link leading to an observed anomaly. In software debugging, system logs are often used for the purpose of root-cause analysis.

Stateful usage data, the next challenge

For some uses, merely counting the number of times an event happened in a given timeframe is not sufficient. We want to know the duration that the system is being in each particular state.

In contrast to applications, platform requirements are more often quantitative performance requirements sometimes known as non-functional requirements. To gain insight relevant to resource usages – power consumption, for example – we would like to know how long the system stays in each state.

For this we need to be able to identify the entry and exit to each state, take note of their timestamps, compute durations and compute cumulative durations.

EMBEDDED DEVICES PROBLEM SPACE

Embedded IoT devices have some typical characteristics that affect how their logs can be processed.

Not always connected

For mobile devices, power consumption remains an issue, and in some countries, notably China and North America the cost of a data plan is still around $10 for

1GB [4], for both of these reasons, there are mobile devices that are not equipped with connectivity to a cellular network and are only connected by WiFi. Consequently, although more devices are connected to the Internet than ever, there are also many devices that are not connected or are only intermittently connected.

Limited space and compute

Because of the capital investment in hardware and the cost of replacing and distributing hardware, embedded systems tend to continue to be used until the hardware resources can no longer sustain the increasing demands of the software. For this reason, for much of the lifetime of an embedded device in the field it is an aging device, running on resources stretched to the limit.

Logging requires both compute and storage requirements, both of which may be scarce. Any project requires a significant increase in logging may hit a barrier in terms of its resource requirements and be deemed infeasible.

Complex integration

Typically, an embedded device, such as a mobile phone, set-top box, etc. will have several major software providers involved in its development: the SoC vendor (e.g. Qualcomm, Broadcom etc.) who provides the reference design including the operating-system kernel and hardware abstraction layer, the hardware manufacturer who adds additional hardware required for the particular product, provides drivers for the additional hardware and customizes the software provided by the SoC vendor, the middleware provider (e.g. Google who provides Android), the system integrator who is responsible for the product as a whole, and application providers who provide applications that may run on the given platform as well as on other platforms. In some cases multiple roles are taken up by the same party, but even then there are still multiple companies involved or at the very least different organizations within a company.

Owing to the multiple parties involved as well as the nature of the software development and testing tools and environments needed to create a software build for an embedded system, the cycle time from the when a feature requirement is presented to the development team until it is deployed in the field can be quite long [data ?].

No standard logging

Different standards for logging exist. For example The Syslog Protocol, standardized as RFC5424 [5]. However, having such standards is only a preliminary step towards enabling the processing of such logs. For one, there is not one globally accepted standard, but multiple standards, that is to say no standard. And secondly, the standards define such things as the timestamp format and a header identifying the source of the event and its type, but the actual content of the event is usually left to a message field that “contains a free-form message that provides information about the event.”[5]. Thirdly, the standards define syntax but not semantics.

A particular challenge is that managing a single dictionary is difficult when multiple parties contribute to the logs. For example, if one wants to assign a unique event id to each event type across all layers of the firmware, this requires coordination between the SoC vendor, the hardware manufacturer, the system integrator and the application providers. But the SoC vendor provides the same firmware to many other platforms, and the applications also may run on other platforms, making this process of coordination nearly impossible.

COMMAND CENTRAL

CommandCentral is Motorola Solutions’ end-to-end suite of command center software, which includes CommandCentral Vault: a cloud-based solution for aggregating and organizing public safety agencies’ digital content in one place.

Body-worn cameras that upload video

The cameras connect to the Internet from time to time to upload videos. It is possible to use this opportunity to upload the logs as well

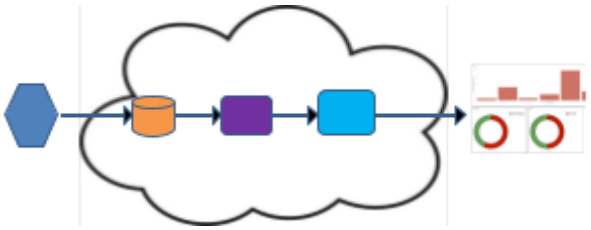

Figure 1. Deployment Architecture

PRODUCT MANAGEMENT

Product Managers are responsible for feature prioritization and road map planning. In order to specify features and assign their relative priority certain things must be known or assumed about how a product is being used. For example: making changes in the software that result in a modest improvement in power consumption makes sense if that modest improvement makes the difference between having to carry a spare battery or not. In order to know if this is the case, the product manager must know how the users actually use the device in the field, and how their usage patterns affect power consumption. Without real data, product managers are limited in the quality of their decision making.

