UBIK: EFFICIENT CACHE SHARING WITH STRICT QoS FOR LATENCY CRITICAL WORKLOADS

Harshad Kasture, Daniel Sanchez

ASPLOS 2014
Low server utilization in datacenters is a major source of inefficiency.
Dedicated machines for latency-critical applications guarantees QoS
Dedicated machines for latency-critical applications guarantees QoS

Under utilization of machine resources
Colocation to Improve Utilization

- Can utilize spare resources by colocating batch apps
Sharing Causes Interference!

- Can utilize spare resources by collocating batch apps
- Contention in shared resources degrades QoS
Outline

- Introduction
- Analysis of latency-critical apps
- Inertia-oblivious cache management schemes
- Ubik: Inertia-aware cache management
- Evaluation
- Large number of backend servers participate in handling every user request
  - Total service time determined by tail latency behavior of backend
Service latency highly sensitive to changes in load
- Short bursts of activity interspersed with idle periods
  - Need guaranteed high performance during active periods
Inertia and Transient Behavior

![Diagram of cores and cache]

IPC vs. Time graph

Core 0, Core 1, Core 2, Core 3, Core 4, Core 5

Last Level Cache
Transient lengths can dominate tail latency!
- Any dynamic reconfiguration scheme has to be inertia-aware

Many hardware resources exhibit inertia
- branch predictors, prefetchers, memory bandwidth...
- LLCs are one of the biggest sources of inertia
Outline

- Introduction
- Analysis of latency-critical apps
- Inertia-oblivious cache management schemes
- Ubik: Inertia-aware cache management
- Evaluation
Inertia-Oblivious Cache Management

Last Level Cache (LLC)

Core 0  Core 1
Batch1  Batch2

Core 2  Core 3

Active  Idle
Active  Idle
Active  Idle

Time
Unmanaged LLC (LRU Replacement)

Unconstrained interference results in poor tail-latency behavior.
Utility Based Cache Partitioning (UCP)

- **High batch throughput**
- **Poor tail latency (low allocation)**

Diagram showing the partitioning of cache space between two batches. LC1 and LC2 represent the last level cache (LLC) divided into two cores (Core 2 and Core 3, respectively). Batch1 and Batch2 represent two separate tasks or workflows. The diagrams illustrate the allocation of active and idle states over time, indicating how the cache space is utilized across the different batches.

Graphs show the allocation of LLC space over time, with active and idle states marked for different batches and time periods. The reconfiguration points are indicated by the arrows, showing when the allocation might change to optimize performance.
OnOff: Efficient but Unsafe

High batch throughput
Cross-Request LLC Inertia

- Other applications qualitatively similar (see paper for details)
StaticLC: Safe but Inefficient

- ✔ Low tail latency (preserve LLC state)
- ✗ Low batch throughput (poor space utilization)
Outline

- Introduction
- Analysis of latency-critical apps
- Inertia-oblivious cache management schemes
- Ubik: Inertia-aware cache management
- Evaluation
Performance as well as overall progress under Ubik after the deadline is identical to static partitioning.
Ubik: Overview

- Activity
- Time
- Size
  - nominal static size
  - idle size

- Target Size
- Actual Size
Ubik: Overview

![Graph showing activity and size over time with different states: nominal static size, boosted size, nominal static size, and idle size.]
Ubik: Overview

Activity

Size

boosted size
nominal static size
idle size

Target Size
Actual Size
Constraint: Cycles lost during \( \text{pink} \) should be compensated for by the cycles gained during \( \text{green} \) before the deadline.
Analyzing Transients

- Need **accurate** predictions for
  - The length of the transient from \( s_1 \) to \( s_2 \)
  - Cycles lost during the transient from \( s_1 \) to \( s_2 \)
Hardware Support

- Utility monitors to measure per-application miss curves
- Fine grained cache partitioning
- Memory Level Parallelism (MLP) profiler
Bounds on Transient Behavior

\[ T_{\text{transient}} = \sum_{s=s_1}^{s_2-1} \frac{c}{p_s} + M \leq (s_2 - s_1) \left( \frac{c}{p_{s_2}} + M \right) \]

\[ L = M \sum_{s=s_1}^{s_2-1} \left( 1 - \frac{p_{s_2}}{p_s} \right) \leq M \left( s_2 - s_1 \right) \left( 1 - \frac{p_{s_2}}{p_{s_1}} \right) \]
Use transient analysis to identify feasible \((\text{idle size}, \text{boosted size})\) pairs.
Use transient analysis to identify feasible (idle size, boosted size) pairs
Use transient analysis to identify feasible (idle size, boosted size) pairs
Use transient analysis to identify feasible \((\text{idle size, boosted size})\) pairs
Use transient analysis to identify feasible (idle size, boosted size) pairs

Choose the pair that yields the maximum batch throughput

See paper for details
Outline

- Introduction
- Analysis of latency-critical apps
- Inertia-oblivious cache management schemes
- Ubik: Inertia-aware cache management
- Evaluation
Workloads

- Five diverse latency-critical apps
  - xapian (search engine)
  - masstree (in-memory key-value store)
  - moses (statistical machine translation)
  - shore-mt (multi-threaded DBMS)
  - specjbb (java middleware)

- Batch applications: random mixes of SPECCPU 2006 benchmarks
Target System

- 6 OOO cores
  - Private L1I, L1D and L2 caches
  - 12MB shared LLC

- 400 6-app mixes: 3 latency-critical + 3 batch apps
  - Apps pinned to cores
- Baseline system has private LLCs
- We report
  - Normalized tail latency
  - Throughput improvement for batch applications
Results: Unmanaged LLC (LRU)

Higher is better
Results: UCP

Higher is better
Results: OnOff

Higher is better
Results: StaticLC

Higher is better
Results: Ubik

Higher is better
Results: Summary

- **LRU**
- **OnOff**
- **UCP**
- **Private LLC**
- **StaticLC**
- **Ubik**

*Higher is better*
To guarantee tail latency, dynamic resource management schemes must be inertia-aware.

Ubik: Inertia-aware cache capacity management
- Preserves tail of latency-critical apps
- Achieves high cache space utilization for batch apps
- Requires minimal additional hardware
THANKS FOR YOUR ATTENTION!

QUESTIONS?