Squeezing Performance: Clickhouse@4GB on K8s

Benchmarking ClickHouse on Low-Memory Kubernetes Environments

By

Pranav Mehta & Shivji Kumar Jha

Safe Harbour Statement

The views, opinions, and conclusions presented in these slides are solely my own and do not reflect the official stance, policies, or perspectives of my employer or any affiliated organization. Any statements made are based on my personal experiences and research and should not be interpreted as official guidance or endorsement.

ABOUT US

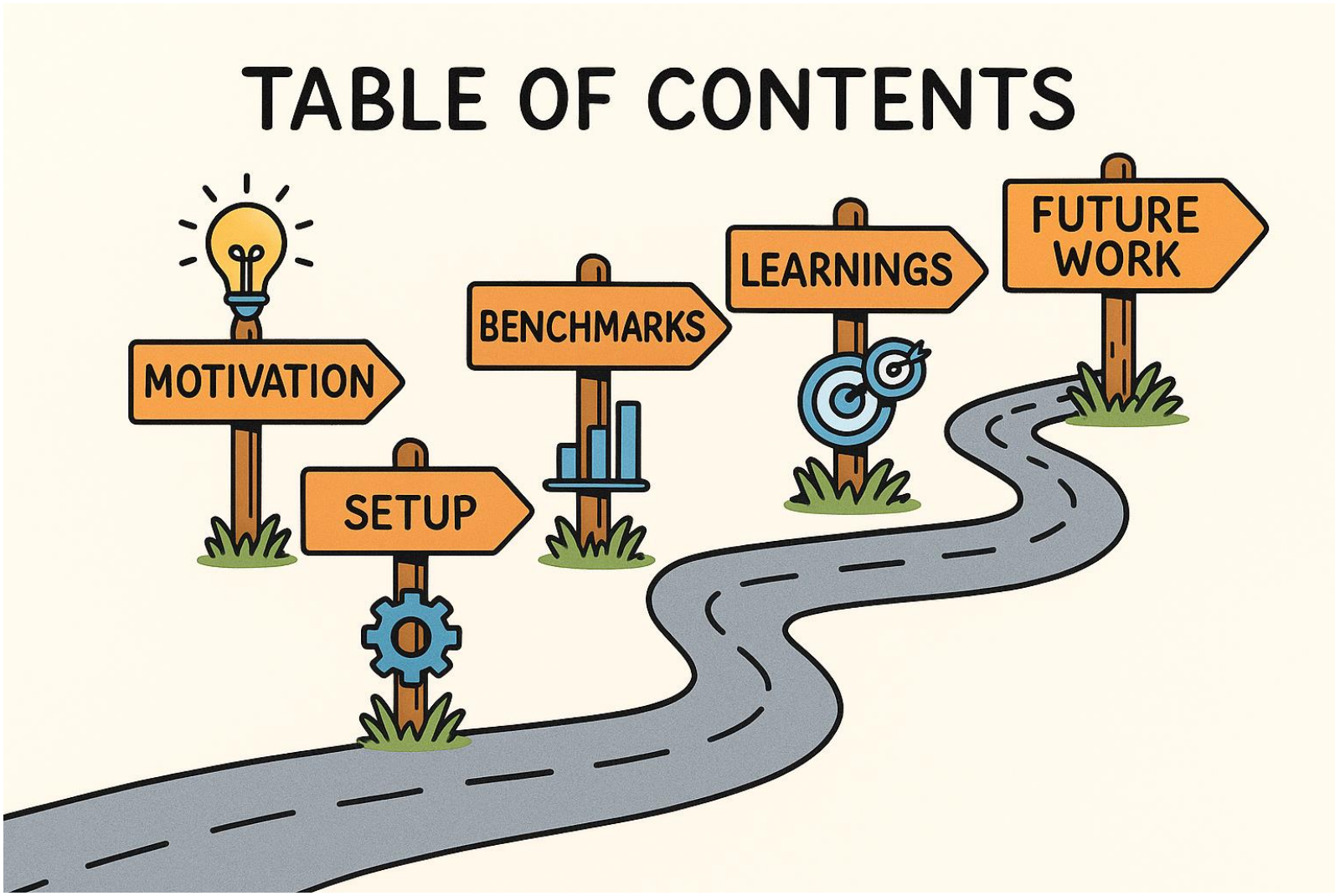
Shivji Kumar Jha
Staff Engineer/Lead
CPaaS Data Platform, Nutanix

- Interests: Databases, Streaming, Infra, App Backends
- Contributed code to MySQL, Pulsar, Clickhouse
- Excited about Open-Source Software & Communities
- Regular Speaker(28*), See tinyurl.com/shiv-slides
- Ping : linkedin.com/in/shivjijha/

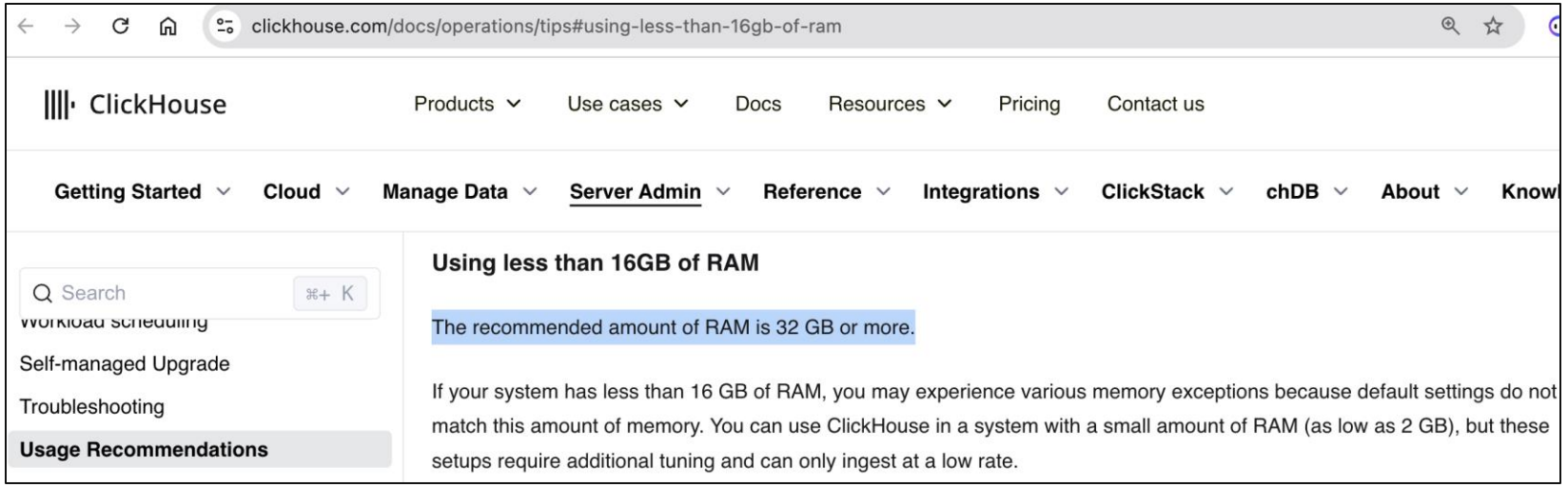
Pranav Mehta
Software Engineer (MTS4)
CPaaS Team, Nutanix

