Detecting Performance Anomaly Using Machine Learning Techniques

Faculty: Abdullah Muzahid, abdullah.muzahid@utsa.edu
Student: Mohammad Mejbah ul Alam, mejbah.alam@utsa.edu
Intel Contact: Justin Gottschlich, justin.gottschlich@intel.com

Performance of a program directly affects users' satisfaction and productivity. Poor performance can lead to increased latency, reduced throughput, and wasted resources (e.g., energy, memory etc.) of a computer system. Therefore, it is important to detect when a program exhibits anomalous performance. However, performance anomaly is not easy to detect. It does not have any clear cut symptoms that can be easily ascertained. A program can have low performance because of poor algorithm, high I/O activity, wrong choice of data structures, redundant memory accesses and computations, contention on hardware resources, contention of synchronizations etc. The goal of this project is to detect performance anomaly and determine its root cause using machine learning techniques.

Performance anomaly detection usually requires significant manual intervention. Programmers use some profiling tool (e.g., hprof [2], gprof [1] etc.) to collect profiles, analyze those profiles to determine *hot* code regions that spend majority of the total execution time, and finally inspect those code regions to detect performance anomalies. Some profiling scheme [3] looks at not only user code information but also information about runtime system, virtual machine etc. Some recent work [4] uses machine learning techniques to predict future performance issues and take corrective action to avoid them.

In this project, we would like to detect and pinpoint root causes of performance anomalies. We would like to use machine learning approach because of its ability to handle complex patterns and diverse workloads. At the high level, the proposed technique works by collecting profiles of some features related to performance. Such features can be number of instructions, number of memory references, number of synchronization operations, waiting time for synchronizations, number of threads, number of cache misses, number of branch mispredictions etc. We will collect profiles of theses features and performance metrics from many executions of a program. We can collect those from a set of selected code locations (e.g., some hot functions). We will build a machine learning model (using deep learning, support vector machine etc.) using the profiles as training data. Lets call this model $M_{correct}$. $M_{correct}$ will predict performance of a program given the values of the features. We will investigate some common performance anomalies (e.g., high contention of locks, too many cache misses, too many branch mispredictions, too few cores etc.) and determine how those anomalies affect the selected features. For a particular anomaly A, we will artificially inject relevant changes in features (e.g., by introducing some wait in a lock, increasing the number of cache misses etc.) during executions. We will collect profiles of those perturbed executions and build another machine learning model for performance. Let us call the model M_A. Thus, M_A will predict performance of a program when it has performance anomaly A. Similarly, we will build models for other performance anomalies. During testing process, given a profile of features, we will use $M_{correct}$ and $\forall_A, M_A$ to predict performance, the model that provides a prediction closet to the actual performance will characterize the execution. For example, if $M_{correct}$ model provides the closet prediction, the execution is anomaly free i.e., as expected. On the other hand, if M_A model provides the closet prediction, then the program suffers from performance anomaly A. An advantage of this proposed approach is that we will be able to detect performance anomaly as well as its cause (e.g., A).

The faculty along with his student recently published a paper that utilizes neural network hardware to diagnose production run software failures [5]. The paper was published in a top tier conference **ISCA 2016**. The work proposed here is an effort to extend similar idea for detecting performance anomalies. We expect the result to be published in some top tier conference as well.

References

[1] GNU gprof. `https://sourceware.org/binutils/docs/gprof/`.

[2] O. Corporation. HPROF: A Heap/CPU Profiling Tool. `http://docs.oracle.com/javase/7/docs/technotes/samples/hprof.html`.

[3] M. Hauswirth, P. F. Sweeney, A. Diwan, and M. Hind. Vertical profiling: Understanding the behavior of object-priented applications. In *OOPSLA*, 2004.

[4] Y. Tan, H. Nguyen, Z. Shen, X. Gu, C. Venkatramani, and D. Rajan. Prepare: Predictive performance anomaly prevention for virtualized cloud systems. In *ICDCS*, 2012.

[5] M. M. ul Alam and A. Muzahid. Production-run software failure diagnosis via adaptive communication tracking. In *ISCA*, 2016.