Software Performance Analysis – Industry Perspectives

Kingsum Chow, Xinyu Jiang, Chengdong Li, Anil Rajput

Who are we?

Kingsum Chow, Professor / School of Software Technology, Zhejiang University

Xinyu Jiang, Postgraduate Student / School of Software Technology, Zhejiang University

Chengdong Li, Founder & CEO / Optimatist

Anil Rajput, AMD Fellow / Datacenter Ecosystems and Application Engineering, also Chair, Java Committee, SPEC

Agenda

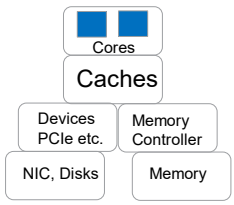
- Part 1: Performance perspective with focus on production deployments
- Part 2: Performance analysis in the industry, methodology and case studies

Part 1: Performance perspective with focus on production deployments

Part-1

- Life in 2010 vs. Now
- Performance Monitoring Universe
- Production Deployments Asks
- Detour: essential STEPs of Performance Analysis
  - Profiling
  - Architecture features intertwined with Analysis
- Large Scale Deployments
  - App Telemetry
  - Tools, data collection, methodology and Analysis
- Challenges

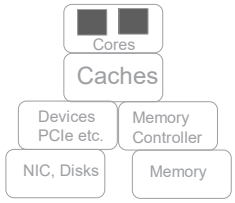
Life used to be simple…



- Single or Dual Cores
- Fixed Frequency
- Uniform Memory
- Simple I/O
- Baremetal OS

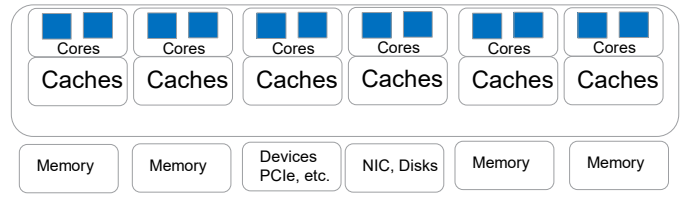
….and performance analysis ☺

Life used to be simple… Now…



- Dual Cores
- Fixed Frequency
- Uniform Memory
- Simple I/O
- Baremetal OS

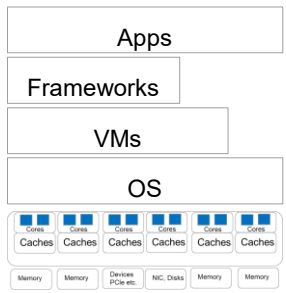
….and performance analysis ☺



- vCPUs > 512
- Frequency/Boost
- Per core power control
- NUMA
- Multi-nodes to Microservices
- Cloud VMs, Hypervisors, Containers
- System and Application telemetry
- Open-source and proprietary tools
- AI based scheduling and problem analysis

Optimatist AMD

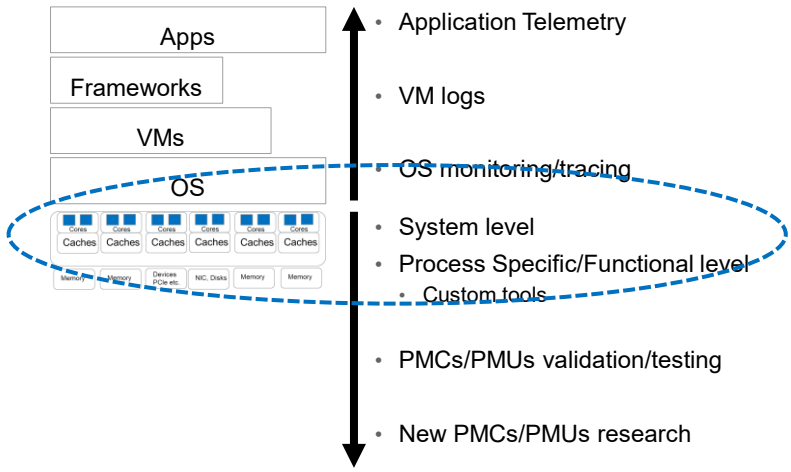
Performance monitoring universe…



- Application Telemetry
- VM logs
- OS monitoring/tracing
- System level
- Process Specific/Functional level
  - Custom tools
- PMCs/PMUs validation/testing
- New PMCs/PMUs research

Optimatist AMD

Performance monitoring universe … this tutorial focus…



Latest production deployment focus

What typical production deployments requesting…

- Few important counters … all the time at right granularity
- On demand or based on some alerts … detailed counters capture

➔All above … using open-source tools (Linux Perf)

➔Automation to correlate … above with application telemetry!

➔Should lead to … better configurations, optimizations and TCO

What typical production deployments requesting…

- Few important counters … all the time at right granularity
- On demand or based on some alerts … detailed counters capture

➔All above … using open-source tools (Linux Perf)
➔Automation to correlate … above with application telemetry!
➔Should lead to… better configurations, optimizations and TCO

➔In 1-2 years… generate enough training data set for LLMs ☺
Short term: Apply RAG, create Agents
Long Term: AIPerfEngineer!

What typical production deployments requesting…

- Yes, we have a proposed methodology which is not perfect, but pretty good!

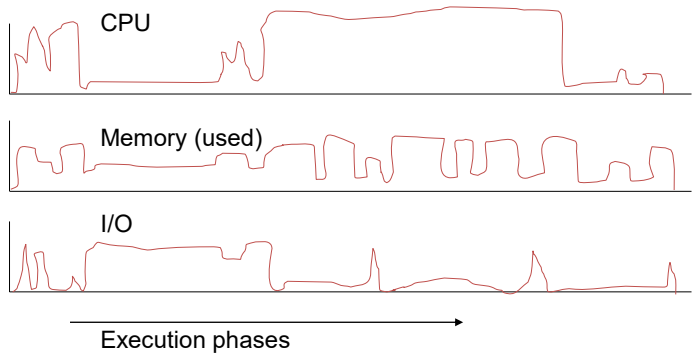
- It starts with basic fundamentals…

Before executing the application … ensure…

- Platform Configuration
  - Log the config details using many tools
  - Develop automatic compare and alerts
    - CPU models, Memory DIMMs, Disks, etc.
- System Health check
  - Collect quick basic performance evaluation
    - Memory bandwidth and latency
    - I/O perf, etc.

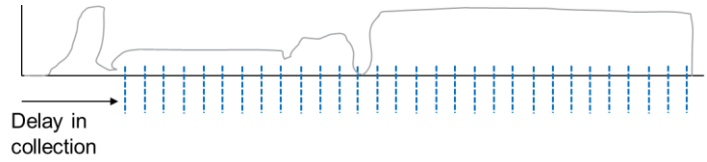
STEP 1 of any performance analysis … understand…

- The high-level characteristics of the application

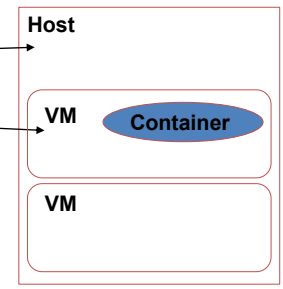


How to profile an application…

- Many tools available
  - Basic vmstat
    - Collect and view in histogram



  - Check if running at
    - Host level or
    - VM level



How to profile an application … vmstat

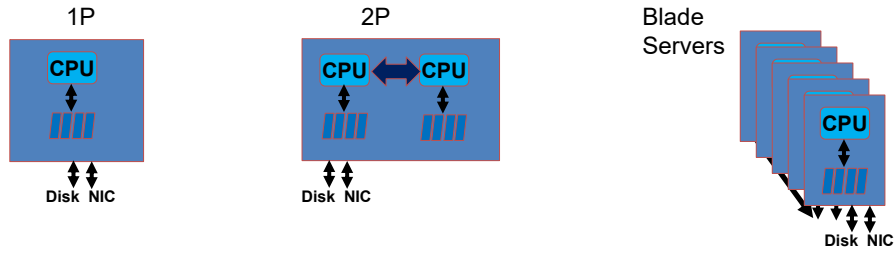
- vmstat https://access.redhat.com/solutions/1160343

- Waiting threads
- Memory use
- I/O
- Interrupts
- Context Switch
- CPU %: user, system, I/O wait, and idle

```
[user@fedora9 ~]$ vmstat 1 5
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 3  0      0  44712 110052 623096    0    0    30    28  217  888 13  3 83  1  0
 0  0      0  44408 110052 623096    0    0     0     0   88 1446 31  4 65  0  0
 0  0      0  44524 110052 623096    0    0     0     0   84  872 11  2 87  0  0
 0  0      0  44516 110052 623096    0    0     0     0  149 1429 18  5 77  0  0
 0  0      0  44524 110052 623096    0    0     0     0   60  431 14  1 85  0  0
[user@fedora9 ~]$
```

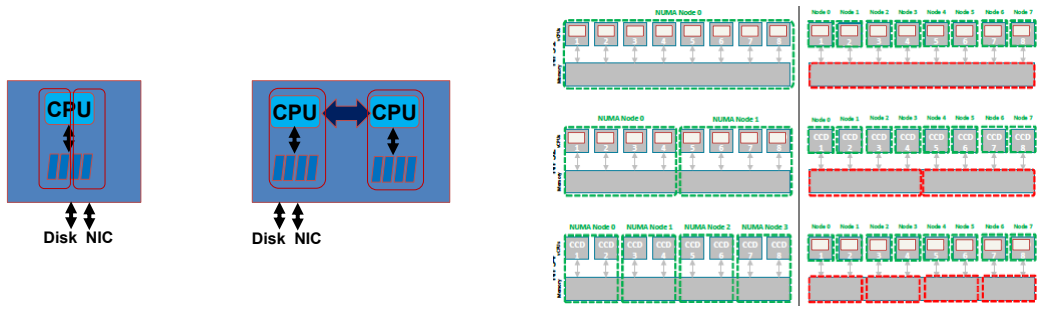
STEP 2 … understand the platform and its features

- 1P, 2P or more…



Understand the platform and its features: NUMA

- NUMA & Memory interleaving



→Impacts memory bandwidth and latency

Understand the platform and its features: SMT / HT

- AMD® EPYC™ Simultaneous Multithreading (SMT)
- Intel® Hyper-Threading Technology (HT)

- IBM® Power9™ Simultaneous Multithreading (SMT)
  - More replication of resources

→Significantly impacts *CPU utilization %* and *Performance analysis*!

