Microsoft

What I learned benchmarking Citus & Postgres performance with HammerDB

Naisila Puka

Software Engineer @ Microsoft

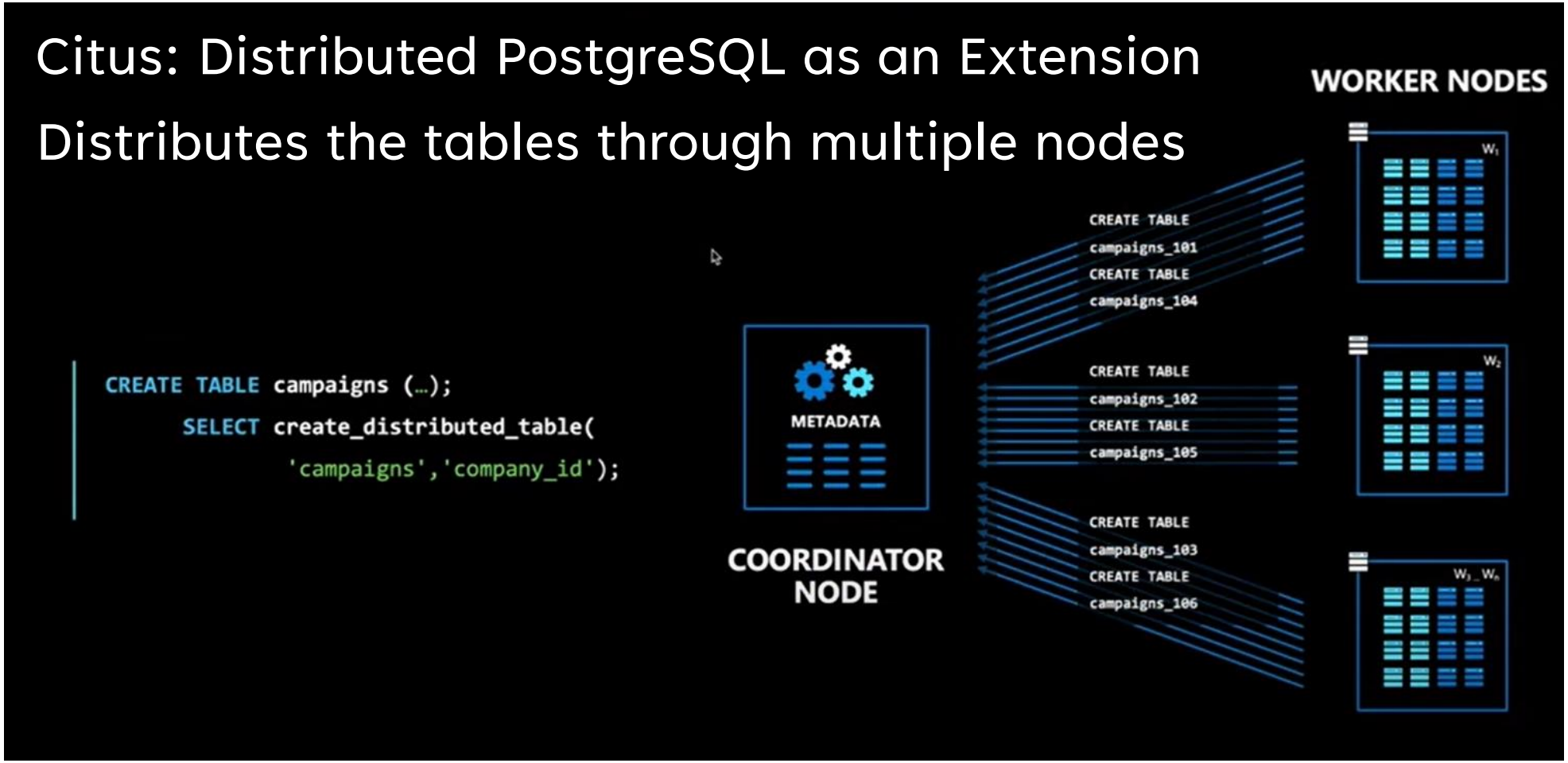
Outline

- Introduction: Citus and HammerDB
- Configuring the benchmark and the cluster
- Interpreting benchmark results
- Tweaking parameters based on results

What is Citus?

Citus: Distributed PostgreSQL as an Extension

Distributes the tables through multiple nodes



OLTP and Citus

- OLTP: common workload category for distributed databases
- Online Transaction Processing: lots of small, short-running transactions
- Citus is good at OLTP, routes each query to proper worker node.
- HammerDB helps us measure that performance!

