Proxy Benchmarks for Emerging Big-data Workloads

Reena Panda and Lizy Kurian John
The University of Texas at Austin
reena.panda@utexas.edu, ljohn@ece.utexas.edu

***Abstract*—Early design space evaluation of computer systems is usually performed using performance models (e.g., detailed simulators, RTL-based models, etc.). However, it is very challenging (often impossible) to run many emerging applications on detailed performance models owing to their complex software-stacks and long run times. To overcome such challenges in benchmarking these complex applications, we propose a proxy generation methodology, PerfProx to generate miniature proxy benchmarks, which are representative of the performance of real-world applications and yet, converge to results quickly and do not need any complex software-stack support. Past proxy generation research utilizes detailed micro-architecture independent metrics derived from detailed simulators, which are often difficult to generate for many emerging applications. PerfProx enables fast and efficient proxy generation using performance metrics derived primarily from hardware performance counters. We evaluate the proxy generation framework on three modern databases (Cassandra, MongoDB and MySQL) running data-serving and data-analytics applications. The proxy benchmarks mimic the performance (IPC) of the original applications with ∼94% accuracy, while significantly reducing the instruction count.**

I. Introduction

Early computer design evaluation is performed using performance models such as execution-driven simulators or RTL-based models. However, several emerging applications are often complex targets to evaluate on early performance models owing to their complex software-stacks, significantly long run times, system dependencies, etc. Moreover, detailed performance models are significantly slower than real hardware that makes it difficult to analyze complete execution characteristics of these long-running applications. A set of standard benchmarks are typically used for performing computer design space exploration. Cloudsuite [1] and BigDataBench [2] benchmark suites have been recently proposed to represent the broad range of emerging big-data applications. Few research studies [3] have also tried to simplify database benchmarking by using smaller data-sets etc. However, these efforts suffer from similar challenges as the real-world applications, i.e., they rely on the ability of early performance models to support complex software stacks with back-end database support. On the other hand, benchmarks like SPEC CPU2006 [4] are comparatively simpler targets for performance evaluation but several prior research studies [2], [5] have showed that their performance behavior is very different from the big-data applications.

In order to overcome the difficulties in evaluating emerging applications, in this paper, we propose a proxy synthesis methodology, "**PerfProx**", to create miniature proxy benchmarks that are representative of the performance of real-world database applications and yet, converge to results quickly and do not need any complex software-stack support. Past research on proxy generation [6], [7] utilizes micro-architecture independent metrics derived from detailed functional simulators, which are often very difficult to generate for many emerging applications. Program profilers such as Pin [8] also face difficulties when dealing with Java based databases etc. PerfProx enables fast and efficient proxy generation for such applications using performance metrics derived primarily from hardware performance counters. Several big-data workloads do not work reliably with many profiling tools and thus, performance-counter based characterization and associated extrapolation into generic parameters that the code generator can take is an important contribution. We evaluate the effectiveness of the proxy benchmarks using three real-world databases (Cassandra [9], MongoDB [10] and MySQL [11]) running data-serving and data-analytics applications and demonstrate that the proxy benchmarks closely follow the performance behavior of the original applications while significantly reducing the instruction counts. The mean error in IPC between the proxy benchmarks and the database applications is 5.1% for data-serving applications and 6.5% for data-analytics applications.

