TailWAG: Tail Latency Workload Analysis and Generation

Heng Zhuo

University of Wisconsin-Madison

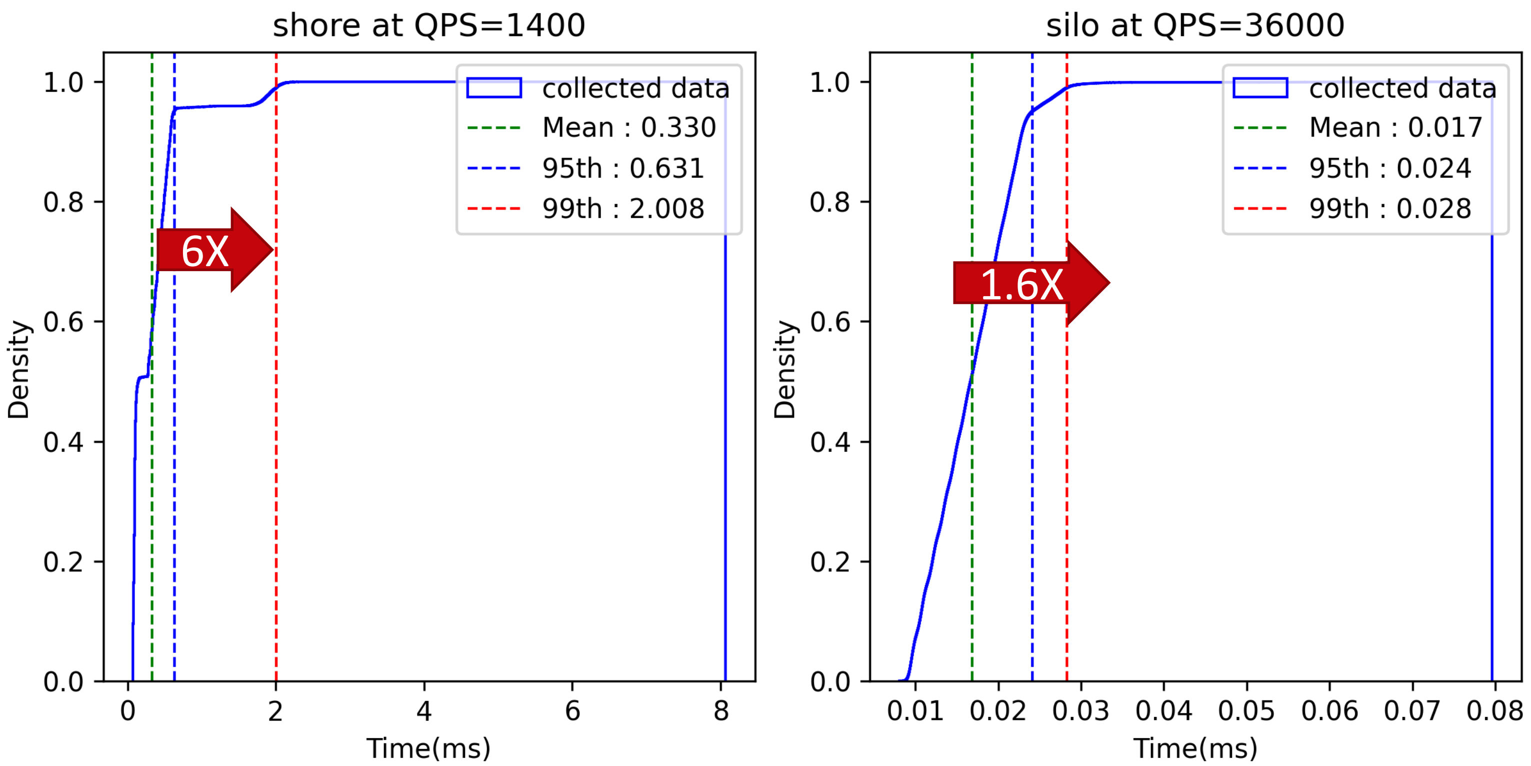
Executive Summary

- Introduction and Background
  - Tail Latency.
  - Problems and challenges.
- TailWAG
  - Workload analysis.
  - Workload generation and validation.
  - Case Study.
- Conclusion

Cloud Computing, Data Center, Supercomputer

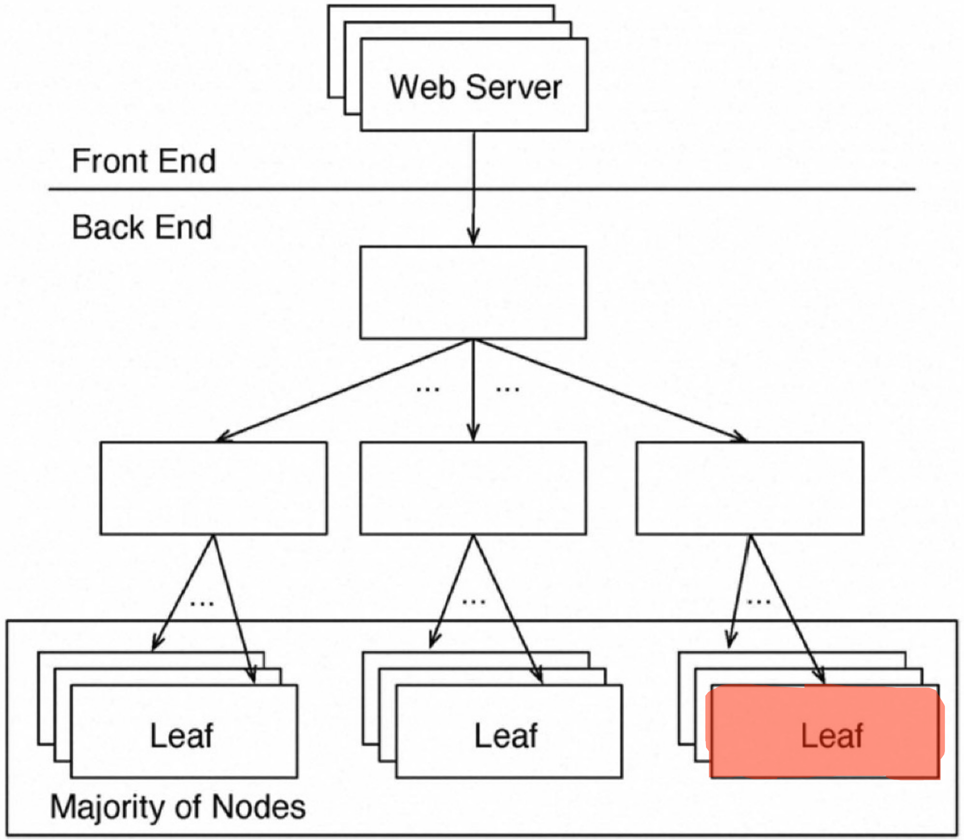
An eight-rack pod of Google's liquid-cooled TPU version 3 servers for artificial intelligence workloads. (Image: Google)

Tail latency in server workload



Mean latency may not be representative for a system over time.

