Asymmetric Network Processing to Reduce Jitter

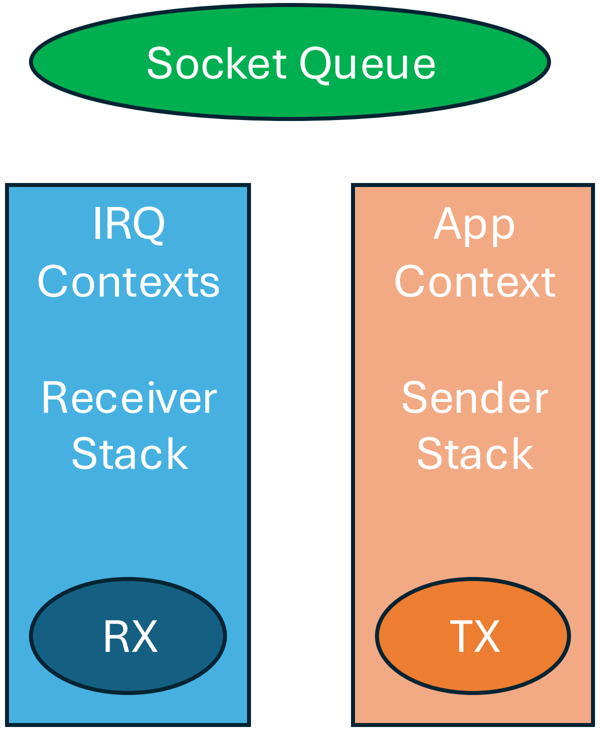
Satish Kumar
satish.kumar@bytedance.com
ByteDance
System Technology Engineering (STE)

Agenda..

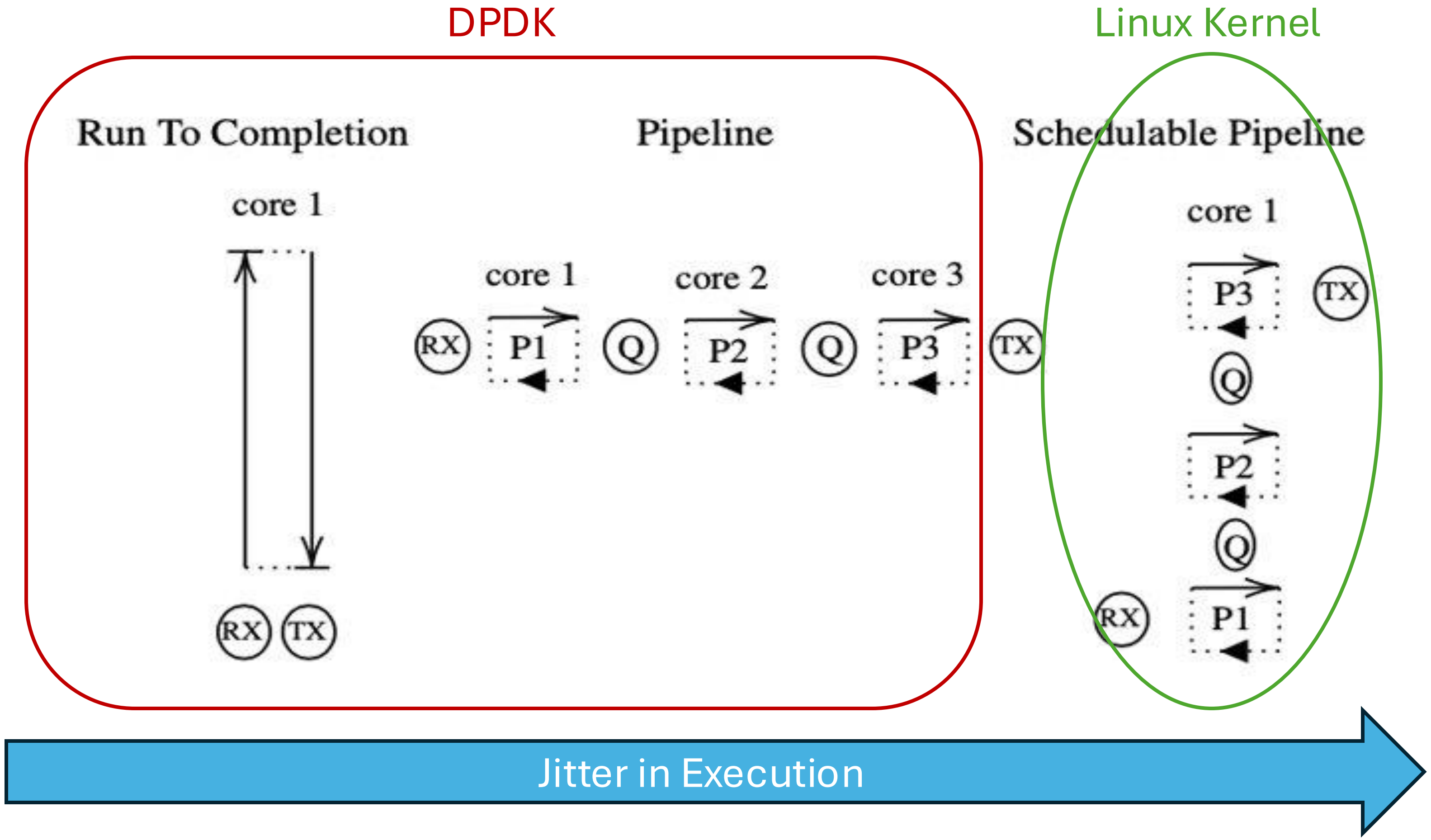
- Commonly used event loop designs.
- Preferred network configuration.
- Analyze the Jitter introduced.
- A dynamic approach to handle the jitter.