CPU utilization % examples … next level detail…

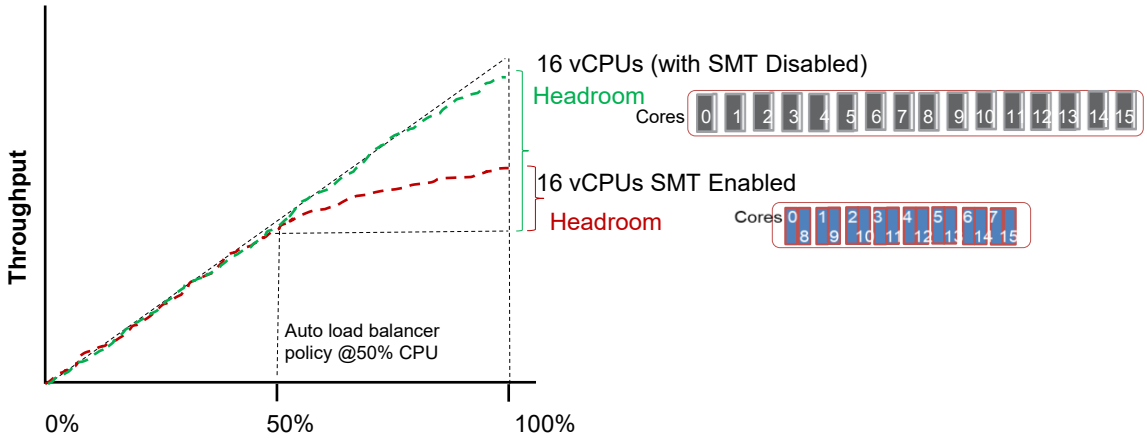
System	Total CPU %	User vs. Sys
A	40%	35% User 5% Sys
B	40%	25% User 15% Sys

System	Total CPU %	User vs. Sys	Effective Frequency
A	40%	35% User 5% Sys	3.50 GHz
B	40%	35% User 5% Sys	2.8 GHz

System	Total CPU %	User vs. Sys	Effective Frequency	Scheduling
A	40%	35% User 5% Sys	3.50 GHz	Cores + SMT threads 
B	40%	35% User 5% Sys	3.50 GHz	Across cores 

→Just knowing CPU % utilization may not be sufficient!

CPU utilization % not linear when SMT is enabled…



→In production, it is critical for auto load balancer policies!

Performance analysis with SMT / HT…

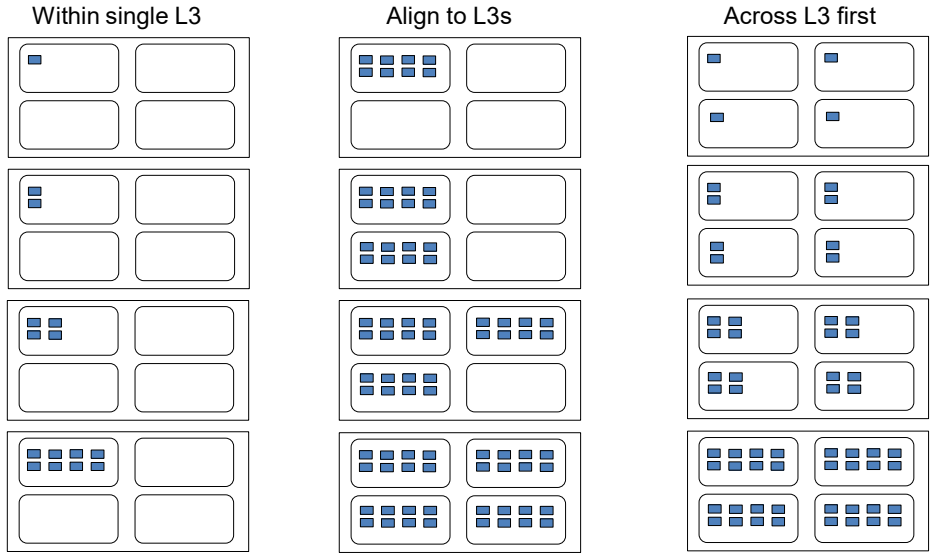
- Fundamentals of Performance Analysis

  Example:

	SMT OFF	SMT ON	
Total Instruction retired / sec	96 x 10^9	110 x 10^9	
Total CLKs / sec	48 x 10^9	96 x 10^9	SMT OFF 16C/16T vCPUs x 3.0 GHz SMT ON 16C/32T vCPUs x 3.0 GHz
Total Ops / sec	1000	1200	20% SMT Uplift
IPC (Instructions / CLK)	2.00	1.15	
CPI (CLKs / Instruction)	0.50	0.87	

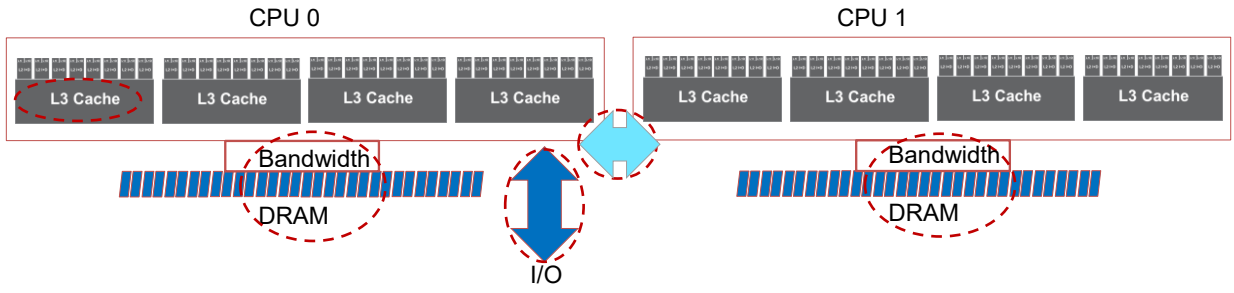
→Special attention when mixing IPC from SMT ON and OFF configs!

STEP 3 Understand the scaling of the deployed application…

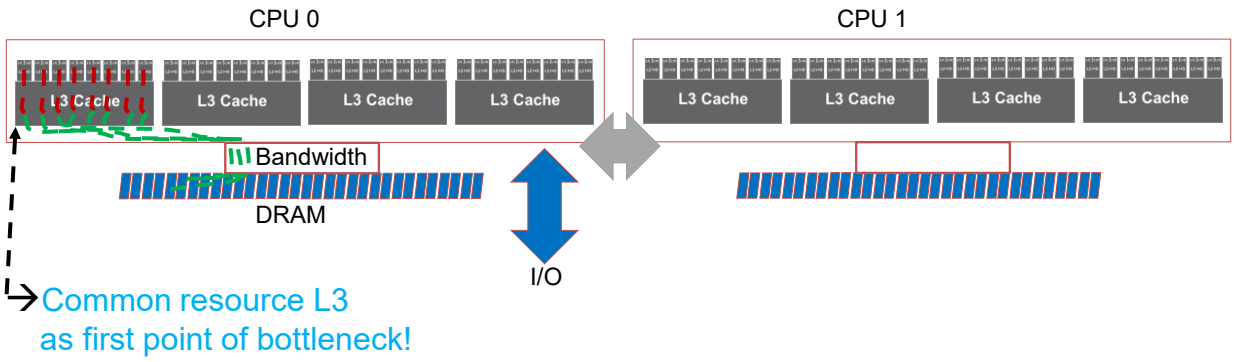


→Hypervisor may schedule VMs using different policies!