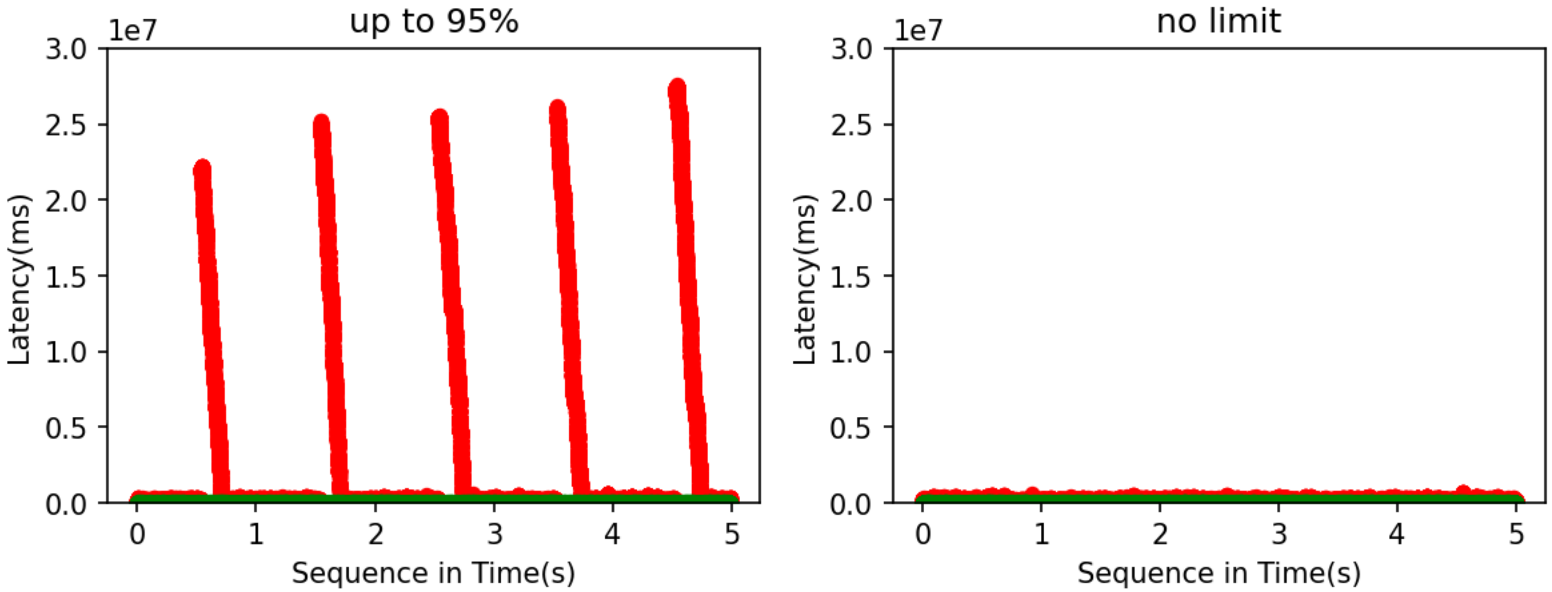
Server at scale



Singler user request may end up using over hundreds of server nodes.

Meisner, D. & Sadler, C.M. & Barroso, Luiz & Weber, W. & Wenisch, Thomas. (2011). Power management of Online Data-Intensive services.

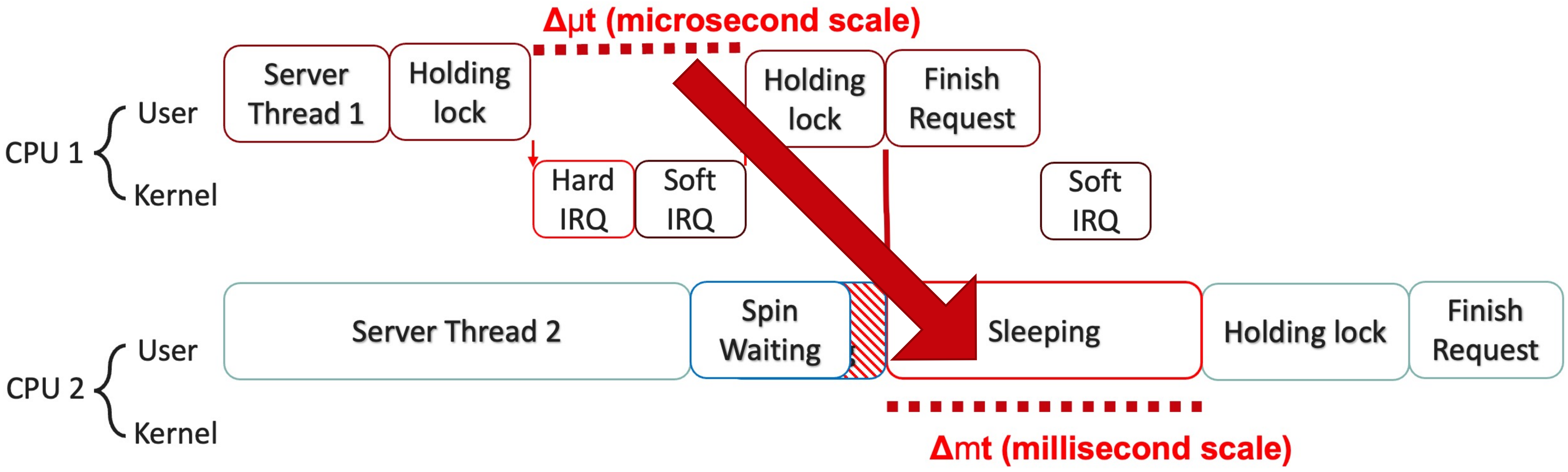
Causes for tail latency: Kernel Real-Time Scheduling



- Application can be run on Real-Time(higher) priority. (FIFO/Round-Robin).
- Linux Kernel, by default: only 95% of CPU time can be used by Real-Time.

Note, this be dangerous, such as deadlock.

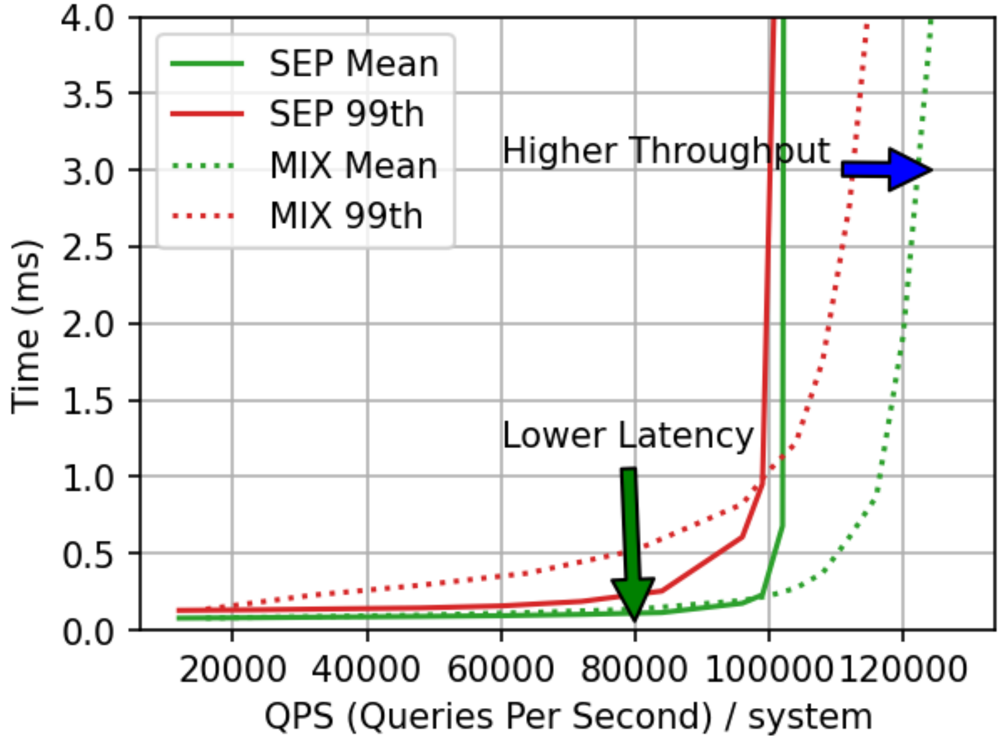
Causes for tail latency: (Ethernet) Interrupt handling



Interrupt Handling is bad, we should avoid it.

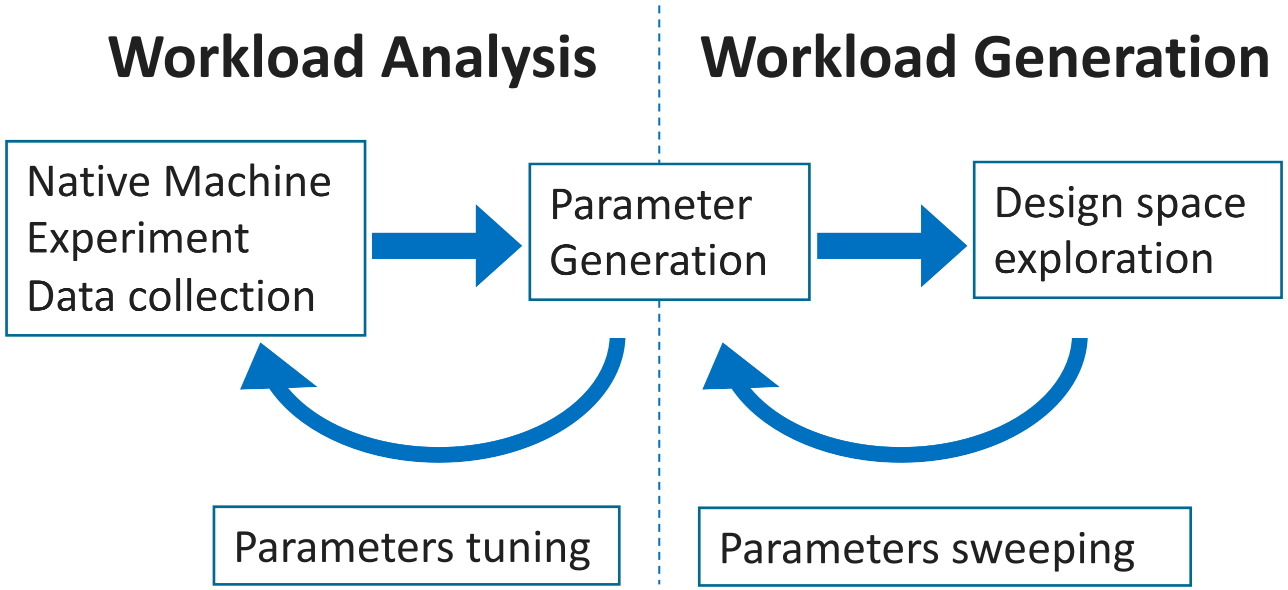
Causes for tail latency: (Ethernet) Interrupt handling

- For a 4-core system:
- SEP: reserving core 1 for interrupt handling;
- MIX: using all 4 cores for both interrupts and server threads.