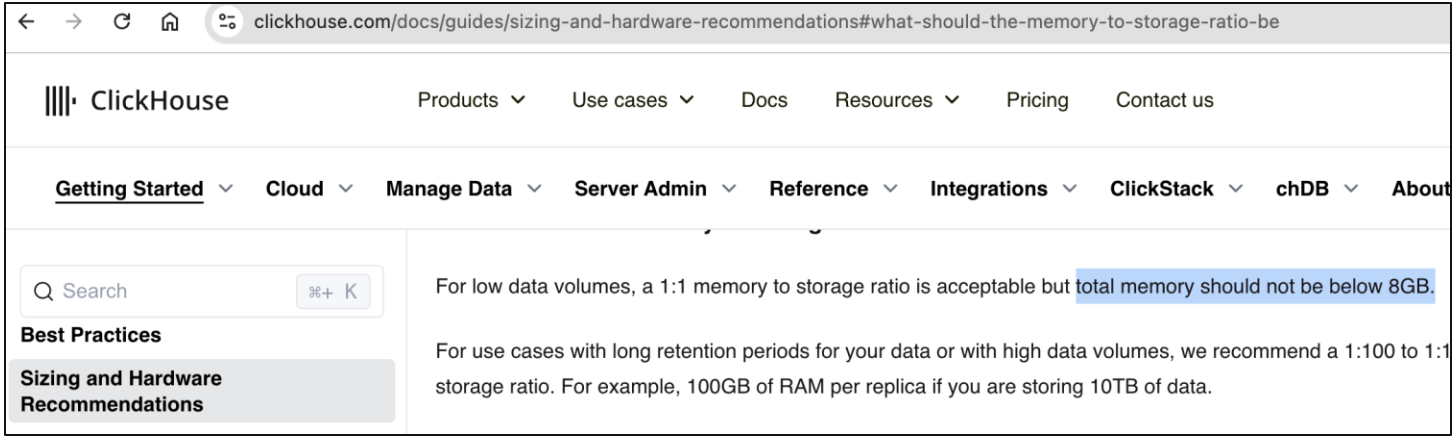
- Passionate About Distributed Systems
- Excited about:
    - Clickhouse, NATS, P2P, WASM, Linux
- Ping: linkedin.com/in/pranavmehta94/

TABLE OF CONTENTS



Docs

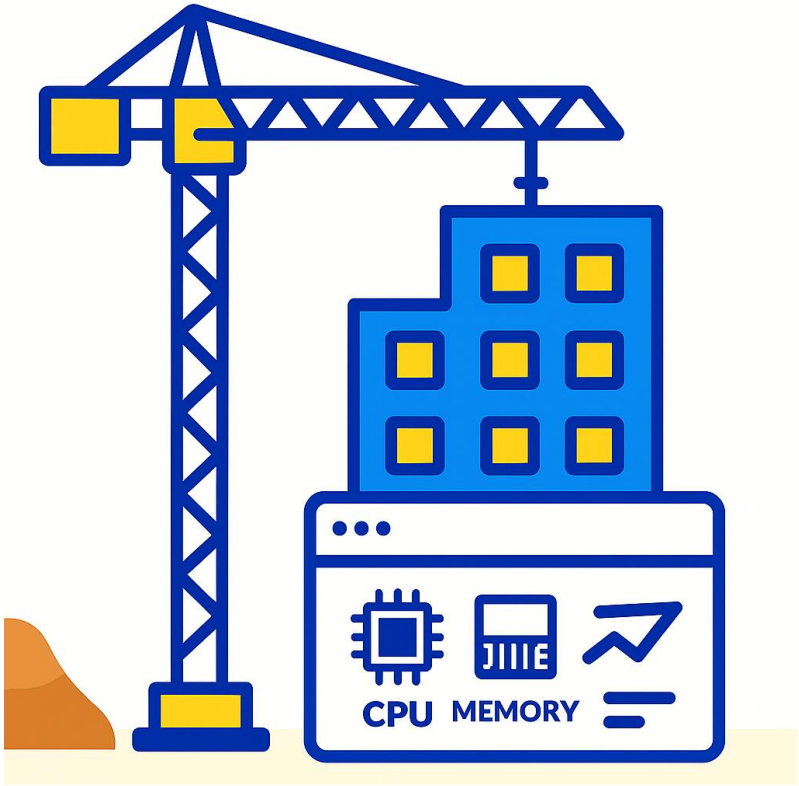

clickhouse.com/docs/operations/tips#using-less-than-16gb-of-ram
ClickHouse
Products
Use cases
Docs
Resources
Pricing
Contact us
Getting Started
Cloud
Manage Data
Server Admin
Reference
Integrations
ClickStack
chDB
About
Search
⌘+ K
Self-managed Upgrade
Troubleshooting
Usage Recommendations
Using less than 16GB of RAM
The recommended amount of RAM is 32 GB or more.
If your system has less than 16 GB of RAM, you may experience various memory exceptions because default settings do not match this amount of memory. You can use ClickHouse in a system with a small amount of RAM (as low as 2 GB), but these setups require additional tuning and can only ingest at a low rate.


clickhouse.com/docs/guides/sizing-and-hardware-recommendations#what-should-the-memory-to-storage-ratio-be
ClickHouse
Products
Use cases
Docs
Resources
Pricing
Contact us
Getting Started
Cloud
Manage Data
Server Admin
Reference
Integrations
ClickStack
chDB
About
Search
⌘+ K
Best Practices
Sizing and Hardware Recommendations
For low data volumes, a 1:1 memory to storage ratio is acceptable but total memory should not be below 8GB.
For use cases with long retention periods for your data or with high data volumes, we recommend a 1:100 to 1:1
storage ratio. For example, 100GB of RAM per replica if you are storing 10TB of data.


