Alibaba Cloud
Official Cloud Services Partner

System Memory QoS Challenge

More Than Just Cloud

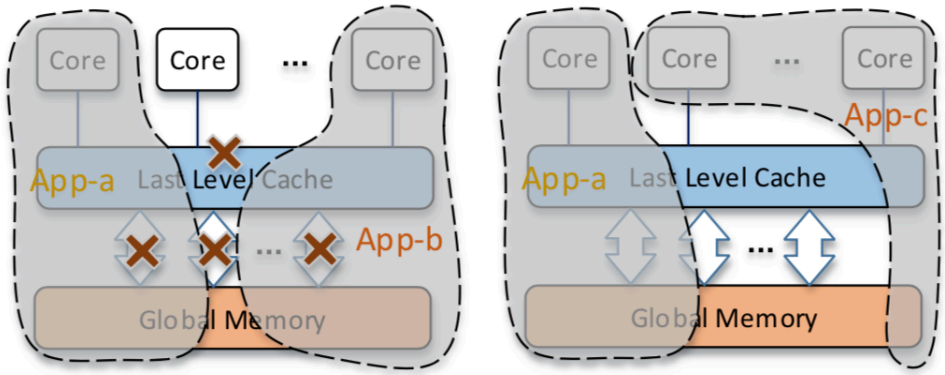
Oliver Yang

Jun 5, 2018

1. Background

Service Consolidation & Co-location Practice

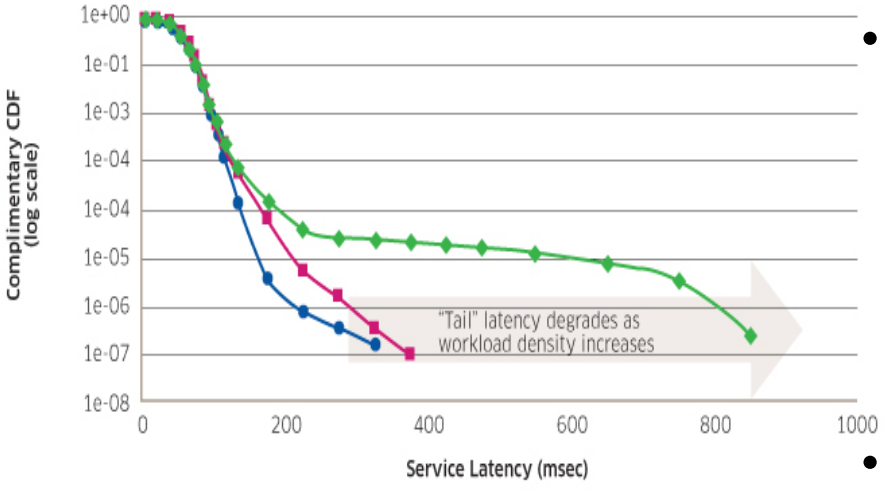
- Goal: Increase data center hardware utilization
- Idea: Consolidate multiple workloads with acceptable QoS loss
  - Borg/Kubernetes, VMware DRS etc.
- Challenge: Avoid contentions among multiple workloads
  - Workload characterizations
    - Resource model: Compute bound + IO bound
    - Time sharing: Peak time + Off-peak time
  - OS resource control & isolation features



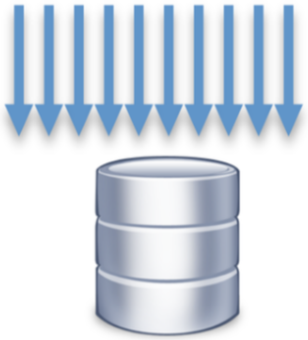
More Challenges In Webscale Data Center

- Scale-out optimization at large scale
  - Resource QoS classifications & scheduling
  - Workload placements(Bin-packing) algorithm
  - Dynamic distributed load balance & scheduling
- Constantly varying load challenges from cloud native applications
  - The synthetic test environment has poor coverage
  - Many issues can be only found on real online environment
  - Using statistics to determine the impact of a change
  - Performance or resource characters could be totally changed after a time period
  - Paper: Performance Analysis of Cloud Applications – NSDI 2018

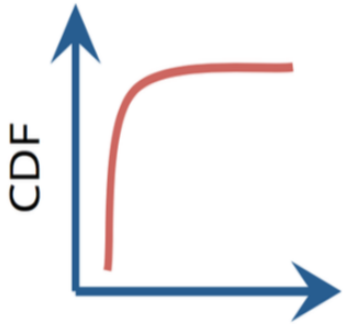
Co-location Jitters & Causes



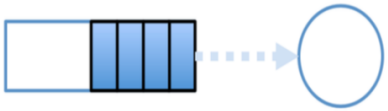
- System performance jitters RCA (Root Cause Analysis) can be very challenge…
  - Burst randomly
  - In a very short period
  - Reported/found very late
  - Always postmortem analysis
  - Without enough debug data
- Tradeoff between utilization and QoS
  - Avoid jitters by resource isolations



HW Resource Contentions



Skewed Access Pattern



Queuing Delays

Background Activities

OS Resource Control & Isolation

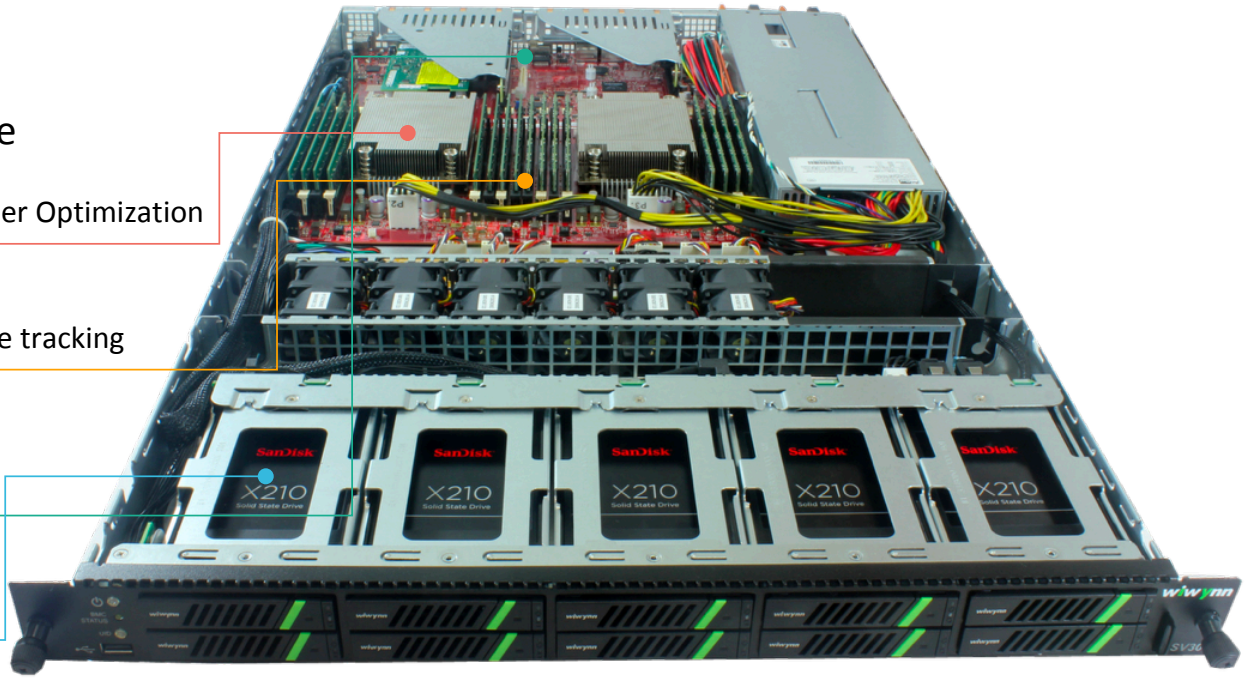
- System wide optimizations
  - Hardware & Software cooperate together
  - Full stack optimizations
- 3 major aspects
  - Performance isolation
  - Fault isolation
  - Utilization Increase