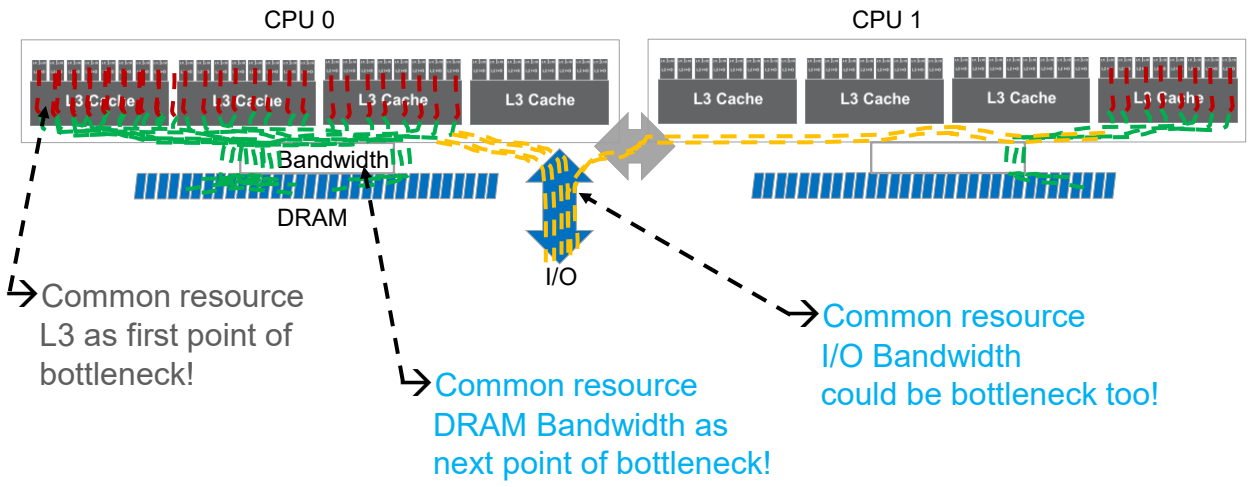
Understand the scaling bottlenecks…



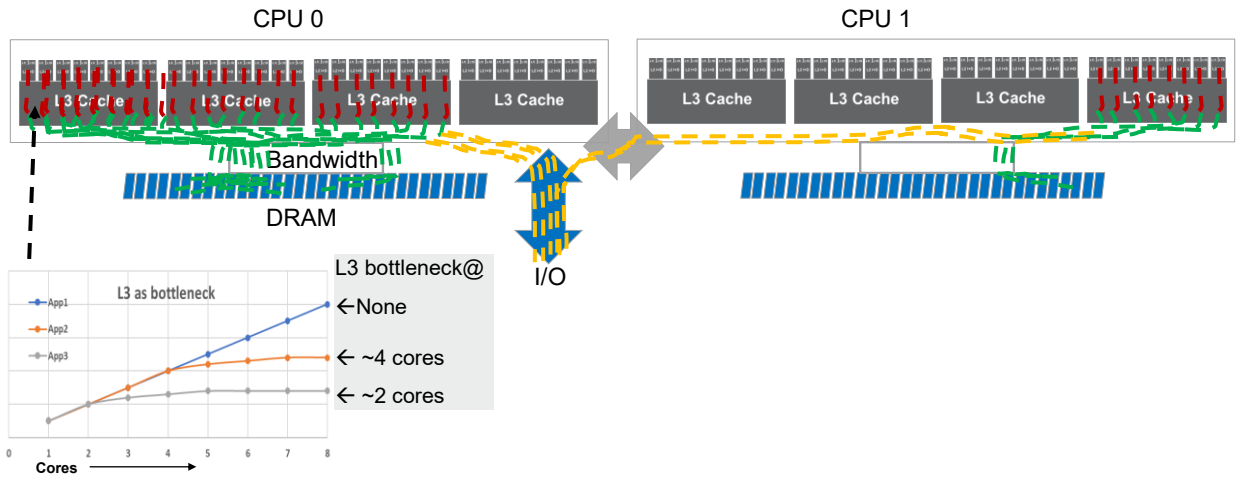
Understand the scaling of the deployed application…



STEP 3 Understand the scaling of the deployed application…



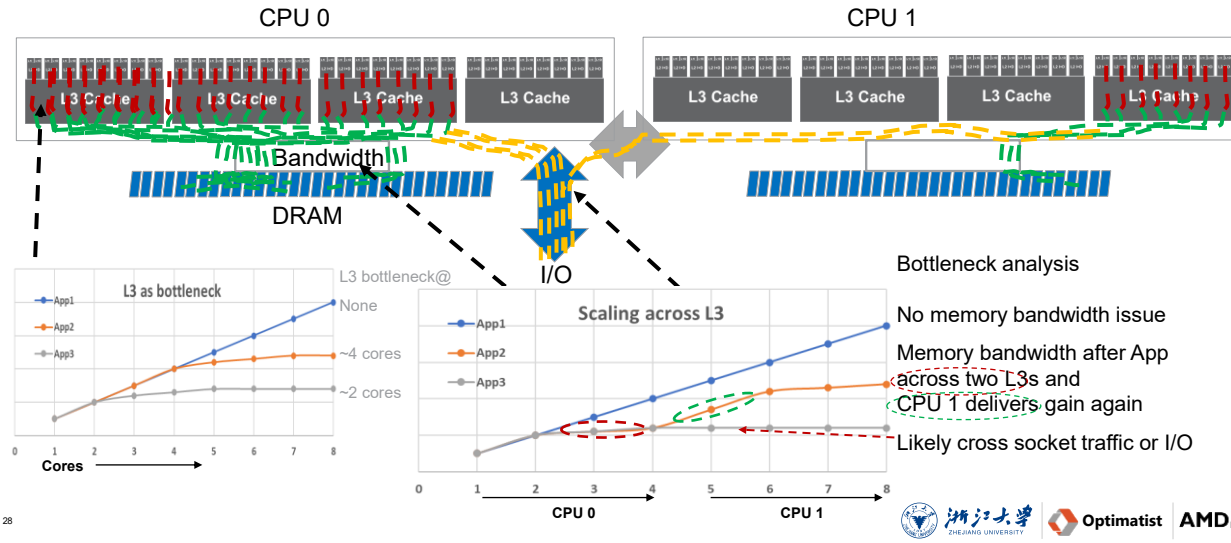
STEP 3 Understand the scaling of the deployed application…



27

Optimatist AMD

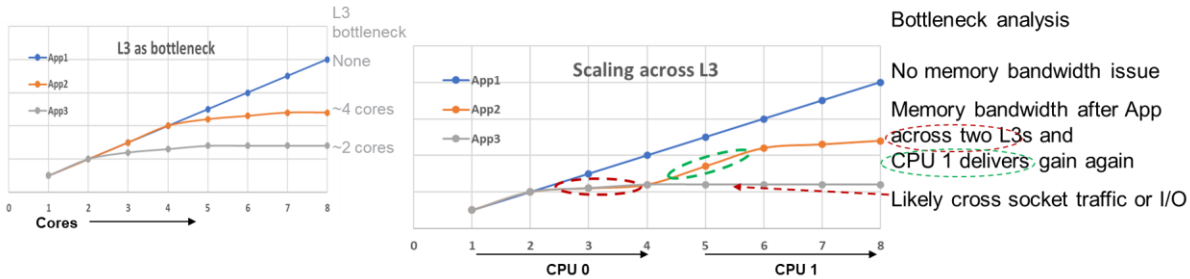
STEP 3 Understand the scaling of the deployed application…



28

Optimatist AMD

STEP 4 How to validate these observations…

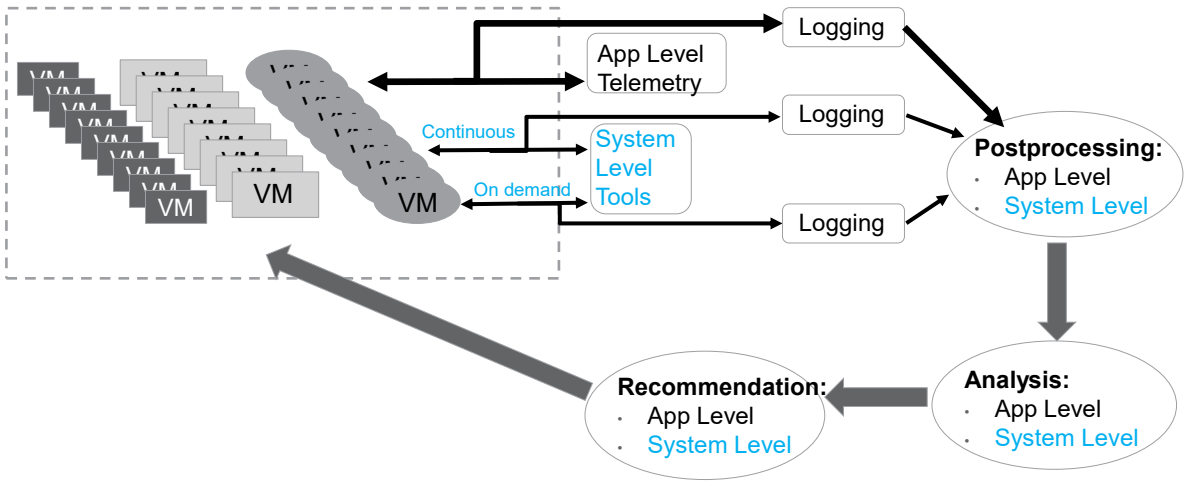


→Tools, data collection, post processing and analysis

Typical production deployments … asks…

- In-depth optimization
  - Profiling tools with architecture specific analysis
- Optimal sizing of VMs
  - High level characterization of the application + TCO analysis
- Large scale production deployments telemetry and analysis…

