Workload Colocation: Pitfalls & Potential Solutions

Kubernetes Task Force of the 10th China Open Source Hackathon

Dec. 2019

Why Workload Colocation Is Difficult

Complicated Nature of Resource Sharing

- CPU resource scheduling
- On-chip resource management
- I/O interference
- Evolving hardware

Lack of Infrastructure Support

- Cluster management & monitoring mechanism

We Are Not Ready for It

- Mindset: static optimization vs. runtime data-driven optimization

CPU

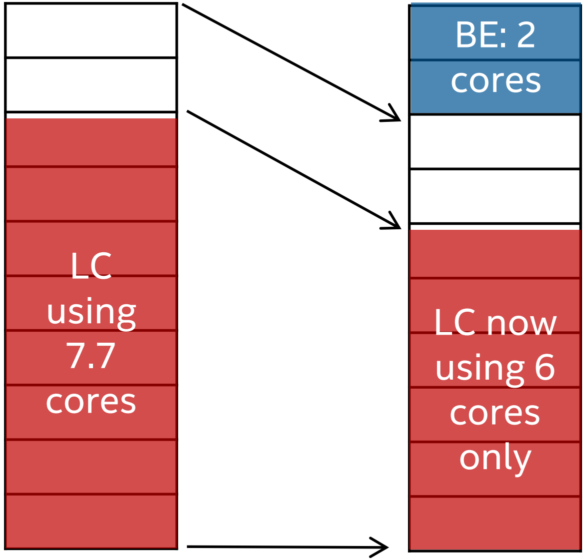
Probably the Most Invested Area in Resource Sharing

- (Supposed to be) well monitored
- Established scheduler in OS
- Completely fair scheduler (CFS) & Linux control group mechanism

Yet CPU Performance Isolation Is an Unsolved Problem

CPU Scheduling Is Not Perfect (1)

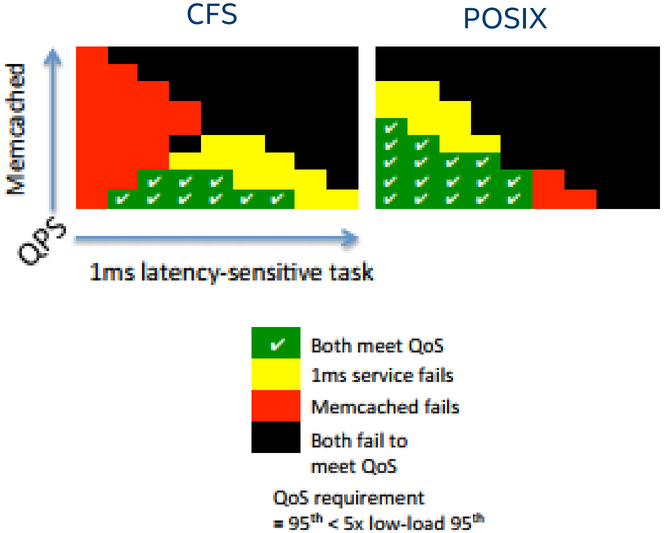
NoSQL database (LC) vs CPU hog (BE)



Median: 0.4ms
P99: 1.0ms

Median: 0.4ms
P99: 4.1ms

Scheduling delay impacts low-latency jobs: Memcached vs 1ms RPC service under CFS & POSIX real-time scheduler + CPU quota (Leverich & Kozyrakis 2014)



J. Leverich & C. Kozyrakis (2014). Reconciling high server utilization and sub-millisecond quality-of-service. In *EuroSys '14*.

CPU Scheduling Is Not Perfect (2)

Linux Scheduler Has Bugs (Lozi et al. 2016)