CPU: CAT, Cgroup, Noise Clean, Scheduler Optimization

Memory: MBM, MBA, Cgroup, Idle page tracking

Network: TC, HTB, Lockless Qdisc

IO: Write back Cgroup, Low/Upper limit，Weight, File Level-Based IO Throttle



Why Memory Matters

- DRAM cost is close to CPU recently
- Memory is incompressible resource
- Challenges
  - Capacity planning is difficult: OOM, Various Jitters, Bad utilization
  - Memory overcommit has more risks: OOM, Jitter
  - The bottlenecks of compute density
    - Big data, Tensorflow…
    - 4 NUMA node servers
- Opportunities
  - New NVM hardware
    - AEP, Optane, NVME SSD
  - AI technologies

Memory QoS Requirements

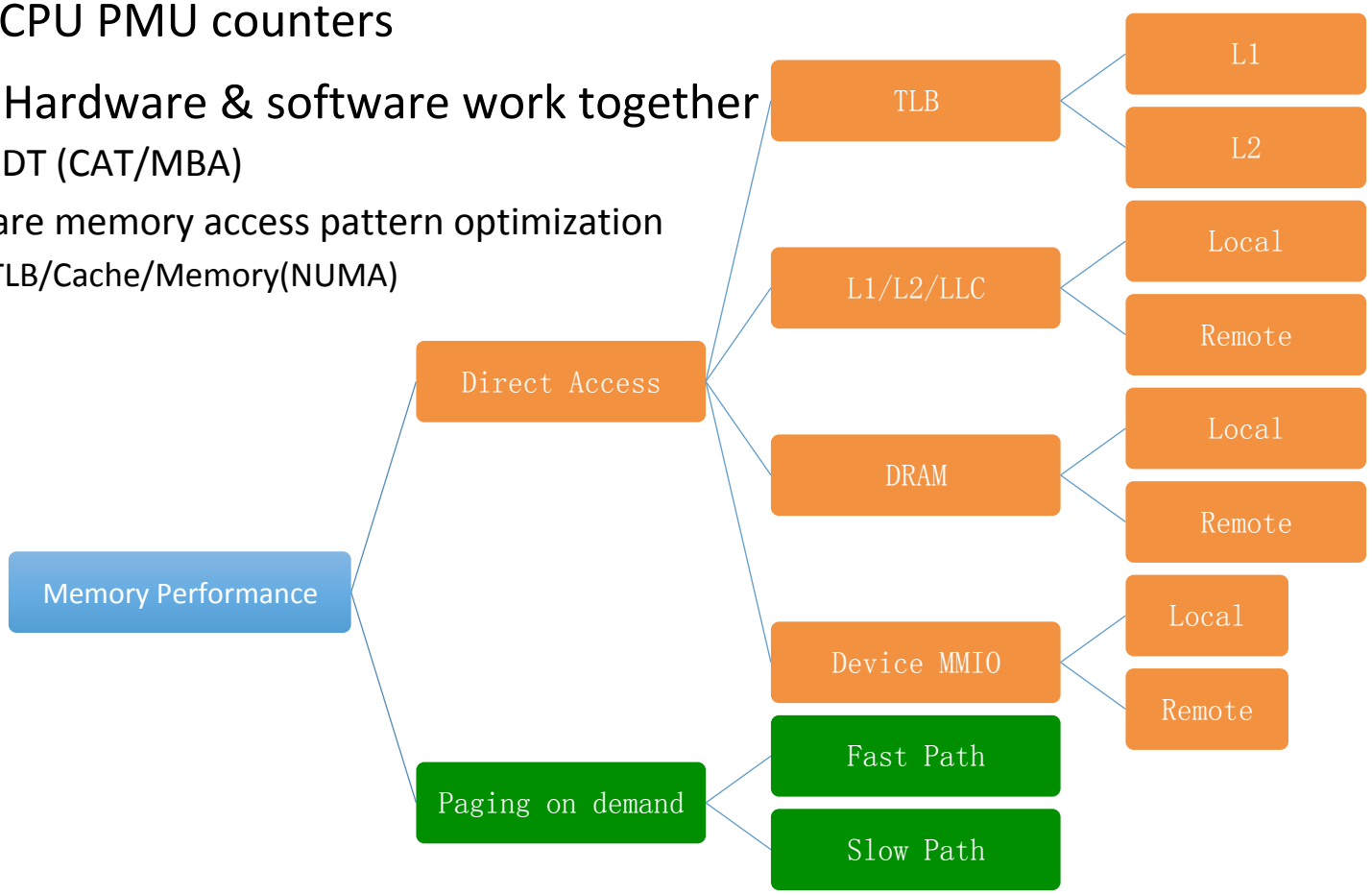
- Performance Isolation
  - Memory access performance shouldn’t be interfered by other containers/VMs.
- Fault Isolation
  - Memory faults should be self-contained and shouldn’t have global QoS impacts
- Utilization Increase
  - Memory utilization should be increased without QoS regressions

Perf Isolation	Fault Isolation	Utilization Increase
• Throughput • Latency	• Software fault • Hardware fault	• Capacity Planning • Memory Overcommit

2. Performance Isolation

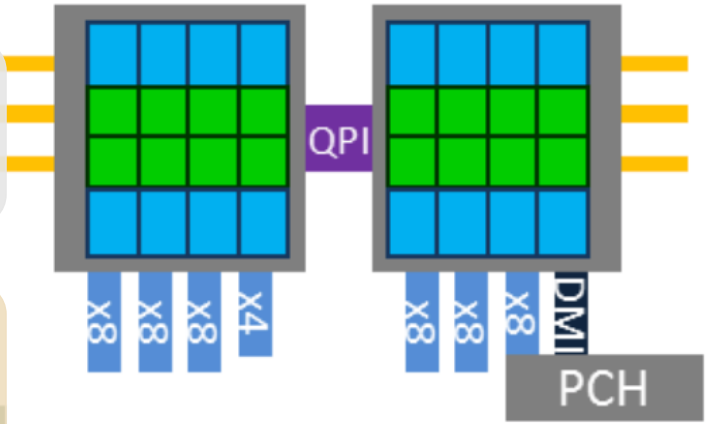
Performance Isolation - Direct Memory Access