Large scale deployments … proposed architecture…



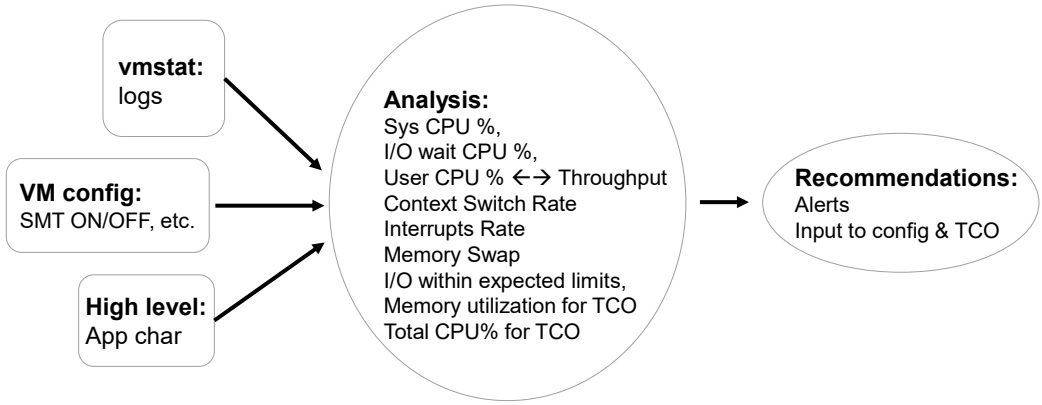
→Collaboration areas between research and industry…

Continuous data collection…

- vmstat
- Effective Frequency
- Etc.

Analysis … vmstat or similar…

- Mostly rule based analysis



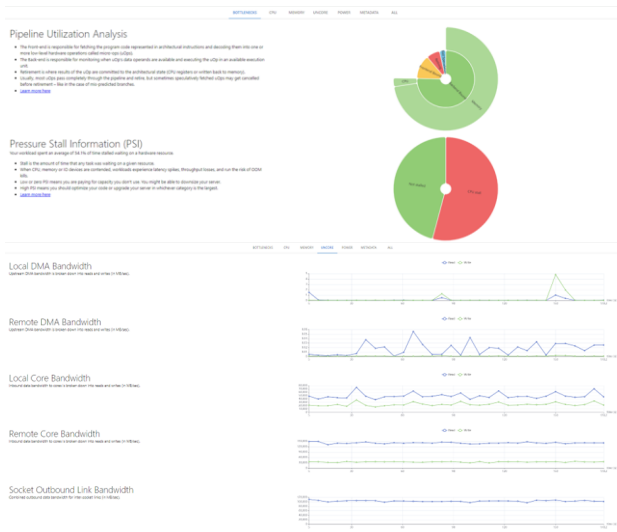
Effective Frequency…

- Due to core level power management and total power capping, measuring effective frequency of the cores associated to the VM is very important to determine performance!

On-demand data collection … system level tools…

- PerfSpect
- ProcessWatch

PerfSpect…



- `pip3 install -r requirements.txt`
- `make`
- `cd build && ./perf-collect -t 60` (sets duration of 60s)
- `./perf-collect --socket -t 60` (sets duration of 60s and collects Socket level information)
- `./perf-collect --cpu -a $PWD/sample.sh` (option `--cpu` to collect per cpu data and disables uncore events, `-a` for application to run with perf-collect and ends after workload completion)
- `./perf-collect -m 80 --cpu -t 60` (option `-m` is the mux interval)
- `./perf-collect -p 5041 -t 100` (option `-p` to collect data for a given PID process)
- `./perf-postprocess` (generates reports in readable csv format and HTML file to view the graphical plots)

ProcessWatch…

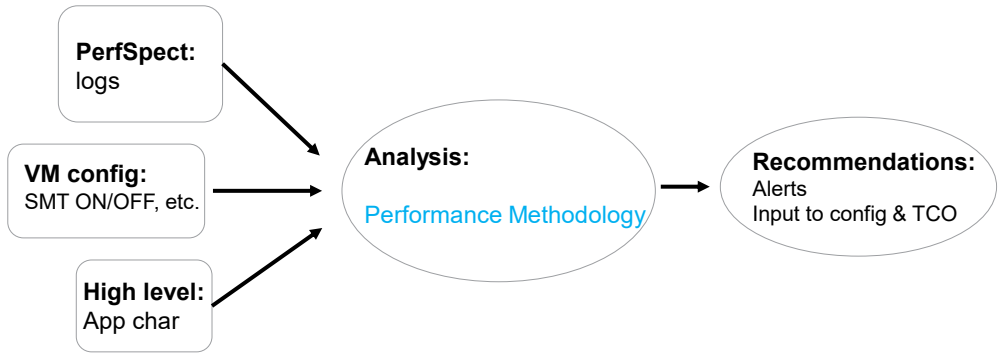
- Showcase

Analysis…

- Mostly rule based analysis

Analysis … using architecture specific counters…

- Mostly rule based but following … performance methodology…



Performance methodology … rule + heuristic…

$$\text{Performance} = \frac{\text{Ops}}{\text{CLKs}} = \frac{\text{Ops}}{\text{Instructions}} \times \frac{\text{Instructions}}{\text{CLKs}}$$

$$= \frac{\text{Ops / sec}}{\text{Instructions / sec}} \times \frac{\text{Instructions / sec}}{\text{CLKs / sec}}$$

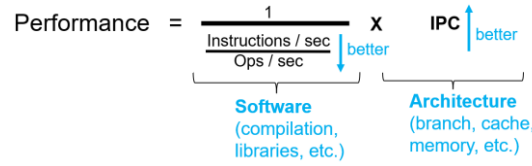
$$\text{Performance} = \frac{1}{\frac{\text{Instructions / sec}}{\text{Ops / sec}}} \times \text{IPC}$$

(better) (better)

Software (compilation, libraries, etc.)

Architecture (branch, cache, memory, etc.)

PerfSpect logging data…



- Ops per sec (App Telemetry)
- Instruction / sec
- CLKs / sec

- Calculate IPC (Instructions, CLKs)
  - Compare Architecture Counters
    - Next in-depth: Branches, caches, memory bandwidth and latency

- Calculate Instructions / Ops (Instructions, Ops / sec)
  - Compare for SW stack changes

Challenges…

- Many cloud deployments only allow core level PMCs
  - Only core level PMCs available within a VM or container
- Manually it is not possible to analyze data collected at scale
- Performance analysis is rule based + heuristic + changes with deployment and load etc.

→Very important for overall TCO to develop analysis and recommendation using (App telemetry + System tools data)

Part 2: Performance analysis in the industry, methodology and case studies

Workload in the many core era

A workload that has been running fine for a few cores may not do well with dozens or hundreds of cores.

Core Scaling Analysis:

- Run a workload from a few cores to dozens or even hundreds of cores
- If performance doesn't scale well, identify the bottlenecks.

Case study: Bottleneck analysis with many cores

LevelDB

LevelDB is a fast key-value storage library written at Google that provides an ordered mapping from string keys to string values. Authors: Sanjay Ghemawat (sanjay@google.com) and Jeff Dean (jeff@google.com)

RocksDB

A library that provides an embeddable, persistent key-value store for fast storage.

RocksDB is developed and maintained by Facebook Database Engineering Team. It is built on earlier work on LevelDB.

https://github.com/google/leveldb
https://github.com/facebook/rocksdb

45

浙江大学 ZHEJIANG UNIVERSITY | Optimatist | AMD

Case study: Bottleneck analysis with many cores

LevelDB

`db_bench` is used to evaluate the performance of LevelDB and is part of the LevelDB project.

RocksDB

Facebook conducted secondary development based on LevelDB's `db_bench` to evaluate the performance of RocksDB.

They also built `benchmark.sh` on top of it to provide recommended running methods.

https://github.com/google/leveldb
https://github.com/facebook/rocksdb

46

浙江大学 ZHEJIANG UNIVERSITY | Optimatist | AMD

Case study: Bottleneck analysis in multithreaded scenarios

Some of RocksDB's users:

- Apache Spark
- Apache Flink
- Apache Kafka
- Apache Doris
- Apache Kvrocks
- Alluxio
- ByteDance ByteGraph
- Microsoft Bing search engine
- Netflix
- Uber
- Airbnb
- Tencent PaxosStore (for WeChat)
- Yahoo
- LinkedIn
- PingCAP, TiDB, TiKV
- Snowflake

https://github.com/facebook/rocksdb/blob/main/USERS.md

47

Case study: Bottleneck analysis in multithreaded scenarios

https://github.com/facebook/rocksdb/wiki/Performance-Benchmarks

Test 2. Random Read (benchmark.sh readrandom)

NUM_KEYS=900000000 CACHE_SIZE=6442450944 DURATION=5400 benchmark.sh readrandom

Measure performance to randomly read existing keys. The database after bulkload was used as the starting point.

