Research Topic: Telemetry/Logging

Focus sub-topic: Logging and Debugging Methodologies

1. Introduction

Telemetry is the process of gathering the performance data of any product and communicating it to a remote location for monitoring and analysis. Telemetry helps the developers stay aware of the performance of the software and notifies them if a problem occurs with the application. It plays an important role in ensuring that the end-user gets the best version of the application. Once an application is introduced in the market, it gets scattered across various parts of the world. At that point, it becomes nearly impossible to keep track of all the application instances and ensure that they are all working at optimal performance. Telemetry gives you the tools to keep track of performance from all outlets, no matter how remote the end user's location is.

Telemetry encompasses all the data collected in the system - including logs, performance metrics, and even trace information etc. It's used for proactive monitoring and identifying potential issues before they become problems. Logging is more for troubleshooting and debugging after an issue has occurred, where the focus is on recording events and messages within the system.

Logging is a widely adopted practice in software engineering. I personally believe that no large-scale application can function efficiently in the long term without logging. When several teams are involved in developing features for the applications, and many customers are using the application, there are bound to be bugs or issues that must be triaged and resolved. There is no way to debug and trace the root cause of these kinds of issues without proper logging, especially if they are not replicable on the developer's end.

Log messages, which are generated at runtime by logging statements that developers inserted into the source code, present rich details about the runtime behavior of software applications. The valuable information in logs is leveraged in various software development and operation activities including bug fixing, test results analyses, anomaly detection, and system monitoring. Due to the limited resources on mobile devices, making optimal logging decisions is challenging. For example, logging too much may cause additional performance overhead, which may lead to both slow response of the mobile apps and additional battery consumption.

2. Industry trends and needs

Logging is necessary to maintain a high-quality user experience and improving app functionality. Various tools for logging provide the following solutions -

- Collect and analyze logs in real-time to identify and address issues quickly. This is critical for maintaining a high-quality user experience and reducing crashes.
- Logging solutions that work seamlessly across all platforms, providing a unified view of app behavior regardless of the device.
- Understanding how users interact with the app by capturing user actions, track events, and measure performance metrics.
- Lightweight solutions that minimize overhead and battery drain on mobile devices.

But logging comes with its own set of challenges. Simply enabling logging and adding a bunch of logging statements within the code does not serve the purpose. Some of the challenges are-

- Log Volume and Complexity: Mobile apps can generate a massive amount of log data, making it difficult to store, manage, and analyze effectively.
- Identifying Root Causes: Logs often contain a lot of information, making it challenging to pinpoint the exact cause of an issue. Developers need tools with advanced filtering and analysis capabilities to identify root causes efficiently.
- Data Security and Privacy: Ensuring user data privacy is paramount. Developers need logging solutions that comply with data privacy regulations and offer secure storage and access controls.
- Limited Device Resources: Mobile devices have limited processing power and storage. Logging solutions need to be efficient and use resources judiciously.

Due to some of these challenges, logging and debugging for mobile apps differs from its implementation in desktop or web applications. During my experience as a web application developer, my primary responsibility was to address issues reported in the app by customers and provide code fixes. I spent a lot of time collect and analyzing logs. For a large-scale application, logging is not enabled by default and only severe types of errors get logged by default. Customers had to enable specific profiles and do some setup to enable logging. These logs would be recorded to different servers based on the service, for example UI Server, Service server for REST API specific logs, and other servers for WebLogic etc. For certain logs, we would also have to involve internal teams for server-side setup for logging. Even after collecting the logs, we would have to look for logs related to our specific issue by searching for the user who performed the operation and packages in huge log files.

3. Current Solutions

Logging solutions vary based on the development platform and the scale of the application. Some common tools used for logging and debugging are-

i. **Remote Debugging:** Traditional in-device debugging is giving way to remote debugging tools. These tools allow developers to inspect and debug apps running on real devices from the comfort of their development machines. This is especially helpful for geographically dispersed teams and testing on a wider range of devices.

Tools like *Chrome DevTools Remote Debugging* integrates with Chrome browser for remote debugging of Android web views and hybrid apps. *Flipper* (for React Native) is an open-source platform for debugging, inspecting, and visualizing React Native apps on real devices.

ii. **Cloud-based Logging:** Storing logs in the cloud offers scalability, accessibility, and advanced analytics capabilities. Developers can access logs from anywhere and leverage tools to filter, analyze, and identify trends in application behavior.

