A Way To Wakeup More Efficiently

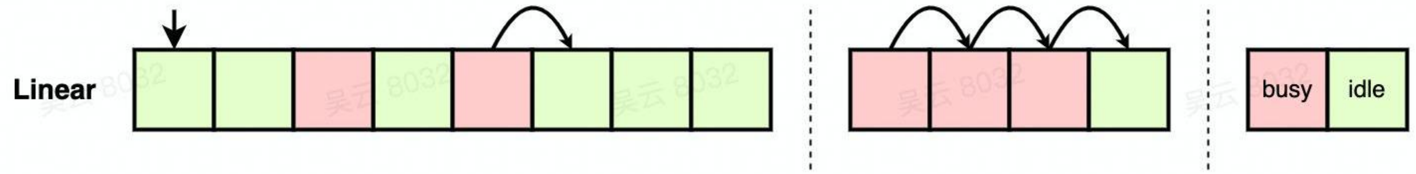
Abel Wu <wuyun.abel@bytedance.com>
Bytedance Infrastructure Kernel Team

Content

- Background
- Design
  - RFC
  - Cpuidle
  - Anti-NOHZ
  - Per-Core
  - Failover
- Benchmark

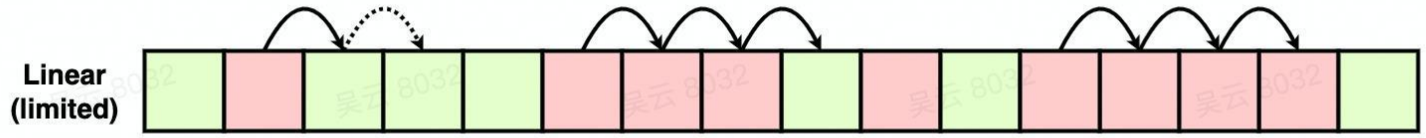
Background

- Wakeup plays an important role in performance
  - for sufficient use of cpu capacity
  - for better data locality
  - ...
- SIS now scans linearly which works well when:
  - under light pressure → not hard to find an idle cpu
  - with small LLCs → well bounded worst case



Background

- Trends in the real world
  - CSPs aim at TCO optimization → Co-location
  - Workloads are becoming more complicated and hungry
  - LLCs are getting bigger
- Scalability → Limiting scan depth is not enough
  - SIS_{UTIL,PROP} don't contribute to success rate
  - Would be good to scan more wisely

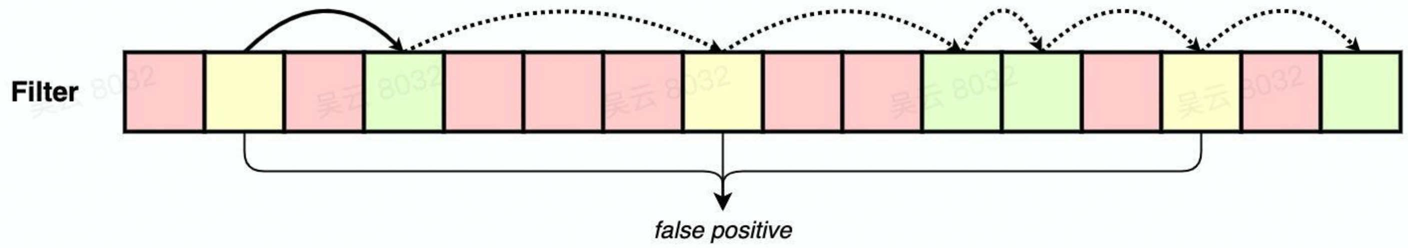


Design

- SIS strategy retained: aiming at better performance
  - Cache-hot idle cpus are given the most precedence, if none then...
  - Idle cores are preferred than sched-idle/idle cpus to maximize cpu utilization, or
  - Cache-hot non-idle cpus are last choice
- Better know in advance which cpus are idle
  - All cpus need to know the related information → shared or private?
  - The information from each cpu's sight → single data, array, ...?