Version	Opts	ops/sec	mb/sec	usec/op	p50	p75	p99	p99.9	p99.99
7.2.2	None	136915	34.7	467.4	615.5	772.8	1270	1801	2840
7.2.2	DIO	189236	47.9	338.2	419.6	539.1	1022	1693	2297
7.1.1	None	145490	36.8	439.9	599.7	753.7	1252	1809	2813
7.1.1	DIO	189242	47.9	338.2	419.0	539.1	1037	1696	2294
7.0.3	None	145540	36.8	439.7	599.8	753.3	1251	1803	2803
7.0.3	DIO	189243	47.9	338.2	419.2	539.2	1029	1691	2246
6.29.1	None	145577	36.9	439.6	606.3	751.0	1204	1292	2091
6.29.1	DIO	189243	47.9	338.2	430.0	540.9	854	969	1291
6.29.0	None	145590	36.9	439.6	606.2	751.0	1204	1292	1936
6.29.0	DIO	189241	47.9	338.2	430.0	540.8	854	932	1289
6.28.0	None	146980	37.2	435.4	604.3	748.9	1195	1291	1984
6.28.0	DIO	189232	47.9	338.2	430.0	540.9	854	991	1293
6.27.0	None	146921	37.2	435.6	604.4	748.8	1194	1291	1980
6.27.0	DIO	189250	47.9	338.2	430.1	540.8	854	902	1287
6.26.0	None	128341	32.5	498.7	639.6	805.7	1272	1298	2156
6.26.0	DIO	189244	47.9	338.2	430.1	540.8	854	894	1287
6.25.0	None	128517	32.5	498.0	639.0	804.6	1272	1298	2220

`benchmark.sh` and `db_bench` have been used as the benchmark for performance testing in many versions of RocksDB.

48 https://github.com/facebook/rocksdb/wiki/Performance-Benchmarks

Case study: Bottleneck analysis in multithreaded scenarios

Random Read: Measure performance to randomly read existing keys. Uniform Distribution. Mason rotation method.

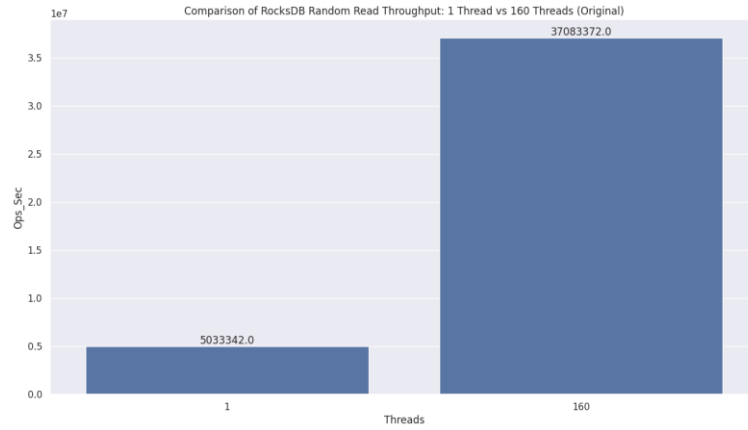
- **RocksDB**: version 9.2.0
- **CPU**: 2 * Intel(R) Xeon(R) Platinum 8383C CPU @ 2.70GHz
  - HyperThreading ON
  - 40 cores per socket
  - 160 hardware threads
- **Memory**: 512 GB
- **OS:** Ubuntu 22.04 5.15.0-102-generic

```
# the parameters of benchmark.sh
export DB_DIR=./db
export WAL_DIR=./wal
export NUM_KEYS=900000000
export CACHE_SIZE=6442450944
export DURATION=300
export NUM_THREADS=1 # only this changed in the following different experiments

./tools/benchmark.sh randomread
```


浙江大学 ZHEJIANG UNIVERSITY | Optimatist | AMD

Case study: Bottleneck analysis in multithreaded scenarios



Threads	Instructions	Transactions	Path_Len	Cycles	CPI	MicroSec_Ops	Ops_Sec	Total_Time(s)	CPU_Util(%)
1	3129398329928	1510002999	2072.445109	1136230674488	0.363083	0.199	5033342.0	300.000	99.710000
160	22625805888251	11125908840	2033.614172	171717420118090	7.589450	4.314	37083372.0	300.024	99.462063

浙江大学 ZHEJIANG UNIVERSITY | Optimatist | AMD

Case study: Bottleneck analysis in multithreaded scenarios

1 thread, data collected by perf-record, parsed by perf-report

```
#        Overhead  Command          Shared Object
Symbol
# ...............  ...............  ..........................................  ..........
#
     6.37%   7.70%  db_bench         db_bench                                    [.] rocksdb::DBImpl::GetImpl
     3.47%   4.19%  db_bench         db_bench                                    [.] rocksdb::Version::Get
     3.26%   3.94%  db_bench         db_bench                                    [.] rocksdb::Stats::FinishedOps
     2.94%   3.56%  db_bench         db_bench                                    [.] rocksdb::Benchmark::ReadRandom
     2.88%   3.48%  db_bench         db_bench                                    [.] rocksdb::GetContext::GetContext
     2.13%   2.58%  db_bench         [vdso]                                      [.] 0x00000000000006e8
     1.93%   2.34%  db_bench         db_bench                                    [.] std::mersenne_twister_engine<unsigned long, 64ul, 312ul, 156ul, 31ul,
1304310990599815831ul, 29ul, 6148914691236517205ul, 17ul, 8202884508482404352ul, 37ul, 18444473444759240704ul, 43ul, 6364136223846793005ul>::_M_gen_rand
     1.69%   2.04%  db_bench         db_bench                                    [.] rocksdb::ThreadLocalPtr::StaticMeta::CompareAndSwap
     1.47%   1.77%  db_bench         db_bench                                    [.] rocksdb::DBImpl::Get
     1.40%   1.35%  node             [kernel.kallsyms]                           [k] link_path_walk.part.0.constprop.0
     1.34%   1.62%  db_bench         db_bench                                    [.] rocksdb::ThreadLocalPtr::StaticMeta::Swap
     1.34%   1.29%  node             [kernel.kallsyms]                           [k] __d_lookup_rcu
     0.96%   0.32%  node             [unknown]                                   [.] 0x000000000f5bd5f
     0.95%   1.15%  db_bench         db_bench                                    [.] rocksdb::MemTable::Get
     0.84%   1.01%  db_bench         db_bench                                    [.] rocksdb::MemTableListVersion::GetFromList
```

```
# cmdline : /usr/lib/linux-tools-5.15.0-100/perf record -e {cycles,instructions}:S
-a -F 97 /usr/bin/taskset -c 140 ./benchmark.sh readrandom
```

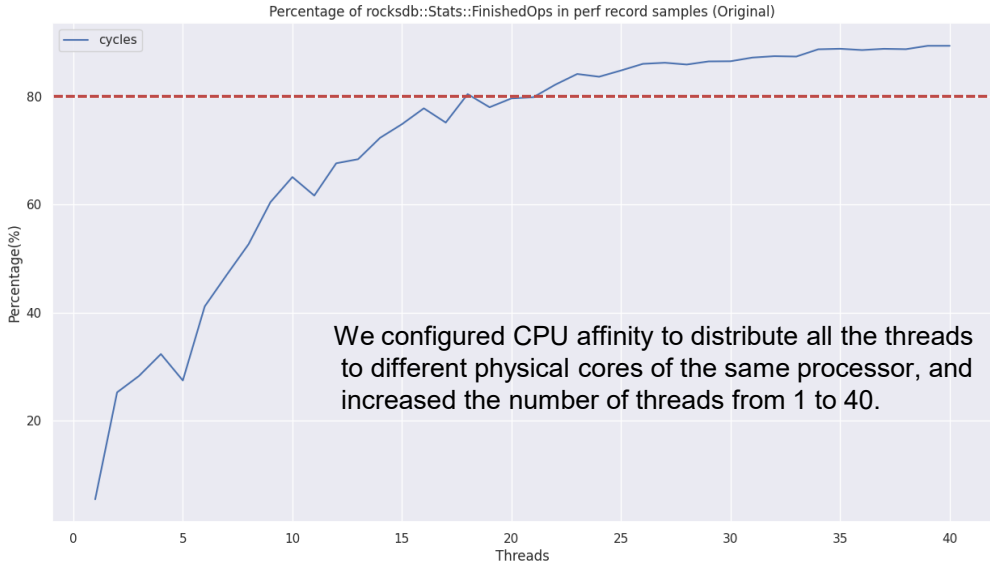

Case study: Bottleneck analysis in multithreaded scenarios

160 threads, data collected by perf-record, parsed by perf-report

```
#        Overhead  Command          Shared Object
Symbol
# ...............  ...............  ..........................................  ..........
#
    87.10%  86.67%  db_bench         db_bench                                    [.] rocksdb::Stats::FinishedOps
     2.06%   2.09%  db_bench         db_bench                                    [.] rocksdb::Version::Get
     1.71%   1.58%  db_bench         db_bench                                    [.] rocksdb::ThreadLocalPtr::StaticMeta::Swap
     1.60%   1.64%  db_bench         db_bench                                    [.] rocksdb::DBImpl::GetImpl
     0.97%   0.99%  db_bench         db_bench                                    [.] rocksdb::Benchmark::ReadRandom
     0.52%   0.53%  db_bench         db_bench                                    [.] rocksdb::GetContext::GetContext
     0.41%   0.41%  db_bench         db_bench                                    [.] rocksdb::DBImpl::Get
     0.34%   0.34%  db_bench         db_bench                                    [.] rocksdb::ThreadLocalPtr::StaticMeta::CompareAndSwap
     0.30%   0.31%  db_bench         db_bench                                    [.] rocksdb::HistogramBucketMapper::IndexForValue
     0.25%   0.25%  db_bench         [vdso]                                      [.] 0x00000000000006e8
     0.23%   0.24%  db_bench         db_bench                                    [.] std::mersenne_twister_engine<unsigned long, 64ul, 312ul, 156ul, 31ul,
1304310990599815831ul, 29ul, 6148914691236517205ul, 17ul, 8202884508482404352ul, 37ul, 18444473444759240704ul, 43ul, 6364136223846793005ul>::_M_gen_rand
     0.21%   0.21%  db_bench         db_bench                                    [.] std::mersenne_twister_engine<unsigned long, 64ul, 312ul, 156ul, 31ul,
1304310990599815831ul, 29ul, 6148914691236517205ul, 17ul, 8202884508482404352ul, 37ul, 18444473444759240704ul, 43ul, 6364136223846793005ul>::operator()
     0.19%   0.19%  db_bench         db_bench                                    [.] rocksdb::MemTableListVersion::GetFromList
     0.19%   0.19%  db_bench         db_bench                                    [.] rocksdb::MemTable::Get
     0.18%   0.18%  db_bench         db_bench                                    [.] rocksdb::LookupKey::LookupKey
     0.17%   0.17%  db_bench         db_bench                                    [.] rocksdb::DBImpl::FailIfCfHasTs
```

