April 01, 2019

# DISTRIBUTED STATISTICAL APPLICATION PROFILER

Benjamin Smith

# DISTRIBUTED STATISTICAL APPLICATION PROFILER

Profiling the performance characteristics of an application, in particular a real world application, may be a difficult process. Some profilers may automatically inject code into a program to measure performance. Other profilers may cause a timing interrupt to collect a snapshot of application data at periodic intervals. One type of profiler may specifically instrument portions of code and take performance measurements, such as execution time for those portions of code. Another type of profiler may measure the interactions between user systems. Traditional application profilers generally run on single standalone programs and are typically executed on a single system.

Adding instrumentation to an application for profiling can impact performance of the application. High resolution of an application profile may be burdensome because the more detail required to be measured by the instrumentation the greater the impact on performance. Profiling a standalone application also provides a limited profile of the possible use cases of the application and does not provide profiling data for performance of the application across different systems and hardware. Therefore, traditional profilers are limited in their ability to profile application performance of use cases as seen in real world use of the application.

Furthermore, there may be a large number of possible actions and combinations of actions that a user may perform in the application. Testing of all permutations, or even a large number of permutations, of the possible use cases of an application is significantly limited. Testing of use cases may be limited by possible actions imagined by a human tester and, for an automated system, the set of actions the system is configured to explore. Correlation of application performance with relevant code sections may also be challenging. Correlating performance regression with particular code may require back-and-forth communication with users, deductive problem solving, or outright failure to fix the issue. Such troubleshooting can be time consuming and ineffective.

We present a Distributed Statistical Application Profiler (DSAP) to solve the above issues with application profiling. By partially profiling a large number of instances of an application, such as instances of an application deployed to a large user base, an accurate and detailed performance profile may be generated. For example, an application distributed to users may include, in each instance, instrumentation to profile a fraction of the entire code of the application. Each user may use the application in an individualized way and therefore a number of use cases may be analyzed due to the different uses of the application. The profile data from each partial profile of the instances of the application may be collected at a centralized system to aggregate the partial profiles into a complete profile of the application. An algorithm may be defined to recombine the samples of partial profiles into a statistical model of the application.

After the partial profile samples are collected and aggregated from the number of instances of the application, a complete application profile may be reconstructed from the samples. When compiled, the samples can produce both an entire profile of the application and a statistical representation of the use cases of the application. The statistical representation of the use cases may model the different behaviors the application exhibits when being executed for various users. Some use cases may be used by many users of the application while other use cases might be used minimally. Thus, when implementing solutions to performance regression, high use cases may be identified and engineers may prioritize addressing the high use cases to obtain the greatest improvement of performance for their efforts.

Sample data points from partial profiler instrumentation may include references that identify parts of the application which they instrument. Using these references, an engineer may be able to identify where a problem needs to be addressed in the source code of the application. For example, data points returned to a central collecting mechanism may include the source file and line from which the sample was taken. The data sent to the collecting mechanism may be encoded to prevent leaking of information about the application. A hash or encoding of the information included with the data point may be created upon sending the samples to the central collecting mechanism.

Additionally, when the instrumentation is compiled in the code of the program, there may be an indication of where that instrumentation occurs within the code. For example, the instrumentation may indicate the file and line of code at which it is compiled. When the sample is returned to the central collection mechanism it can be determined which part of the code the measurement is referring to. For more detail, the sample may further include reference to a hierarchy of information regarding the execution of the code at which it is located. For example, the instrumentation may collect and send call stack information, including where the code was called from and so forth. The reference data may be combined into a hash that may be decoded at the collection mechanism. The collection mechanism may include a code table that maps to a set of locations upon receiving a sampling from particular code.

The sampling from the instrumentation can be tuned or adjusted in terms of level-of-detail. The level-of-detail of the sampling may have an effect on the performance of the application, so applications with a smaller user base may be required to collect a less overall detailed profile to prevent effects on application performance. For any application there may be a certain level of detail of sampling at which the performance of the application begins to suffer. The adjustment of the level-of-detail of the sampling allows the profiler to balance the tradeoff between the performance impact and the resources needed to process the samples against the fidelity of the model created from the samples. The more samples the profiler takes the better the resolution of the model but at a greater risk of a negative performance impact. If an application has a large user base then highly detailed sampling may be easily supported without impact on application performance because the sampling can be distributed across the numerous users.

