# Leveraging Approximation to Improve Resource Efficiency in the Cloud

Neeraj Kulkarni, Feng Qi, Glyfina Fernando, and Christina Delimitrou
Cornell University
{nsf49, fq26, gsf52, delimitrou}@cornell.edu

## Abstract

Although cloud computing has increased in popularity, datacenter utilization has remained for the most part low. This is in part due to the interference that comes as a result of applications sharing hardware and software resources. When interference occurs, the resources of at least one co-scheduled application need to be reduced forcing it to take a performance penalty. In current proposals, the penalized application is typically a low-priority, best-effort workload. Approximate computing applications present an opportunity to improve datacenter efficiency without performance degradation, since they can absorb the enforced resource reduction as a loss in output quality.

In this paper we present Pliant, a runtime system that improves datacenter utilization by co-scheduling interactive services with approximate computing applications. When the runtime detects QoS violations in the interactive service, it employs approximation to reduce interference, and absorbs the resource reduction as a loss in output accuracy.

## 1. Introduction

Cloud computing has reached proliferation by offering *resource flexibility* and *cost efficiency* [2, 1, 8]. Cost-efficiency is achieved through multi-tenancy, i.e., by co-scheduling multiple jobs on the same physical platform. Unfortunately multi-tenancy also leads to unpredictable performance due to interference [12, 4, 13]. When the applications suffering from interference are high priority, interactive services, like websearch and social networking, multi-tenancy is disallowed hurting utilization, or - at best - interactive services are co-scheduled with low priority, best-effort applications whose performance can be sacrificed [16, 9, 5]. Approximate computing applications offer the potential to break this utilization versus performance trade-off.

In this work we present Pliant, a cloud runtime system that achieves both high quality of service (QoS) and high utilization by leveraging the ability of approximate computing applications to tolerate some loss of output quality. Pliant enables aggressive co-scheduling of interactive, latency-critical services with - also high priority - approximate computing applications. It consists of a lightweight performance monitoring system based on adaptive sampling [15, 10] that continuously checks for QoS violations, and a dynamic compilation system that adjusts the level of accuracy the approximate computing application can sustain in an online manner. When interference surfaces due to resource sharing, Pliant employs suitable approximation techniques that alleviate contention without penalizing the execution time of the approximate computing application. Specifically, Pliant determines the appropriate approximation technique(s) needed based on the type of interference measured in the system, e.g., memory, compute, network, storage I/O, and incrementally increases the degree of approximation until the interactive service can once again meet its QoS constraints.

We evaluate Pliant with a distributed in-memory low latency caching service, memcached [7], and two benchmark suites with applications that can tolerate approximation [3, 17]. In server platforms with 20 physical (40 logical) cores, Pliant enables 90-95% CPU utilization, while ensuring that memcached achieves the same throughput (QPS) and tail latency as when run in isolation, and the approximate computing applications achieve 18.3% lower execution time on average, with a maximum of 15% loss in accuracy.

## 2. Pliant Design & Evaluation

Pliant consists of three components. First, a lightweight *performance monitor* continuously samples the throughput and end-to-end latency (average and tail) of the interactive service, and notifies the runtime system in the event of a QoS violation. Second, an *interference monitor* runs on the server and collects performance counter information that identifies the resource(s) suffering from contention. Third, a *runtime system* that enforces a degree and method of approximation based on the output of the performance and interference monitors. The system uses DynamoRIO [6] to switch between the precise and different approximate versions of the approximate computing applications. Figure 1 shows an overview of the runtime system. The interactive service shares physical cores with the approximate computing applications, although an individual hyperthread (or vCPU) is dedicated to a single application, which is common practice in public clouds [1]. A different client machine is used to drive the load of the interactive service.

**Performance monitor:** This module is integrated in our workload generator and runs on the client side to capture apart from processing time, the network latency of the round trip of a request. It relies on adaptive sampling of requests (based on request rate) to maintain monitoring overheads negligible ($< 0.01\%$ in throughput and $< 0.1\%$ in $99^{th}$ percentile latency) and leverages *systemtap* to provide a breakdown of execution time and identify latency bottlenecks.

