Pliant: Leveraging Approximation to Improve Resource Efficiency in Datacenters

Neeraj Kulkarni, Feng Qi, Christina Delimitrou
Cloud Computing

- **Resource Flexibility**
  - Users can elastically scale their resources on-demand

- **Cost Efficiency**
  - Sharing resources between multiple users and applications
LOW UTILIZATION!

- Servers operate at 10% - 40% utilization most of the time

Major reasons:
- Dedicated servers for interactive services
- Resource over-provisioning – conservative reservations

**Multi-tenancy**

- **Scheduling multiple jobs on the same server**
  - Increases server utilization and cost efficiency
  - Interference in shared resources

- **Interference → Unpredictable performance**
- **Difficult with interactive services**
### Previous Solutions

1. **Allow co-scheduling of apps that would not violate QoS**
   - Bubble-Up, Bubble-Flux, Paragon and Quasar

2. **Partition shared resources at runtime to reduce interference**
   - Heracles, Ubik, Rubik

3. **Reduce interference by throttling applications at runtime**
   - Bubble-Flux, ReQoS, Protean Code

- **But …**
  - Server utilization by disallowing certain co-locations
  - Performance of batch applications by treating them as low-priority
**Approximate computing applications**
- Tolerate some loss in output accuracy in return for
  - Improved performance, or
  - Same performance with reduced resources

**Cloud workloads suitable for approximation**
- Performance can be more important than highest output quality

**Co-locate approximate batch apps with interactive services**
- Meet performance for both applications at the cost of some inaccuracy
1. **Mitigate interference:**
   - Approximation can reduce # of requests to memory system & network
   - Approximation may not be always sufficient

2. **Meet performance of approximate applications:**
   - When approximation is not enough, employ resource partitioning:
     » Core relocation
     » Cache partitioning
     » Memory partitioning
   - Provide more resources to interactive service to meet its QoS
   - Approximation preserves the performance of batch applications
Approximation Techniques

- **Loop perforation**: Skip fraction of iterations
  - Fewer instructions & data accesses $\Rightarrow$ exec time $\downarrow$ & cache interference $\downarrow$

- **Synchronization elision**: Barriers, locks elided
  - Threads don’t wait for sync $\Rightarrow$ exec time $\downarrow$
  - Reduces memory accesses for acquiring locks

- **Lower precision**: Reduce precision of variables
  - e.g., replace ‘double’ with ‘float’ or ‘int’
  - Reduces memory traffic

- **Tiling**: Compute 1 element & project onto neighbors
  - Fewer instructions & data accesses $\Rightarrow$ exec time $\downarrow$ & cache interference $\downarrow$
**Approximation Trade-offs**

- 100s of approximate variants
- Pruning design space:
  - Hint-based:
    - Employ approximations hinted by ACCEPT* tool
  - Profiling-based (gprof):
    - Approximate in functions which contribute most to execution time

*ACCEPT: A Programmer-Guided Compiler Framework for Practical Approximate Computing, A. Sampson et. al.
PLIANT: GOALS

- **High utilization**
  - Co-schedule interactive services with approximate applications

- **High QoS**
  - Satisfy QoS of all co-scheduled jobs at the cost of some accuracy loss

- **Minimize accuracy loss**
  - Adjust approximation at runtime using slack in tail latency

- **Techniques used to reduce interference at runtime**
  - Approximation
  - Resource relocation (core relocation, cache & memory partitioning)
Pliant - Overview

- Continuously monitors the tail latency
- Dynamic recompilation
- Runtime allocation

Diagram:
- Client: CPU, LLC, Main Memory, interactive service, workload generator, approximate computing app
- Server: CPU, LLC, Main Memory
- Pliant: Performance monitor, Actuator
- QoS violation
- Design Space Exploration
- Performance monitor
- Actuator
- Requests
- Meet QoS as fast as possible
- Minimize accuracy loss using latency slack when QoS met

**Batch:**
- Precise
- Most-1 Approx
- Most Approx

**Interactive:**
- +1 core

**Latency slack > 10%**

QoS not met

Interference ↓

Performance of approx ↑

Latency slack > 10%

Batch: -1 core Interactive: +1 core
Multiple resources: cores, LLC and memory
**Pliant – Varying Approximation Degree**

- **Dynamic recompilation system**
  - Aggregated approximate variants to construct tunable app
  - Linux signals for DynamoRIO to switch to an approximate variant
  - `drwrap_replace()` interface is used to replace functions
    - Coarse granularity $\rightarrow$ low overheads
Pliant – Runtime Resource Allocation

- All applications run in Docker containers

- Core relocation
  - Docker update interface to allocate cores to each container

- Cache allocation
  - Intel’s Cache Allocation Technology (CAT) to allocate cache ways

- Memory capacity
  - Docker update interface to assign memory limits
**Experimental Setup**

- **Interactive services:** NGINX, memcached, MongoDB
- **24 approximate computing applications:**
  - PARSEC, SPLASH2x, MineBench, BioPerf benchmark suites

- **Systems**
  - 44 physical core dual-socket platform, 128 GB RAM, 56 MB LLC/socket
  - Interactive services & approximate applications pinned to different physical cores of same socket

- **Baseline**
  - Approximate application run in precise mode
  - Cores, cache, and memory shared fairly among the applications
EVALUATION - DYNAMIC BEHAVIOR
EVALUATION – DYNAMIC BEHAVIOR

- Across interactive services
  - memcached and NGINX need to reclaim resources
  - In case of MongoDB, approximation is enough
**Evaluation – Dynamic Behavior**

- **Across approximate applications**
  - Bayesian shows bursty behavior - approximation usually enough
  - In case of SNP, no resource reclamation is required

For all co-schedulings, show QoS is met for all apps at an accuracy loss of up to 5% (2.8% on average)
Approximation can break performance vs utilization trade-off
Many cloud applications can tolerate some loss of quality

Pliant – practical runtime system
- Incremental approximation using dynamic recompilation
- Dynamic allocation of shared resources

Achieves high utilization
- Enabled co-scheduling of approximate batch apps with interactive services

Achieves high QoS
- Meets QoS for all apps at cost of small accuracy loss (max 5%, avg 2.8%)
Approximation can break performance vs utilization trade-off
Many cloud applications can tolerate some loss of quality

Pliant – practical runtime system
- Incremental approximation using dynamic recompilation
- Dynamic allocation of shared resources

Achieves high utilization
- Enabled co-scheduling of approximate batch apps with interactive services

Achieves high QoS
- Meets QoS for all apps at cost of small accuracy loss (max 5%, avg 2.8%)
THANK YOU!