Identifying On-/Off-CPU Bottlenecks Together with Blocked Samples

Minwoo Ahn[1], Jeongmin Han[1], Youngjin Kwon[2], Jinkyu Jeong[3]
[1]*Sungkyunkwan University,* [2]*KAIST,* [3]*Yonsei University*
{minwoo.ahn, jeongmin.han}@csi.skku.edu, yjkwon@kaist.ac.kr, jinkyu@yonsei.ac.kr

Abstract

The rapid advancement of computer system components has necessitated a comprehensive profiling approach for both on-CPU and off-CPU events simultaneously. However, the conventional approach lacks profiling both on- and off-CPU events, so they fall short of accurately assessing the overhead of each bottleneck in modern applications.

In this paper, we propose a sampling-based profiling technique called *blocked samples* that is designed to capture all types of off-CPU events, such as I/O waiting, blocking synchronization, and waiting in CPU runqueue. Using the blocked samples technique, this paper proposes two profilers, *bperf* and *BCOZ*. Leveraging blocked samples, bperf profiles applications by providing symbol-level profile information when a thread is either on the CPU or off the CPU, awaiting scheduling or I/O requests. Using the information, BCOZ performs causality analysis of collected on- and off-CPU events to precisely identify performance bottlenecks and the potential impact of optimizations. The profiling capability of BCOZ is verified using real applications. From our profiling results followed by actual optimization, BCOZ identifies bottlenecks with off-CPU events precisely, and their optimization results are aligned with the predicted performance improvement by BCOZ's causality analysis.

1 Introduction

Application profiling encompasses the analysis of two types of events: on-CPU events and off-CPU events. Profiling on-CPU events aims to analyze instructions executed on a CPU [1,4,7,15,17,19–21,33,45,53,54]. In contrast, profiling off-CPU events aims to analyze waiting events within an application such as waiting for blocking I/O completion, locks, scheduling, etc [27,35,38,39,55,58].

In the past, applications were clearly characterized as either CPU-boud or I/O-bound due to the use of slow I/O devices such as HDD or SATA SSD. Therefore, existing profiling tools have applied bottleneck analysis techniques separately for on-CPU events or off-CPU events. However, with the recent advancements in fast storage and many-core CPUs, modern applications often exhibit complex behaviors. Especially, their performance bottleneck is combined by on-CPU events and off-CPU events, necessitating the need for comprehensive profiling of both on-CPU and off-CPU events *simultaneously* and capturing their interactions. For example, with the emergence of NVMe SSDs and ultra-low latency SSDs, the critical path of I/O-intensive applications often shifts from I/O to CPU events [23, 30–32]. Consequently, studies have focused on optimizing on-CPU events rather than I/O events to enhance the performance of I/O-intensive applications [23, 30–32].

However, existing profilers focus on analyzing only on-CPU [17, 20, 33] or off-CPU events [3, 38], so they cannot analyze the complicated behaviors of modern applications. COZ [15], a state-of-the-art causal profiler, estimates performance gain through its virtual speedup approach. COZ intentionally delays competing threads to estimate the performance impact by optimizing certain code lines without actually optimizing them. However, COZ applies virtual speedup profiling exclusively to on-CPU events as it lacks the capability to incorporate execution information from off-CPU events. wPerf [58], a state-of-the-art off-CPU analysis profiler, traces waiting events between threads during the execution and reports the result in the form of a graph (called a wait-for graph). While wPerf can analyze interactions between on-CPU and off-CPU events, it falls short in assessing the actual impact of bottlenecks on application performance. Furthermore, although wPerf identifies off-CPU bottlenecks, it lacks detailed information about the application contexts related to these bottlenecks. These limitations require programmers to attempt optimizations for various bottlenecks to achieve actual performance improvements and demand additional effort to pinpoint the application code to be optimized (Section 2.2).

This paper introduces a new profiling technique called *blocked samples*, which is designed to capture off-CPU events. Drawing inspiration from the event-based sampling (e.g., Linux perf subsystem [17]), our approach employs sampling-based profiling of off-CPU events. Similar to Linux perf, the blocked samples technique periodically captures snapshots of end overhead of 27.6% on average and up to 64.7%. However, such profilers show similar performance overheads. For example, COZ has demonstrated its overhead of 17.5% on average and up to 65% [15]. As these profilers insert additional delays while the application is running, the end-to-end execution time can be increased. These overheads can be reduced when the inserted delays are limited, cooling-off times are inserted between virtual speedup experiments, etc [15]. Such remedies can be effective in reducing the profiling overhead, but it remains uncertain whether the results provided are sufficient to identify bottlenecks and their potential for performance gain.