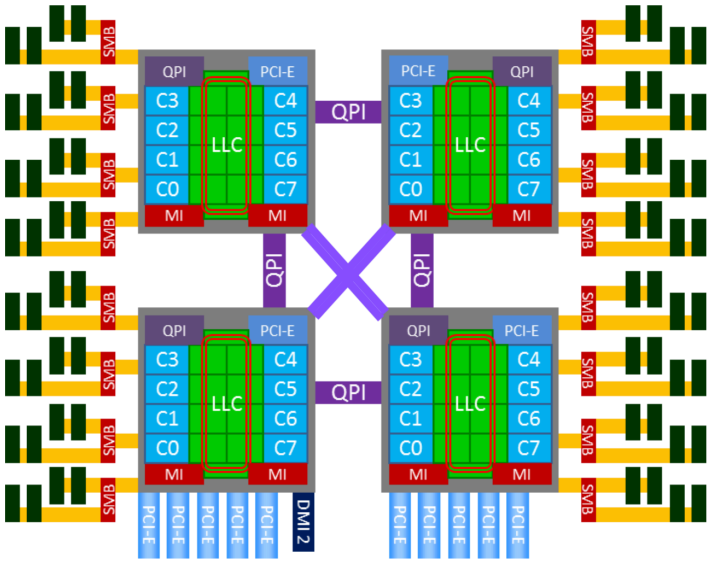
- Problem: Hardware resource contention
- Analysis: CPU PMU counters
- Solution: Hardware & software work together
  - Intel RDT (CAT/MBA)
  - Software memory access pattern optimization
    - TLB/Cache/Memory(NUMA)



Perf Metrics For Direct Memory Access

CPI/IPC/MIPS	• Perf stat - IPC • Pmu-tools – toplev, ocperf
TLB Hit/Miss/ Flush	• Perf stat – tlb miss, flush events • perf mem – access ratio
Cache Hit/Miss	• Perf stat – cache miss events • Perf mem – access ratio • Perf c2c – access ratio, latency
Memory Latency/BPS/ IOPS	• Resctrl - CQM and MBM • Numatop - Local/Rmote • Perf c2c – access ratio, latency





Notes: Some metrics have performance overheads

PMU Example 1 - IPC/Branch Miss/Cache Hit Profiling

- PMU counters analysis based on perf stat

  https://github.com/brendangregg/pmc-cloud-tools

```
$sudo ./pmcarch -p 34756
K_CYCLES    K_INSTR      IPC BR_RETIRED    BR_MISPRED   BMR% LLCREF      LLCMISS     LLC%
1583546     950812      0.60 189045873     2518242      1.33 29417171    4461836     84.83
1656505     1047047     0.63 192707027     2695387      1.40 30859621    4631101     84.99
1654197     1053233     0.64 192701567     2629283      1.36 30106516    4585228     84.77
1638629     992180      0.61 193536802     2729231      1.41 30132460    4537869     84.94
1599697     1075443     0.67 191747754     2687716      1.40 30495914    4545437     85.09
1701018     1033800     0.61 189594544     2643653      1.39 29421056    4617382     84.31
1653737     1031279     0.62 193806093     2621252      1.35 31090183    4519754     85.46
1616909     1032686     0.64 191847208     2675521      1.39 30611086    4634774     84.86
1658340     1060700     0.64 192448679     2549817      1.32 30981661    4565961     85.26
1605295     1076172     0.67 193239820     2702305      1.40 29376881    4538589     84.55
1646236     1064070     0.65 191875322     2702808      1.41 29424180    4450857     84.87
1607596     967716      0.60 188095209     2458454      1.31 30369607    4588897     84.89
1582847     1078681     0.68 190811604     2727363      1.43 28932521    4501466     84.44
1643928     1047685     0.64 196892387     2585929      1.31 30914223    4503705     85.43
1667067     1042170     0.63 189144235     2542124      1.34 29852498    4555005     84.74
1626384     1036131     0.64 193389218     2612645      1.35 30568228    4559701     85.08
1645485     1052296     0.64 197918946     2702345      1.37 30188062    4658338     84.57
1664933     1028350     0.62 191910705     2668139      1.39 30427900    4489223     85.25
1604389     1055862     0.66 193450208     2715620      1.40 29934664    4486881     85.01
1632274     1062373     0.65 186048650     2609909      1.40 30109942    4671752     84.48
^C
```

PMU Example 2 - Direct Memory Access Sampling

```
# To display the perf.data header info, please use --header/--header-only options.
#
# Samples: 58K of event 'cpu/mem-loads/pp'
# Total weight : 868972
# Sort order   : comm,tlb,mem,,,,,
#
# Overhead       Samples  Command          TLB access              Memory access
# ........  ............  ...............  ......................  ........................
#
    35.29%         32510  sysbench         L1 or L2 hit            L1 hit
    24.42%          1471  sysbench         L1 or L2 hit            LFB hit
    20.74%         21045  swapper          L1 or L2 hit            L1 hit
     5.26%           299  sysbench         L1 or L2 hit            L3 miss
     4.37%           482  swapper          L1 or L2 hit            LFB hit
     1.64%           165  swapper
     1.37%           181  sysbench
     1.24%           596  sysbench
     1.21%           176  swapper
     0.89%           433  swapper
     0.68%            37  swapper
     0.47%            26  sysbench
     0.26%           110  sysbench
     0.25%             8  swapper
     0.23%            17  sysbench
     0.15%           118  swapper
```

```
# To display the perf.data header info, please use --header/--header-only options.
#
# Samples: 53K of event 'cpu/mem-loads/pp'
# Total weight : 933774
# Sort order   : comm,tlb,mem,,,,,
#
# Overhead       Samples  Command   TLB access              Memory access
# ........  ............  ........  ......................  ........................
#
    53.12%         48581  sysbench  L1 or L2 hit            L1 hit
    32.78%          2239  sysbench  L1 or L2 hit            LFB hit
     7.55%           546  sysbench  L1 or L2 hit            L3 miss
     2.27%           339  sysbench  L1 or L2 hit            L3 hit
     2.17%          1067  sysbench  L1 or L2 hit            L2 hit
     0.58%            38  sysbench  L2 miss                 L3 miss
     0.43%            34  sysbench  L2 miss                 L3 hit
     0.40%           211  sysbench  L1 or L2 hit            Uncached hit
     0.33%            36  sysbench  L2 miss                 L2 hit
     0.25%            26  sysbench  L2 miss                 L1 hit
     0.08%             4  sysbench  L1 or L2 hit            Remote RAM (1 hop) hit
     0.03%             2  sysbench  L2 miss                 LFB hit
     0.01%             1  sysbench  L1 or L2 hit            Local RAM hit
```

- Using perf mem sampling
- System wides & per-task

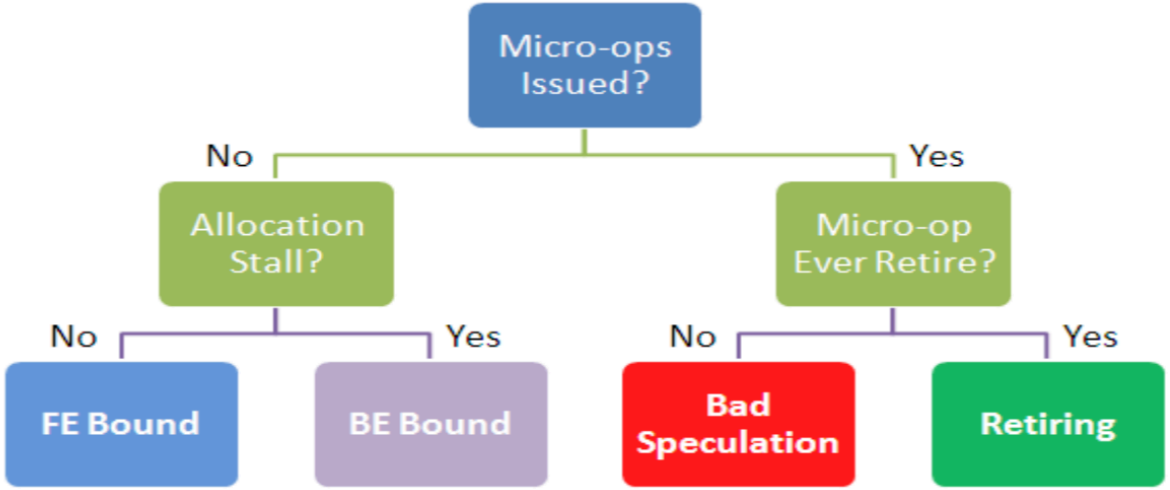
Detecting Noisy Neighbors - CPI/IPC

- CPI - Cycle Per Instruction
  - IPC = 1 / CPI
  - Paper - CPI2: CPU performance isolation for shared compute clusters, Eurosys 2013