Tradeoff question: Better Latency or Better Throughput?

TailWAG: Workflow



Executive Summary

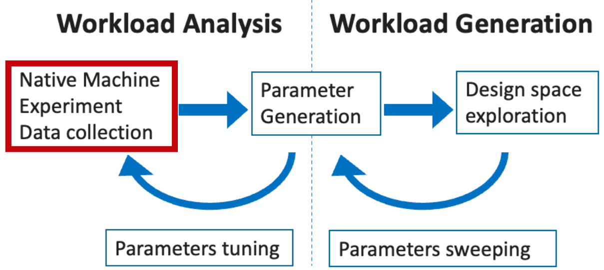
- Introduction and Background
  - Tail Latency
  - Problems and challenges
- TailWAG:
  - Workload analysis.
  - Workload generation and validation.
  - Case Study.
- Conclusion

Executive Summary

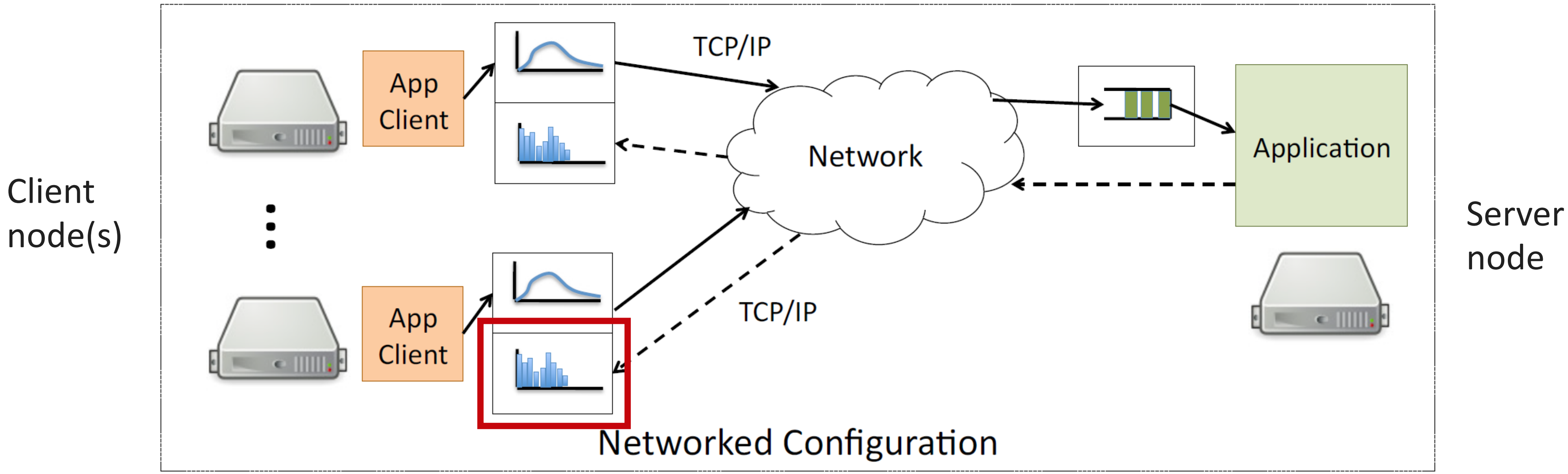
- Introduction and Background
  - Tail Latency
  - Problems and challenges
- TailWAG:
  - Workload analysis.
  - Wo
  - Cas

Spoiler alert, we use 3 sets of parameters.

- Conclusion

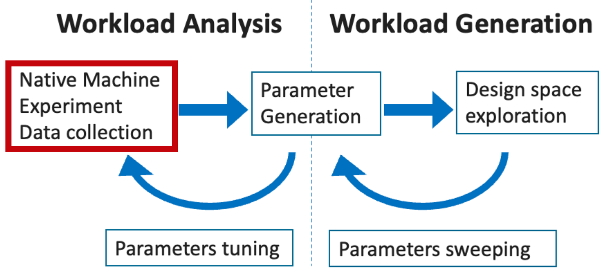


Workload Analysis: Tailbench



- Harness: single/multiple server thread(s) and client thread(s).
- Applications: online search, key-value store, image recognition, etc.

H. Kasture and D. Sanchez, "Tailbench: a benchmark suite and evaluation methodology for latency-critical applications," 2016 IEEE International Symposium on Workload Characterization (IISWC), 2016

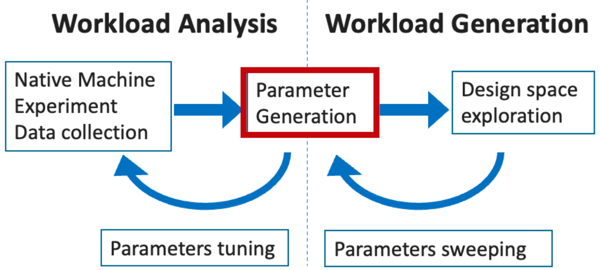


Analysis: Experimental Setup

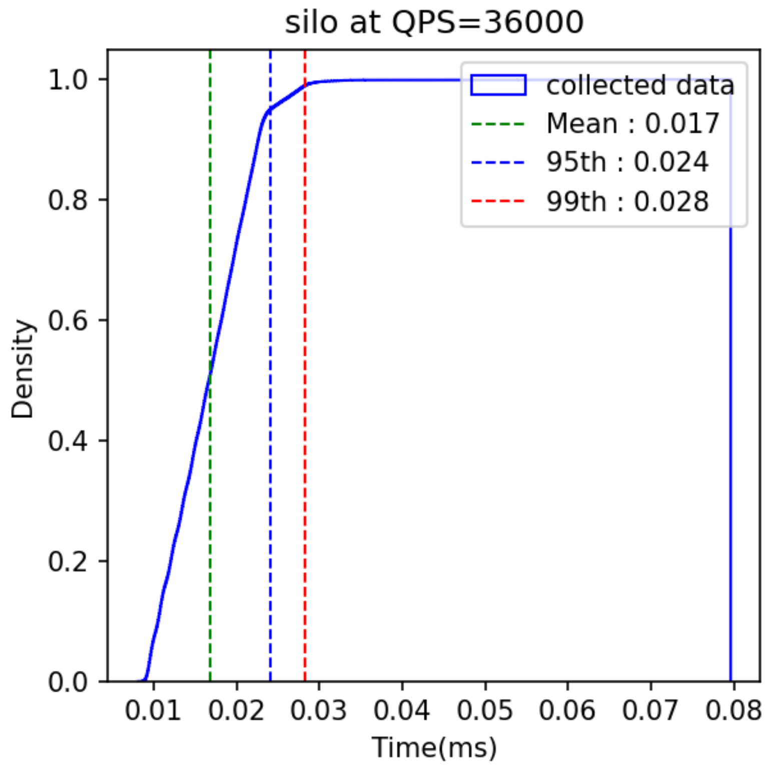
- Generated data from server running on :