Design - Filter

- The cpumask is selected to be the holder of information
  - Simple & straight-forward, good as a start
  - Stored in the LLC shared domain → Unified view from related cpus
  - Contain all the possibly idle cpus within each LLC
  - The key is the way it is maintained!



Design - Milestones

1 RFC — 2020.9

Set when entering tickless idle
Clear in 1st tick after leaving idle
At per-cpu granule

2 Cpuidle — 2020.11

Set when stop tick
Update at every tick when NOHZ

3 Anti-NOHZ — 2021.3

Set when entering idle, update at tick

4 Per-Core — 2022.6

Update at load balancing at SMT level

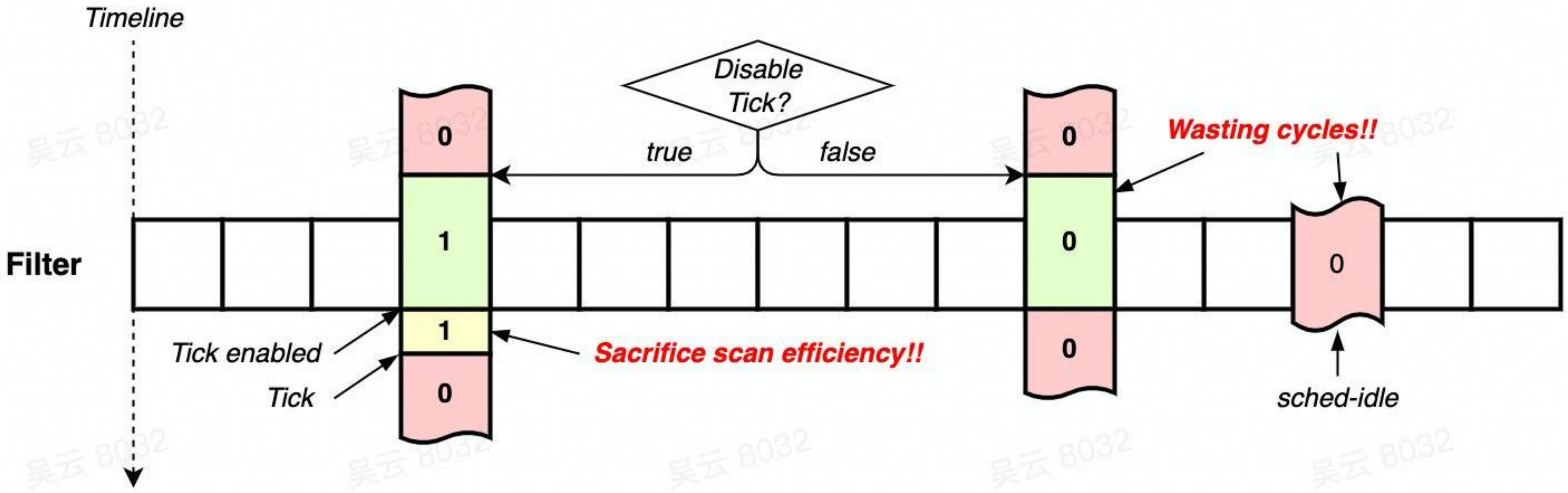
5 Failover — 2022.10

Update at SIS failure

Design

- RFC by Aubrey Li
  - Set the cpus to the filter when they enter idle with tick stopped
  - Cleared in the 1st tick after leaving idle → reduce cacheline bouncing
- Problems
  - The sched-idle cpus are not taken into consideration
  - Tick may not be stopped if cpus are likely to stay idle only for a short period
    - A tick can be long enough to waste lots of cpu cycles
  - The false positives are handled at ms-scale
  - Depend on NOHZ (with no strong reason)