SERVER-SIDE APPROACH TO CLIENT LOGS

This section will explain our approach to generating stateful information about device use by processing the existing logs in the cloud, rather than introducing changes in device software to produce the data explicitly.

Development velocity

Any update to firmware requires extensive testing and a lengthy process of release.

When a new metric is required, rather than changing the firmware to explicitly add the required metric, the existing logs can be processed in the cloud to elicit the required metric from the existing information.

We rely on existing debug logs. Debug logs generally contain a wealth of information, since every action is logged. Generating the desired information from the data may require some effort.

For example, we would like to know what percentage of the time WiFi is on in the device. Rather than adding a requirement for the firmware to keep track of the cumulative time in which WiFi is on and off and output the ratio to the log, our approach is to search the existing logs for events indicating, directly or indirectly, the turning on or off of WiFi and use these events to calculate the ratio.

Log analysis in the cloud enables fast introduction of new metrics because no update to firmware is necessary.

Generating metrics retroactively

Relying on existing logs content and format enables the introduction of new metrics and its application retroactively to historical data. In contrast, relying on the addition of explicit logging would apply only from the time of introduction of the new metric.

PROCESSING STEPS

The following steps are implemented in Logstash by customizing the Logstash configuration file.

Ingesting the log file from storage

The devices upload their log files to cloud storage. From there they are ingested into Logstash, where they are uncompressed and parsed.

Parsing into fields

The first step of parsing is to dissect each line in the log file into its constituent fields according to the log format used.

Re-mapping the timestamp

In general each record reported in Elasticsearch is given the timestamp of the time of its reporting. For events generated from device logs, special processing needs to be performed to assign them the correct timestamp.

The log files contain events that were reported hours or days before the time they are ingested into the system. The timestamps need to be read and remapped to the Elasticsearch timestamp metadata field or another field that will be used as the timestamp.

Dealing with gaps in the log

The log file buffer for in embedded devices is limited, typically providing for a few days of logging, long enough to enable troubleshooting a problem by analyzing the logs. If logs are not uploaded and the buffer is filled then the start of the log will be lost so that between the previous log file and the present one there is a gap. When processing the log, this possibility should be taken into account.

Dealing with overlaps in time

When the buffer is not filled up, it still contains the part of the log that was already uploaded previously. This causes duplicate records. Elasticsearch identifies the duplicates and discards them.

Dealing with timestamps when clock not initialized

Many devices have a coin cell battery that provides backup power when the main battery is depleted, so that the system clock can continue to run. If this is not the case, then when a device is turned on, if its battery is totally depleted then the system clock reverts to an uninitialized state, usually showing the Unix epoch.

The events with such a timestamp will be placed on an invalid date in the timeline and should be filtered out when performing queries (e.g. by setting the date range in Kibana).

Filtering the relevant events

Debug logs contain huge amounts of data, as every meaningful action taken by the software is logged. In analyzing the log, only events of interest are indexed in the log database. In the context of the ELK stack, this means filtering out unnecessary data in Logstash.

Inference of State

We assume all states of interest are observable by the way of log events that indicate state transitions from which the entry and exit to each state of interest can be inferred. However, the transitions may be implied by transitions at a higher level, rather than explicitly recorded in the logs.

Dealing with device power on and off

We conceive of the system as having a hierarchical state space, such as that of communicating hierarchical state machines [6], in which a high-level state transition also implies transition from the final state of a low-level state machine included in the previous state to an initial

state of a low-level state machine included in the next state.

An example of this scenario is when the device is turned off and all the components in it are implicitly also turned off, while the log may only indicate a global '*off*' event.

This means there is a one-to-many relationship between events recorded in the log and observations noted by log analysis. One event recorded in the log e.g. "device off"may imply multiple observations, e.g.both the device and all its components are turned off.

Calculating the duration in each state

The duration in each state of interest is calculated by taking the difference between the timestamp of entering and exiting the state. By aggregating and filtering the durations, statistical insights can be gained on the distribution of the time that each component resides in each state, thereby characterizing the usage of the device.

Example

In the following example we illustrate how a log is processed. We have a device component that includes two components: Bluetooth and GPS (**Figure 2**).

