# Title for Submission to Eurosys 2018

Paper #XX

NN pages

## Abstract

MacOS is well known for its friendly user experience. However, performance bugs still exists, which annoys users. For instance, spinning cursors are threw out from time to time. Although the reason why it appears is straightforward based on its mechanism that the main UI thread is too busy to process user input, its root cause is still hard to diagnose. Tracking the performance bug where the lagging or unresponsiveness of applications stem, is hard due to the massive interactions among threads. Without instruments in the source code, it is challenging to detangle the user transactions running concurrently in the system, as well as the background heartbeat threads.

Fortunately, instruments in low level libraries and system could help developers to recognize interations of threads, by generating a dependancy graph over the tracing data. A tracing point, the statement where the instrument resides, contains the name of an event and attributes correlated to the event. Each of them either represents the execution boundary of a particular request/user input or reflects the potential connections with other events. With events inside the boundary of particular request/user input as node, and connections as edges, a dependancy graph can be generated for each user transaction. The dynamically generated dependancy graph is useful, in that it helps developers to figure out the critial path of user transations and enlight them to improve their code.

However, without the in-depth understanding of a system, especially when it is not open source, the integrity and completeness of the generated dependancy graph is hard to gurantee. The incorrect graph becomes less useful, event misleading, for the developers to figure out the root cause of performance bugs. In this paper, we focus on Apple's Mac system and address the challenges by inspecting the over connections and under connections in the dependancy graph, which is generated with well-known programing paradigms. Revealing the problematic edges or nodes caused by specific programming paradigms provides hints to the developers. With these hints, the developers can further explore pragraming paradigms in various levels, including kernel, dynamic libraies and middle-ware frameworks. As a result, the dependancy graph gets improved cumulatively and becomes more helpful to figure out the root cause of the performance bugs.

## 1. Introduction

Performance bugs, usually cause significant performance degration, have vital impact on user experience. However, it is hard to diagnose for a knowledge gap between the symptom and the inefficient code for app developers. To identify causes of performance degreadation, the general idea shared by multiple previous work is to track the system activity and extract dependancy graphs for individual user transaction for further analysis. However, multithread programming are widely applied in modern systems. Under these circumstances, massive thread activities emerge in the system, from kernel, daemons, various applications and even the interleaved user transactions from the same application. Those activities in interleaved manner make it difficult to extract a correct and complete dependancy graph for a corresponding user input.

To address the challenge, some previous works in the area depend on the input from developers. Magpie is designed for distributed system to extract control path and resource demand of workloads from system wide traced events. It adds necessary tracing points in application, middleware and kernel by extending the tracing tool (Event Trace for Windows). A user provided event schema is used to correlate events from the same request. While Pip is an infrastructure developed to compare the expectation of an application with the actual behavior so as to reveal bugs of applications in distributed system. It designs a language for developers to describe their expectations of application behavior. An annotation library and a set of tools for gathering and checking the traced events. However, either the schema or the source code abstraction can be error prone. In that different components in a system, for example libraries, are usually developed by different developers, and even the worse, some of them can be closed source. Without the help from system authors, it is hard for the app developers to construct a complete and sound schema or expected application behavior.

There also exist works that do not need developers' inputs. AppInsight minimize the noise by only concentrating on the activities from the Application. It tracks an user transaction from the begin of user input to the end of UI update to diagnose performance bottlenecks and failures of apps on Windows Phone, including UI manipulation, Layout update, Asynchronous function call, begin and end of callback functions from framework into the app code (upcalls), thread synchronization and exception data. Without tracking daemons or system level activity, the dependancy graph may be less useful in pinpoint bugs residing in daemons or underlying frameworks. Panappticon instruments system wide with fine-grained tracing points and captures the correlations and causality of events across thread/process boundaries. Its usefulness highly depends upon the knowledge of Android property and the understanding of the whole system to define the correlations of temperally ordered events and the causality of events between threads. Hueristics that the locking primitive in the background thread indicated the producer/consumer of a task queue and no unrelated work will be performed while processing a particular task from a queue may be not true for other systems. The causality, deduced from asychronous call of MessageQueue and ThreadPoolExecutor, interprocess communication via Binder, and sychronization mechanisms, may be not complete either. To sum up, the integrity and completeness of dependancy graph is hard to guarantee without in-depth understanding of the whole system. Previous work from various platforms, such as Windows, Android and etc, could hardly directly applied to a new system, especially one with different design of programming paradigms.

Our work is a complementary part to the previous work. We proposed XXX to help the developer to detect the under-connection and over-connection in a dynamically generated dependancy graph, discover and explore the potentially missing vital tracing points to rectify the errors. To justify the integrity of the dependancy graph, we need to rule out the over connections. One simple way is to audit the path between every two processes in the graph. If two irrelevant applications appear in the same graph, there must be some over connections in their connecting path. Further exploring and fixing can be done by checking the nodes in the path. To improve the completeness of the dependancy graph, we need to inspect the under connections. To prevent the dependancy graph from over-connection and under-connection, tracing points can be added either by hooking dynamic libraries or by setting hardware watch pointers.

We apply the method on the application running on MacOS and figure out both false negtive(under-connection) and false positive(over-connection) with the tranditional tracking technique.

There are missing thread asynchronous mechanisms. False negatives we detected:

- Timers can be armed to delay the event processing.
- Grand Central Dispatcher is frequently referred by the high level frameworks to schedule tasks.
- Daemons like WindowServer may postpond event processing via shared variable.
- Rendering with layer can be batched for later with flags in layer objects.

False positive we detected:

- Runloop is a more complicated programing modle than MessageQueue or ThreadExecutor. Dividing it merely depending on the user event would result in false connection of thread activities in temporal order.
- Some kernel_task threads in MacOS running as heartbeat thread, can always introduce over connections.
- One block from dispatch queue, invoked with dispatch_mig_server, could serve for multiple unrelated work.
- Daemons like WindowServer will also serve for different work in interleave manner to support postponding processing of messages.