Design - RFC



Timeline
Disable Tick?
true
false
0
1
1
0
0
0
0
Wasting cycles!!
0
Filter
Tick enabled
Tick
Sacrifice scan efficiency!!
sched-idle

Design

- Using the stop_tick signal...
  - Set to the filter when entering tickless idle unless cpuidle driver not available
    - Shortly idle cpus are not candidates
  - Updated at the tick to also consider sched-idle cpus

- Problems
  - ~~The sched-idle cpus are not taken into consideration~~
  - ~~Tick may not be stopped if cpus are likely to stay idle only for a short period~~
  - Prediction on idle length can be misleading
    - A tick can be long enough to waste lots of cpu cycles
  - The false positives are handled at ms-scale
  - Depend on NOHZ (with no strong reason)

Design

- Getting rid of NOHZ...
  - Set to the filter when entering idle
  - Updated at the tick
- Problems
  - ~~The sched-idle cpus are not taken into consideration~~
  - ~~Tick may not be stopped if cpus are likely to stay idle only for a short period~~
  - ~~Prediction on idle length can be misleading~~
  - ~~Depend on NOHZ (with no strong reason)~~
  - The false positives are handled at ms-scale

Design

- Per-CORE, rather than per-CPU...
  - Updated during load balancing at SMT level
    - Triggered when newly-idle or at tick-boundary
    - Periodic LB has admission control → reduce by O(k)
    - Re-use intermediate data during LB
  - Set one idle cpu of a core to the mask at a time → introduce true negatives
    - Cores not set in the filter must be fully busy
    - Avoid stacking tasks on same core

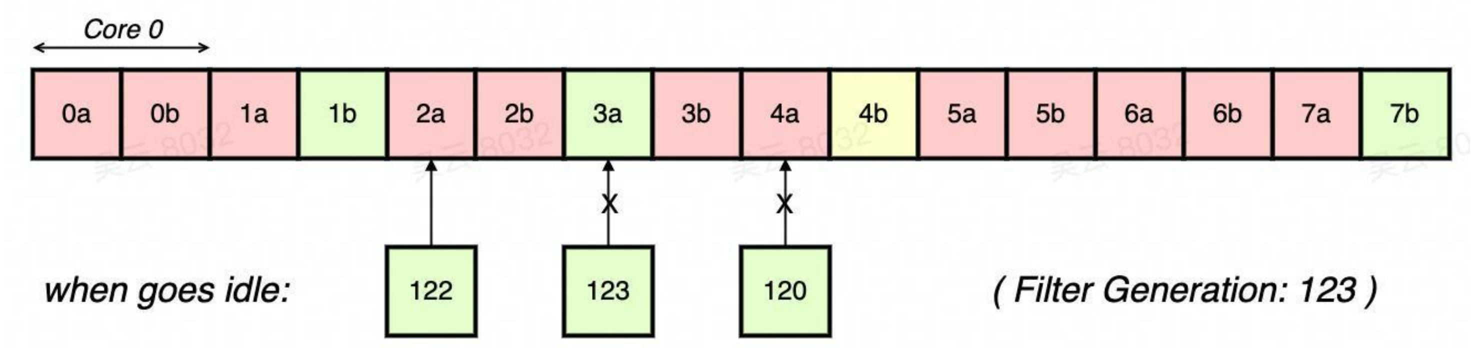
- Problems
  - Newly-idle LB can be skipped → true negatives
  - The true negatives may spend jiffies to go into filter
  - The false positives are handled at ms-scale

False Positive

- Clear periodically on tick/LB/...
  - Can be too late → Wakeup can be way more frequent
  - Tick-based proposals are just **NOT** work!!
- Clear on task enqueue
  - Multiple updates before being selected → Not efficient
  - Can be frequent → The filter is LLC-shared
- Clear when scan starts to fail
  - Adjust when necessary → Lazy & Ondemand
  - What's the strategy?