Linux Kernel Networking Stack

- Pipeline by design
- Runs inside two context:
  1. IRQ Context (NIC IRQ + SoftIRQ)
     - Receiver stack
  2. Application Context
     - Sender stack



Network Execution Models



- Kernel does the scheduling and provide concurrent safe environment.

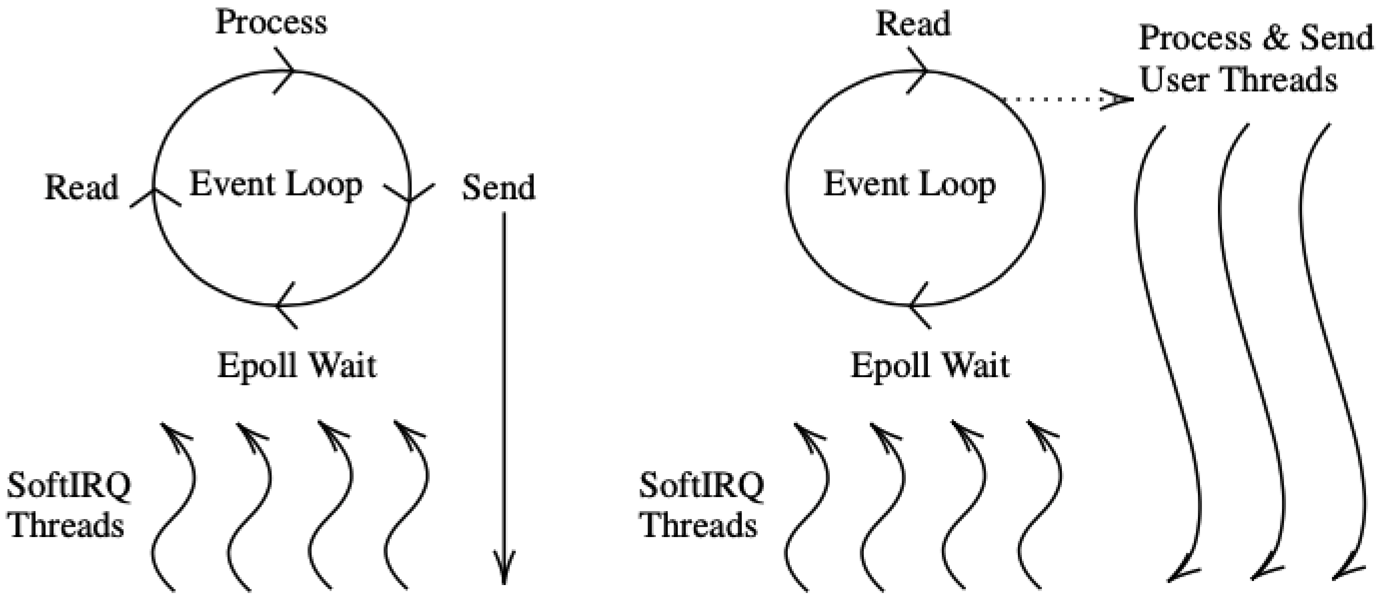
Scale-Out Configuration

- Ideally
  - The receiver and sender stack should run on the same CPU
    - For example, by using RFS/aRFS (Receiver Flow Steering)