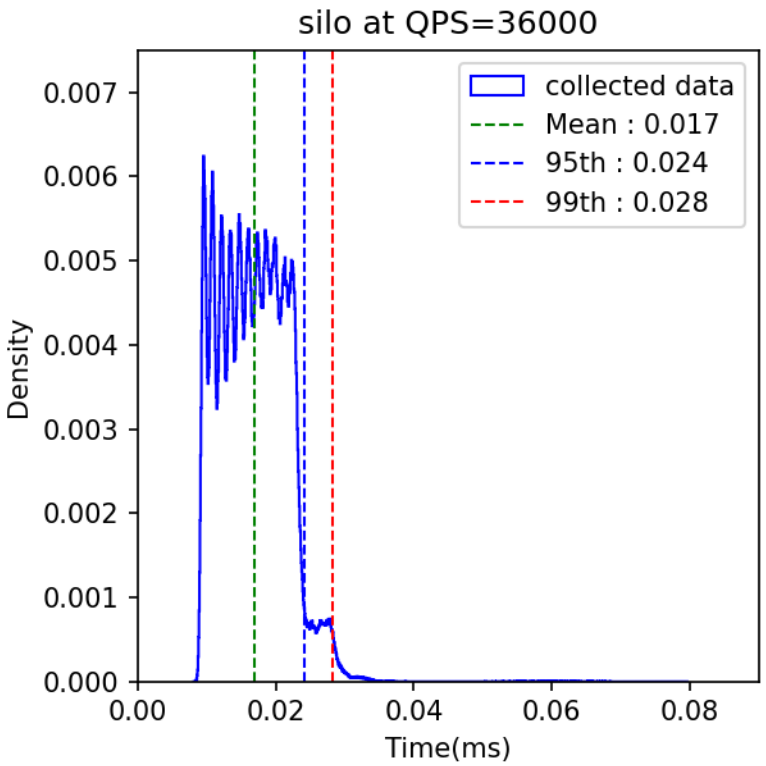
Model	Supermicro SYS-2029GP-TR
Cores	6 Cores Intel Xeon Gold 6128, 3.5 GHz
Caches	32KB L1, 6MB L2, 19.25MB L3
Main Memory	96GB, 1333 MHz
Operating System	Ubuntu 22.04 , Linux kernel 5.15.0

- SMT(Simultaneous multithreading), deep sleep disabled.
- Client connected with direct Gigabit Ethernet.

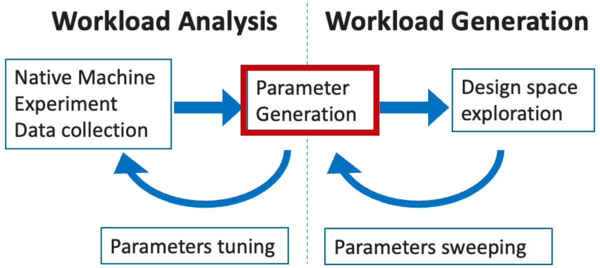


Analysis: Service Time Distribution



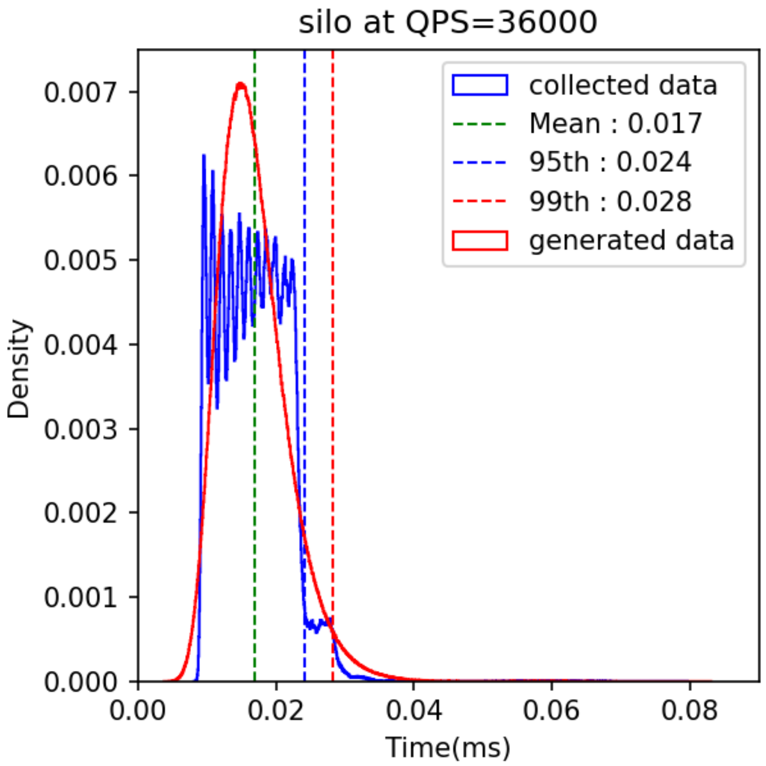


Probability function is easier to identify the shape

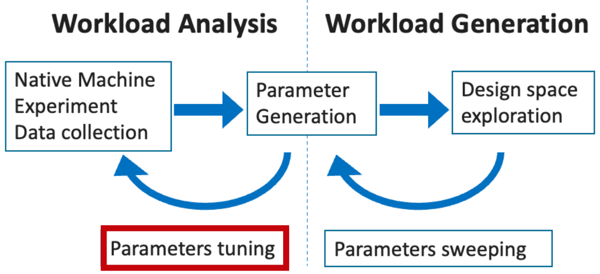


Analysis: Service Time Distribution

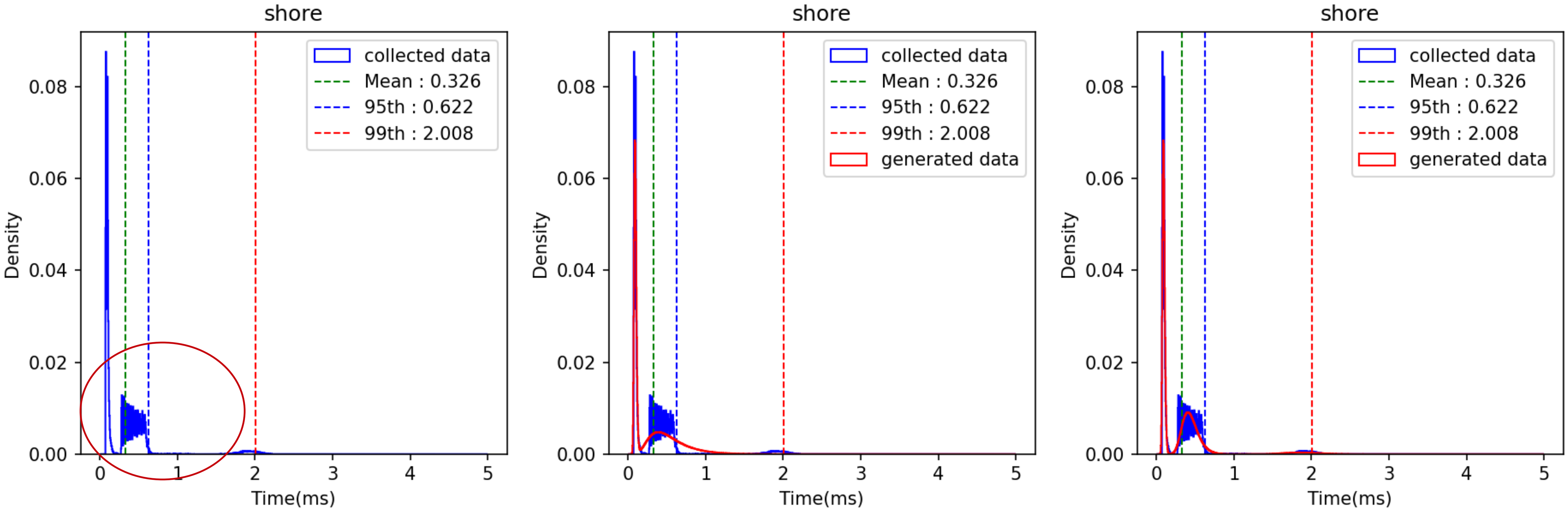
- Using SciPy provided stats, get:
  - Mean;
  - Variance.
- Feed into NumPy random, generate data.



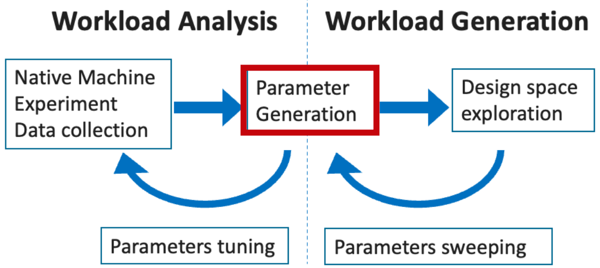
Note: only analysis now, everything in Python for now.