5 Related Work

On-CPU Event Profilers. Conventional profilers [17, 19–21, 33] that rely on existing on-CPU events (such as CPU usage and execution time) face challenges when identifying bottlenecks in modern applications. This is primarily because the event with the longest execution time in a multi-threaded application does not necessarily represent the critical performance path, and they do not account for off-CPU events. In the context of multi-threaded applications, there are causal profiling studies aimed at analyzing the impact of optimizing each individual event on overall application performance [1, 4, 7, 15, 45, 53, 54]. COZ [15], for instance, provides performance improvement predictions by applying virtual speedup to each event using the sampling results from the Linux perf subsystem. However, COZ's virtual speedup is limited to on-CPU events sampled by the Linux perf subsystem. It is not capable of estimating virtual speedup of events that include off-CPU events.

Off-CPU Event Profilers. Existing studies have focused on analyzing off-CPU event bottlenecks [27,35,38,39,55,57,58]. Some studies analyze application bottlenecks by measuring the duration of off-CPU events [27,39,55]. However, in multi-threaded applications, the longest event may not always represent the critical path of the application [15, 58]. Furthermore, nested off-CPU events can have varying performance impact on event duration and overall application performance [58]. Therefore, analyzing performance using the duration of off-CPU events leads to incorrect conclusions.

Other studies identify application bottlenecks by specifically targeting off-CPU events related to synchronization [35, 57]. However, as mentioned earlier, the off-CPU bottlenecks in modern applications are diverse and encompass various aspects, including device I/O. Therefore, relying on profiling specific off-CPU events has limitation of supporting the various applications.

wPerf [58] is a state-of-the-art study focused on analyzing off-CPU bottlenecks in applications, wait-for graphs are constructed to identify off-CPU events that act as bottlenecks. However, as discussed in Section 2.2, wPerf has several limitations. wPerf does not precisely pinpoint the performance bottleneck of applications. Also, wPerf lacks the capability of causality analysis, so it could not analyze the actual impact on application performance when optimizing a performance bottleneck. Finally, wPerf identifies bottlenecks but misses detailed information, requiring additional efforts from developers to understand the exact performance bottleneck.

6 Conclusion

Existing profilers face limitations when it comes to identifying modern application bottlenecks that involve a mix of on- and off-CPU events. These profilers treat on- and off-CPU events as separate dimensions, making it difficult to perform comprehensive profiling and interpret the results. Moreover, even if the bottleneck of an application is identified, it remains uncertain whether optimizing the bottleneck will result in actual performance improvements. To address this problem, this paper introduces a sampling technique called blocked samples, which enables the identification of application bottlenecks by integrating on- and off-CPU events within the same dimension. We present bperf, a Linux perf tool that utilizes the proposed blocked samples technique to identify application bottlenecks based on event execution time, and BCOZ, a causal profiler that offers a virtual speedup for off-CPU events. By profiling the RocksDB application using these two profilers, we are able to uncover previously unidentified bottlenecks related to I/O and synchronization tasks. Furthermore, by virtually speeding up these tasks, we identify optimization possibilities that were overlooked in existing RocksDB optimization studies.

We plan to extend blocked samples to include richer information for profiling. The current blocked samples consider the operations inside an I/O device as a black box. However, I/O devices may have their internal operations, which can be the hint of performance optimization opportunities for applications. For example, disk-internal events, such as garbage collection, and valid page copying, are important events for storage applications to establish their optimization strategies. In this regard, we plan to augment blocked samples with I/O device-internal operations thereby allowing applications to employ expanded optimization strategies.

Acknowledgments

We would like to thank the anonymous reviewers and our shepherd, for their valuable comments and feedback. This work was supported in part by Samsung Electronics and by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2023-00321688). Jinkyu Jeong is the corresponding author.

Availability

The source code is available at `https://github.com/s3yonsei/blocked_samples`.