  - To have better cache relevance

- But that’s not the case, due to
  - Application uses single thread receiver architecture
    - Multiplex read IOs
  - Depends upon
    - Kernel threads for scalability

- Normally used is scale-out configuration
  - By using NIC multi-queue (RSS: Receive Side Scaling)

Event Loop Architectures



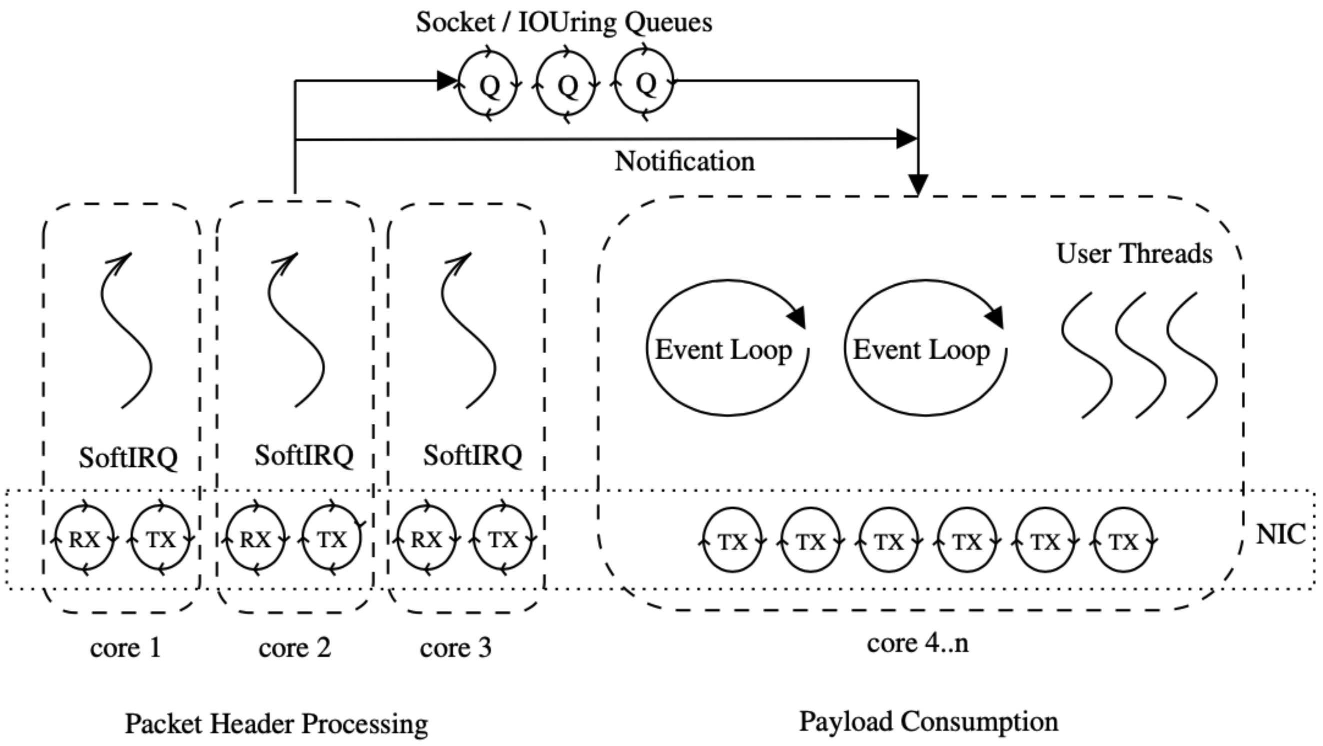
- Redis and NetPoll RPC Framework
- One to Many relation between application receiver thread and kernel SoftIRQ threads.