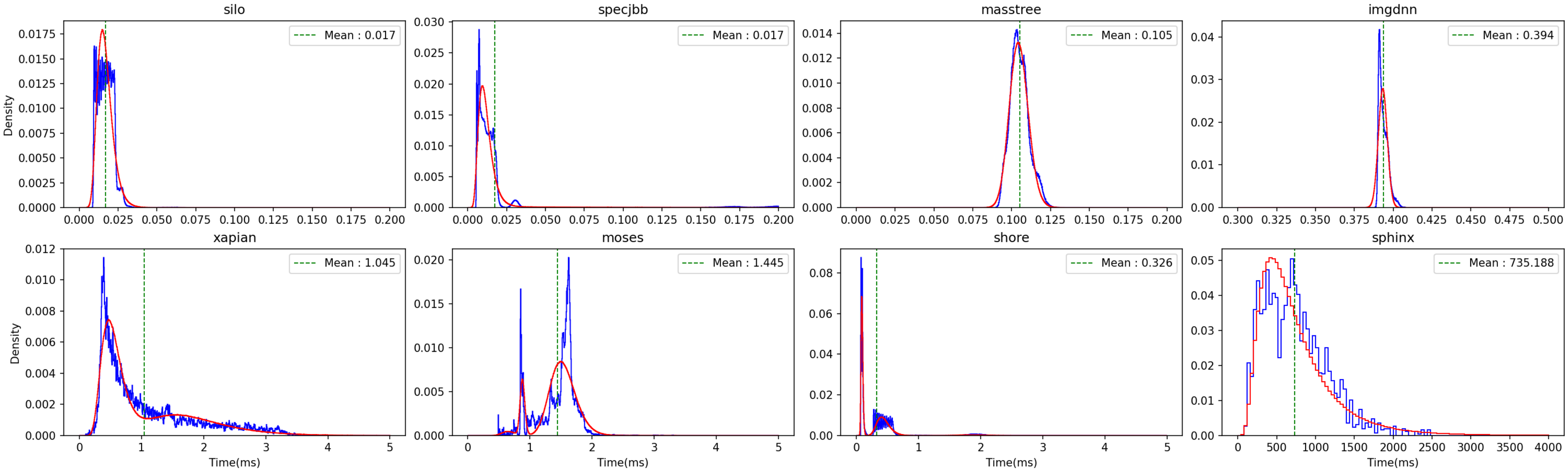
Analysis: Parameter Tuning



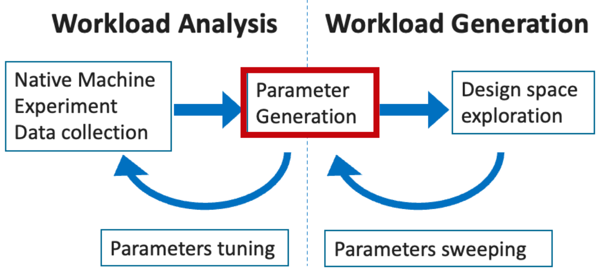
Using one distribution is not enough for some application.



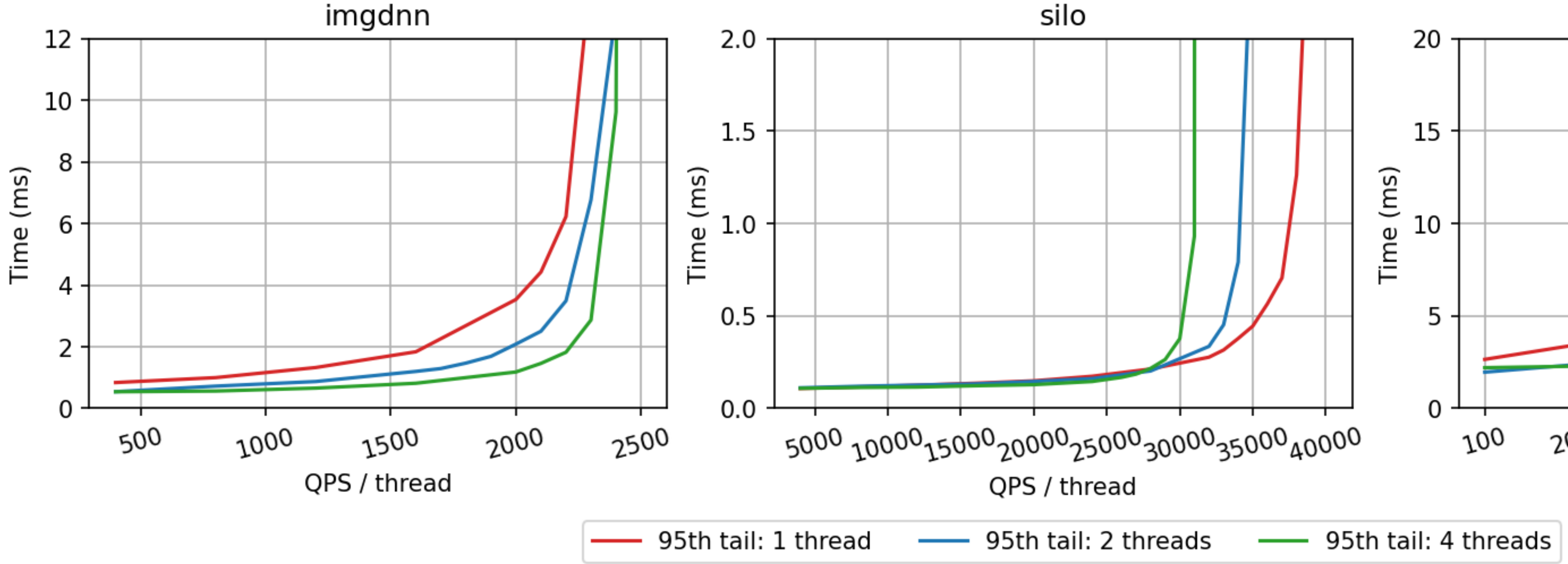
Analysis: Service Time Distribution



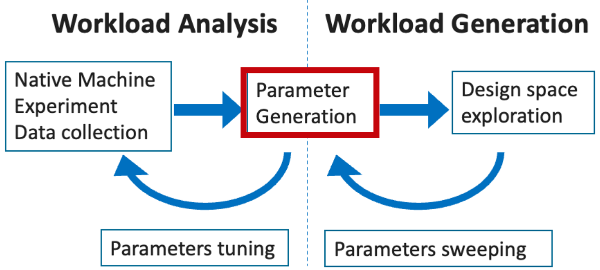
Using 1-3 distributions can cover service time behavior.



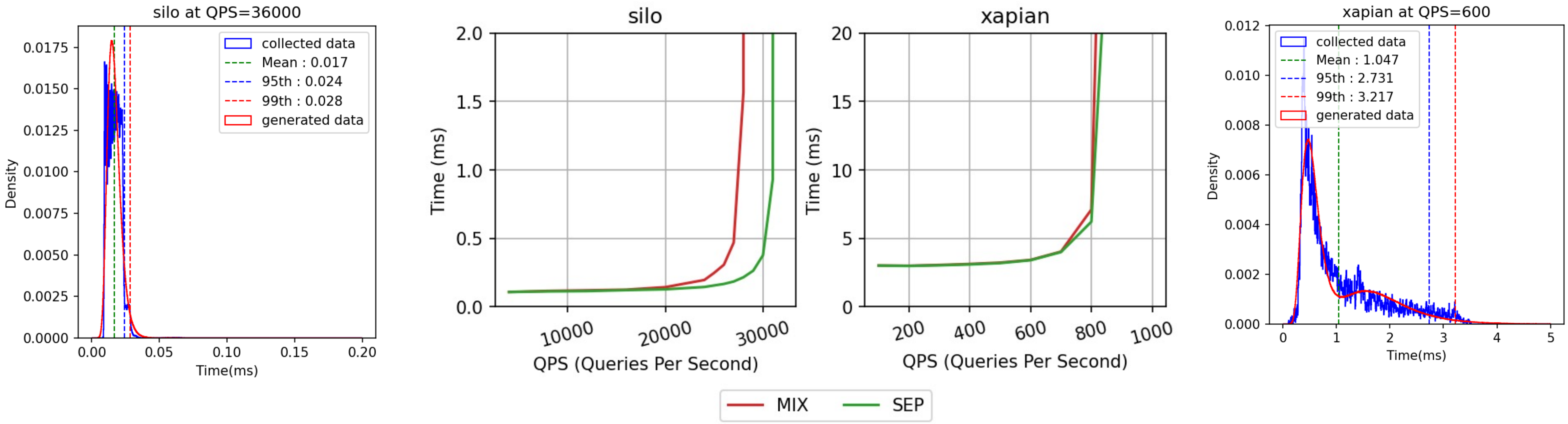
Analysis: Critical Section



- Imgdnn: better than linear.
- silo: worse than linear, due to critical section relating to TCP stack.
- moses: worse than linear after 2 threads, due to memory bottleneck.