HammerDB TPROC-C

- HammerDB: open-source benchmarking suite for databases
- TPROC-C full benchmark suite for OLTP
- Version 4.4 support for Citus: pg_citus_compat:
- Before running the transactions, all tables in HammerDB are distributed with `create_distributed_table`
- NOPM (new orders per minute) – the higher the better

Let’s start, and learn along the way

- What do we want?
  Measure performance, achieve HIGH NOPM
- Where will the benchmark run?
  Azure CosmosDB for PostgreSQL: Citus’s home on Azure!
- How to start?
  With a powerful Citus cluster, and work from there

Coordinator: Standard_D32ds_v5, 512 GB
Worker: Standard_D16ds_v5, 2048 GB
Number of worker nodes: 10

10*16+32

Choosing benchmark parameters

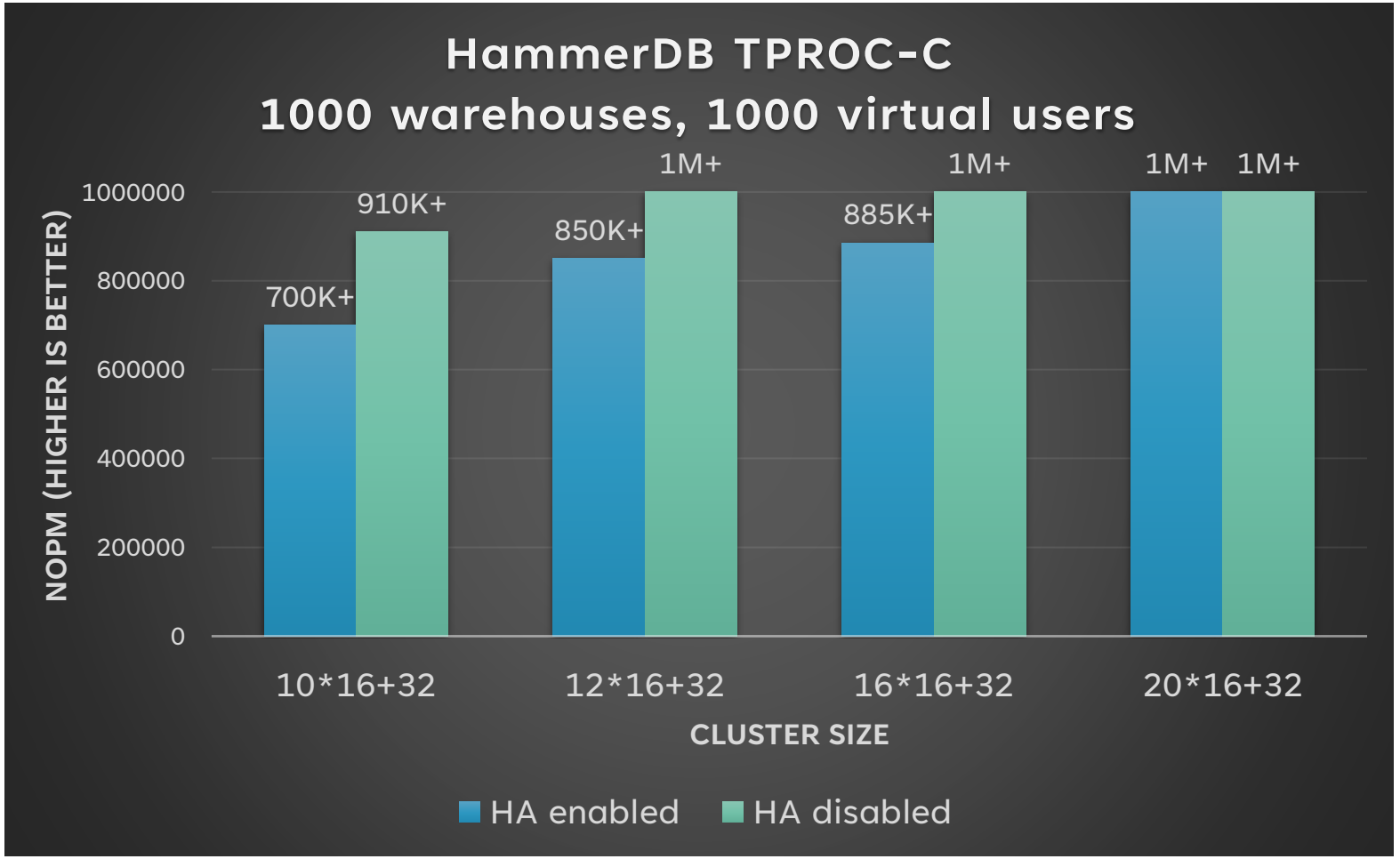
- TPROC-C is based on TPC-C specification
  - Fulfil orders from customers to supply products from a company.
  - The company keeps its stock in warehouses.
  - Warehouse size ~ 100MB
  - Larger schema ⇔ More warehouses
- A powerful Citus cluster (10+ 16-core workers) can process the increased level of transactions that a larger schema comes with.