Jitter in Scale-Out

- Kernel receiver stack runs on most CPUs
  - Polluting caches
- Frequently interrupts the application execution:
  - NIC IRQ
  - SoftIRQ

- So if we think (and highlighted in multiple documents):
  - There is no logical sharing between the receiver and application context.
  - One process packet header, other consumes the payload.
  - It make sense to separate the two context.

Asymmetric Network Processing (ANP)



Isolates receiver context from application context by:

1. Identifying CPUs to reserve for receiver stack.
2. Modify the flow hash indirection table in the NIC to divert all packet equally to the reserved CPUs.
    - Ethtool –X <dev> equal <#reserved_cpus>
3. Change application task affinity to remaining CPUs.

- TX queues needs to be mapped to

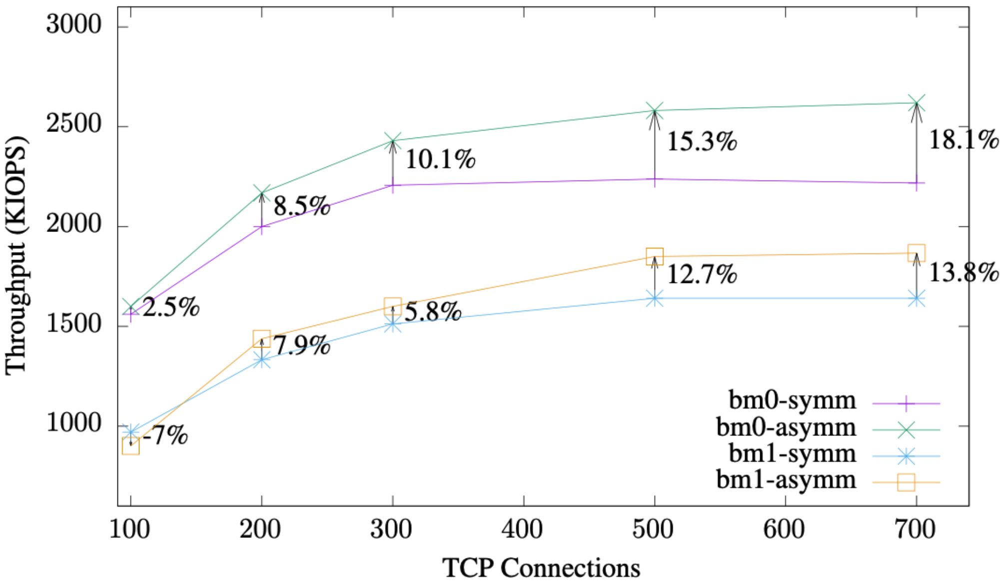
Analysis

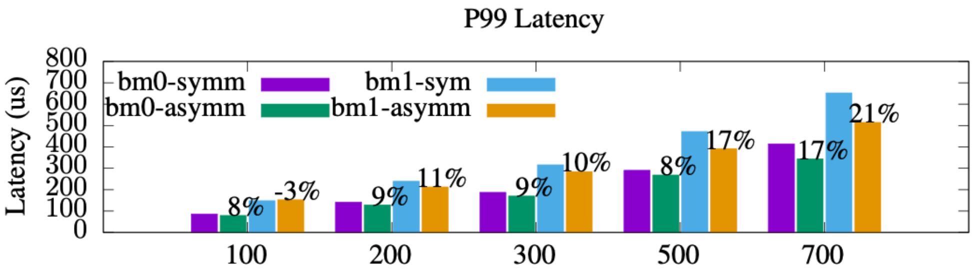
- Ping pong client server benchmark
  - Bm0: redis like architecture
  - Bm1: netpoll like architecture

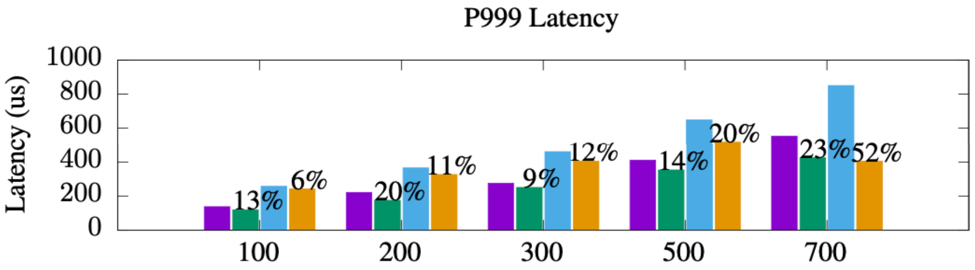
  - The server modifies the data.
  - Pfifo qdisc is in use.

- Value size 1KB
- Total CPUs 20
- CPUs reserved in ANP for bm0 and bm1 are 8 and 4 respectively.
- Numbers measured on server side, where each server creates four event loop of similar type.

Analysis continues..

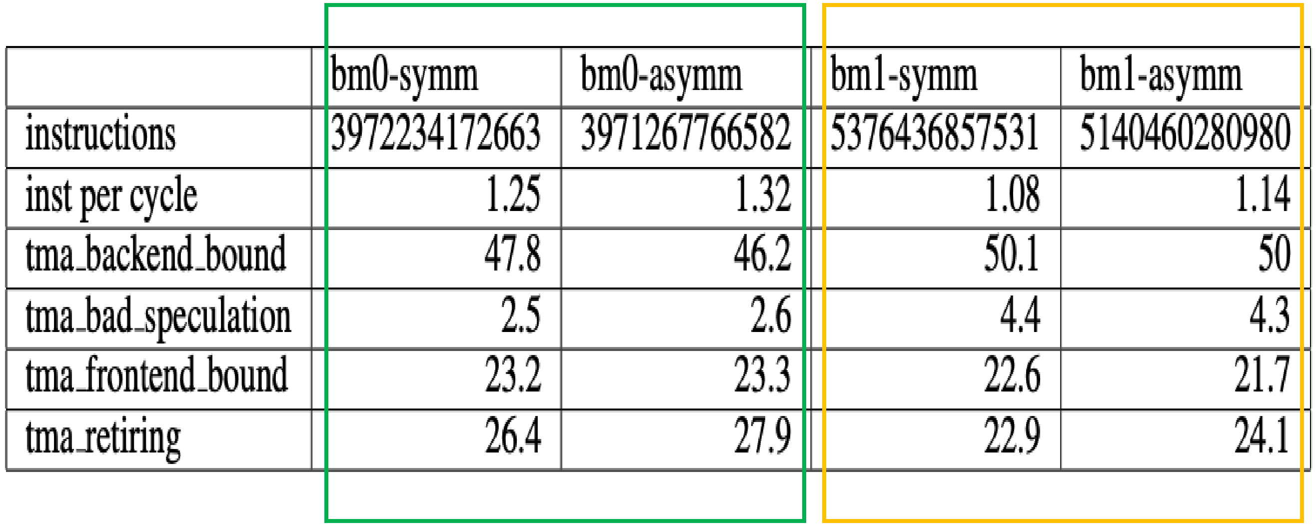






- Throughput increase with number of connections.
  - 2-18% increase
- Significant reduction in 99 and 99.9 percentile latencies.
  - 10-50%

Analysis continues..