**Dynamic recompilation:** Pliant relies on DynamoRIO [6] to switch between the precise and different approximate versions of an application; the trigger for a switch is one of several Linux signals (e.g., SIGTERM, SIGQUIT). We examine the following approximation techniques:

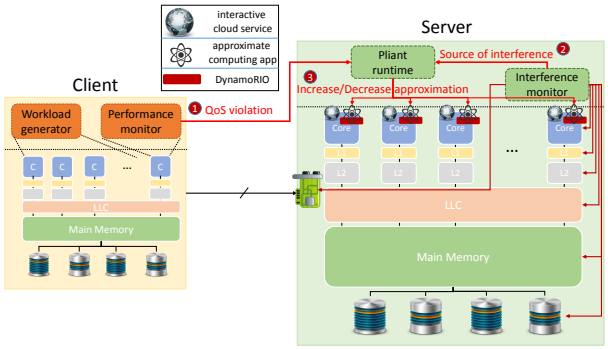


**Figure 1: Pliant overview.**

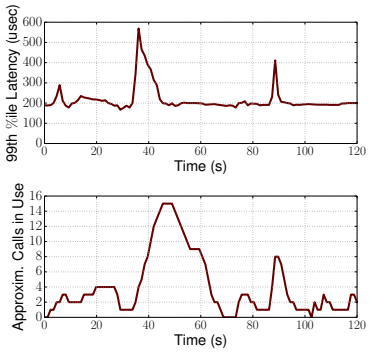


**Figure 2: (a) Tail latency for memcached, (b) number of approximate "functions" in use.**

- *Loop perforation*: Reducing the number of executed loops by (i) only executing the first $(max_iter/p)$ iterations, (ii) executing every $p^{th}$ iteration, or not executing every $p^{th}$ iteration to reduce memory pressure.
- *Data type precision*: Reducing memory footprint by lowering the precision of suitable data types. To identify eligible data types we use the ACCEPT framework [14].
- *Algorithmic exploration*: Using different algorithms, such as search, sort, graph traversal, that optimize for reduced resource usage instead of performance.
- *Synchronization elision*: Optimistically executing parallel regions of the code without acquiring a lock avoids long serialization delays for a penalty in output accuracy.

**Incremental approximation:** Pliant only uses the *minimum degree of approximation* necessary to restore the performance of the interactive service. This means that for each of the techniques above there are several approximate versions, each with different resource requirements and quality loss, for example, loop perforation by a factor of /2, /4, etc. When Pliant receives a QoS violation signal it first switches to approximate versions with small quality loss, and progressively moves to more aggressive methods until QoS recovers. Similarly when the interactive service's performance is better than needed, Pliant reverts back to the precise version of the application (also incrementally).

**Interference-aware approximation:** Pliant also uses the output of the interference monitor to guide its selection of approximation techniques. For example, if the monitor signals high last level cache interference, Pliant will prioritize loop perforation to alleviate memory contention. Similarly with CPU contention and synchronization elision. The interference monitor also tracks contention in network and storage I/O, and although our current applications are contained in single-machine setups, the same methodology can extend to distributed workloads, e.g., machine learning and data mining that can also tolerate approximation.

### 2.1 Evaluation

We use two servers with 20 physical (40 hyperthreaded) cores each, and 128GB of RAM, one as server and one as client. We use memcached as the latency-critical application, and PARSEC and SPLASH-2 as the approximate computing applications. We also set QoS for memcached to 200usec for the $99^{th}$ percentile latency and the throughput to 1.5mil QPS (memcached's throughput close to saturation when it's running on the same platform in isolation). A QoS violation is signaled when either tail latency or throughput deviate from their required values. On each physical core we also pin a thread of an approximate computing application. Figure 2a shows the tail latency of memcached (throughput behaves similarly) as it runs alongside PARSEC and SPLASH-2 benchmarks with Pliant. Initially ($t = 0$) memcached runs alone. Every time latency increases, Pliant employs approximation to restore it. Figure 2b shows how many approximation techniques (or functions) Pliant employed throughout the duration of the experiment. $t = 40$sec shows how incremental approximation works, with an increasing number of approximate functions being called to reduce contention. In the specific example contention was in the memory system, so the method of approximation chosen was loop perforation. CPU utilization is 90-95% and output quality loss is at most 15% (6% on average).