Analysis: Timing Disturbance



Wide distribution make application more robust to timing disturbance.

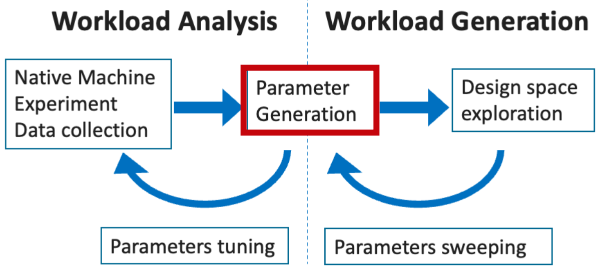
SEP: reserving core 1 for interrupt handling.
MIX: using all 4 cores for both interrupts and server threads.

Executive Summary

- Introduction and Background
  - Tail Latency
  - Problems and challenges
- TailWAG:
  - Workload analysis.
  - Workload generation and validation.
  - Case Study.
- Conclusion

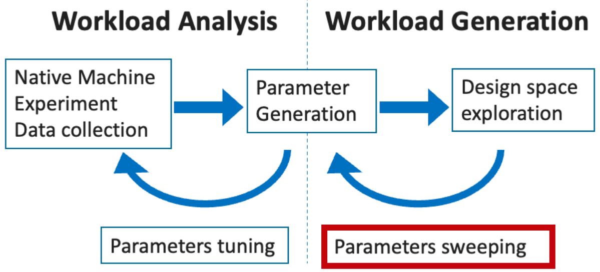
Workload Generation

- Parameters assemble:
  - Written in C++.
  - 30 lines of code.
- Each loop = one query:
  - Service time distribution.
  - Critical section (Receive/ Sent).
  - Any timing disturbances.

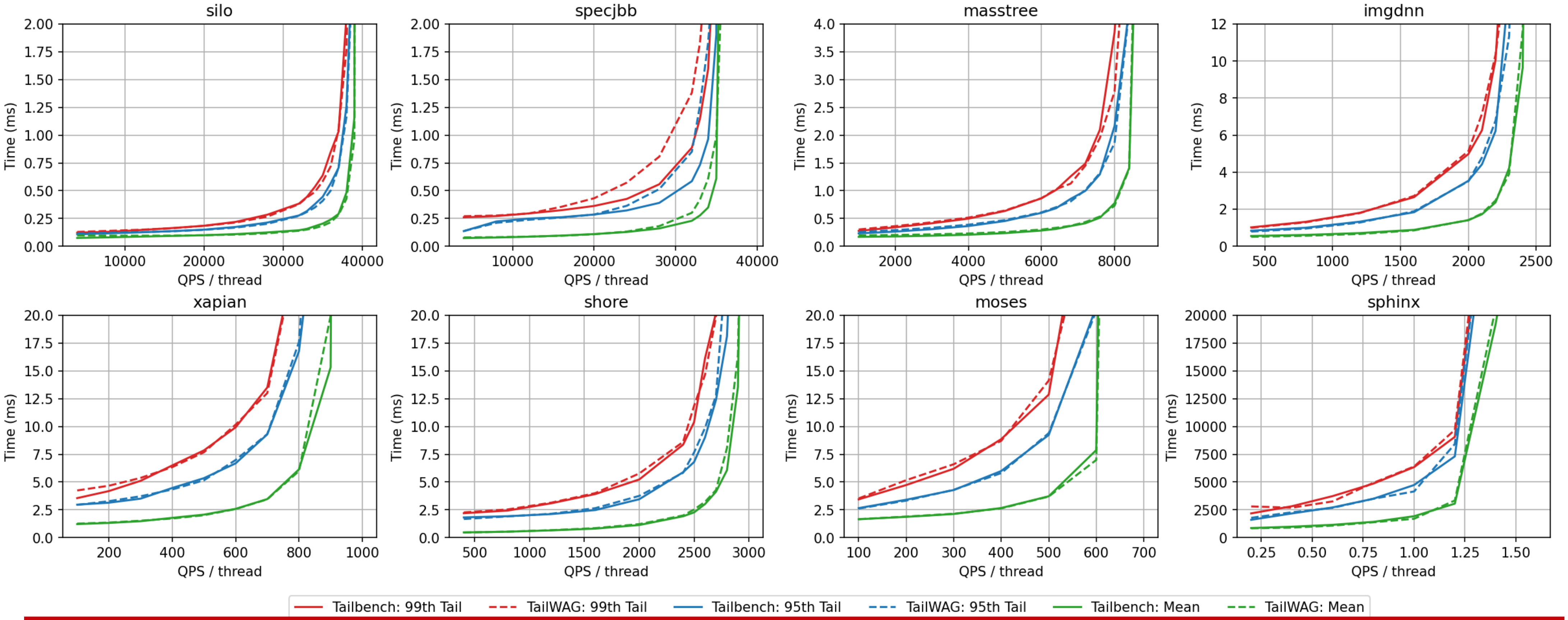



```
void doRun(){
  //setup parameters
  ...
  //each iteration
  //is one query
  while (true) {
    recvRequest();

    lock(recvLock);
    spin(recvLockCycles);
    unlock(recvLock);

    p=uniform_random(0.001%,100%);
    if p < prob1
      distCycles=log_random(mean1,var1);
    else if p < prob1+prob2
      distCycles=log_random(mean2,var2);
    else
      distCycles=log_random(mean3,var3);
    spin(distCycles);

    //can be multiple noise injection
    spin(noiseCycles);

    lock(sentLock);
    spin(sentLockCycles);
    unlock(sentLock);

}
```

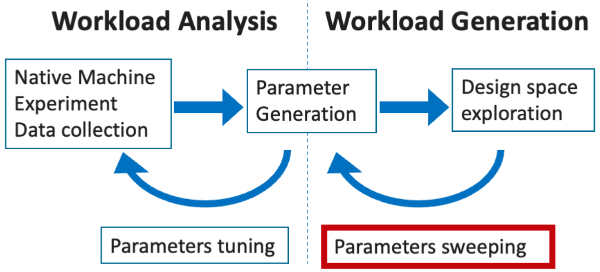
30 lines of code representing millions of lines?



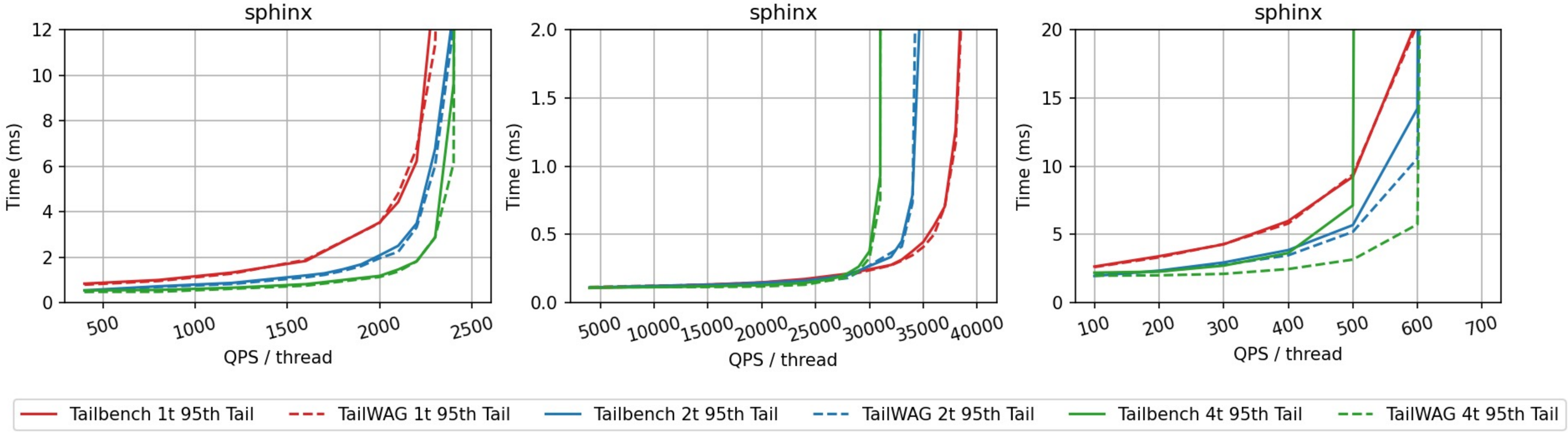
Validation: Single Thread



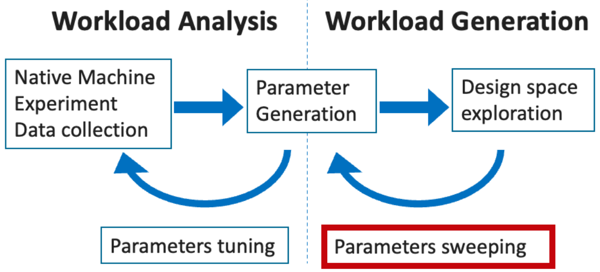
Using a set of main distribution mimics latency and throughput performance.