Firebase Crashlytics is one such free service from Google that automatically collects crash reports, user behavior data, and non-fatal errors. *AWS CloudWatch Logs* offers centralized log storage, monitoring, and analysis for mobile apps.

iii. **Crash Reporting and Bug Tracking Tools -** Repetitive tasks like log filtering and identifying common issues are being automated using AI and machine learning. These tools go beyond simple crash reporting by offering features to help developers track, prioritize, and resolve bugs effectively.

Sentry and BugSnag are real-time crash reporting, error tracking, and performance monitoring platforms with detailed error diagnostics.

iv. Open Source Logging Libraries - These libraries provide developers with granular control over how application data is logged.

Logback (Android) is a popular logging framework offering different logging levels and output destinations (e.g., console, file). *Timber* (Android) is a Lightweight logging library with a simple API and focus on crash-safety.

While the above tools provide more than just basic logging features, Android and iOS also provide sufficient centralized logging functionality. Another aspect to consider is that with stricter data privacy regulations, developers also need to be mindful of what data is logged and how it's stored. Techniques like anonymization and user consent management are becoming crucial.

4. Critical Analysis

Solution	Pros	Cons
Remote Debugging	Convenient to debug from anywhere	Latency – Network delays can affect responsiveness
	Allows for wider testing on a broad range of devices	Requires secure connections for remote device access
	Ability to inspect real-time behaviour for fast triaging	

Cloud-based logging	Scalable – can handle large amounts of data efficiently	Cost – storage and analytics features come with ongoing fees
	Accessibility – Access logs from anywhere	Lock-in - Switching cloud providers can be complex if deeply integrated with their logging infrastructure.
	Advanced Analytics – Powerful filtering, analysis and visualization tools for deeper insights	
Crash Reporting and Bug Tracking	Automatically capture and report crashes and errors. Reduces manual developer workload, allowing them to focus on complex problems.	Cost - Pricing plans often depend on the number of apps and events tracked.
	Help prioritize critical issues based on severity and frequency.	Customization options for reports and error handling might be restricted.
	Facilitate communication and collaboration between developers to resolve issues.	AI models require training and may not always be accurate.
Open-Source Logging Libraries	Developers can tailor/customize logging behavior to their specific needs.	Requires developers to manage and maintain logging infrastructure.
	Freely available and customizable, fostering community support.	May lack advanced features for filtering and analyzing log data.
	Lightweight- Libraries are often designed to be efficient and minimize performance impact.	

5. Propose a new solution or an improvement to an existing solution that minimizes or removes a limitation of existing solutions

Automated tools and advanced analytics have proven to be very helpful during triage of issues. These reduce developer workload of having to manually filter through thousands of log statements. But as stated earlier, these automated tools also have some limitations. Especially tools powered by AI, which require training of models, and can still produce inaccurate results.

One improvement to further reduce workload for both the developer and the tools, is to introduce edge computing. Lightweight logging libraries that can embedded within mobile apps to collect and pre-process log data on the device itself, focusing on capturing essential

information and anonymizing sensitive data. This reduces overall size of data to be transmitted and further analyzed by analysis tools. Then, a central server coordinates the communication between edge agents and facilitates collaborative learning. It doesn't store raw log data but utilizes federated learning algorithms to train a global model for anomaly detection and issue identification. This, coupled with a visualization and analytics dashboard to assist developers with a user-friendly interface to view aggregated insights, identify trends, and receive alerts for potential issues and easily filter out data with search terms to look for specific log statements.

This solution minimizes risk to user data as the edge computing would remove any sensitive information from logs, i.e. sensitive data never leaves the user device. It also reduces the load on centralized servers and handles basic log processing, improving scalability. Edge agents can enhance offline functionality by buffering logs and transmitting them later when an internet connection becomes available. Moreover, this solution would aid in the quick resolution of bugs and issues reported by customers in large-scale commercial applications, so that developers can provide quick fixes with minimal impact to their business processes.

6. Citations and links to examples

i. https://www.logicmonitor.com/blog/what-is-telemetry
ii. https://www.logicmonitor.com/blog/anomaly-detection-for-devops
iii. https://petertsehsun.github.io/papers/fdroid_emse2019.pdf
iv. https://gemini.google.com – for understanding, content, ideas

Tools/tech examples cited -

v. https://developer.chrome.com/docs/devtools/remote-debugging
vi. https://fbflipper.com
vii. https://firebase.google.com/docs/crashlytics
viii. https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/WhatIsCloudWatchLogs.html
ix. https://sentry.io/
x. https://www.bugsnag.com/
xi. https://logback.qos.ch/documentation.html
xii. https://github.com/JakeWharton/timber
xiii. https://developer.android.com/reference/android/util/Log.html
xiv. https://developer.apple.com/documentation/os/logging