Why would you do that?
SYSADMIN
4 GB

Nutanix: Virtualization at Scale

- Customer hardware to private cloud platforms
- Collect time-series metrics for every VM, disk etc
- Lots of VMs at times
- At times fewer VMs

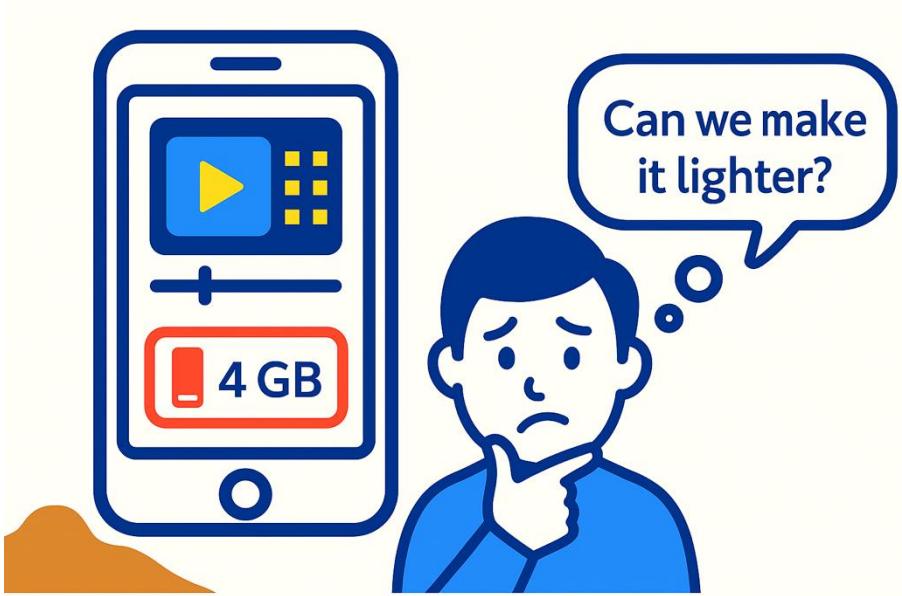


📦 ClickHouse Inside the Product

- Apps are shipped to customer environments
- ClickHouse is embedded in some products for fast, local analytics
- Must run reliably on **customer hardware**
- Even under **tight memory** and resource constraints

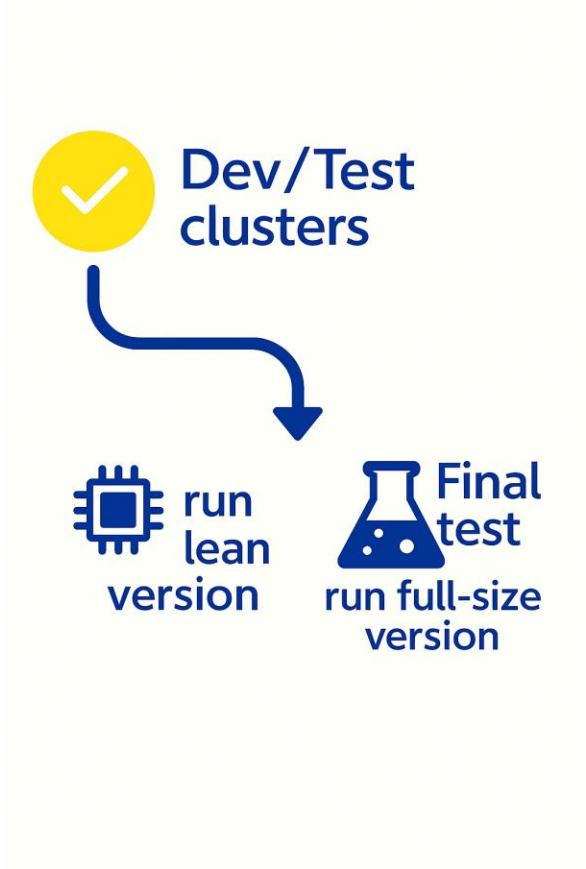
Small Customers, Smaller Machines

- In smaller deployments, resources are limited
- ClickHouse may get as little as **4GB RAM**
- Still needs to deliver **reliable performance** for critical analytics