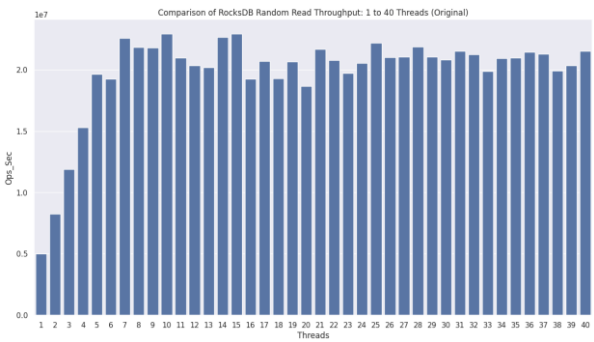

```
# cmdline : /usr/lib/linux-tools-5.15.0-100/perf record -e {cycles,instructions}:S
-a -F 97 ./benchmark.sh readrandom
```

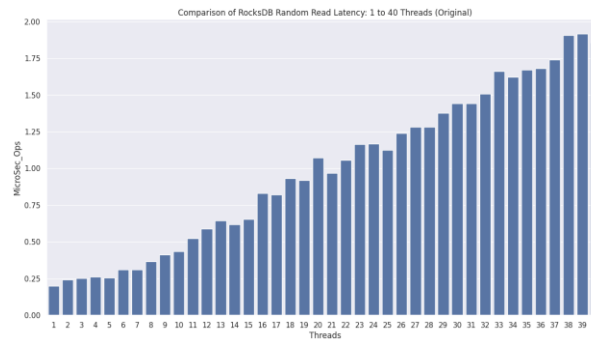
Case study: Bottleneck analysis in multithreaded scenarios



We configured CPU affinity to distribute all the threads to different physical cores of the same processor, and increased the number of threads from 1 to 40.

Case study: Bottleneck analysis in multithreaded scenarios





Throughput

Latency

Different physical cores on the same processor

Case study: Bottleneck analysis in multithreaded scenarios

```
2282  void FinishedOps(DBWithColumnFamilies* db_with_cfh, DB* db, int64_t num_ops,
2283                   enum OperationType op_type = kOthers) {
2284    if (reporter_agent_) {
2285      reporter_agent_->ReportFinishedOps(num_ops);
2286    }
2154  class Stats {
2155  private:
2156    SystemClock* clock_;
2157    int id_;
2158    uint64_t start_ = 0;
2159    uint64_t sine_interval_;
2160    uint64_t finish_;
2161    double seconds_;
2162    uint64_t done_;
2163    uint64_t last_report_done_;
2164    uint64_t next_report_;
2165    uint64_t bytes_;
2166    uint64_t last_op_finish_;
2167    uint64_t last_report_finish_;
2168    std::unordered_map<OperationType, std::shared_ptr<HistogramImpl>,
2169                       std::hash<unsigned char>>
2170        hist_;
2171    std::string message_;
2172    bool exclude_from_merge_;
2173    ReporterAgent* reporter_agent_;  // does not own      Igor Canadi, 9 years ago
2174    friend class CombinedStats;
```

The function `FinishedOps` is called once after each operation is completed.

```
6035  void ReadRandom(ThreadState* thread) {
6058    while (!duration.Done(1)) {      Lei Jin, 10 years ago • db_bench cleanup
6138
6139      thread->stats.FinishedOps(db_with_cfh, db_with_cfh->db, 1, kRead);
6140    }
6141
6142    char msg[100];
6143    snprintf(msg, sizeof(msg), "(%" PRIu64 " of %" PRIu64 " found)\n", found,
6144             read);
6145
6146    thread->stats.AddBytes(bytes);
6147    thread->stats.AddMessage(msg);
6148  }
```

Global Variable with Mutex Lock

Case study: Bottleneck analysis in multithreaded scenarios

Before modification

Threads	Instructions	Transactions	Path_Len	Cycles	CPI	MicroSec_Ops	Ops_Sec	Total_Time(s)	CPU_Util(%)
1	3129398329928	1510002999	2072.445109	1136230674488	0.363083	0.199	5033342.0	300.000	99.710000
160	22625805888251	11125908840	2033.614172	171717420118090	7.589450	4.314	37083372.0	300.024	99.462063

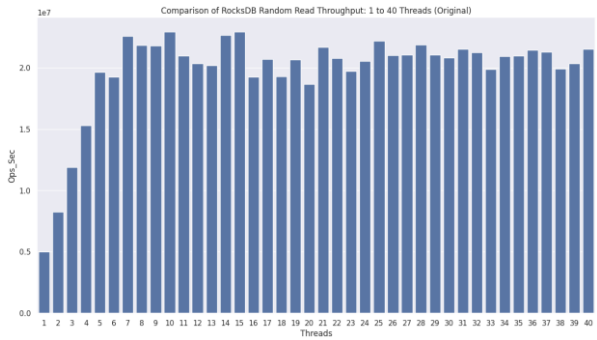
After modification

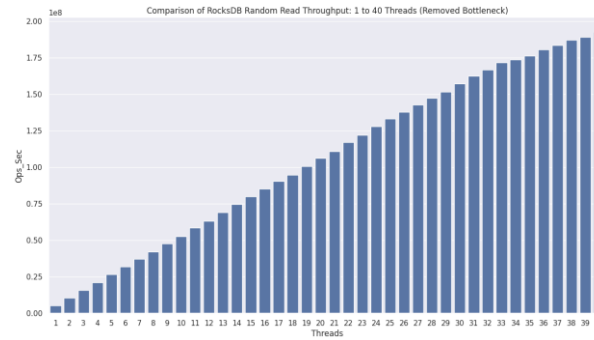
323,795,384 vs 37,083,372

Threads	Instructions	Transactions	Path_Len	Cycles	CPI	MicroSec_Ops	Ops_Sec	Total_Time(s)	CPU_Util(%)
1	3259113189316	1588156999	2052.135394	1124524900793	0.345040	0.189	5293853.0	300.000	99.71000
160	194183272380812	97151942840	1998.758509	151013619154778	0.777686	0.494	323795384.0	300.041	99.55725

浙江大学 ZHEJIANG UNIVERSITY | Optimatist | AMD

Case study: Bottleneck analysis in multithreaded scenarios



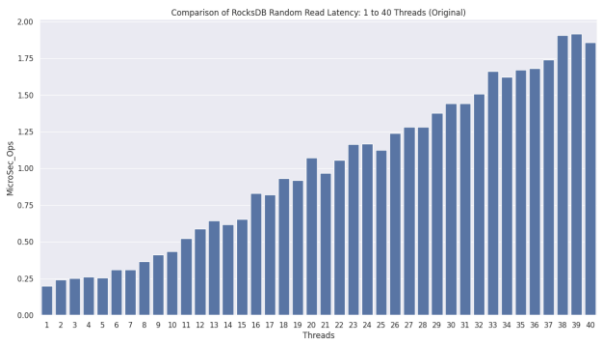


Before modification

After modification

Throughput

Case study: Bottleneck analysis in multithreaded scenarios



Before modification

After modification

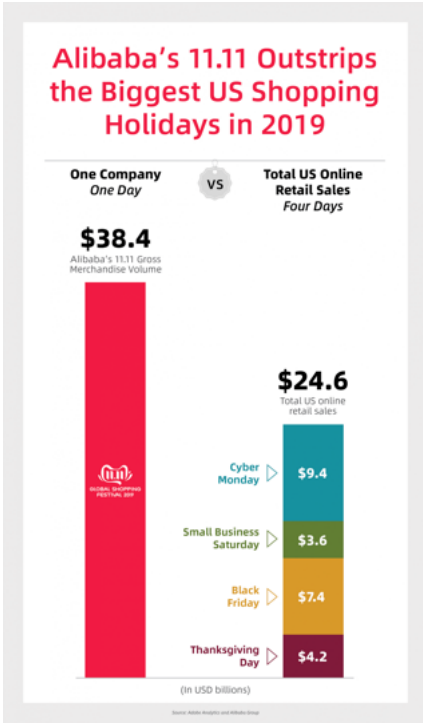
Latency

Analysis shared on GitHub and Google Group

https://github.com/facebook/rocksdb/issues/12594

https://groups.google.com/g/rocksdb/c/ORtpFcXMf8w

Problem agreed last month. Will submit a PR to fix it.