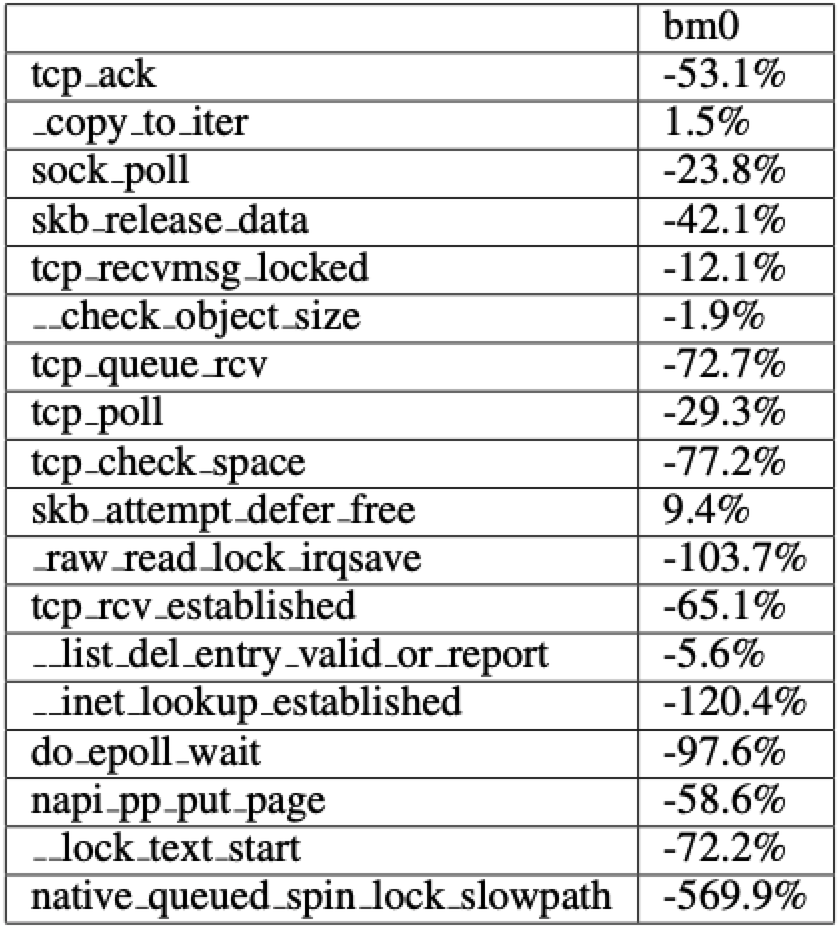
	bm0-symm	bm0-asymm	bm1-symm	bm1-asymm
instructions	3972234172663	3971267766582	5376436857531	5140460280980
inst per cycle	1.25	1.32	1.08	1.14
tma_backend_bound	47.8	46.2	50.1	50
tma_bad_speculation	2.5	2.6	4.4	4.3
tma_frontend_bound	23.2	23.3	22.6	21.7
tma_retiring	26.4	27.9	22.9	24.1

	bm0	bm1
MEM_INST_RETIRED.ANY	0.3%	4.7%
CYCLE_ACTIVITY.STALLS_MEM_ANY	2.8%	7.9%
EXE_ACTIVITY.BOUND_ON_STORES	15.4%	36.1%
CYCLE_ACTIVITY.STALLS_L1D_MISS	-5.6%	5%
CYCLE_ACTIVITY.STALLS_L2_MISS	-5.3%	5%
CYCLE_ACTIVITY.STALLS_L3_MISS	94.6%	91.1%
ICACHE_TAG.STALLS	33.3%	24.7%
ICACHE_DATA.STALLS	14.8%	29.8%