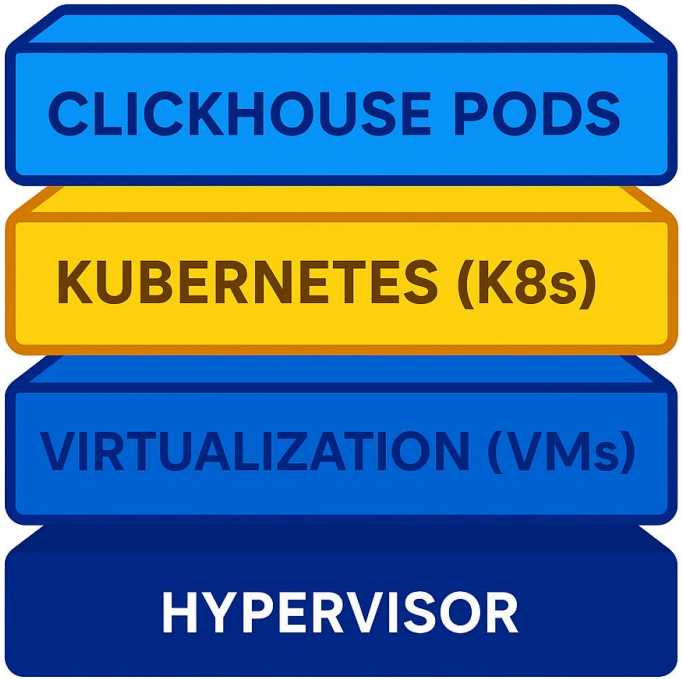


🎯 Bonus Use Case: Smarter Resource Usage

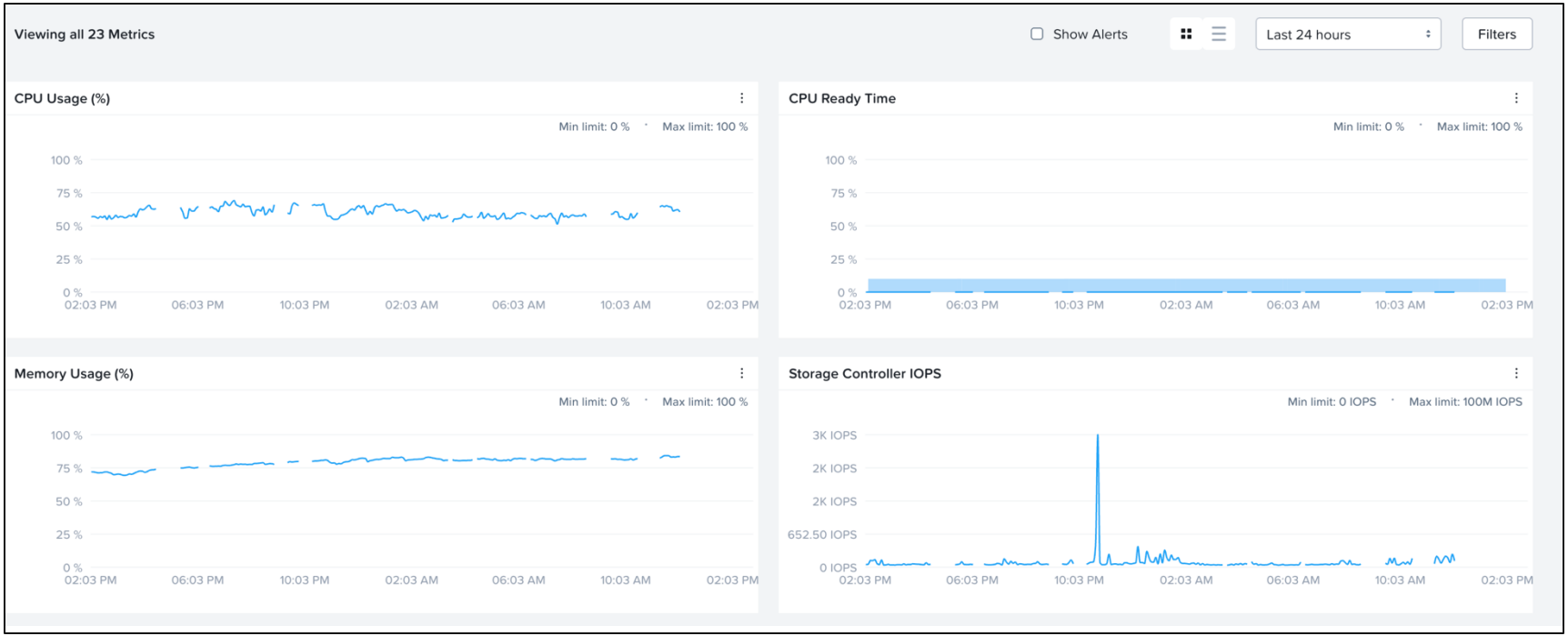
- Dev/Test clusters don’t need full-scale config
- Run lean ClickHouse (4GB?)
- Saves memory across **many environments**
- Final test/pre-prod?
  - full-size config for realistic performance



Clickhouse in a cloud-native world



Powered by Clickhouse



All Benchmarks are Lies!

At the minimum, do your own benchmarks

Setup

- Clickhouse version: 24.8.8.17
  - 1 shard
  - 1 replica
  - ReplicatedMergeTree
  - 4GB RAM, 1vCPU, 100 GB disk
- Clickhouse Keeper: 24.1.2
  - 3 nodes
  - 1 GB RAM, 0.33 cores CPU
- Kubernetes version: v1.29.6
- OS version: Rocky Linux 9.4
- Kernel Version: 5.14.0-427.31.1.el9_4.x86_64

Dataset

- Synthetic workload with **~5000 entities** per run
- For each entity, insert **(25-1000) metrics** every 5m
- MVs for (hourly) rollups
- Types of queries (examples)
  - SELECT values for VMs /5m over a 1h/1d/3m period
  - SELECT AVG(memory) for VMS for 1h ranges
  - SELECT VMs with > 20% utilization
  - COUNT VMs with 60% utilization
- Longevity duration: from 1hr to 12hr