### 2.2 Limitations & Active Work

First, while DynamoRIO provides an easy way for recompilation it can add considerable overheads in execution time (on average 9% and up to 18%). We are actively developing an LLVM-based compiler [11, 9] that enables online code transformations by diverting the program's control flow at a set of virtualized points, with $< 1\%$ overhead.

Second, we currently do not use isolation beyond not sharing a single hyperthread. Modern platforms offer several isolation techniques, including containers, thread pinning, memory capacity partitioning, and network and storage bandwidth, and last level cache partitioning. These can further reduce resource interference, and we are exploring how they can assist Pliant's decisions.

## 3. Conclusions

We have presented Pliant, a practical runtime system that leverages approximate computing applications to increase datacenter utilization. Pliant co-schedules interactive services with approximate applications and employs incremental and interference-aware approximation, selecting the appropriate type of approximation based on the level and cause of the QoS violation of the interactive application.

## References


[1] Amazon ec2. http://aws.amazon.com/ec2/.

[2] Luiz Barroso and Urs Hoelzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. MC Publishers, 2009.

[3] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. Toronto, CA, October, 2008.

[4] Christina Delimitrou and Christos Kozyrakis. Paragon: QoS-Aware Scheduling for Heterogeneous Datacenters. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Houston, TX, USA, 2013.

[5] Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-Efficient and QoS-Aware Cluster Management. In *Proceedings of the Nineteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Salt Lake City, UT, USA, 2014.

[6] Dynamorio: Dynamic instrumentation tool platform. http://www.dynamorio.org.

[7] Brad Fitzpatrick. Distributed caching with memcached. In *Linux Journal, Volume 2004, Issue 124, 2004*.

[8] Google container engine. https://cloud.google.com/container-engine.

[9] Michael A. Laurenzano, Yunqi Zhang, Lingjia Tang, and Jason Mars. Protean code: Achieving near-free online code transformations for warehouse scale computers. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-47, pages 558–570, Washington, DC, USA, 2014. IEEE Computer Society.

[10] Jacob Leverich and Christos Kozyrakis. Reconciling high server utilization and sub-millisecond quality-of-service. In *Proceedings of EuroSys*. Amsterdam, The Netherlands, 2014.

[11] The llvm compiler infrastructure. http://llvm.org.

[12] Jason Mars and Lingjia Tang. Whare-map: heterogeneity in "homogeneous" warehouse-scale computers. In *Proceedings of ISCA*. Tel-Aviv, Israel, 2013.

[13] Ripal Nathuji, Aman Kansal, and Alireza Ghaffarkhah. Q-clouds: Managing performance interference effects for qos-aware clouds. In *Proceedings of EuroSys*. Paris,France, 2010.

[14] Adrian Sampson, Andre Baixo, Benjamin Ransford, Thierry Moreau, Joshua Yip, Luis Ceze, and Mark Oskin. Accept: A programmer-guided compiler framework for practical approximate computing. Technical Report UW-CSE-15-01-01, University of Washington.

[15] Benjamin H. Sigelman, Luiz Andr Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. Dapper, a large-scale distributed systems tracing infrastructure. Technical report, Google, Inc., 2010.

[16] Lingjia Tang, Jason Mars, Wei Wang, Tanima Dey, and Mary Lou Soffa. Reqos: Reactive static/dynamic compilation for qos in warehouse scale computers. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, pages 89–100, New York, NY, USA, 2013. ACM.

[17] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The splash-2 programs: characterization and methodological considerations. In *Proceedings of the 22nd International Symposium on Computer Architecture (ISCA)*. Santa Margherita Ligure, Italy, 1995.