II. Methodology

The proposed proxy benchmark generation framework is shown in Figure 1. During the workload characterization step Ⓐ, PerfProx characterizes the low-level dynamic execution characteristics of the database application using the following metrics: (i) **Instruction mix (IMIX) -** PerfProx captures the IMIX of the original application (categorized into fraction of loads, stores, control instructions, etc.) using hardware performance counters; (ii) **Control-flow performance -** PerfProx estimates the average branch transition rate of the database application in a directly correlated fashion based on the application's branch misprediction rate (measured using hardware performance counters). It assigns an appropriate transition rate to the component branch instructions of the proxy benchmark to achieve this target branch transition rate; (iii) **Memory access model -** PerfProx models data memory accesses using strided stream classes over fixed-size data arrays. PerfProx calculates the memory strides by using a pre-computed table that holds the correlation between L1/L2 cache hit rates and the corresponding stride values [6]. TLB performance is modeled by controlling the degree of concurrency within the active memory streams. The proxy data footprint is scaled based on the data-set size of the database application; (iv) **Instruction-level parallelism (ILP) -** PerfProx models application ILP using the inter-instruction dependency distance metric. PerfProx estimates the application's inter-instruction dependency distance based on the measured dependency-related stall events of the database application; (v) **System activity -** PerfProx

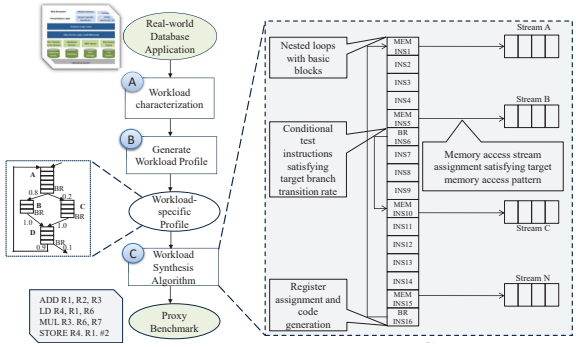

Fig. 1: Proxy generation framework

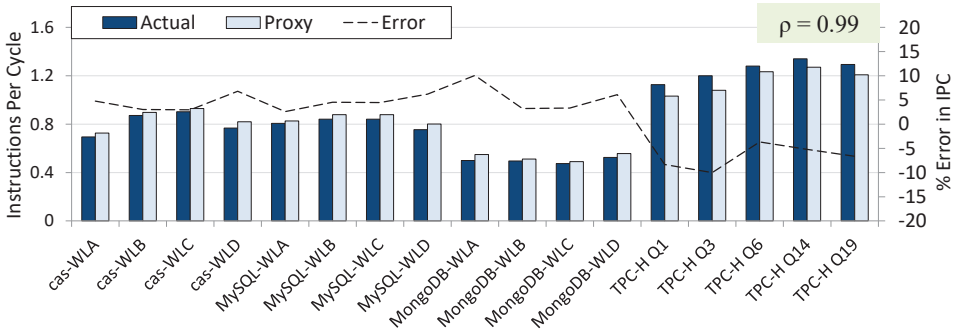

Fig. 2: IPC of real databases and proxy applications

tracks system activity using the *STRACE* tool and the fraction of executed user-mode and kernel instructions measured using hardware performance counters.

Together, the above metrics form a unique "***workload-specific profile***" for the database application that summarizes the application's low-level runtime behavior over its entire execution. The workload synthesis algorithm uses this workload-specific profile to generate the proxy benchmark ©. Synthesizing using statistics rather than the actual source code hides the functional meaning of code/data, which can address any proprietariness concerns about sharing end-user workloads. To generate the proxy, PerfProx first estimates the number of static basic blocks to instantiate in the proxy benchmark. It then chooses a basic block based on its access frequency and assigns its size to satisfy the mean and standard deviation of the target basic block size. For every basic block, PerfProx assigns the individual instructions an appropriate instruction type (last instruction is always a conditional branch), dependency distance, memory stride (for memory operations) etc. System calls are injected based on the target system call frequency. These steps are repeated till the target number of static basic blocks are generated. Finally, the generated instruction sequence is nested under a two-level loop, where the inner loop controls the dynamic data footprint and the outer loop controls the dynamic instruction count of the proxy benchmark. The outer loop iterations reset each data-stream access to the first element of the memory array. The proxy synthesizer generates C-language based proxy benchmarks with embedded x86-based assembly instructions using the *asm* construct.