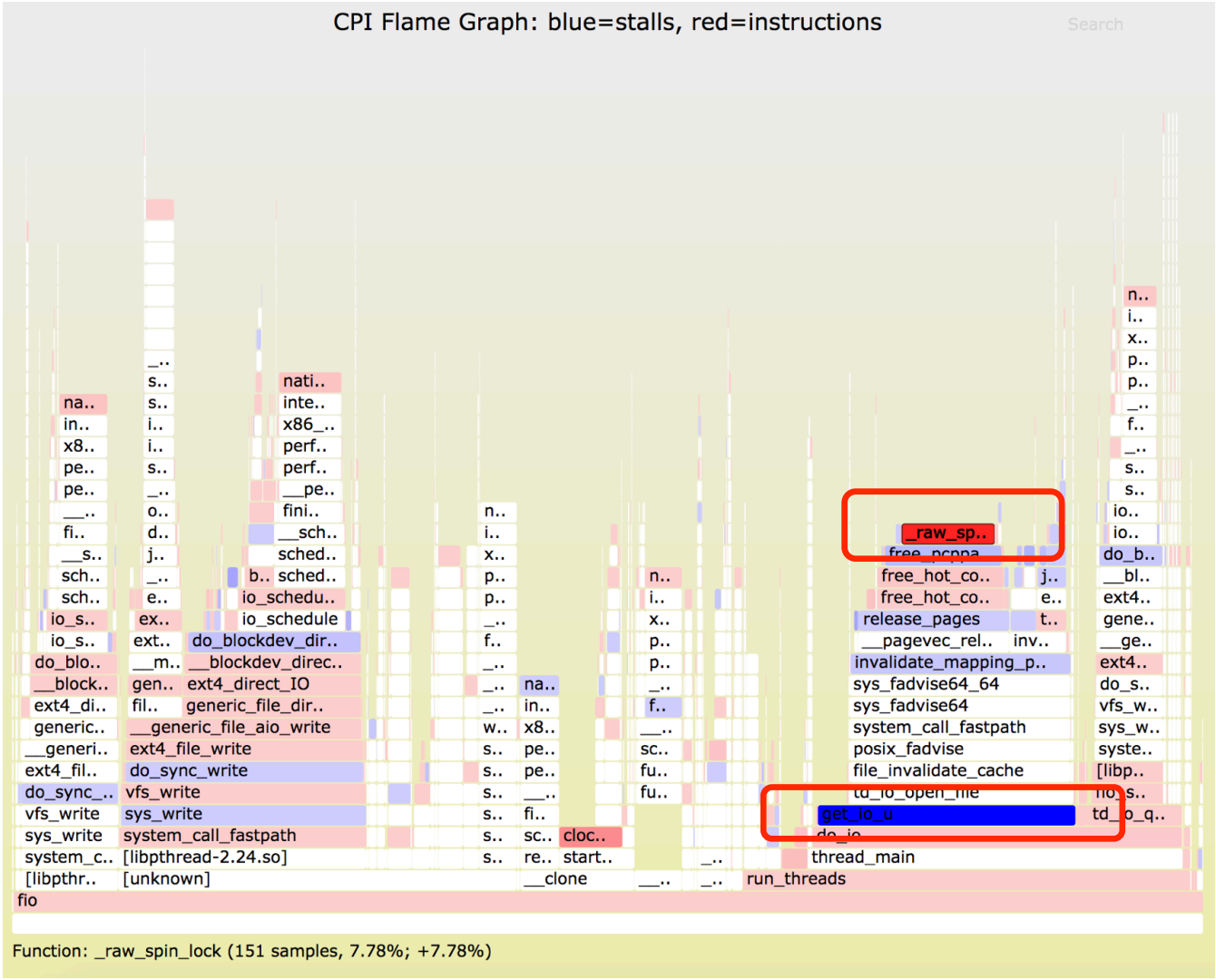
$$\text{Execution time}(T) = \text{CPI} \times \text{Instruction count} \times \text{clock time}$$

$$\text{Effective processor performance} = \text{MIPS} = \frac{\text{clock frequency}}{\text{CPI}} \times \frac{1}{1\ \text{Million}}$$

- Intel Top-down Microarchitecture Analysis Method
  - PMU tools: https://github.com/andikleen/pmu-tools/wiki/toplev-manual