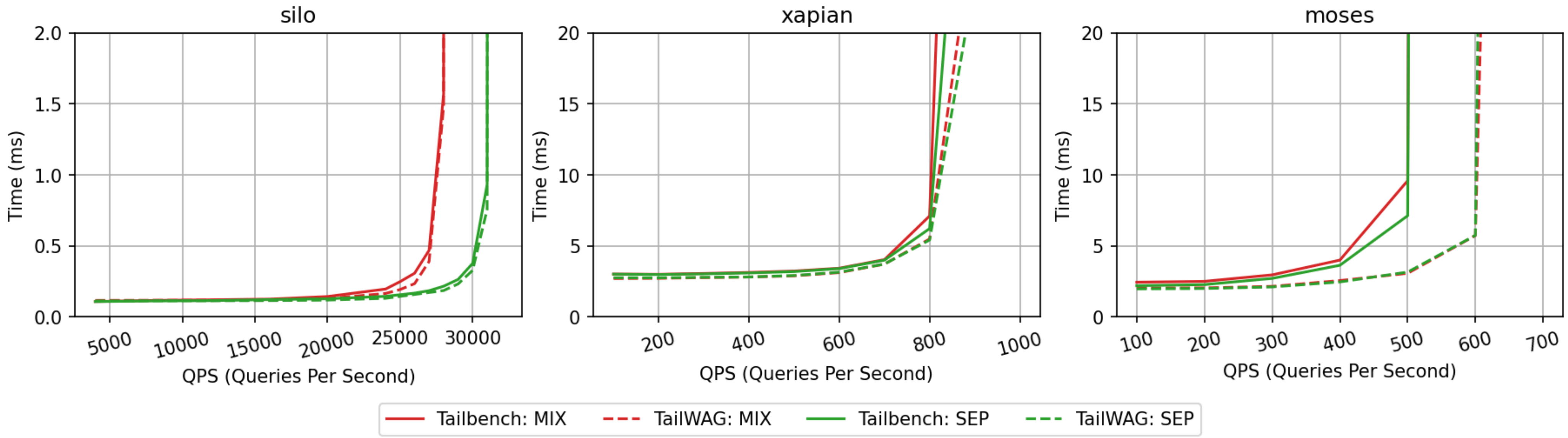
Validation: Critical Section



- img-dnn and silo: behaves close with critical section parameter.
- moses: showing optimal upper bound without memory bottleneck.



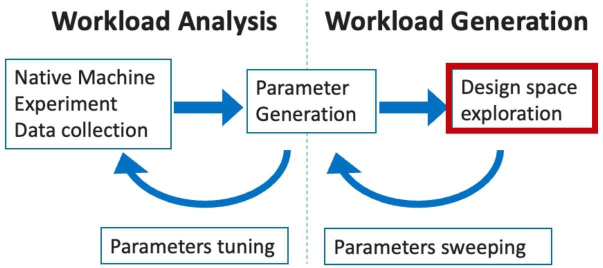
Validation: Timing Disturbance



- silo and xapian: behaves close with timing disturbance parameter.
- moses: showing optimal upper bound without memory bottleneck.

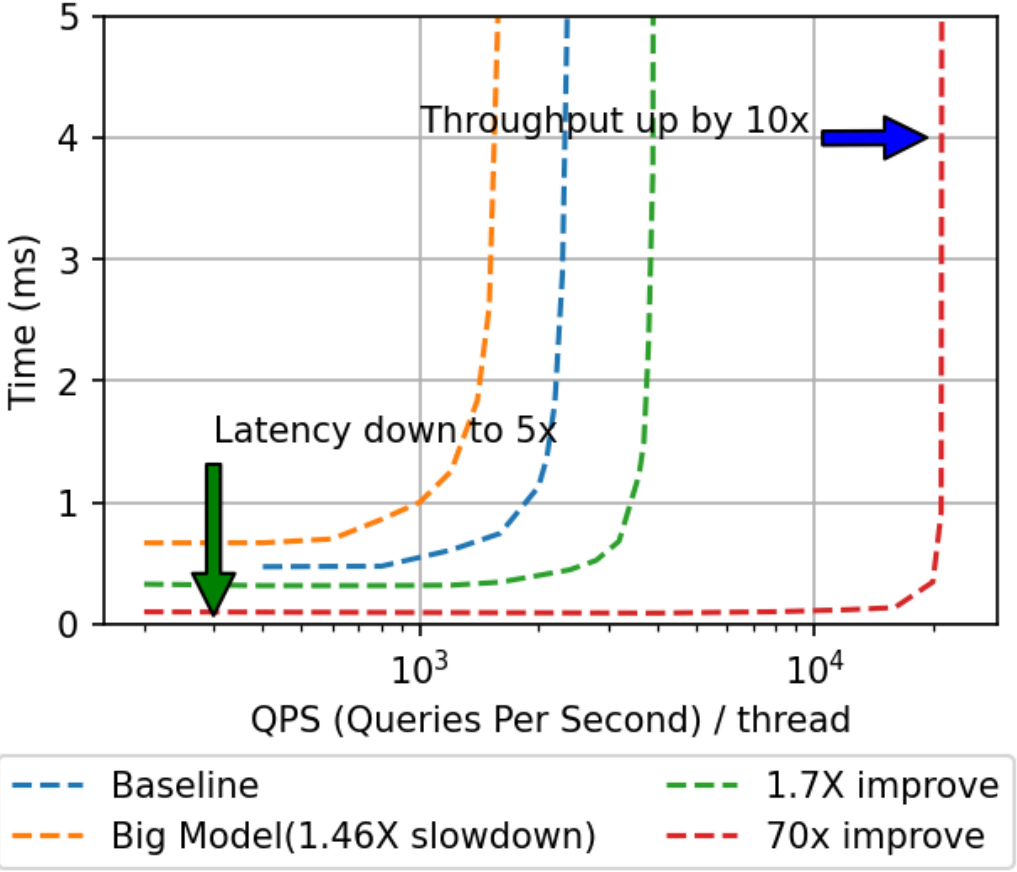
Executive Summary

- Introduction and Background
  - Tail Latency
  - Problems and challenges
- TailWAG:
  - Workload analysis.
  - Workload generation and validation.
  - Case Study.
- Conclusion



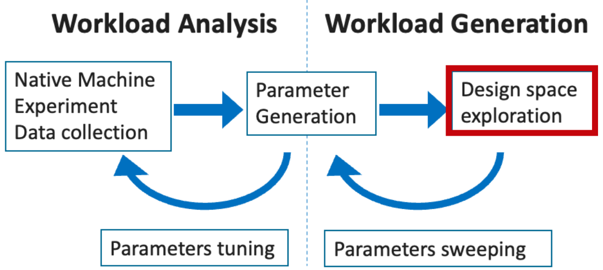
Case Study: Hardware Innovation

- DNN algorithm revolution:
  - Complex model for better accuracy
  - Annual 1.46X FLOPS increase
- DNN hardware revolution
  - TPU v1: 1.7X speedup
  - TPU v2: 70X speedup
- Parameter change, only service time distribution :
  - Baseline: 395 ms
  - Big Model: 671 ms
  - TPU v1: 232 ms
  - TPU v2: 5.6 ms