- Topdown analysis shows reduction of bottleneck on frontend as well as backend.
- Improves instruction retiring and provide better instruction per cycle
- Stalls on Frontend caches (TLB and L1 Instruction cache) reduce by ~30%.
- Slight increase in local core caches (L1 and L2) and decrease on shared cache (L3) signify cross core communication after the isolation.

Analysis continues..

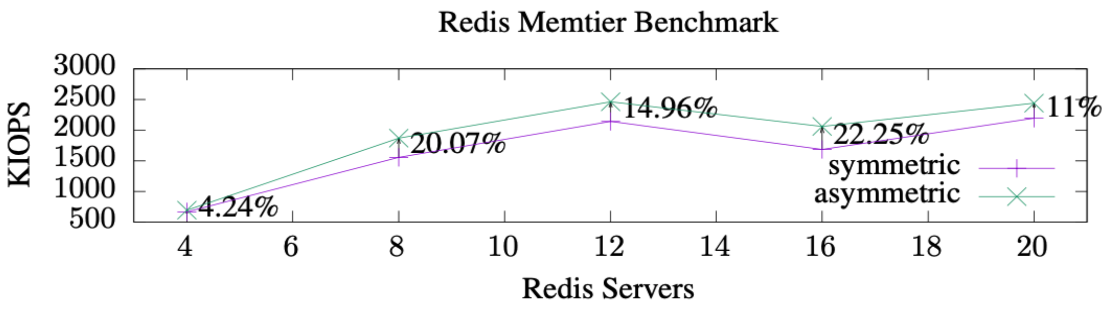
- Sampling L2 miss
  - Cross core communication on epoll event subsystem, socket queue and skb is expected.
  - More investigation is needed to reduce data sharing.
  - Same delta on _copy_to_iter confirm no logical sharing between kernel receiver and application context.

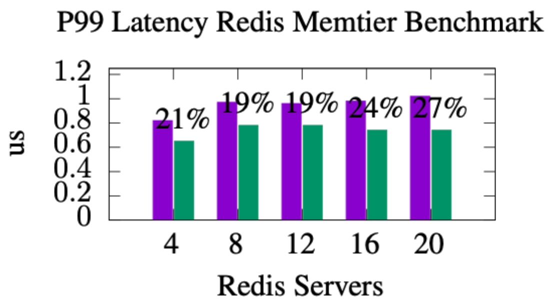


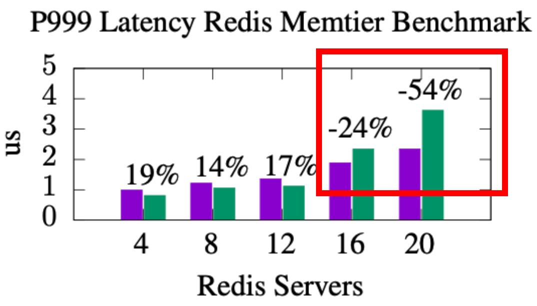
	bm0
tcp_ack	-53.1%
_copy_to_iter	1.5%
sock_poll	-23.8%
skb_release_data	-42.1%
tcp_recvmsg_locked	-12.1%
__check_object_size	-1.9%
tcp_queue_rcv	-72.7%
tcp_poll	-29.3%
tcp_check_space	-77.2%
skb_attempt_defer_free	9.4%
_raw_read_lock_irqsave	-103.7%
tcp_rcv_established	-65.1%
__list_del_entry_valid_or_report	-5.6%
__inet_lookup_established	-120.4%
do_epoll_wait	-97.6%
napi_pp_put_page	-58.6%
__lock_text_start	-72.2%
native_queued_spin_lock_slowpath	-569.9%