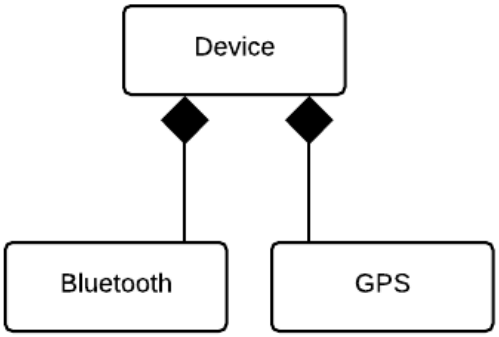

Figure 2. Component Model

Each component is associated with a list of states.
Device States: {Device_on, Device_off }
BT States: {BT_on, BT_off }
GPS States: {GPS_on, GPS_off }

The hierarchy of the state space is based on the composition hierarchy of the components. At the top level are the Device States. Under the state Device_On are two concurrent state machines: one for BT and one for GPS. In each state machine one state is marked as initial and one (possibly the same one) as final (**Figure 3**).

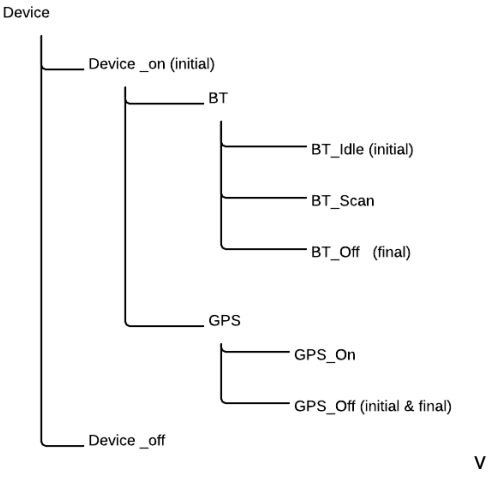

Figure 3. Hierarchical State Space. BT and GPS are concurrent state machines contained in state Device_On. When entering Device_On state, the system implicitly enters GPS_Off and BT_Idle.

CONCLUSION

In this paper we presented an approach to generating stateful information from device logs without making changes to the device firmware. Within the context of the ELK stack, our implementation used Logstash as the locus of log processing. Future directions include using traces to gain insights about system behavior.

REFERENCES

1. Microsoft App Center Analytics Website (https://docs.microsoft.com/en-us/appcenter/sdk/analytics/android#local-storage-size)
2. G. Shroff, *Enterprise Cloud Computing: Technology, Architectire, Applications*, Cambridge University Press, pp. 139, 2010.
3. M. N. Vora, "Predicting Utilization of Server Resources from Log Data," *International Journal of Computer Theory and Engineering,* Vol. 6, No. 2, April 2014.
4. Forbes (2019, March), "The Cost Of Mobile Internet Around The World," [Online]. Available: https://www.forbes.com/sites/niallmccarthy/2019/03/05/the-cost-of-mobile-internet-around-the-world-infographic/#7d47cee5226e
5. Network Working Group, Request for Comments: 5424 The Syslog Protocol [Online].Available: https://tools.ietf.org/html/rfc5424#page-17
6. R. Alur, S. Kannan, M. Yannakakis "Communicating Heirarchical State Machines," Proc. International Colloquium on Automata, Languages, and Programming, pp 169-178, 1999.

Tal Pasternak is a System Architect at Motorola Solutions Inc. and an adjunct lecturer at the Hebrew University in Jerusalem. Dr. Pasternak received his B.S. degree from the Technion, Israel Institute of Technology, and his M.S. and Ph.D degrees from Vanderbilt University. He is a member of the IEEE and the IEEE Computer Society. Contact him at tpasternak@computer.org.

Dafna Langerman is a Senior Software Engineer at Motorola Solutions Inc. Dr Langerman received her PhD. from Bar Ilan University. Contact her at dafna.langerman@motorolasolutions.com.

Steven Tine is a Product Manager at Motorola Solutions Inc. He received his B.S from Boston University. Contact him at steve@tinefamily.com.

Yonadav Munk is a Software Engineer at Motorola Solutions Inc. He received his B.S from Bar Ilan University. Contact him at yonmunk@gmail.com.

Alisa Faingold is a Software Engineering Intern at Motorola Solutions Inc. She is studying for a B.S at Ben Gurion University. Contact her at alisa.faingold@motorolasolutions.com.