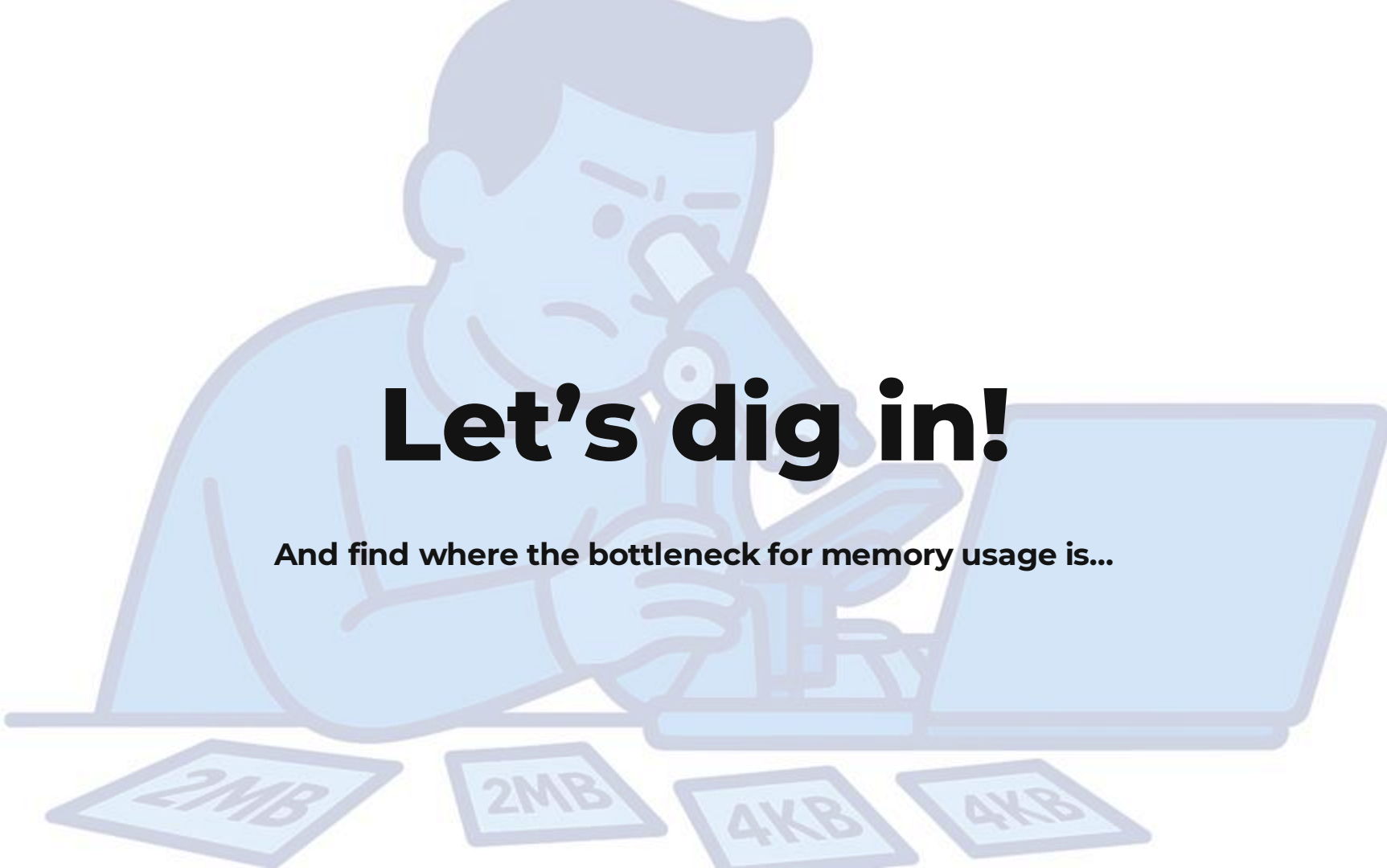
Progression of Load

	#Metrics /entity	duration	QPS	rows inserted	batch size	insert interval	insert time	total mem	mem	cpu	status
Test1	25	1hr	NA	125050	10K	5m	1848ms	4GB	~1.06GB	0.04m	success
Test2	100	1hr	NA	500200	10K	5m	6318ms	4GB	~1.20GB	0.14m	success
Test3	25	48hr	3	125050	10K	5m	2414ms	4GB	~2.1GB	0.25m	success
Test4	100	1hr	3	500200	10K	5m	9982ms	4GB	~2.25GB	0.4m	success
Test5	100	1hr	10	500200	10K	5m	8208ms	4GB	~2.28GB	0.28m	success
Test6	500	1hr	12	2501000	10K	5m	58457ms	4GB	~3.2GB	1	success
Test7	1000	1hr	12	5002000	10K	5m	155615ms	4GB	~3.8GB	1	failed
Test9	1000	3hr	10	5002000	5k	3m	226226ms	4GB	~3.8GB	1	failed
Test15	500	8hr	18	250100	10K	5m	58613	6GB	~5.2GB	1	failed

Failures with 1000 events per entity

- Failures seen with 1000 entities (1 hr)
- OOM kills on insert queries
- Merge tasks terminated
- Frequent OOM kills on SELECT queries
- Pod-level restarts querying query_log

Code: 241. DB::Exception:
Memory limit (total) exceeded



Let’s dig in!

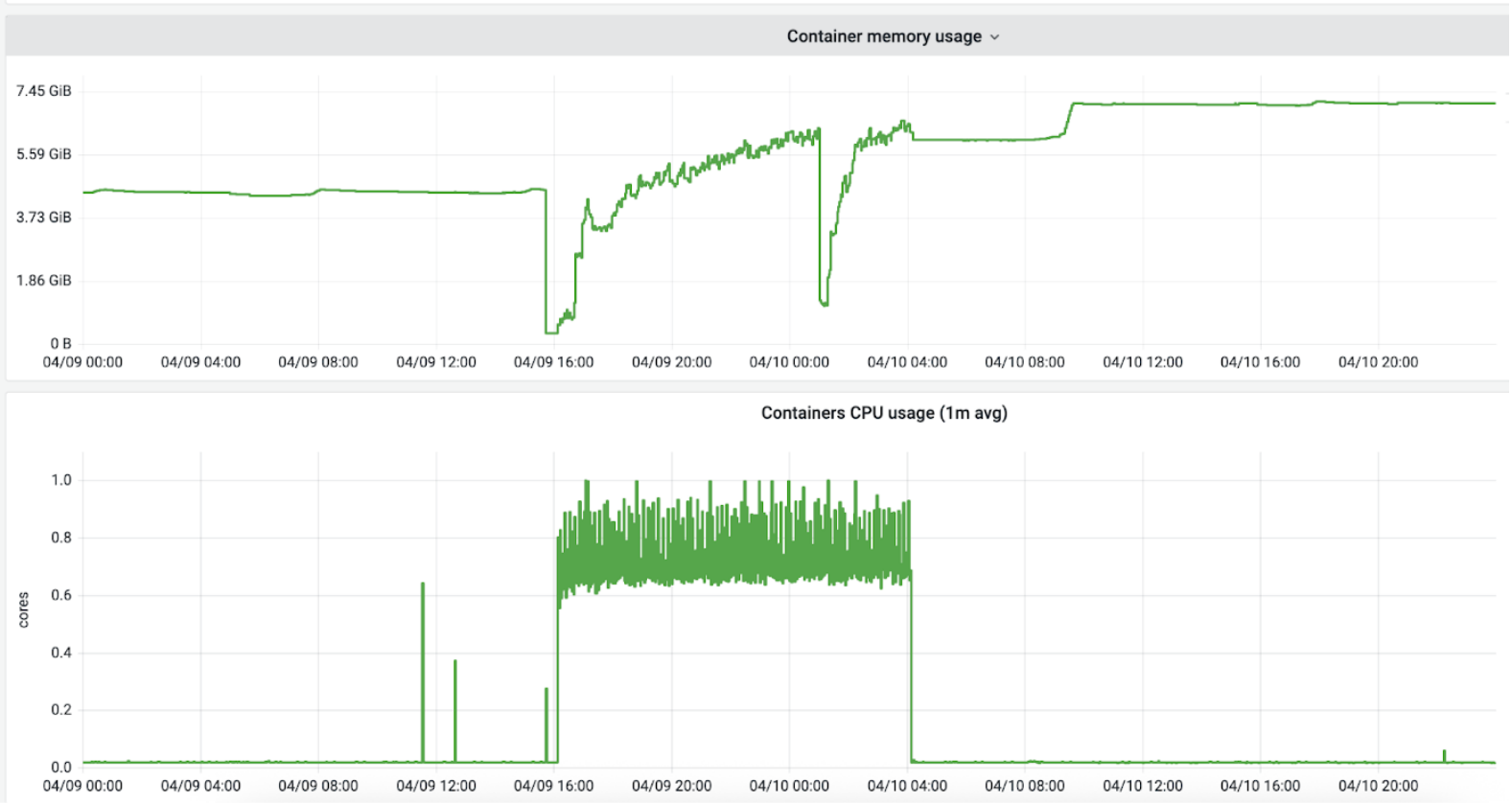
And find where the bottleneck for memory usage is...

What are our options?

- Tune Clickhouse configurations
- Memory allocations
- OS Kernel configurations
- And whatever else the graphs tell us!

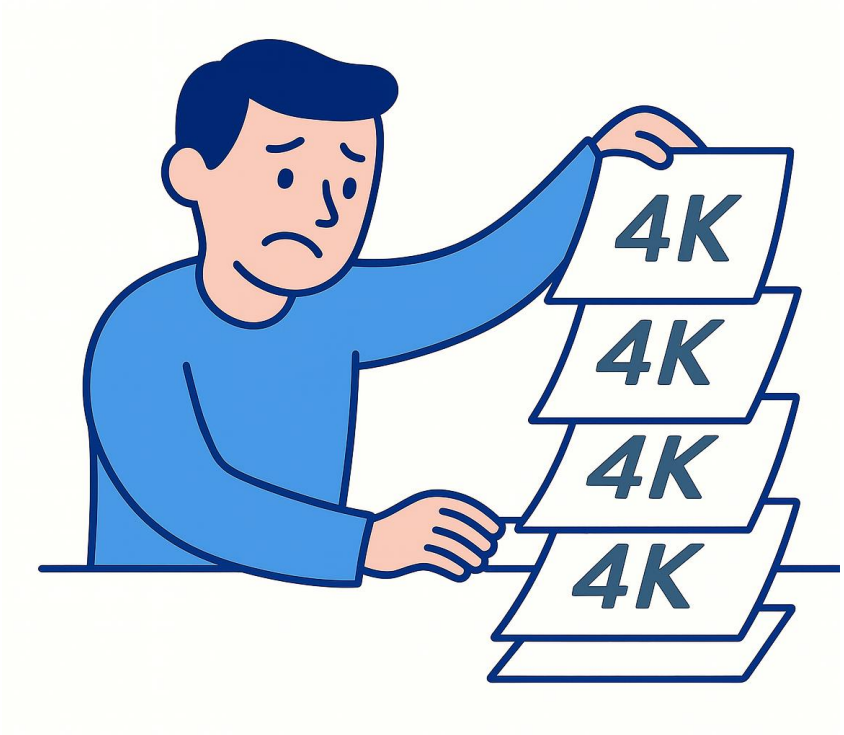
Points of Interest

Memory Analysis(1) – OS Page Cache



Memory Analysis(1) – OS Page Cache

- OS Page cache ~70% of memory
- Not reclaimed in Cgroup under stress (K8s ≤ 1.22)
- Kernel config needed for better reclamation
  - e.g. vm.swappiness



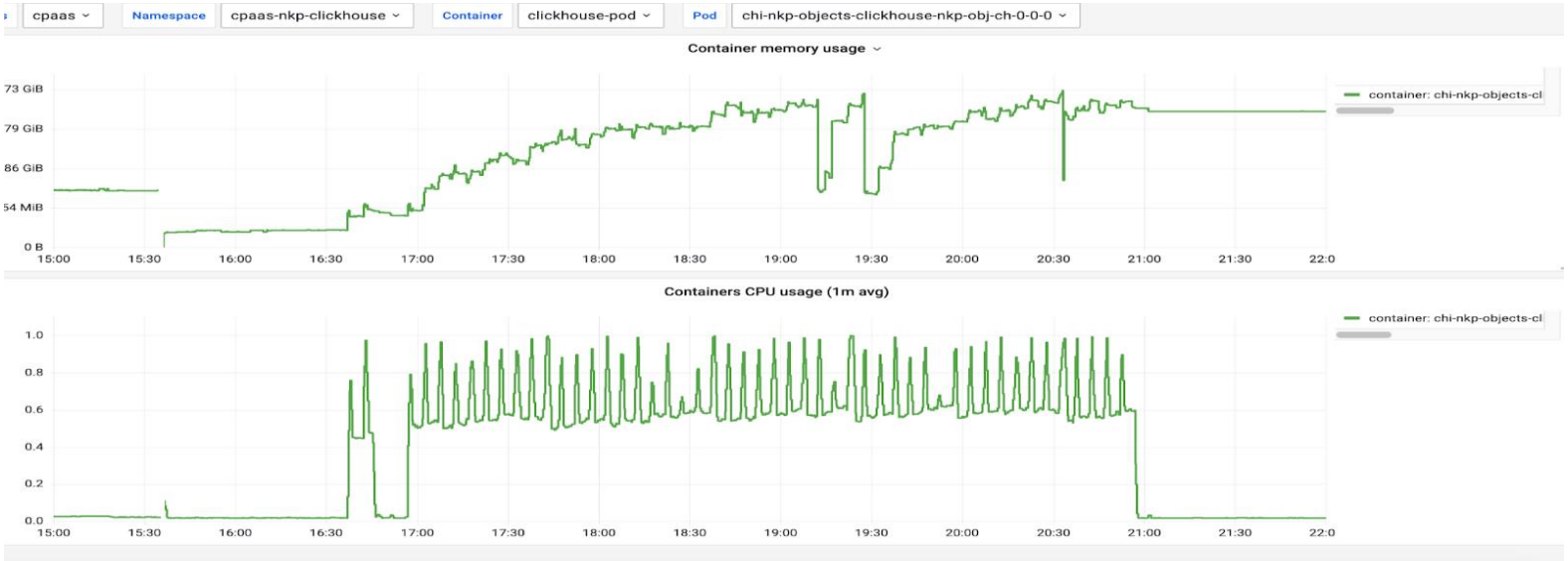
Referneces:

- https://biriukov.dev/docs/page-cache/4-page-cache-eviction-and-page-reclaim/
- https://phoenixnap.com/kb/swappiness
- https://www.schutzwerk.com/en/blog/linux-container-cgroups-01-intro/

Memory Analysis(2) – jemalloc

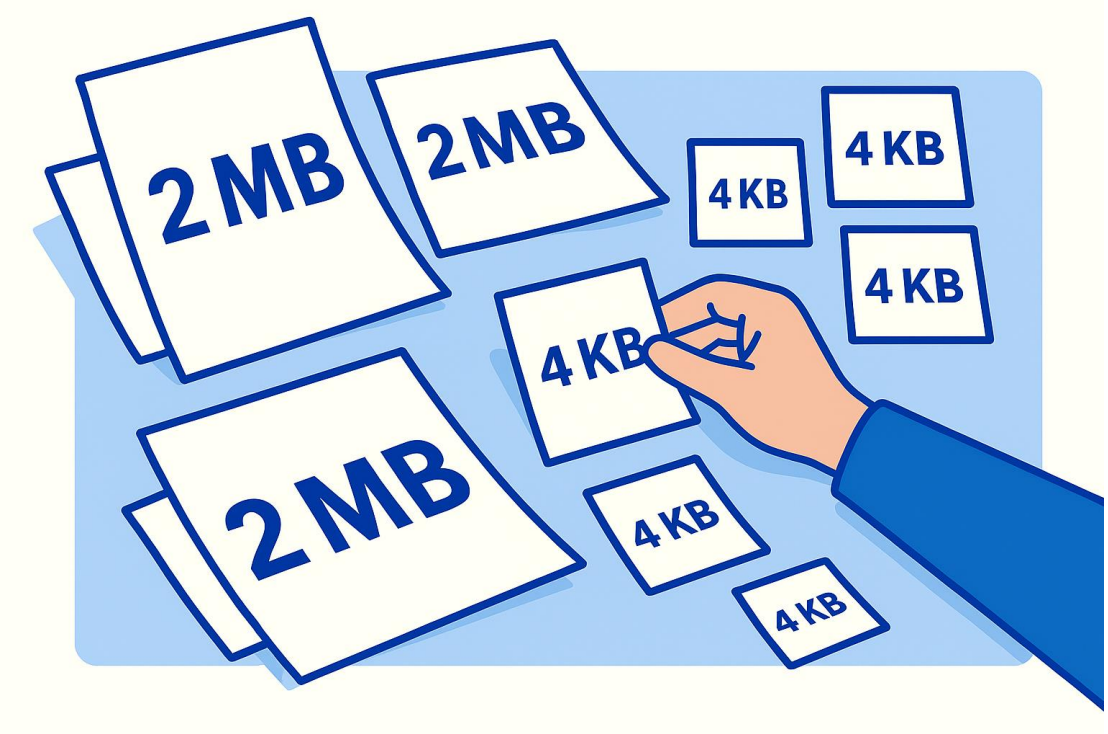
jemalloc designed to be a high-performance alternative to the default memory allocator

- jemalloc tuning reduces RSS
- OS Page cache still grew aggressively



https://people.freebsd.org/~jasone/jemalloc/bsdcan2006/jemalloc.pdf

Transparent Huge Pages (THP)



https://www.percona.com/blog/settling-the-myth-of-transparent-hugepages-for-databases/

Memory Analysis(3) – Transparent Huge Pages

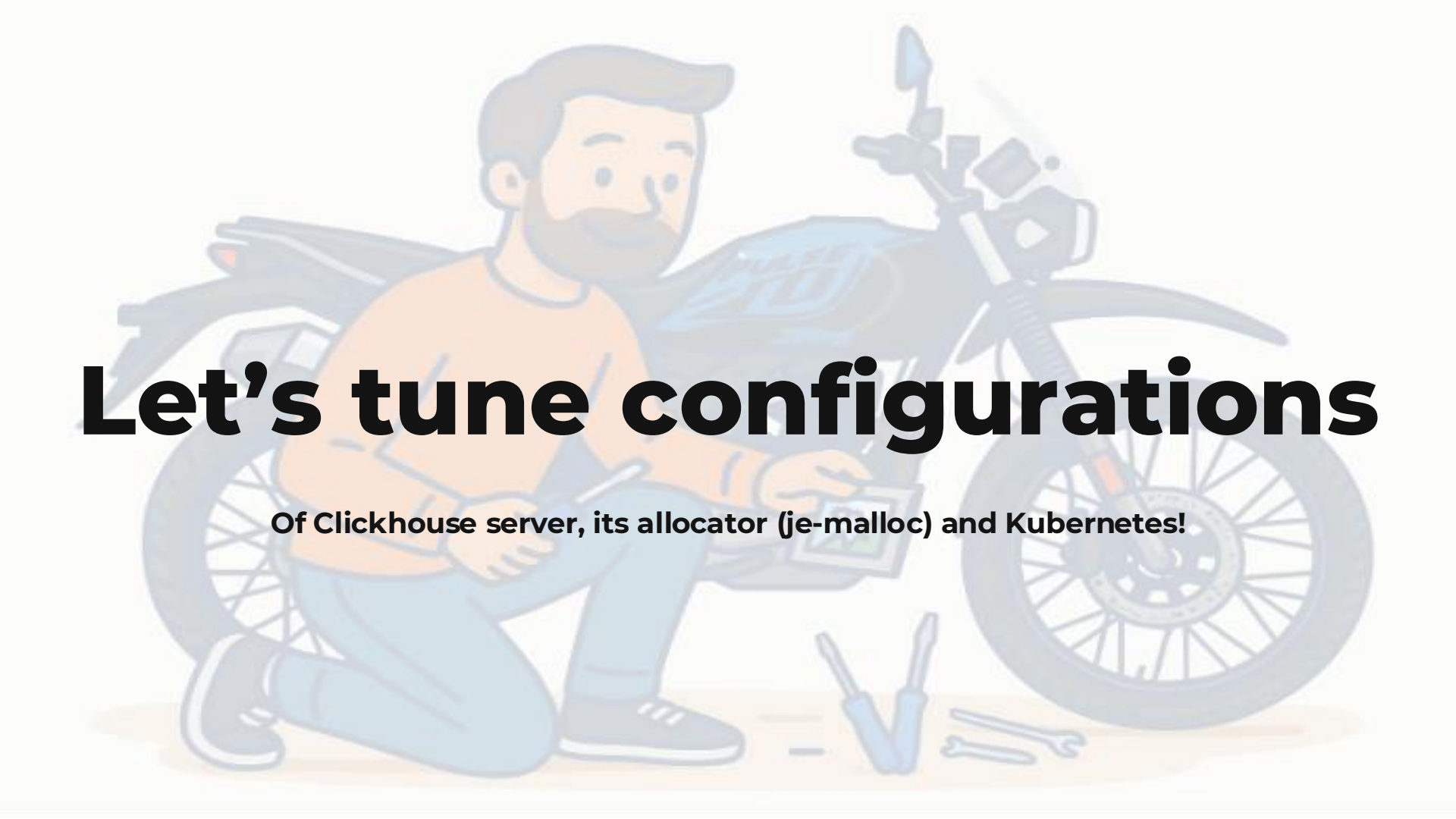
- Enabled THP causes memory spikes

https://www.kernel.org/doc/Documentation/vm/transhuge.txt

Memory Analysis(3) – Transparent Huge Pages

- Enabled THP causes memory spikes
- Disabling THP stabilizes usage

https://www.kernel.org/doc/Documentation/vm/transhuge.txt



Let’s tune configurations

Of Clickhouse server, its allocator (je-malloc) and Kubernetes!