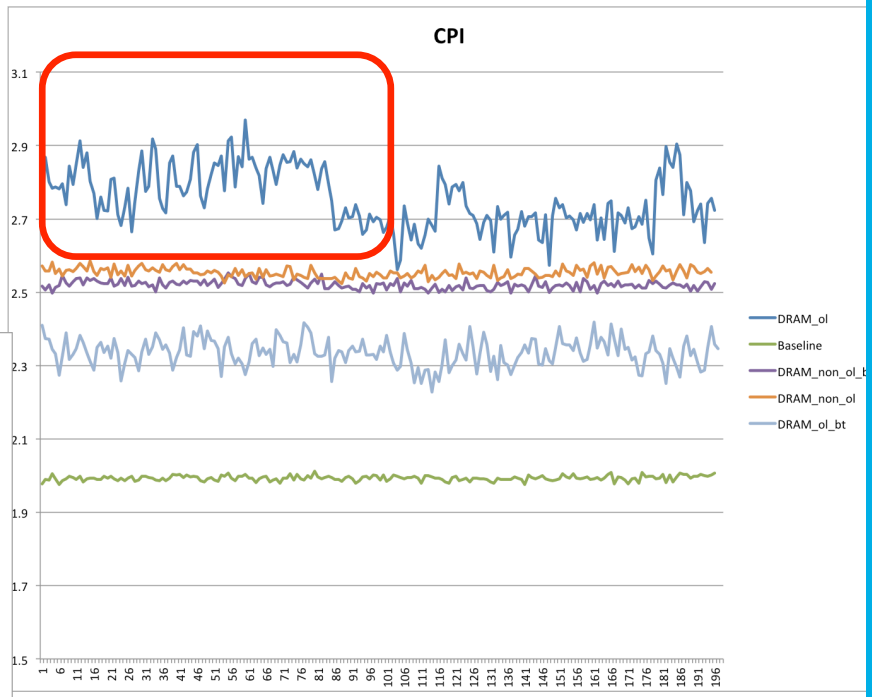


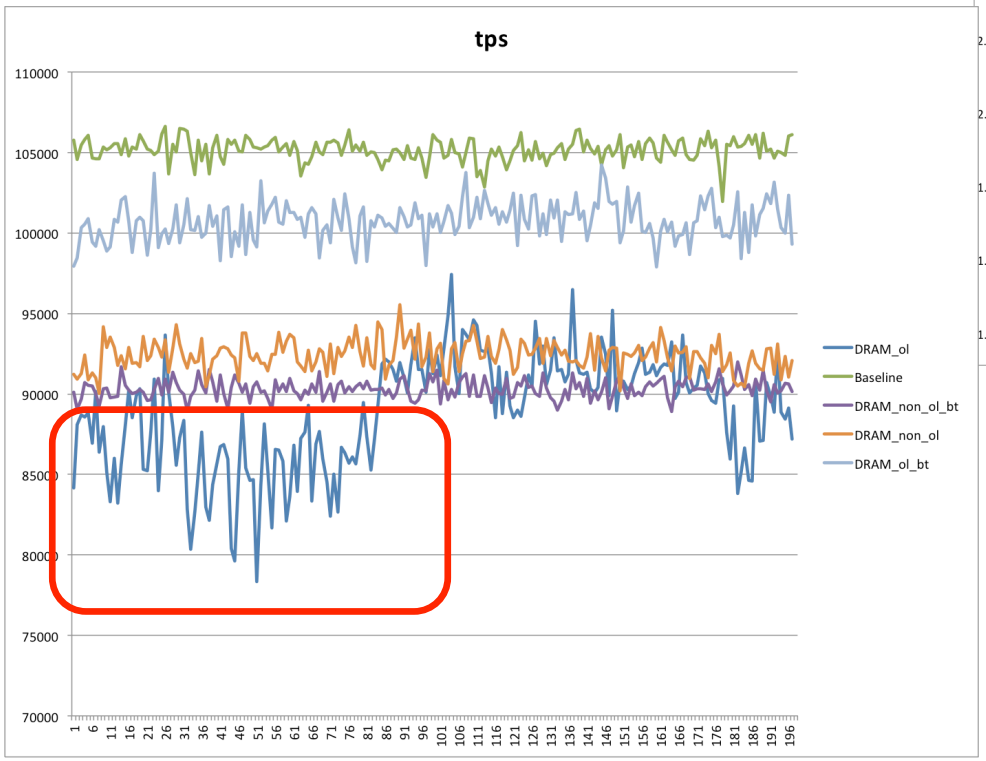
Detecting Noisy Neighbors - CPI Flame Graph



Case Study 1 - Contention Detection By CPI

- Under sysbench & memory interference testing, MySQL TPS showed strong correlation with CPI





Notes: While MySQL TPS got dropped, the CPI was increased significantly

Case Study 2 - Fix Skew Memory Access Pattern

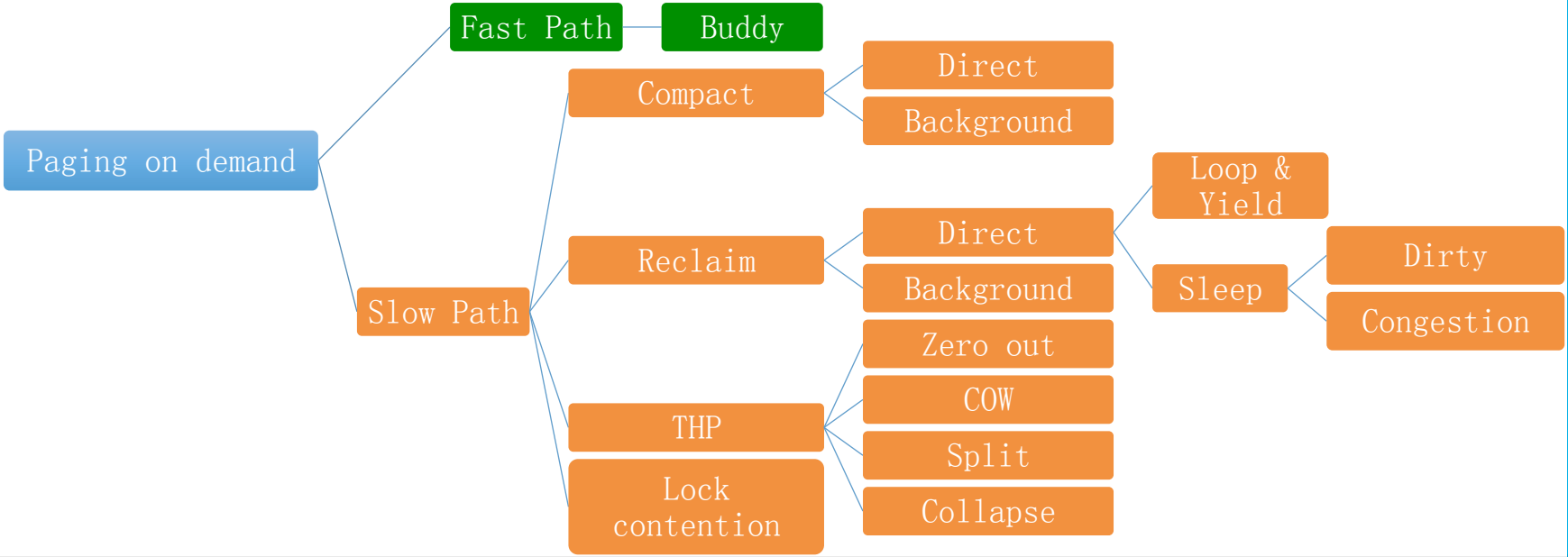
```
=================================================
          Shared Data Cache Line Table
=================================================
#
#                            Total      Tot  ----- LLC Load Hitm -----  ---- Store Reference ----  ---
# Index           Cacheline  records     Hitm    Total      Lcl      Rmt    Total    L1Hit   L1Miss  ...
# .....  ..................  .......  .......  .......  .......  .......  .......  .......  .......  ...
#
      0  0xffffc90041261080      466   25.79%      172        0      172       99       99        0
      1  0xffff887e7f355440       55    4.20%       28        0       28        2        2        0
      2  0xffff887e7f2d5440       46    3.75%       25        0       25        3        3        0
      3  0xffff887e7f255440       46    3.45%       23        0       23        1        1        0
      4  0xffff887e7f215440       37    3.15%       21        0       21        3        3        0
      5  0xffff887e7f335440       47    3.15%       21        0       21        4        4        0
      6  0xffff887e7f2b5440       41    2.70%       18        0       18        5        5        0
      7  0xffff887e7f375440       31    1.80%       12        0       12        3        3        0
      8  0xffff887e7f3b5440       31    1.3         95.35%    0.00%    0.00%    0.00%                0x1c     62900  0xffffffffa05bbeb7
      9      0x7f8263ff2800       10    1.0           |
     10  0xffff887bcd597080       67    0.7           |--0.34%--0xffffffffffffffff
     11  0xffff887713d257c0        9    0.6           |           |
     12  0xffff88771e84f400       11    0.6           |           |--0.31%--__GI___ioctl
     13  0xffff88775872a5c0        8    0.6           |           |          system_call
     14  0xffff887782a5cd00        6    0.6           |           |          sys_ioctl
     15  0xffff887e7f2f5440        6    0.6           |           |          do_vfs_ioctl
     16      0x7f8263fdec00        4    0.6           |           |          0x5f8
     17      0x7f8263feac00        7    0.6           |           |          0x429b
     18      0x7f8263ff1400        5    0.6           |           |          0xeb7
     19  0xffff887713d27540        6    0.4           |           |
                                                      |           --0.03%--__clone
                                                      |                     start_thread
                                                      |                     vrbd_polling_run
                                                      |                     |
                                                      |                     |--0.03%--vrbd_null_do_req
                                                      |                     |           |
                                                      |                     |           |--0.02%--vrbd_req_done
                                                      |                     |           |          __GI___ioctl
                                                      |                     |           |          system_call
                                                      |                     |           |          sys_ioctl
                                                      |                     |           |          do_vfs_ioctl
                                                      |                     |           |          0x5f8
                                                      |                     |           |          0x429b
                                                      |                     |           |          0xeb7
```