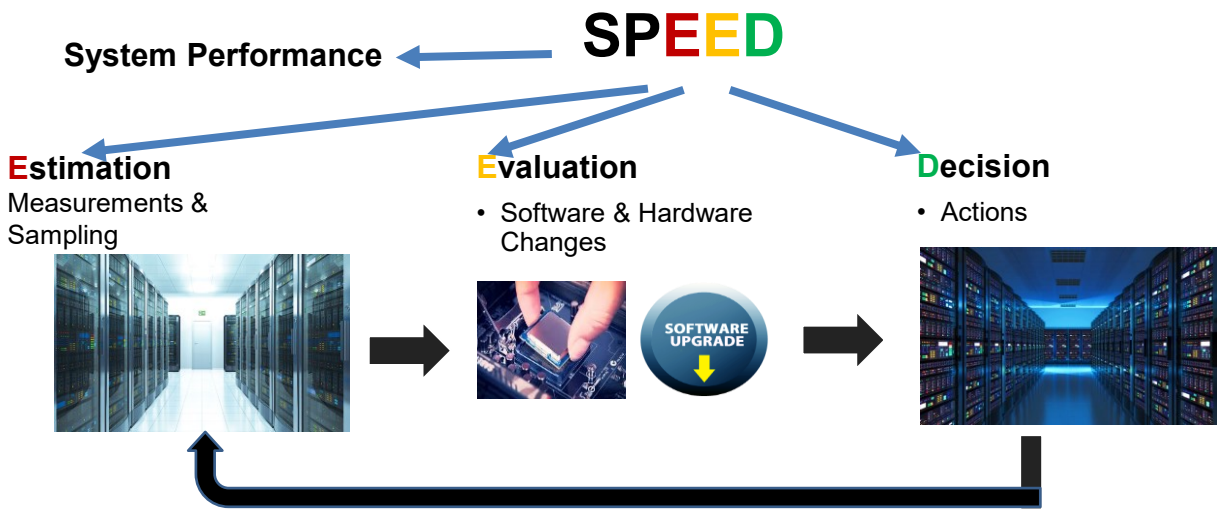


Performance Matters!

https://www.alizila.com/alibabas-11-11-outstrips-biggest-us-shopping-holidays-in-2019

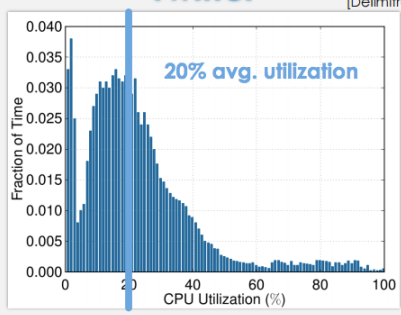
To save 1000 servers

Servers	Needed Performance Improvement (%)	Examples
10,000	10	Workload Characterization Iron Law
100,000	1	Software Configuration at Scale Hardware Configuration at Scale

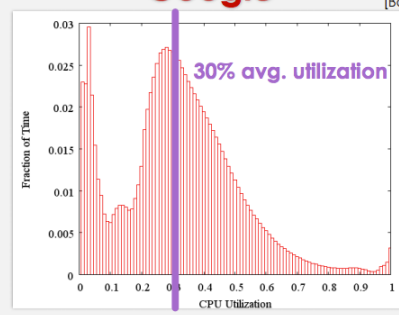


But the datacenters are poorly utilized!

Twitter [Delimitrou'14]



Google [Barroso'09]



- Low utilization in large-scale clouds, even with automated management systems

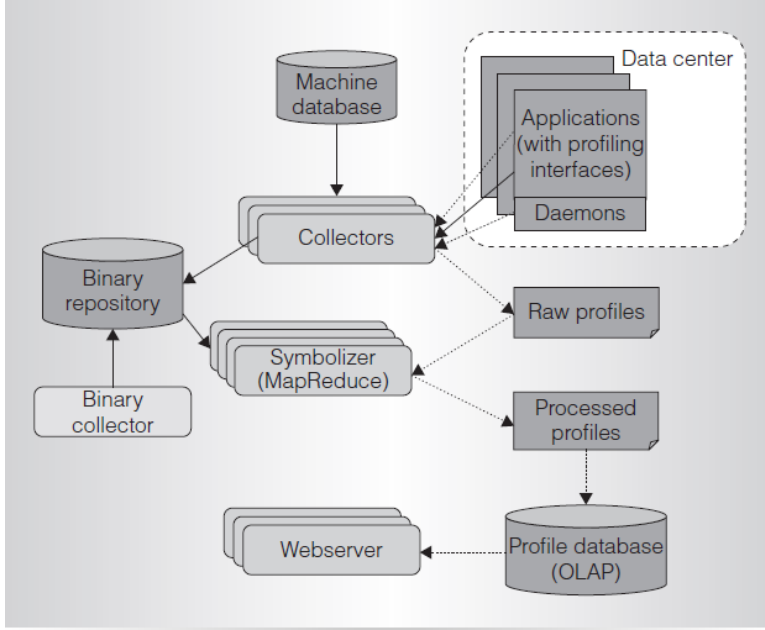
浙江大学 ZHEJIANG UNIVERSITY | Optimatist | AMD

Performance Analysis at Scale

- Google-wide Profiling (GWP)
- CPI^2
- Performance Scaling in many cores
- Resource Usage Effectiveness (RUE)
- System Performance Estimation, Evaluation and Decision (SPEED)

浙江大学 ZHEJIANG UNIVERSITY | Optimatist | AMD

GWP



Gang Ren, Google-wide Profiling @ IEEE Micro 2010

GWP Optimization

- CPI_{ij}, the measured CPI of application j on platform i.
- $TotalLoad_j$, the total measured number of instruction samples of application j.
- $Capacity_i$, the total capacity for platform i, measured as total number of cycle samples for platform i.

The equation is:

$$\text{Minimize} \sum_{i,j} CPI_{ij} * Load_{ij}$$

$$\text{where} \sum_{j} Load_{ij} = TotalLoad_j$$

$$\text{and} \sum_{j} CPI_{ij} * Load_{ij} \leq Capacity_i$$

Gang Ren, Google-wide Profiling @ IEEE Micro 2010

CPI²

In one study, CPI correlates well with throughput

If that is true for your data, then the next slide may help you

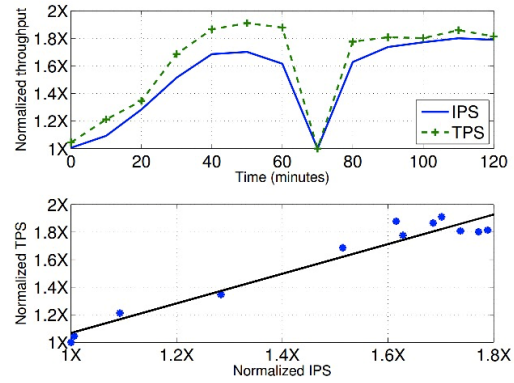


Figure 2: *Normalized application transactions per second (TPS) and instructions per second (IPS) for a representative batch job: (a) normalized rates against running time; (b) scatter plot of the two rates, which have a correlation coefficient of 0.97. Each data point is the mean across a few thousand machines over a 10 minute window. The data is normalized to the minimum value observed in the 2-hour collection period.*

Xiao Zhang, et al. CPI² @ EuroSys 2013

CPI²

CPI Distribution
Ignore small samples (e.g., less tha
Select outlier detection based on a

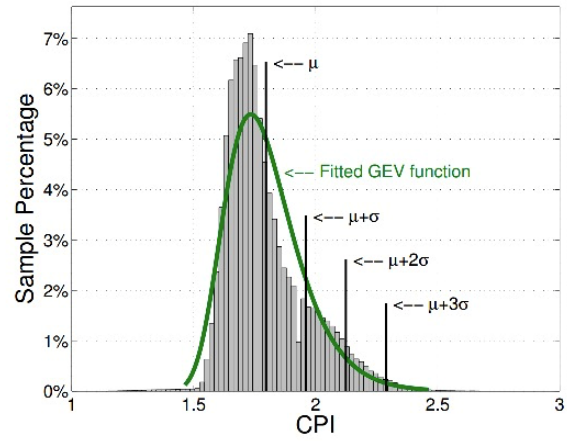
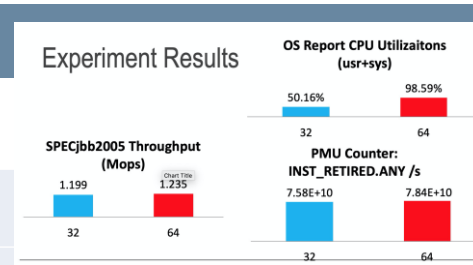


Figure 7: *CPI distribution for a web-search job in a cluster running on thousands of machines of the same type over a 2-day period. The graph includes more than 450k CPI samples and has mean $\mu = 1.8$ and standard deviation $\sigma = 0.16$. We also show the best-fit generalized extreme value curve GEV $(1.73, 0.133, -0.0534)$.*