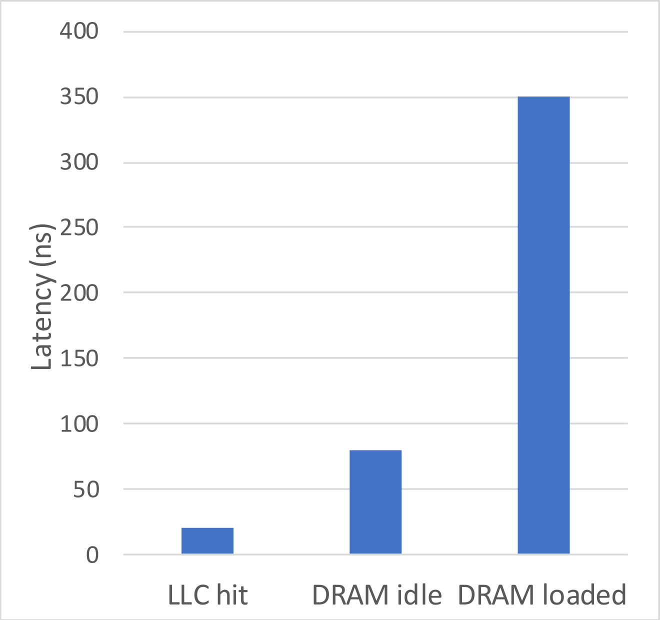
- Colocating a 64-thread workloads w/ a 2-thread workload in a supposedly fully loaded scenario → sometimes idle cores
- Caused by
  - Load tracking metrics relevant to thread number
  - Load stealing heuristics in hierarchical load balancing

Potential Solutions to Imperfect CPU Scheduling

- Leave enough headroom, e.g., effective headroom concept in Platform Resource Manager (PRM: https://github.com/intel/platform-resource-manager/)
- Pin workloads to cores whenever applicable (e.g., Iorgulescu et al. 2018)
- Look at scheduling delay, improve scheduler, etc.

J.-P. Lozi, B. Lepers, J. Funston, F. Gaud, V. Quéma & A. Fedorova (2016). The Linux scheduler: a decade of wasted cores. In *EuroSys '16*.
C. Iorgulescu, R. Azimi, Y. Kwon, S. Elnikety, M. Syamala & V. Narasayya (2018). PerfIso: Performance Isolation for Commercial Latency-Sensitive Services. In *USENIX ATC '18*, 519-531.

On-chip Resource Contention Matters (1)



Last-level Cache (LLC)

- Shared among cores
- Replacement policy biased towards keeping recently used items
  - Colocating a workload w/ a sequential scan → swipes out the other workload's useful cache items

Memory Controller

- Queueing memory referencing requests at the CPU socket level
  - Processing delay adds to the waiting time of other requests in queue
  - Workload colocation → more likely a longer queue → more likely longer waiting time

On-chip Resource Contention Matters (2)

Example: Extent of last-level cache contention varies (Lo et al. 2015)

websearch

	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%
LLC (small)	134%	103%	96%	96%	109%	102%	100%	96%	96%	104%	99%	100%	101%	100%	104%	103%	104%	103%	99%
LLC (med)	152%	106%	99%	99%	116%	111%	109%	103%	105%	116%	109%	108%	107%	110%	123%	125%	114%	111%	101%
LLC (big)	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	264%	222%	123%	102%

ml_cluster

	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%
LLC (small)	101%	88%	99%	84%	91%	110%	96%	93%	100%	216%	117%	106%	119%	105%	182%	206%	109%	202%	203%
LLC (med)	98%	88%	102%	91%	112%	115%	105%	104%	111%	>300%	282%	212%	237%	220%	220%	212%	215%	205%	201%
LLC (big)	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	276%	250%	223%	214%	206%

memkeyval

	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%
LLC (small)	115%	88%	88%	91%	99%	101%	79%	91%	97%	101%	135%	138%	148%	140%	134%	150%	114%	78%	70%
LLC (med)	209%	148%	159%	107%	207%	119%	96%	108%	117%	138%	170%	230%	182%	181%	167%	162%	144%	100%	104%
LLC (big)	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	280%	225%	222%	170%	79%	85%

One size does not fit for all!

D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, & C. Kozyrakis (2015). Heracles: improving resource efficiency at scale. In *ISCA '15*, 450-462.

Data-Driven Contention Management

Application-Aware Isolation (e.g., Google's Heracles)

- Detect issues based on application performance monitoring
- Manage resource based on performance slack & application profile using core allocation & Intel Resource Director Technology (RDT)

Application-Agnostic Contention Detection & Isolation (e.g., Google's CPI^2 (Zhang et al. 2013), PRM)

- Monitor low-level platform counters only
- Identify contention w/ outlier (deviation from normal cases) detection or meta learning in noisy environment (Shen & Li 2019), & determine contention resource type
- Throttle CPU usage (in CPI^2) or low-level resource usage using Intel RDT (in PRM)

X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, & J. Wilkes (2013). CPI^2: CPU performance isolation for shared compute clusters. In *EuroSys '13*, 379-391.
H. Shen & C. Li (2019). Detecting last-level cache contention in workload colocation with meta learning. To appear in *IISWC '19*.

Cache Contention Detection & Isolation in PRM

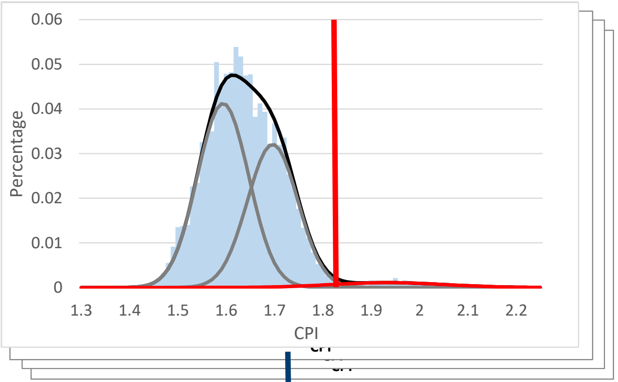
Performance counter monitoring

Counter	Value
Cycles	59637914522
Instructions	36587677620
Cache misses	542595259
CPU load	0.67
…	…

Multiple key metrics

Metrics	Value
CPI	1.63
MPKI	14.83
CPU load	0.67
…	…

Modeling historical metrics w/ [finite mixture model](#) & compute the outlier detection threshold



Runtime metrics on individual servers

CPI > 1.83 & **MPKI > 15.72 & CPU load ∈ [0.65, 0.7]** → LLC contention

[BE Throttling using Cache Allocation Technology](#)

Disk I/O

Unstable Factor in Performance

- Early days: rotating hard drive → significant penalty in seeking
- Modern age: flash translation layer in NAND SSD (write on erased page & erase at block level) → unpredictable performance in scattered write
- Both exacerbated in workload colocation

Potential Solutions

- Cgroup v2 I/O bandwidth limiting
- Budget fair queueing scheduler to improve responsiveness (Valente & Andreolini 2012)
- Rearchitect modern multi-queue mechanism (Bjørling et al. 2013)
- Physical isolation: one drive for LC job & another for BE

P. Valente & M. Andreolini (2012). Improving application responsiveness with the BFQ disk I/O scheduler. In *SYSTOR '12*.
M. Bjørling, J. Axboe, D. Nellans & P. Bonnet (2013). Linux block IO: introducing multi-queue SSD access on multi-core systems. In *SYSTOR '13*.

Network I/O

Interference on Network I/O

- Can happen in both incoming & outgoing direction

Potential Solutions

- Outgoing direction: Linux traffic control like qdisc scheduler w/ hierarchical token bucket (HTB) queueing discipline
- Incoming direction: solutions similar to EyeQ (Jeyakumar et al. 2013)

V. Jeyakumar, M. Alizadeh, D. Mazieres, B. Prabhakar, C. Kim & A. Greenberg (2013). EyeQ: Practical Network Performance Isolation at the Edge. In NSDI '13, 297-311.

Evolving Hardware

Evolving Hardware

- Accelerators, e.g., GPU, FPGA, ASIC
- New memory, e.g., Intel Optane memory
- New storage, e.g., key-value SSD (Kang et al. 2019)
- SmartNIC, e.g., Azure Accelerated Networking (Firestone et al. 2018)

Each Brings Unique Challenge to Interferences

- Need case-by-case analysis, e.g.,
  - Accelerators not good for sharing (Zhu et al. 2019)
  - Workload colocation on Optane memory needs to be handled carefully

Y. Kang, R. Pitchumani, P. Mishra, Y.-S. Kee, F. Londono, S. Oh, J. Lee, J. Lee, D. D. G. Lee. Towards Building a High-performance, Scale-in Key-value Storage System. In *SYSTOR '19*, 144-154.
D. Firestone et al. (32 authors) (2018). Azure accelerated networking: SmartNICs in the public cloud. In *NSDI '18*, 51-64.
H. Zhu, D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan & M. Erez (2019). Kelp: QoS for accelerated machine learning systems. In *HPCA '19*, 172-184.

Monitoring Is Tricky

Monitoring Is Important

- Application performance monitoring to understand performance implication
- CPU utilization monitoring for resource allocation (initial quota & workload placement) & adjustment (in second granularity)
- Low level platform telemetry monitoring for contention detection & isolation

Monitoring Is Difficult

- Application performance monitoring is not always possible, & is difficult at the right granularity for contention mitigation
- System reported utilization can be misleading: CPU utilized in stalling for code/data
- Low level platform telemetry monitoring for workloads may sometimes introduce 7%~10% impact to tail latency of sub-millisecond QoS

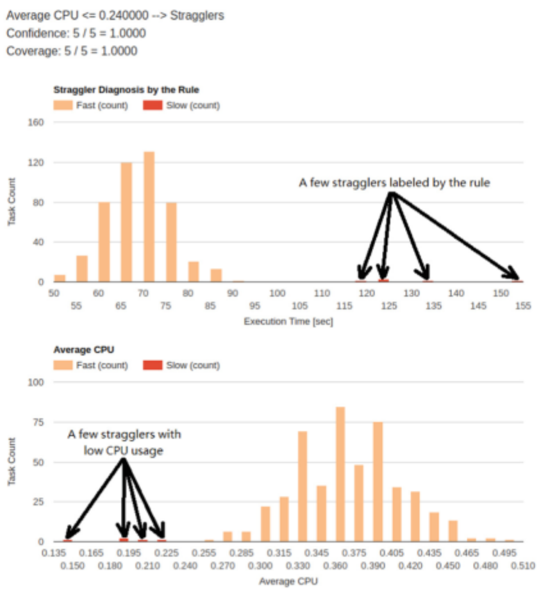
Cluster Management Infrastructure

Basic Support

- Monitoring, resource allocation & oversubscription
- Placement, core-pinning (optional), migration (optional)

Advanced Analytics

- Contention-aware colocation based on history statistics (Kambadur et al. 2012)
- Throttling impact analysis at cluster level
  - Arbitrarily throttling BE can sometimes be costly (Shen & Li 2018)



M. Kambadur, T. Moseley, R. Hank, & M. A. Kim (2012). Measuring interference between live datacenter applications. In *SC '12*.
H. Shen & C. Li (2018). Zeno: a straggler diagnosis system for distributed computing using machine learning. In *ISC High Performance '18*, 144-162.

Are We Ready?

Mindset Change

- Old days: optimizing single benchmark instance at peak intensity
- Nowadays: runtime optimization in realistic workload colocation environment given varying load at cluster level

Focus

- Right context: multiple workloads, varying load, large-scale analysis (Lee et al. 2018)
- Right metrics in evaluation
  - Example: device idling in BFQ improves responsiveness of interactive application but degrades total system throughput (Valente & Avanzini 2015)

J. Lee, C. Kim, K. Lin, L. Cheng, R. Govindaraju, & J. Kim (2018). WSMeter: a performance evaluation methodology for Google's production warehouse-scale computers. In *ASPLOS '18*, 549-563.
P. Valente & A. Avanzini (2015). Evolution of the BFQ storage-I/O scheduler. In *MST '15*, 15-20.

Backup

PRM: Finite Mixture Model & Outlier Detection

Finite Mixture Model

- $P(X = x) = \sum_{i=1}^{k} P(M = m_i) P(X = x | M = m_i)$
  - Each model $P(X = x|M = m_i)$ in a mixture is a normal distribution
- All the parameters estimated through Expectation-Maximization algorithm
  - Initialized with k-means algorithm
- Number of models (k) in a mixture determined by Bayesian Information Criterion
  - $-2\sum_{x \in \aleph} \log P(X = x) + 2k\log|\aleph|$

Outlier Detection Threshold

- Filtering minor components in the mixture within a small probability δ
- Determine the threshold based on the next component

Back to presentation

PRM: BE CAT Throttler

Naïve Heuristic

- Significantly biased to LC performance
- Taking minimum input
  - Contention signal only, but not on any contention metrics or performance implication
- Idea similar to Papadakis et al. (2017)

Simple Algorithm

- Once contention detected, then go for maximum level of throttling
- Once no contention detected in d cycles, then reduce throttling by 1 level only
  - LLC way-partitioning

I. Papadakis, K. Nikasm, V. Karakostas, G. I. Goumas, & N. Koziris (2017). Improving QoS and utilisation in modern multi-cores with dynamic cache partitioning. In *COSH-VisorHPC 2017*.

Back to presentation

PRM: Performance Isolation Experiments

Workloads

- LC: Memcached (pinned to cores), Cassandra
- BE: TensorFlow

Baselines

- No colocation
- Static CPU quota
- Dynamic CPU control (w/ effective headroom)
- Dynamic CPU control + CPU throttling for cache contention
- Dynamic CPU control + CAT throttling (Intel RDT) for cache contention

PRM: Performance Isolation Results