False Positive

- Reset the filter when full scan fails
  - How to know if a cpu is set in filter
    - Straightforward cpumask_test_cpu() → Costly due to LLC-shared
    - Cache locally in runqueue → Need to maintain coherence
  - Filter generation
    - Cache generation in runqueue when set cpus to the filter
    - Generation iterates when reset, expiring all caches



False Positive

- Reset the filter when full scan fails
  - How to know if a cpu is set in filter
    - Straightforward cpumask_test_cpu() → Costly due to LLC-shared
    - Cache locally in runqueue → Need to maintain coherence
  - Filter generation
    - Cache generation in runqueue when set cpus to the filter
    - Generation iterates when reset, expiring all caches
- Problems
  - There might be true negatives outside the filter
  - Reset can be triggered when the filter only contains one/two cpus
  - What about partial scan failure?

True Negative

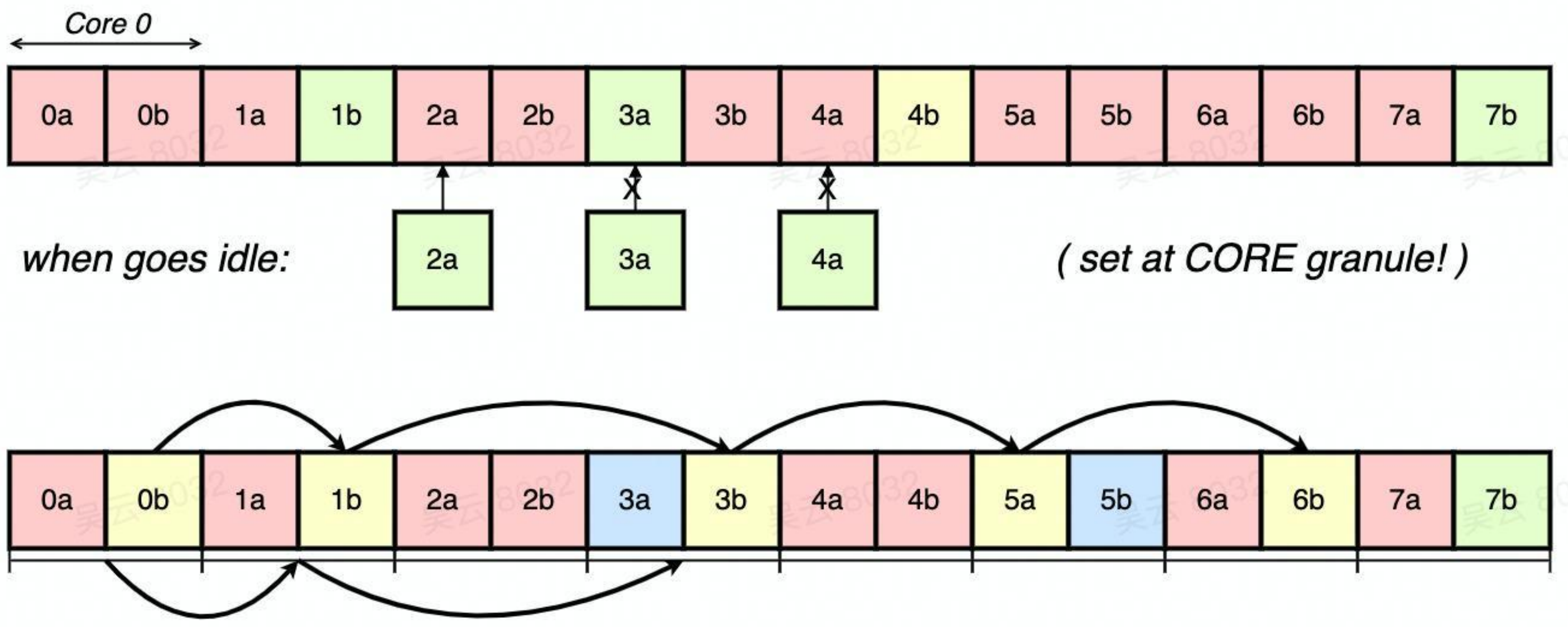
- Even more important than false positive
  - Missing idle cores in filter will lead to performance degradation
  - Containing false positives only affects scan efficiency
  - Should make addition more aggressive than deletion
- Can be treated as fallbacks of the false positives
  - Checked when the filter fails to provide any idle cpus
  - Not conflict with the idea of load spreading

Design

- Failover based on Per-Core updating
  - Updated at Core granule when entering idle
    - Only one cpu of a core can be set to the filter → reduce by O(k)
    - Avoid stacking tasks on same core
  - Strategy against scan failures
    - Scan verbosely on SMT siblings if fail-prone → true negatives handled
    - Fix on verbose scan → false positives handled
    - The inaccuracy is handled on demand

- Problems
  - ~~Update before task migration → State of cpus can change~~
  - ~~Newly-idle LB can be skipped → Wasting cpu cycles~~
  - ~~The false positives are handled at ms-scale~~

Design - Failover



Benchmark

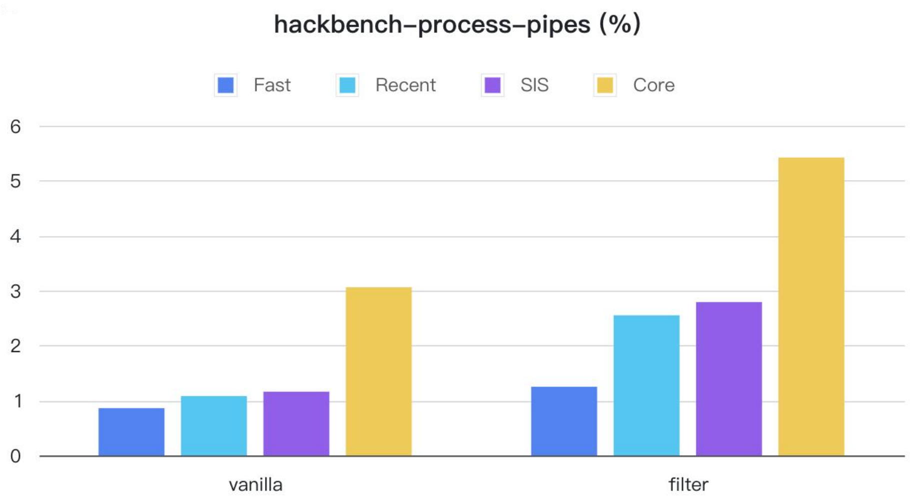
```
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
Address sizes:       46 bits physical, 48 bits virtual
CPU(s):              96
On-line CPU(s) list: 0-95
Thread(s) per core:  2
Core(s) per socket:  24
Socket(s):           2
NUMA node(s):        2
Vendor ID:           GenuineIntel
CPU family:          6
Model:               85
Model name:          Intel(R) Xeon(R) Platinum 8260 CPU @ 2.40GHz
```

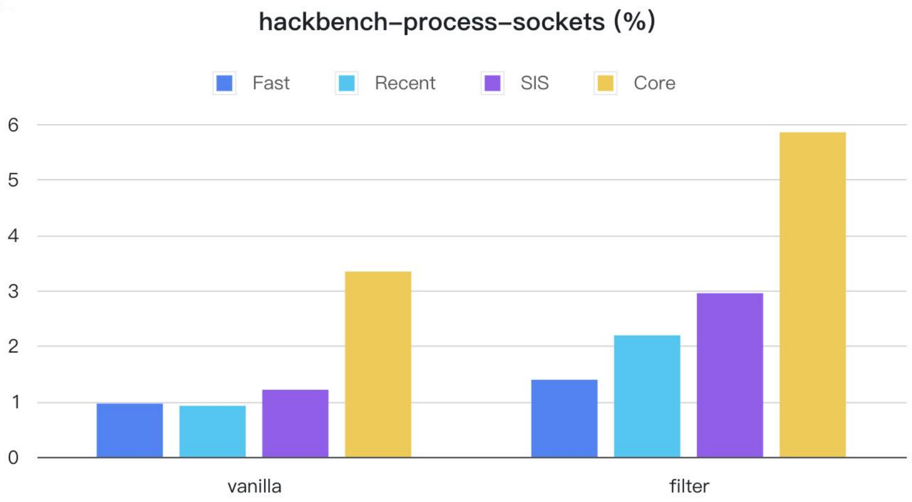
Benchmark

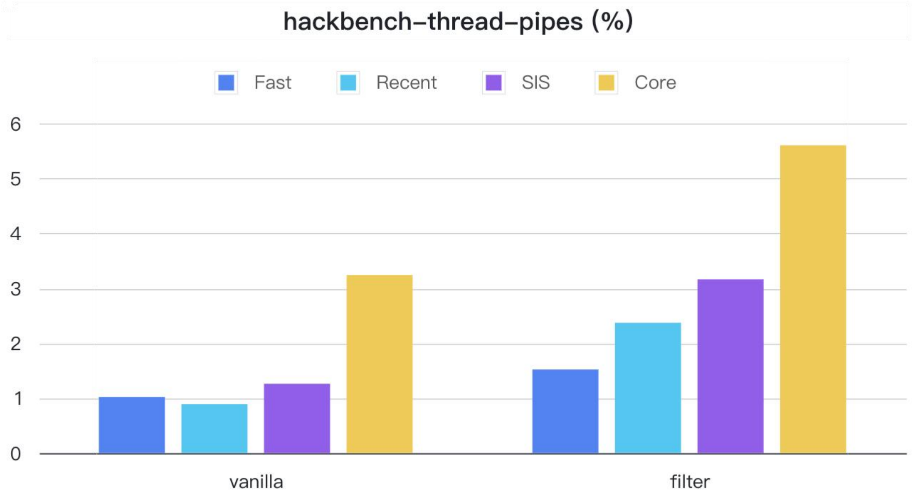
hackbench-process-pipes

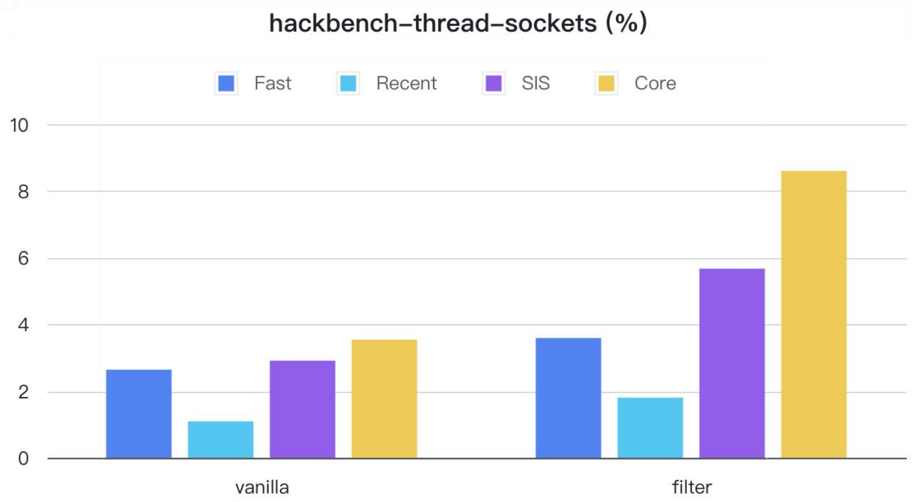
Busy	Amean	1	0.2550	(0.00%)	0.2420	(5.10%)	Neutral
	Amean	4	0.6120	(0.00%)	0.6100	(0.33%)	
	Amean	7	0.7863	(0.00%)	0.7673	* 2.42%*	
Overloaded	Amean	12	1.3557	(0.00%)	1.0857	(19.92%)	Great Win
	Amean	21	3.9463	(0.00%)	2.1170	* 46.36%*	
	Amean	30	6.7407	(0.00%)	3.2443	* 51.87%*	
	Amean	48	9.3920	(0.00%)	8.0733	(14.04%)	
	Amean	79	10.7680	(0.00%)	9.5433	* 11.37%*	
Saturated	Amean	110	13.5903	(0.00%)	12.4097	* 8.69%*	Win
	Amean	141	16.2363	(0.00%)	15.0823	* 7.11%*	
	Amean	172	19.1173	(0.00%)	17.8967	* 6.39%*	
	Amean	203	21.6250	(0.00%)	19.8717	* 8.11%*	
	Amean	234	24.7833	(0.00%)	22.9517	* 7.39%*	
	Amean	265	27.2227	(0.00%)	26.2753	(3.48%)	
	Amean	296	31.1530	(0.00%)	28.8437	* 7.41%*	

SIS Success Rate









netperf-udp

Hmean	send-64	210.32	(	0.00%)	210.70	(	0.18%)
Hmean	send-128	420.16	(	0.00%)	420.08	(	-0.02%)
Hmean	send-256	821.66	(	0.00%)	827.69	(	0.73%)
Hmean	send-1024	3142.47	(	0.00%)	3180.49	*	1.21%*
Hmean	send-2048	5937.27	(	0.00%)	6080.53	*	2.41%*
Hmean	send-3312	9218.16	(	0.00%)	9323.67	*	1.14%*
Hmean	send-4096	11161.95	(	0.00%)	11286.52	*	1.12%*
Hmean	send-8192	17981.63	(	0.00%)	17879.61	(	-0.57%)
Hmean	send-16384	28103.88	(	0.00%)	28228.77	(	0.44%)

netperf-tcp

Hmean	64	1199.11	(	0.00%)	1223.04	*	2.00%*
Hmean	128	2322.14	(	0.00%)	2344.40	(	0.96%)
Hmean	256	4270.41	(	0.00%)	4285.72	(	0.36%)
Hmean	1024	13129.62	(	0.00%)	13016.83	(	-0.86%)
Hmean	2048	20484.06	(	0.00%)	20723.21	(	1.17%)
Hmean	3312	24405.74	(	0.00%)	24636.75	*	0.95%*
Hmean	4096	26019.02	(	0.00%)	26365.89	*	1.33%*
Hmean	8192	30330.13	(	0.00%)	30815.83	*	1.60%*
Hmean	16384	34516.75	(	0.00%)	34596.67	(	0.23%)

tbench4 Throughput

Hmean	1	279.31	(0.00%)	281.48 *	0.78%*
Hmean	2	552.40	(0.00%)	574.04 *	3.92%*
Hmean	4	1111.10	(0.00%)	1136.80 *	2.31%*
Hmean	8	2147.94	(0.00%)	2259.07 *	5.17%*
Hmean	16	4259.90	(0.00%)	4451.94 *	4.51%*
Hmean	32	7107.08	(0.00%)	7183.24 *	1.07%*
Hmean	64	8649.92	(0.00%)	8801.72 *	1.75%*
Hmean	128	19352.16	(0.00%)	19420.56 *	0.35%*
Hmean	256	19183.01	(0.00%)	19591.02 *	2.13%*
Hmean	384	19039.79	(0.00%)	19503.20 *	2.43%*

Conclusion

- Hackbench
  - Neutral when LLC not loaded
  - Great improvement in loaded even saturated LLCs
  - Also benefit hit rate on prev/recent cpus
- Netperf / tbench4
  - The cost of maintaining filter seems acceptable
  - No regression under fast idling workloads

THANKS.

ByteDance 字节跳动