CPI from SPECjbb2005 experiments



Number of logical CPU's	64				
Frequency /GHz	2.5	(Fixed)			
	Throughput /Mops	CPU Util. /%	Cycles /s	Insts /s	CPI
32 cores (1 thread each)	1.199	50.16	80,256,000,000	75,800,000,000	1.06
32 cores (both threads per core)	1.235	98.59	157,744,000,000	78,400,000,000	2.01

Performance Estimation at Scale

System Tools

$$RUE = \frac{Resource\ Usage}{Work\ Done}$$

Workload Specific

Smaller is better

Resource usage: CPU, Memory, Storage, Network
Work Done: Queries, Tasks

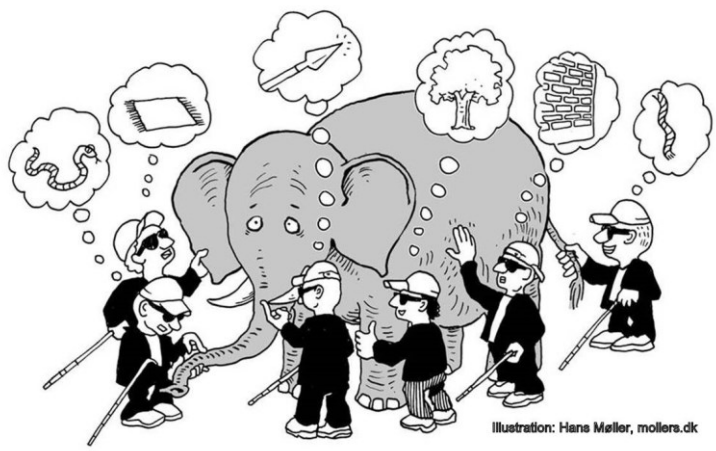
Performance Evaluation at Scale

$$Speedup = \frac{RUE_1}{RUE_2}$$

Bigger is better

RUE_1 is the RUE of configuration 1
RUE_2 is the RUE of configuration 2

Performance Data Collection in the Large



The law of large numbers

a theorem that describes the result of performing the same experiment a large number of times. According to the law, the average of the results obtained from a large number of trials should be close to the expected value, and will tend to become closer as more trials are performed.

https://en.wikipedia.org/wiki/Law_of_large_numbers

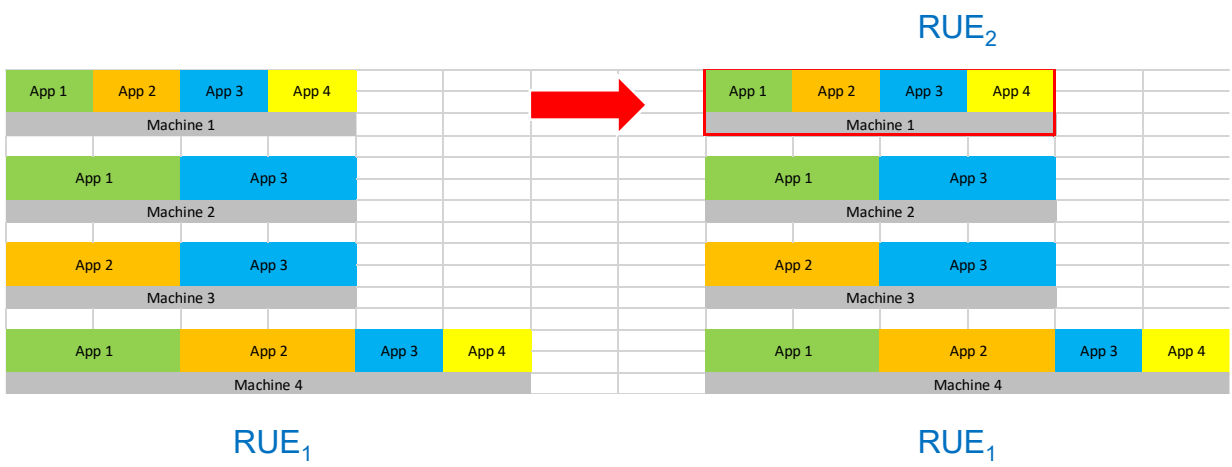
Example: Testing a new feature

To reduce the cost of testing

- 1% of instances of an application ran on the new config (config 2), 99% of instances ran on the old config (config 1)
- No change in deployments, each app might run on the new config or the old config

We still have a large number of samples, even with 1% of the instances

Performance Evaluation at Scale



Big Data

	Config 1		Config 2		Speedup
	Proportion of App Instances	RUE_1	Proportion of App Instances	RUE_2	
App Total	99.00%	885	1.00%	815	1.09

Looks really promising, let's change ??
More samples needed?
More analysis needed?

Big Data Paradox

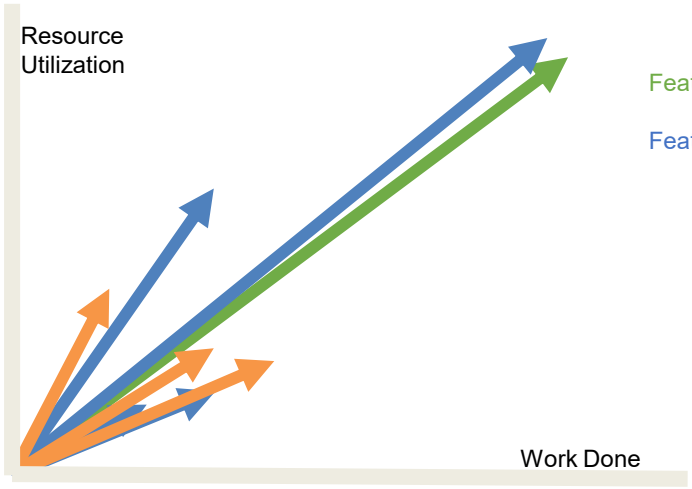
	Config 1		Config 2		Speedup
	Proportion of App Instances	RUE_1	Proportion of App Instances	RUE_2	
App Total	99.00%	885	1.00%	815	1.09
App Group 1	50.10%	1289	0.30%	1484	0.87
App Group 2	31.50%	428	0.40%	434	0.99
App Group 3	17.40%	550	0.30%	655	0.84

Simpson's Paradox

A trend appears in several different groups of data but disappears or reverses when these groups are combined

https://plato.stanford.edu/entries/paradox-simpson/

Simpson's Paradox



Feature on

Feature off

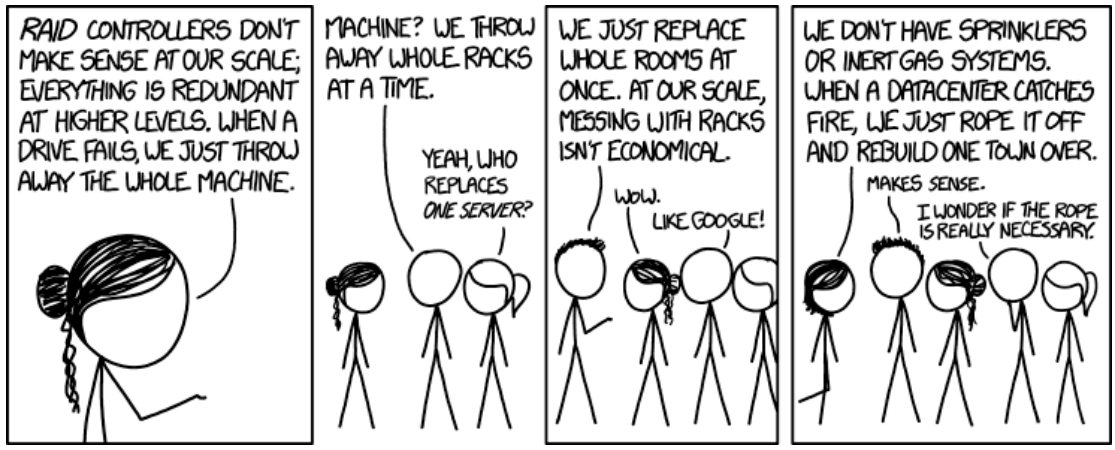
Overall performance increases

Is that real?

References

- G. Ren, E. Tune, T. Moseley, Y. Shi, S. Rus and R. Hundt, "Google-Wide Profiling: A Continuous Profiling Infrastructure for Data Centers," in *IEEE Micro*, vol. 30, no. 4, pp. 65-79, July-Aug. 2010, doi: 10.1109/MM.2010.68.
- Xiao Zhang, Eric Tune, Robert Hagmann, Rohit Jnagal, Vrigo Gokhale, and John Wilkes. 2013. CPI2: CPU performance isolation for shared compute clusters. In Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys '13). Association for Computing Machinery, New York, NY, USA, 379–391. https://doi.org/10.1145/2465351.2465388
- Intel gProfiler
  - https://www.intel.com/content/www/us/en/newsroom/news/intel-releases-continuous-profiler-for-cpu-performance.html#gs.785y33
- Facebook RocksDB
  - https://github.com/facebook/rocksdb
- Simpson's Paradox
  - https://github.com/ninoch/Trend-Simpsons-Paradox
  - https://github.com/CamDavidsonPilon/simpsons-paradox
  - https://github.com/ijmbarr/simpsons-paradox
  - https://github.com/ehart-altair/SimpsonsParadox

☺ THANK YOU ☺



https://xkcd.com/1737/