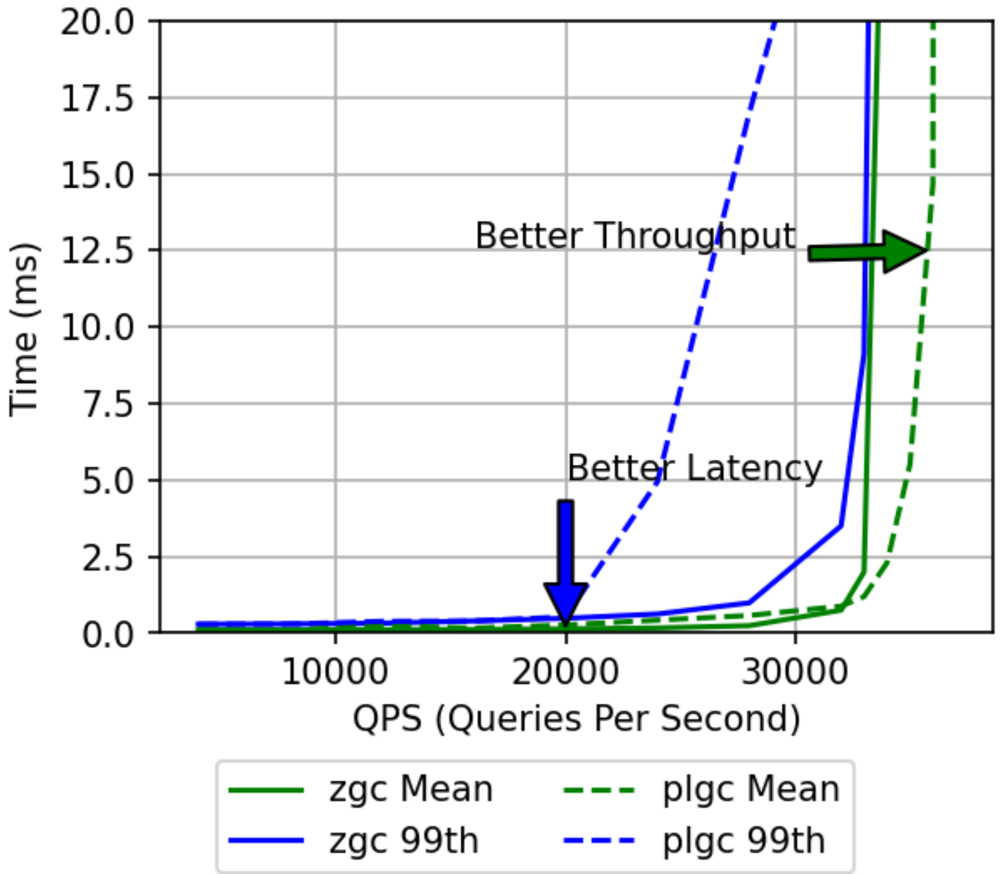
Easy design space exploration.

[1] Jouppi, Norman P., et al., "In-datacenter performance analysis of a tensor processing unit., ISCA, 2017.
[2] Jouppi, Norman P., et al., "Ten Lessons From Three Generations Shaped Google's TPUv4i : Industrial Product," ISCA, 2021.



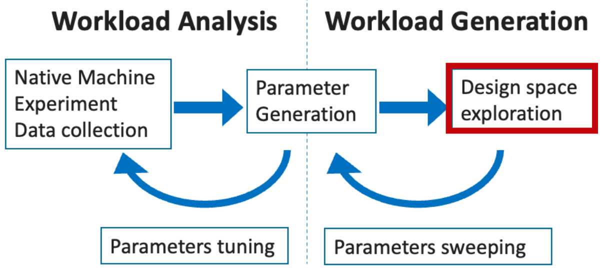
Case Study: Garbage Collection

- plgc(Parallel Garbage Collector):
  - Classic, throughput oriented.
  - Longer pausing time, can be over 100 ms.
- zgc(Z Garbage Collector):
  - Newer feature, latency oriented, evolving every java version.
  - Often but short pausing time, less than 1 ms.



- Parameter change:
- plgc: timing disturbance (50 ms every 10 s).
- zgc : 10% more o

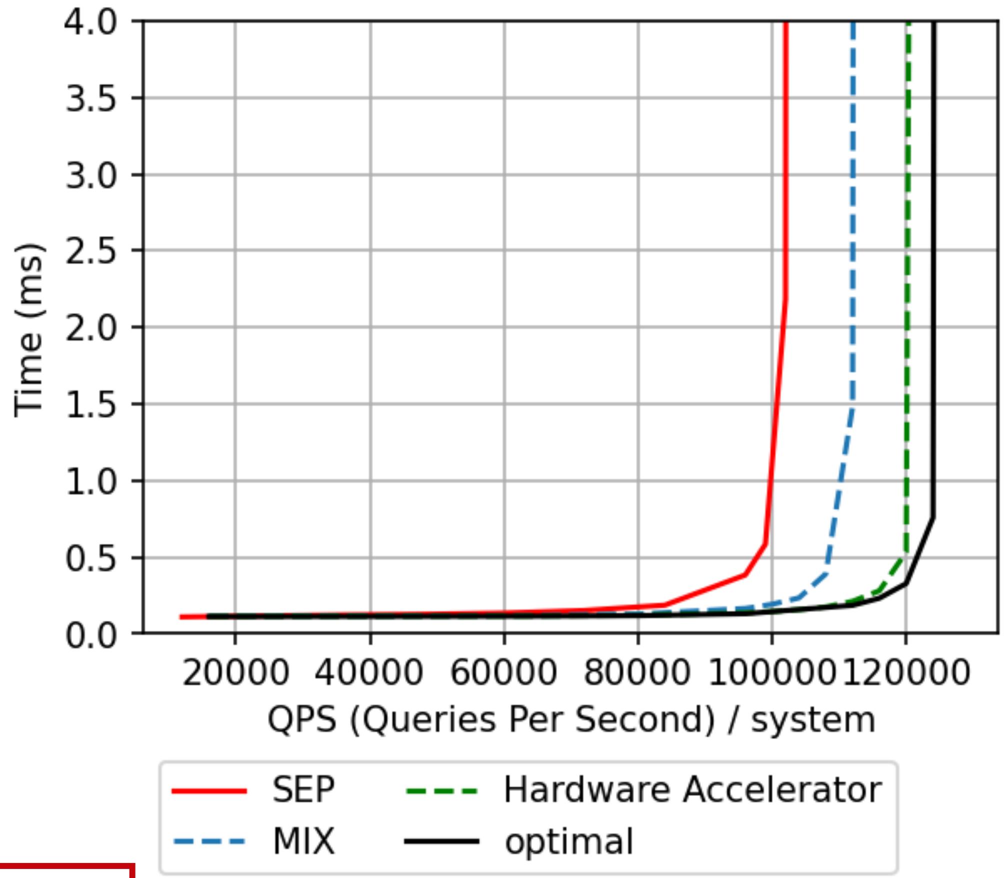
Easy run time configuration exploration.



Case Study: Hardware Accelerator

- System with 4 cores:
- SEP: 3 threads, for better latency.
- MIX: 4 threads, for better throughput.

- Parameter change on timing disturbance:
- MIX: 4 µs.
- Hardware Accelerator: reduce to 0.5 µs.
- Optimal: Without any, 0 µs.



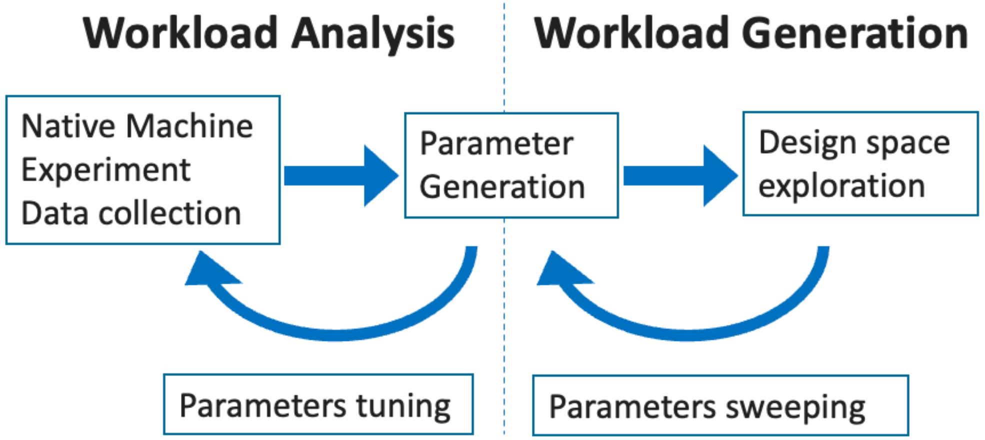
Easy design space exploration.

SEP: reserving core 1 for interrupt handling.
MIX: using all 4 cores for both interrupts and server threads.

Executive Summary

- Introduction and Background
  - Tail Latency
  - Problems and challenges
- TailWAG:
  - Workload analysis.
  - Workload generation and validation.
  - Case Study.
- Conclusion

Conclusion



- Server Workload:
  - Tail latency and throughput are both important.
  - Tuning system and design space exploration are difficult.
- TailWAG: Tail Latency Workload Analysis and Generation
  - 30 lines of code for generated workload. ✅
  - Validated against real workload. ✅
  - Repeatable behavior and measurements. ✅
  - Enable exploration on future design(hardware/software). ✅

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON