Using Bayesian Learning to Estimate How Hot an Execution Path is
Project Proposal
CS886 - Fall 2010

Karim Ali

1 Problem Domain

In the domain of program analysis, it is always advised to perform code optimizations to the paths that have higher likelihood of occuring in a given run of the program. This prioritization process can be tackled by identifying hot paths (paths of high frequency of execution) and cold paths (paths of low frequency of execution) [1, 2, 3, 4]. The importance of identifying hot paths arises from the empirical observation that most or all of the execution time of a typical program is spent along a small percentage of program paths (i.e. hot paths).

2 Existing Solutions

Static profiling has been commonly used in the literature to identify hot paths. Although static profiling can be very useful and successful, it faces many practical challenges:

1. the frequent lack of appropriate workloads for programs,
2. the questionable degree to which they are indicative of actual usage,
3. the inability of such tools evaluate program modules or individual paths in isolation,
4. and the extra work done by the programmer/developer to write code that generates program profiles.

Analyzing additional information, e.g data flow analysis [5], is also a common practice that helps identify hot paths.

3 Interesting Solution

Raymond Buse and Westley Weimer [6] propose a very interesting approach to the problem of hot paths identification. The proposed solution is modelled as a classification problem, where a path is classified as high frequency, or low frequency. They used a Bayesian classifier for the learning process, where the classifier is trained by analyzing a set of feature-vectors. Each feature-vector consists of the numerical counts of occurances of features that the authors considered sufficient to capture the state changing behavior related to path frequency.

4 Project Idea

I would like to investigate the work done in [6] more, since they did not provide enough explanation for their Bayesian learning process. I would also like to survey similar methods used to identify hot paths. Based on my findings from investigating the work in [6] and other literature, I would like to improve on those techniques and push them one step further.

References

[1] T. Baba, T. Masuho, T. Yokota, and K. Ootsu. Design of a two-level hot path detector for path-based loop optimizations. In *Proceedings of the third conference on IASTED International Conference: Advances in Computer Science and Technology*, page 28. ACTA Press, 2007.

[2] T. Ball and J.R. Larus. Efficient path profiling. In *micro*, page 46. Published by the IEEE Computer Society, 1996.

[3] E. Duesterwald and V. Bala. Software profiling for hot path prediction: Less is more. *ACM SIGARCH Computer Architecture News*, 28(5):202–211, 2000.

[4] M.C. Merten, A.R. Trick, C.N. George, J.C. Gyllenhaal, and W.W. Hwu. A hardware-driven profiling scheme for identifying program hot spots to support runtime optimization. In *Proceedings of the 26th annual international symposium on Computer architecture*, page 147. IEEE Computer Society, 1999.

[5] C. Boogerd and L. Moonen. On the Use of Data Flow Analysis in Static Profiling. In *Source Code Analysis and Manipulation, 2008 Eighth IEEE International Working Conference on*, pages 79–88. IEEE, 2008.

[6] R.P.L. Buse and W. Weimer. The road not taken: Estimating path execution frequency statically. In *Proceedings of the 2009 IEEE 31st International Conference on Software Engineering*, pages 144–154. IEEE Computer Society, 2009.

[7] S. Heckman and L. Williams. A Systematic Literature Review of Actionable Alert Identification Techniques for Automated Static Code Analysis.

[8] B. Calder, D. Grunwald, M. Jones, D. Lindsay, J. Martin, M. Mozer, and B. Zorn. Evidence-based static branch prediction using machine learning. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 19(1):188–222, 1997.