Cache false sharing debug

- Using perf c2c
- Focus on HITM
  HITM - Hit In The Modified
- Perf Boost 17.96%

Performance Isolation - On Demand Paging

- Problem: Overheads from kernel paging mechanism
- Analysis: Kernel counters and dynamic tracing
- Solutions: Avoid running into slow path
  - Warm up and lock the pages
  - Increase min_free_kbytes vs. memory utilization
  - Google patch: per-memcg kswapd (incomplete & need more enhancements)



Memory Performance Isolation Challenges

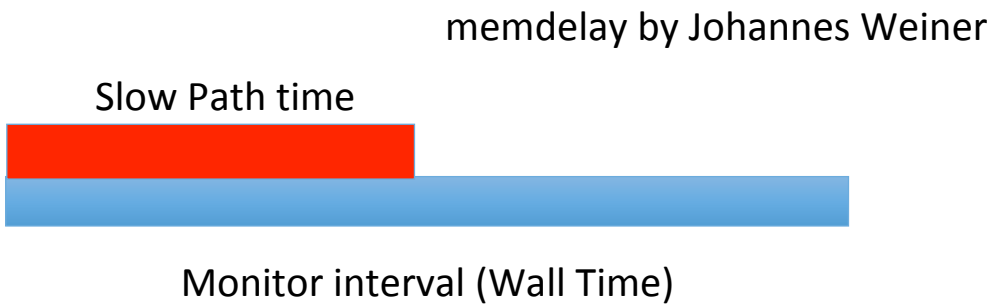
- Constantly varying load breaks static configurations
  - Offline capacity planning is not accurate
  - Static RDT (CAT/MBA) configurations can be broken
  - Static cgroup upper limit, low limit, reclaim watermark can be broken
- Too many slow path corner cases under low free memory conditions
  - Memcg direct reclaim jitters under heavy buffer IO
  - Memory fragmentation and compaction jitters for long uptime system
  - THP (Transparent Huge Page) jitters under latency sensitive use cases
- Online diagnosis, many jitters can only be found on real environment
  - Couldn't be reproduced in the local synthetic test environment
  - Need continuous QoS data collecting and monitoring
  - Need online full stack tracing
- Lack of cluster range QoS data statistics
  - Counter based metrics is not meaningful, and couldn't be used for cluster statistic
  - Fine granularity statistics, for example, cgroup, task levels

Possible Solutions

- Dynamic resource allocator for constantly varying load
  - Real time resource conflicts detection
  - Dynamic resource allocation for resilience handling
- Avoid long-tail performance issue under low free memory conditions
  - Smarter capacity planning, not only offline, but also online
  - Smarter resource resource overcommit algorithm
- Online & offline data collecting, diagnosis and prediction
  - QoS data analysis in cluster range
  - Jitter data collecting and diagnosis
  - Semi-real time analysis and prediction

Cluster QoS Data Normalization

- Counter based metrics are not good for latency analysis
  - TLB miss, Cache miss, Page fault, Alloc stall, Compact stall
- Use time based metrics for cluster data normalization
  - memdelay
    - Translated delays from counters into a pressure percentage of lost wall time



  - PSI - next version of memdelay
    - Expand the ideas to cover pressure stall information for CPU, memory, and IO

Example 1: TLB Miss Data Normalization

- Time based TLB miss analysis

```
$sudo ./tlbmiss -p 54929

Please note the microarchitecture: SandyBridge
The CPU MHz: 2300.000

Tracing PID 54929... Ctrl-C to end.
^C
 Performance counter stats for process id '54929':

           995191      dtlb_load_misses_walk_duration                 (73.68%)
            25553      dtlb_load_misses_walk_completed                (66.72%)
            94356      dtlb_store_misses_walk_duration                (77.75%)
             1997      dtlb_store_misses_walk_completed
            69009      itlb_misses_walk_duration                (56.35%)
              978       itlb_misses_walk_completed                (63.19%)

       7.106366722 seconds time elapsed

Penalty for single dtlb miss :  39.548 cycles  =  17.1948 ns
Penalty for single itlb miss :  70.5613 cycles  =  30.6788 ns
Total miss times dtlb + itlb  :  27550 + 978 = 28528
```

```
Total miss penalties in [7.106366722] seconds:
             dtlb         itld         sum
cycles       1089547      69009        1158556
us           473.716      30.0039      503.72
%            6.66608e-05  4.22212e-06  7.08829e-05
```

- 7% TLB miss time in past 7 seconds

Example 2: Direct Reclaim Data Normalization

- Direct reclaim
  - The allocstall counter in /proc/vmstat
  - Direct reclaim tracepoints

    vmscan:mm_vmscan_direct_reclaim_begin
    vmscan:mm_vmscan_direct_reclaim_end

```
Tracing  Ctrl-C to end.
COMM         PID    START        END              LATms          PAGES
<...>        80936  430042.299345 430042.117869 181.476000   reclaimed=46
<...>        80936  430042.504492 430042.302702 201.790000   reclaimed=57
<...>        80936  430042.707385 430042.504854 202.531000   reclaimed=32
<...>        80936  430042.909637 430042.707611 202.026000   reclaimed=33
<...>        80936  430043.116765 430042.909869 206.905000   reclaimed=32
<...>        80936  430043.321943 430043.117001 204.942000   reclaimed=32
<...>        80936  430043.526633 430043.322179 204.463000   reclaimed=32
<...>        80936  430043.887544 430043.526856 360.689000   reclaimed=29
<...>        80936  430044.679097 430043.887779 791.318000   reclaimed=42
<...>        80936  430044.888294 430044.679386 208.908000   reclaimed=32
<...>        80936  430045.097831 430044.888516 209.315000   reclaimed=32
<...>        80936  430045.334888 430045.098049 236.848000   reclaimed=32
<...>        80905  430045.575300 430045.264685 310.615000   reclaimed=32
<...>        80936  430045.709890 430045.335127 374.763000   reclaimed=32
<...>        80936  430045.978994 430045.710307 268.687000   reclaimed=32
<...>        80905  430046.233338 430045.825409 407.929000   reclaimed=60
<...>        80936  430052.329849 430051.949949 379.900000   reclaimed=32
<...>        80936  430052.882990 430052.330085 552.905000   reclaimed=69
<...>        80905  430053.528546 430051.986333 1542.213000  reclaimed=16671
<...>        80937  430058.936672 430058.935596 1.076000     reclaimed=50
<...>        80937  430059.175154 430058.937047 238.107000   reclaimed=102
<...>        80936  430067.051730 430066.722014 329.716000   reclaimed=32
<...>        80936  430067.386617 430067.051953 334.664000   reclaimed=32
<...>        80936  430067.849535 430067.386836 462.699000   reclaimed=0
```

延时区域	回收次数	百分比	可用性
0-50us	.00	0%	%
50-100us	.00	%	%
100-200us	.00	%	%
200-500us	.00	%	%
500-1000us	.00	%	%
1-5ms	.00	2%	%
5-10ms	.00	5%	%
10-100ms	.00	8%	%
100ms-bigger	.00	%	0%

Direct reclaim latency histogram for a cluster

- Latency range
- Cumulative statistical data

3. Fault Isolation

Memory Faults & QoS Impacts

- Software Fault
  - OOM Handling
  - Split-lock usage
  - Segfault from application
- Hardware Fault
  - CE - Correctable Error
  - UE - Uncorrectable Error
- QoS impacts
  - OS jitters caused by errors handling
  - Memory access latency

Fault Isolation Challenges

- The significant interferences from faulty Cgroup
  - Fault handling code
    - Contention from global locks
    - Priority Inversion
    - TLB flush interrupts
    - IO pressure caused by core dump
  - Split-lock usage from buggy application
  - Error interrupts triggered by hardware errors
    - CMCI, NMI, SMI interrupts
- OOM kill is NOT reliable, and has long latency
  - Resource starving under cgroup OOM handling
- Lack of cluster online/offline faults statistics
- Difficult to predict memory faults

Fault Handling Improvements

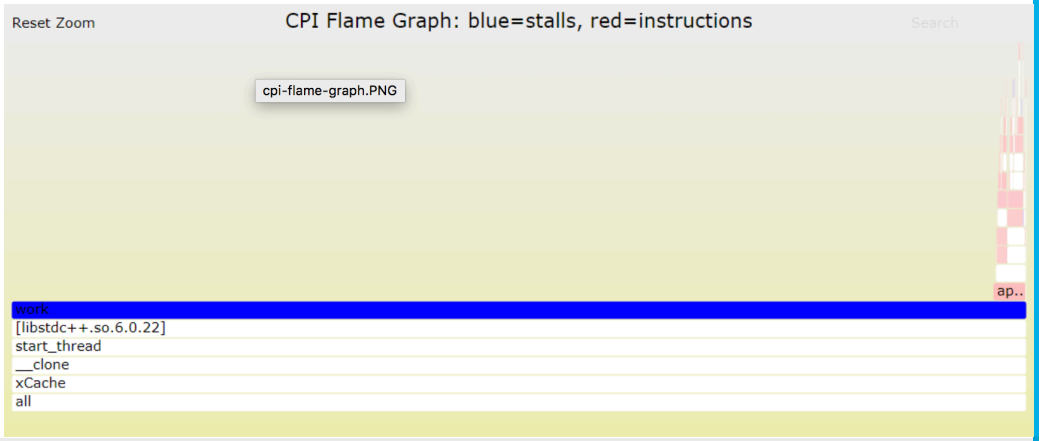
- Cgroup OOM improvements
  - Predictable OOM kill: OOM reaper
  - Reduced overheads: OOM kill fixes
  - Efficient OOM kill: cgroup aware OOM kill
- Fill online/offline memory faults analysis gaps
  - OOM counters: host and cgroup
  - Split-locks monitoring
  - Hardware faults monitoring
  - Interrupts monitoring
  - Memory faults VS. memory utilization, analysis together

OOM Interferences - A Design Issue

- A memcg design problem
  - Memory pages are fully shared among Cgroups
  - Global locking and data structure couldn’t be avoided
  - Always could have jitters problems while cgroup OOM handling
- Solutions
  - Using RunV for better fault isolation.
  - For RunC usage, the best way is to avoid OOM
    - Accurate memory working set estimation
    - A better memory starving prediction algorithm for memory overcommit

Split Lock Isolation - HW & SW solution

- Software bugs
  - Atomic operation cross two cache lines
  - Overall system performance got impacts by faulty application
- Solutions
  - Fix software bugs
  - Detecting split-locks and control split locks
    - #AC exception for new Intel processors
    - PMU data analysis
  - Fault isolation
    - Some error handlings in #AC handlers
    - Other ways…



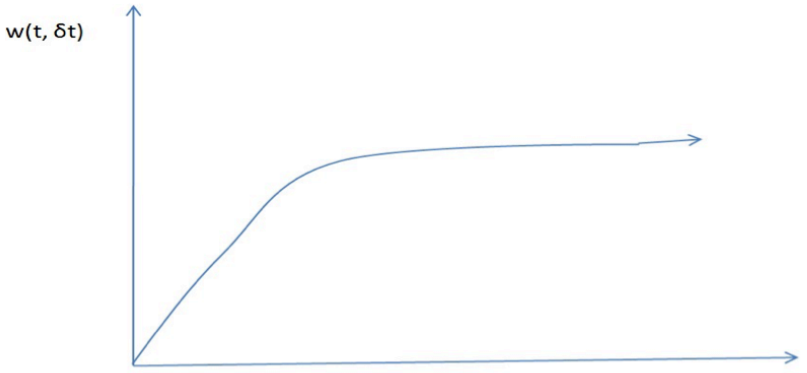
4. Utilization Increase

Memory Utilization Increase Challenges

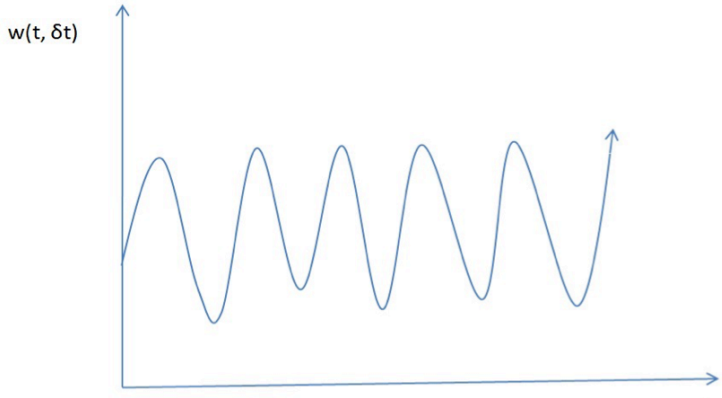
- Memory utilization could be wrong, need to use memory working set

Formula	Comments
(Total - Free) / Total	Not good. There are lots of file read/write once cases.
(Total - Free − Inactive Cache) / Total	Not good. Cache could be still actively used.
(Memory Working Set) / Total	Sounds good. But how do you get the Working Set?

- Rethinking memory working set



Paper: The Working Set Model for Program Behavior – ACM 1968
Working Sets Past and Present − IEEE 1980



Co-location: Memory reclaim always works.

Memory Working Set Estimation

- Use LRU inactive size
  - Kubernetes: WS = Usage – Inactive Page Cache

```
519             workingSet := ret.Memory.Usage
520             if v, ok := s.MemoryStats.Stats["total_inactive_file"]; ok {
521                     if workingSet < v {
522                             workingSet = 0
523                     } else {
524                             workingSet -= v
525                     }
526             }
527             ret.Memory.WorkingSet = workingSet
528     }
```

Source code：https://github.com/google/cadvisor/blob/master/container/libcontainer/handler.go#L519

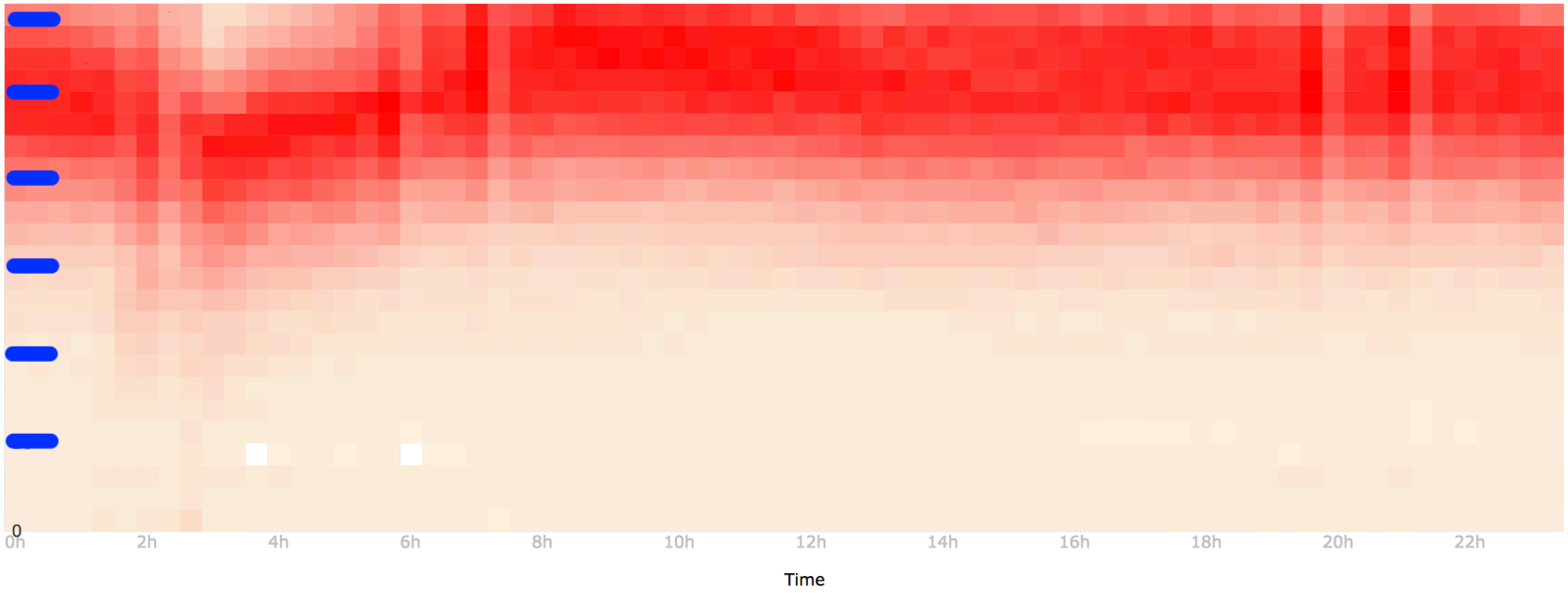
- Use Linux idle page tracking
  - Kubernetes RFE: https://github.com/kubernetes/kubernetes/issues/12422

Linux Idle Page Tracking

- Use Case
  - Documentation/admin-guide/mm/idle_page_tracking.rst
  - Estimate workload's working set size, so that…
    - Setting memory cgroup upper limits for resource control
    - Setting memory cgroup low limits for best efforts working set protection.
    - Deciding where to place the workload within a compute cluster.
    - Determine the effective memory utilization
- Interfaces
  - New: /sys/kernel/mm/page_idle/bitmap
  - Cooperate existing kernel information
    - Per-process: /proc/PID/pagemap
    - Per-cgroup: /proc/kpagecgroup
    - System wide: (0..MAX_PFN]
    - Identify special pages: /proc/kpageflags

Redefine Memory Utilization

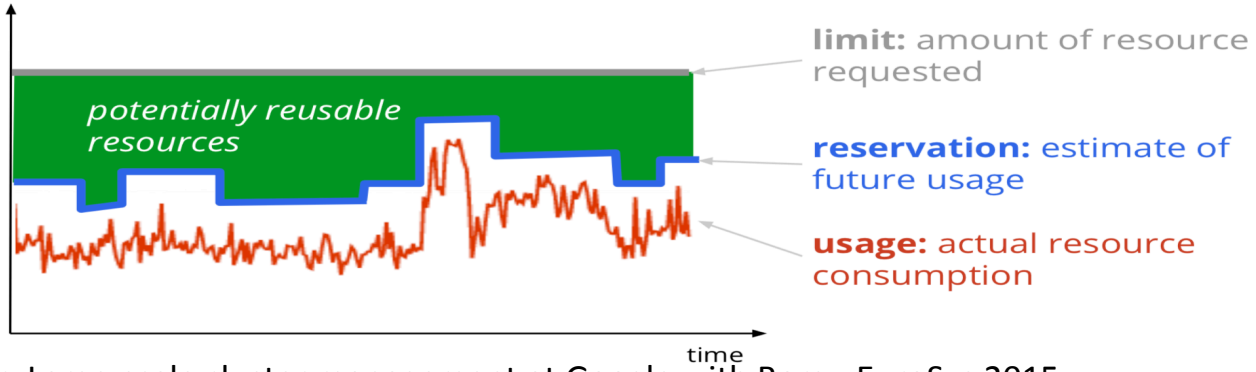
- Effective Memory Utilization
  - No Swap: (Total - Free - Idle Page Cache) / Total
  - With swap: (Total - Free - Idle Memory) / Total
- Online capacity planning by effective memory utilization
  - Online/offline data analysis from any clusters (thousand of machines)
  - Understand spatial and time imbalance of memory resource



Memory Reclamation & Challenges

- Key idea:
  - Find reusable memory for Best-Effort workload
  - Kill Best Efforts workload proactively while memory contentions
- Challenges: Memory utilization & QoS tradeoffs
  - Accurate working-set estimation for continuously-varying load
    - To quickly find reusable memory resource
    - To predict & detect memory starving accurately
- Idle page tracking gives accurate working-set estimation

Efficiency Resource reclamation



Paper: Large-scale cluster management at Google with Borg - EuroSys 2015

5. Summary

Recap

- Memory performance isolation
  - Using PMU counters for hardware resource contention detection
  - Low free memory causes lots of OS Jitters, need a smarter way
  - Using online cluster statistics for utilization and QoS tradeoff
- Memory faults isolation
  - Using RunV solution to meet higher QoS requirements
  - For RunC, try to avoid/predict faults, such as OOM, split-lock, to overcome jitters
- Memory utilization increase
  - Capacity planning can be more accurate by using idle page tracking
  - Continuously varying load requires online capacity planning
  - Memory reclamation solution relies on accurate working-set estimation

References

- Linux Idle Page Tracking
- Intel Top-down Microarchitecture Analysis Method
- Brendan Gregg Blogs
  - CPU utilization is wrong
  - CPI flame graph
  - How To Measure the Working Set Size on Linux
- Resctrl UI
- Perf C2C

Alibaba Cloud

A TEAM OF EXPERTS
6000+ Top Technology Experts Based in Hangzhou, Beijing, Silicon Valley and Seattle.

MISSION
为了无法计算的价值
More Than Just Cloud

VISION
成为全球云数据大计算的领导者
Pioneer of Cloud Data and Big Computing

More Than Just Cloud

Alibaba Cloud
Official Cloud Services Partner


Alibaba Cloud
MORE THAN JUST CLOUD