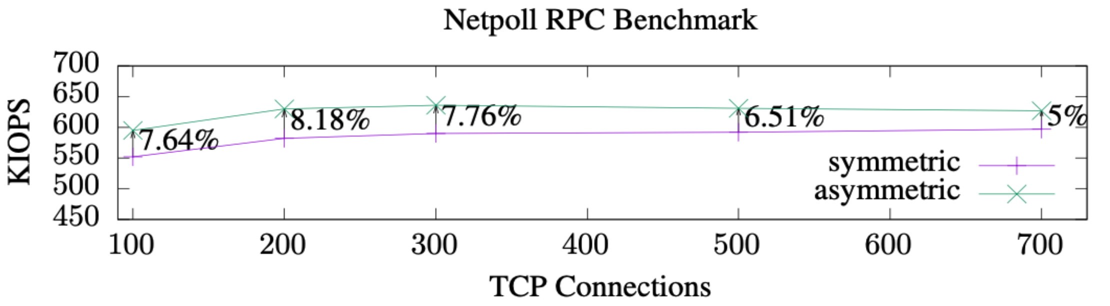
Table 3: CYCLE_ACTIVITY.STALLS_L2_MISS event sampling.

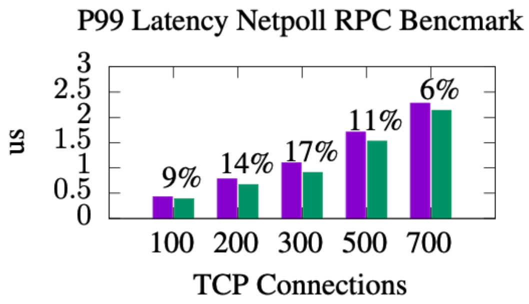
Redis and NetPoll Benchmark Results

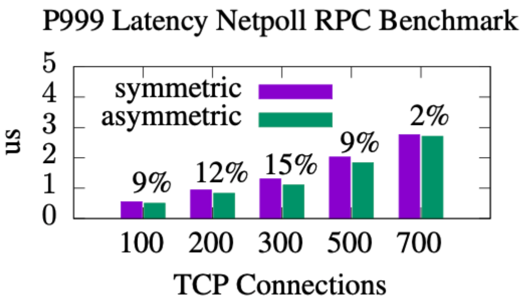












- Throughput increase 4-22% and 5-8%
- Latency decrease 14-27% and 2-17%.
- Increase of -24% and -54% P999 latency with higher Redis server count is due to less available application concurrency after isolation.

ANP: When and How

- Reserving CPUs reduce concurrency available to the application.
  - Not applicable under high concurrency requirements.
  - For ex: Show high tail latency, reduced in throughput etc.

- Not all servers in Datacenter are highly loaded.

- Dynamically identify the reserved CPUs:
  - 'si' CPU utilization.
  - Application Feedback loop: average latency and throughput
  - Kernel Feedback loop?

- Comment on CPU utilization:
  - In most scenarios we found its reduced
  - And in other it is proportional to the throughput increase
  - But rarely we have seen that it consumes more.

- io_uring
  - Netpoll like architecture can benefit from softirq integration with io_uring
  - Three cpu jumps : irq -> epoll,recv -> business
  - data copy to userspace inside softirq context

Conclusion

- Asymmetric processing provide better performance metrics:
  - Efficiency in frontend caches
  - Interference free execution

- Dynamic reservation of CPUs:
  - Concurrency consideration from the application.
  - A feedback loop is needed.

- Provides a good balance between throughtput, latency and cpu utilization.

- Work is needed:
  - Reduce data sharing between the two contexts.
  - Investigate other scenarios where qdisc queues are involved.

- More details on paper.
- Questions?
- Contact:
  - satish.kumar@bytedance.com
  - ByteDance, System Technology Engineering (STE)