Additionally, it may be desired to increase the resolution of the model for certain types of users of the application, such as users of a certain device or a particular class of users. Therefore, the probability that certain users are sampled may be adjusted as well. Increased sampling from a certain class of users may allow for a more detailed model of the application with respect to that class of users. In some implementations, a class of users may merely be test users and therefore the performance impact may be disregarded to increase sampling and provide a very detailed profile of the application based on those test users.

An example of this process of a distributed partial profiler may be illustrated by a simple program that may execute step A, step B, or step C. Step C only executes if step B executes first. In deploying this program for execution by users, N samples may be collected from the users' execution of the application. From the N samples the DSAP can statistically measure the likelihood of step A, step B, and step C being executed in any given execution of the program. If step A and step B are equally likely to be executed, the DSAP would measure N/2 instances of step A and N/2 instances of step B. From the N samples, the DSAP will have samples of instances of step A, step B and step C as well as samples of instances of step B then step C. Using this data, a statistical model of the program may be constructed based on execution of each instance of step A, step B, step C and step B then C. The co-occurrence of different samples may be used to infer structure of the application and to rebuild execution paths. As discussed above, receiving references to code along with the data points, such as call stack information, can aid in the reconstruction of the model of the program.

The performance profile of an application created using the DSAP may be used to diagnose performance regressions of the application. For example, when a performance regression occurs, a performance profile created before the performance regression occurred may be compared to a performance profile that was created after the performance regression occurred. The differences in the performance profiles may indicate which user cases are being affected and what part of the application is causing the performance regression. References to code received from the samples collected from instrumentation may identify particular code sections correlated with the performance regression. Engineers may target their work on the particular sections of code identified by the data points of the performance profile that are causing the performance regression.

Figure 1 is a flow diagram illustrating a method of using a distributed statistical application profiler to create a profile of an application and diagnose performance regressions. At block 102, partial application profile data is received from each of a plurality of application instances. The application instances may be applications being used by a number of users. The source code of the application instances may include instrumentation to collect performance data for a fraction of the entire application. Each instance of the application may include instrumentation of a different part of the application so when the partial profile data is aggregated, a complete picture of the application may be created. The partial profile data may be received at a single central collecting mechanism (e.g., a component residing on a central server connected to multiple client devices or servers running respective instances of the application for the users).

At block 104, the partial application profile data received at block 102 is aggregated into a collection of partial profile data. The aggregated partial profile data may include profile data from each part of the application such that aggregation of the partial profiles includes performance data for the entire application. The aggregated partial profile data may include information about the system and the hardware on which the application is running.

At block 106, a profile of the application is created based on the aggregated partial profile data. The profile of the application may include a performance profile of the application as well as a statistical model indicating high and low use cases of the application. The profile of the application may be a reconstruction of the application. Because each instance of partial profile data reflects only a part of the application, the use cases that are executed by a larger number of users may provide information indicating higher use cases as well as provide higher resolution of the profile for high use cases.

At block 108, a performance regression in the application is identified. A performance regression may result from issues with software, such as a faulty patch, that affect the performance of the application. At block 110, a performance profile created from partial application profile data collected before the performance regression and a performance profile from partial application profile data collected after the performance regression occurred may be compared. The comparison may determine differences in the performance of particular use cases before and after the regression.

At step 112, the cause of the performance regression may be identified based on the differences in the performance profiles from before and after the regression occurred. In this way, the particular performance bottleneck may be identified so that the issue may be directly addressed rather than lengthy bisection of the application. References to sections of code carried along with the partial profile data may help to identify particular parts of the code at which the performance regression is occurring. The comparison may also identify the particular use cases in which the performance regression occurs because not all use cases may experience the performance regression. Because the DSAP can identify performance regressions of different use cases, DSAP may allow engineers to efficiently target correction efforts toward performance regressions of high use cases.

The method of creating an application profile described herein allows identification of performance bottlenecks and prioritization of addressing performance bottlenecks based on an impact predicted by the application profile. Additionally, when performance regressions are identified, comparison of the application profile before and after the regression occurs can pinpoint the source of the problem and link to particular code causing the issue. Thus, the application profile created from distributed statistical sampling can identify the culprit of performance problems without a time consuming and impactful bisection of an application.

Further to the description above, a user may be provided with controls allowing the user to make an election as to both if and when systems, programs or features described herein may enable collection of user information (e.g., information about a user's activities, information about content of documents, a user's preferences, or a user's current location), and if the user is sent content or communications from a server. In addition, certain data may be treated in one or more ways before it is stored or used, so that personally identifiable information is removed. For example, a user's identity may be treated so that no personally identifiable information can be determined for the user. Thus, the user may have control over what information is collected about the user, how that information is used, and what information is provided to the user.