III. Experimental setup

In this paper, we evaluate both data-serving and data-analytics applications using the Yahoo! Cloud Serving Benchmark (YCSB)[12] (in-memory setup, ∼12GB data-set) and TPC-H benchmarks [13] (∼10GB data-set) respectively. The applications are setup using three popular, modern NoSQL and SQL back-end databases (Cassandra, MongoDB and MySQL). Characterization and generation of proxy benchmarks is performed on 64-bit Intel Xeon servers, with 4 out-of-order cores, 2-level cache hierarchy, 16GB DRAM memory and running at a frequency of 2GHz. The proxy benchmarks are compiled using gcc with the -O0 optimization flag to avoid compiler optimizations (e.g., dead code removal). We use Linux perf tool [14] to measure the hardware performance of applications. Intel's Pin tool [8] is used for performing micro-architecture independent workload characterization.

IV. Results and Analysis

Figure 2 compares the instructions per cycle (IPC) of Cassandra, MySQL and MongoDB databases running the YCSB and TPC-H benchmarks along with their corresponding proxies. The IPC of the proxy benchmarks closely follow the IPC of the original applications, with a high correlation (ρ = 0.99). The mean error in IPC is 6.1% (max 10.7% for MongoDB) across all workloads. Considering the data-serving applications alone, the average error in IPC between the proxy and the actual application is 5.1%, while the data-analytics applications have an average error of 6.5%. The dynamic instruction count of the proxy benchmarks is also ∼528 times smaller than the database applications, which can significantly reduce their simulation time on simulation frameworks.

V. Conclusion

In this paper, we proposed PerfProx to generate proxy benchmarks that enable performance evaluation of emerging workloads without needing back-end database or complex software stack support. PerfProx captures application performance using hardware performance counters. We evaluated PerfProx using three modern databases, Cassandra, MySQL and MongoDB for data-serving and data-analytics applications and demonstrated that the proxy benchmarks mimic the performance of the original applications with ∼94% accuracy, while significantly reducing the instruction count.

VI. Acknowledgement

The authors of this work are supported partially by SRC under Task ID 2504 and National Science Foundation (NSF) under grant number 1337393. We wish to acknowledge the computing time we received on the Texas Advanced Computing Center (TACC) system. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF or other sponsors.

References

[1] M. Ferdman *et al.*, "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," in *ASPLOS*. New York, NY, USA: ACM, 2012, pp. 37–48.
[2] W. Gao *et al.*, "Bigdatabench: a big data benchmark suite from web search engines," *CoRR*, vol. abs/1307.0320, 2013.
[3] K. Keeton and D. A. Patterson, "Towards a simplified database workload for computer architecture evaluations," in *In Workload Characterization for Computer System Design, edited byh*. Kluwer Academic Publishers, 2000, pp. 115–124.
[4] "SPEC CPU2006," https://www.spec.org/cpu2006.
[5] R. Panda, C. Erb, M. Lebeane, J. Ryoo, and L. K. John, "Performance characterization of modern databases on out-of-order cpus," in *IEEE SBAC-PAD*, 2015.
[6] R. H. Bell, Jr. and L. K. John, "Improved automatic testcase synthesis for performance model validation," in *Proceedings of the 19th Annual International Conference on Supercomputing*, ser. ICS '05, 2005, pp. 111–120.
[7] K. Ganesan, J. Jo, and L. K. John, "Synthesizing memory-level parallelism aware miniature clones for spec cpu2006 and implantbench workloads." in *ISPASS*, 2010.
[8] C.-K. Luk *et al.*, "Pin: Building customized program analysis tools with dynamic instrumentation," in *PLDI*, 2005, pp. 190–200.
[9] "Cassandra," wiki.apache.org/cassandra/FrontPage.
[10] "MongoDB," mongodb.org.
[11] "MySQL," http://www.mysql.com.
[12] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *SoCC*, 2010, pp. 143–154.
[13] "TPC-H Benchmark Suite," http://www.tpc.org/tpch.
[14] "Linux perf tool," https://perf.wiki.kernel.org/index.php/Main_Page.