Disable OS page cache?

Gotta use the saved RAM now

- Disabling Page Cache for Select Queries and Merges
  - **Stabilised** memory usage
  - But QPS dropped from 18 to 8 (less than half ☹)
- Use the leftover RAM for **clickhouse in-process caches**

Tuning – Clickhouse configurations

- Disable Log Tables
- Adjust asynchronous_metrics_update_period_s
- DIRECT_IO used to bypass OS page cache for Select queries and background merges
  - min_bytes_to_use_direct_io
  - min_merge_bytes_to_use_direct_io
  - Improved stability
- Tune Merges

```
<!-- reconfigure the main pool to limit the merges (those can create problems if the insert pressure is high) -->
<background_pool_size>2</background_pool_size>
<background_merges_mutations_concurrency_ratio>2</background_merges_mutations_concurrency_ratio>
<merge_tree>
  <merge_max_block_size>1024</merge_max_block_size>
  <max_bytes_to_merge_at_max_space_in_pool>1073741824</max_bytes_to_merge_at_max_space_in_pool> <!-- 1 GB max part-->
  <number_of_free_entries_in_pool_to_lower_max_size_of_merge>2</number_of_free_entries_in_pool_to_lower_max_size_of_merge>
  <number_of_free_entries_in_pool_to_execute_mutation>2</number_of_free_entries_in_pool_to_execute_mutation>
  <number_of_free_entries_in_pool_to_execute_optimize_entire_partition>2</number_of_free_entries_in_pool_to_execute_optimize_entire_partition>
</merge_tree>
```

Tuning – Clickhouse configurations

- Disable Log Tables
- Adjust asynchronous_metrics_update_period_s
- DIRECT_IO used to bypass OS page cache for Select queries and background merges
  - min_bytes_to_use_direct_io
  - min_merge_bytes_to_use_direct_io
  - Improved stability
- Tune Merges
- Tune Thread Pools

```
  <!-- shrink all pools to minimum-->
  <background_buffer_flush_schedule_pool_size>1</background_buffer_flush_schedule_pool_size>
  <background_merges_mutations_scheduling_policy>round_robin</background_merges_mutations_scheduling_policy>
  <background_move_pool_size>1</background_move_pool_size>
  <background_fetches_pool_size>1</background_fetches_pool_size>
  <background_common_pool_size>2</background_common_pool_size>
  <background_schedule_pool_size>8</background_schedule_pool_size>
  <background_message_broker_schedule_pool_size>1</background_message_broker_schedule_pool_size>
  <background_distributed_schedule_pool_size>1</background_distributed_schedule_pool_size>
  <tables_loader_foreground_pool_size>0</tables_loader_foreground_pool_size>
  <tables_loader_background_pool_size>0</tables_loader_background_pool_size>
</clickhouse>
```

Tuning – Clickhouse configurations

- Disable Log Tables
- Adjust asynchronous_metrics_update_period_s
- DIRECT_IO used to bypass OS page cache for Select queries and background merges
  - min_bytes_to_use_direct_io
  - min_merge_bytes_to_use_direct_io
  - Improved stability
- Tune Merges
- Tune Thread Pools
- Tune in-process cache
  - Mark Cache
  - Uncompressed Cache (1GB)

```
<!-- decrease the cache sizes -->
<mark_cache_size>268435456</mark_cache_size> <!-- 256 MB -->
<index_mark_cache_size>67108864</index_mark_cache_size> <!-- 64 MB -->
<uncompressed_cache_size>16777216</uncompressed_cache_size> <!-- 16 MB -->
```

Tuning - Jemalloc

- narenas:2
- background_thread:true
- tcache_max:4096
- dirty_decay_ms:5000
- muzzy_decay_ms:5000
- thp:never

Tuning Guide: https://github.com/ClickHouse/jemalloc2/blob/dev/TUNING.md

Tuning – Kubernetes

- Enable Memory QoS feature(Experimental) available in kubernetes version >=1.22 to trigger memory reclamation under pressure

Reference:

- https://kubernetes.io/blog/2021/11/26/qos-memory-resources/
- https://github.com/kubernetes/enhancements/tree/master/keps/sig-node/2570-memory-qos/#kep-2570-support-memory-qos-with-cgroups-v2
- https://github.com/kubernetes/enhancements/tree/master/keps/sig-node/2570-memory-qos/#latest-update-stalled

🎯 We now have it running, Yay!

	#Metrics /entity	duration	QPS	rows inserted	batch size	insert interval	insert time	total mem	mem	cpu	status
Test16	500	12hr	18	250100	10K	5m	59675	8GB	~7.02GB	1	success
Test17	500	12hr	8	250100	10K	5m	61034	8GB	~1.5GB	1	success
Test18	500	12hr	13	250100	10K	5m	57942	4GB	~2.7GB	1	success
Test19	500	12hr	12	250100	10K	5m	58932	4GB	~3.2 GB	1	success


LEARNINGS
4GB

Summarizing Recommendations

- **Disable THP (transparent huge pages)** to prevent unpredictable memory growth.
- **Tune jemalloc** to proactively purge arenas under pressure.
- **Disable page cache** selectively using min_bytes_to_use_direct_io, but balance with acceptable QPS loss.
- Use **uncompressed cache** to offset performance penalties from page cache deactivation.
- Adjust **Index granularity** to limit primary key bytes in memory
- Use Clickhouse settings from **this** article

Future Work

- Explore more kernel configuration for tuning page cache reclamations
- Explore user page cache feature introduced in Clickhouse(v25.3)
- Explore newly introduced cache for primary index(v24.12)

QUESTIONS ?

Liked this? Slides here:

tinyurl.com/shiv-slides

Pranav Mehta
linkedin.com/in/pranavmehta94/

Shivji Kumar Jha
linkedin.com/in/shivjijha/
tinyurl.com/shiv-talks

Thank You!