ABSTRACT

A method for profiling an application using partial profile data from multiple instances of the application deployed for various users is presented herein. The method enables reconstruction of a complete and accurate application profile by distributing instrumentation across many instances of an application to sample only parts of an application by each instance. The method allows for highly detailed and accurate profiling of real world use cases for an application without affecting performance of the application because the sampling is distributed across the user base. The method may allow for tuning of the detail of sampling in order to collect as much detail as possible without affecting the performance of the application. Tuning may also allow for selective sampling of particular classes of users and devices.

**Keywords:** application profiling, data sampling, application performance, statistical profiling, distributed profiling, performance regression

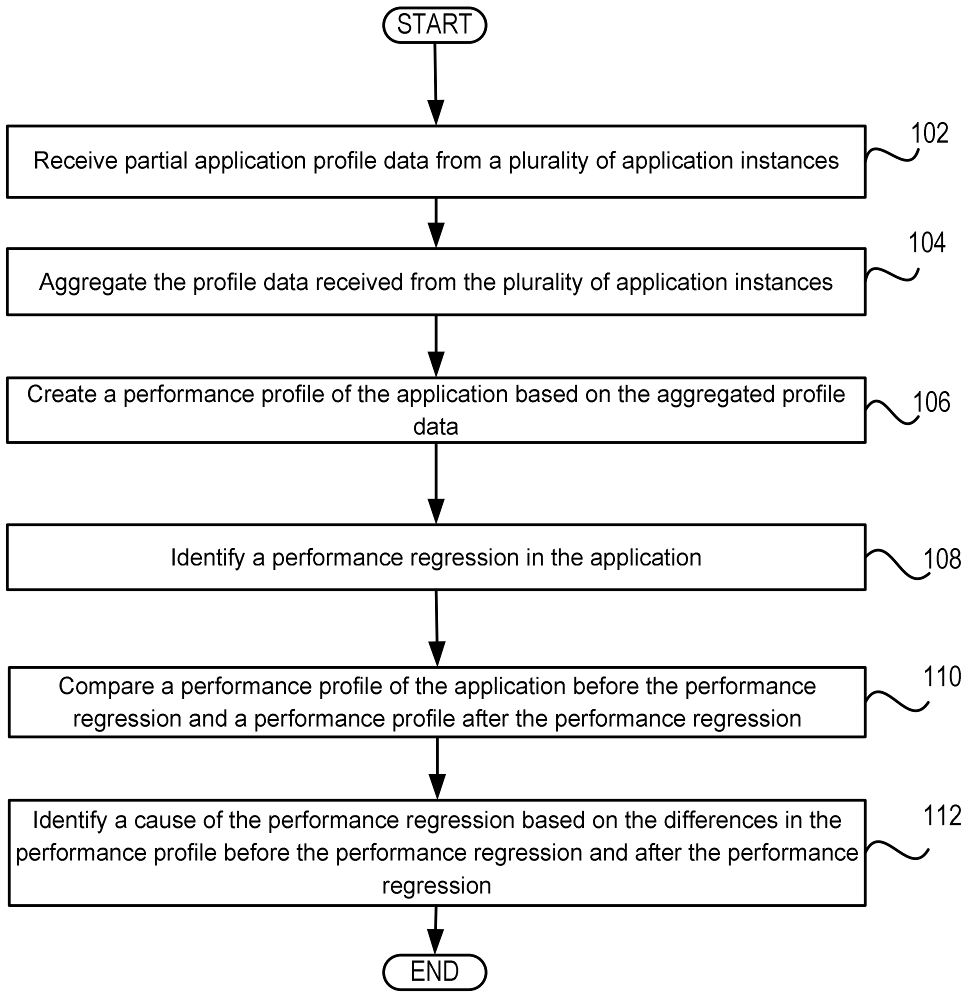
START
Receive partial application profile data from a plurality of application instances
102
Aggregate the profile data received from the plurality of application instances
104
Create a performance profile of the application based on the aggregated profile data
106
Identify a performance regression in the application
108
Compare a performance profile of the application before the performance regression and a performance profile after the performance regression
110
Identify a cause of the performance regression based on the differences in the performance profile before the performance regression and after the performance regression
112
END

FIG. 1