Choosing benchmark parameters

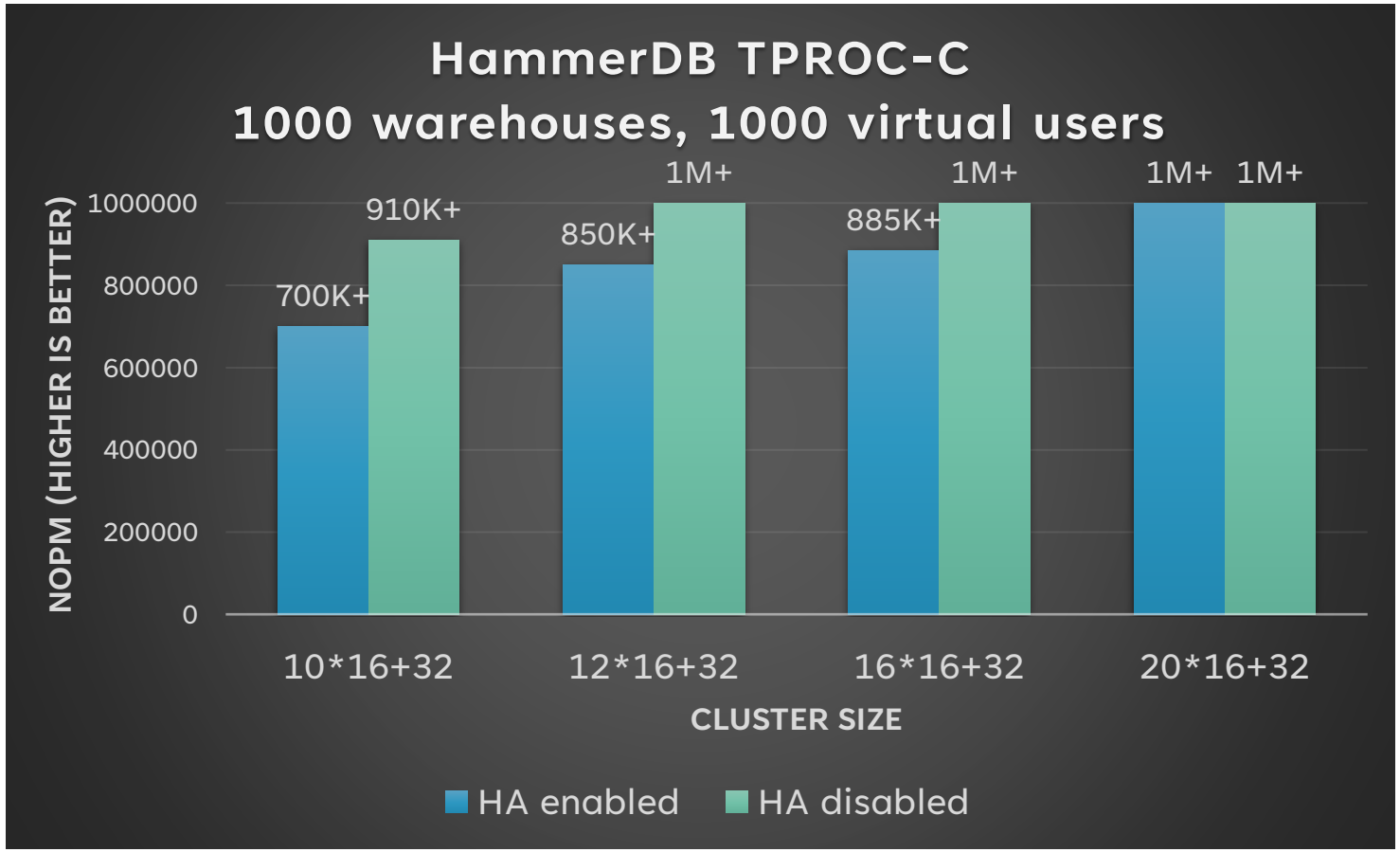
- Virtual users: OS threads, concurrent connections to the database.
- At least one warehouse per virtual user to avoid lock contention
- Azure CosmosDB For PostgreSQL has a limit of 1000 client connections.
- So, we utilize all we have at hand, and pick a minimum number of warehouses needed:

1000 virtual users, 1000 warehouses (~100GB)

Interpreting results

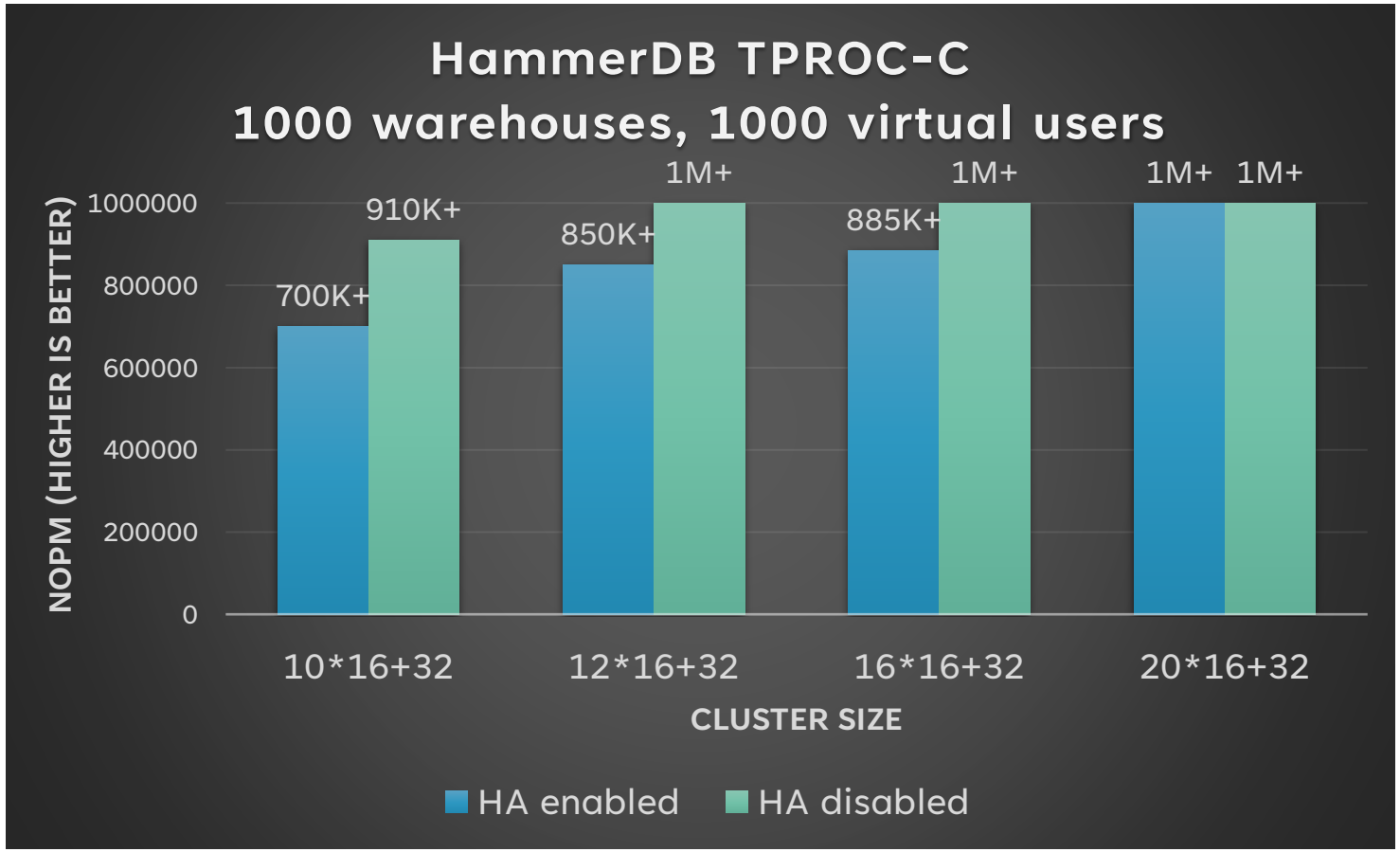


Interpreting results



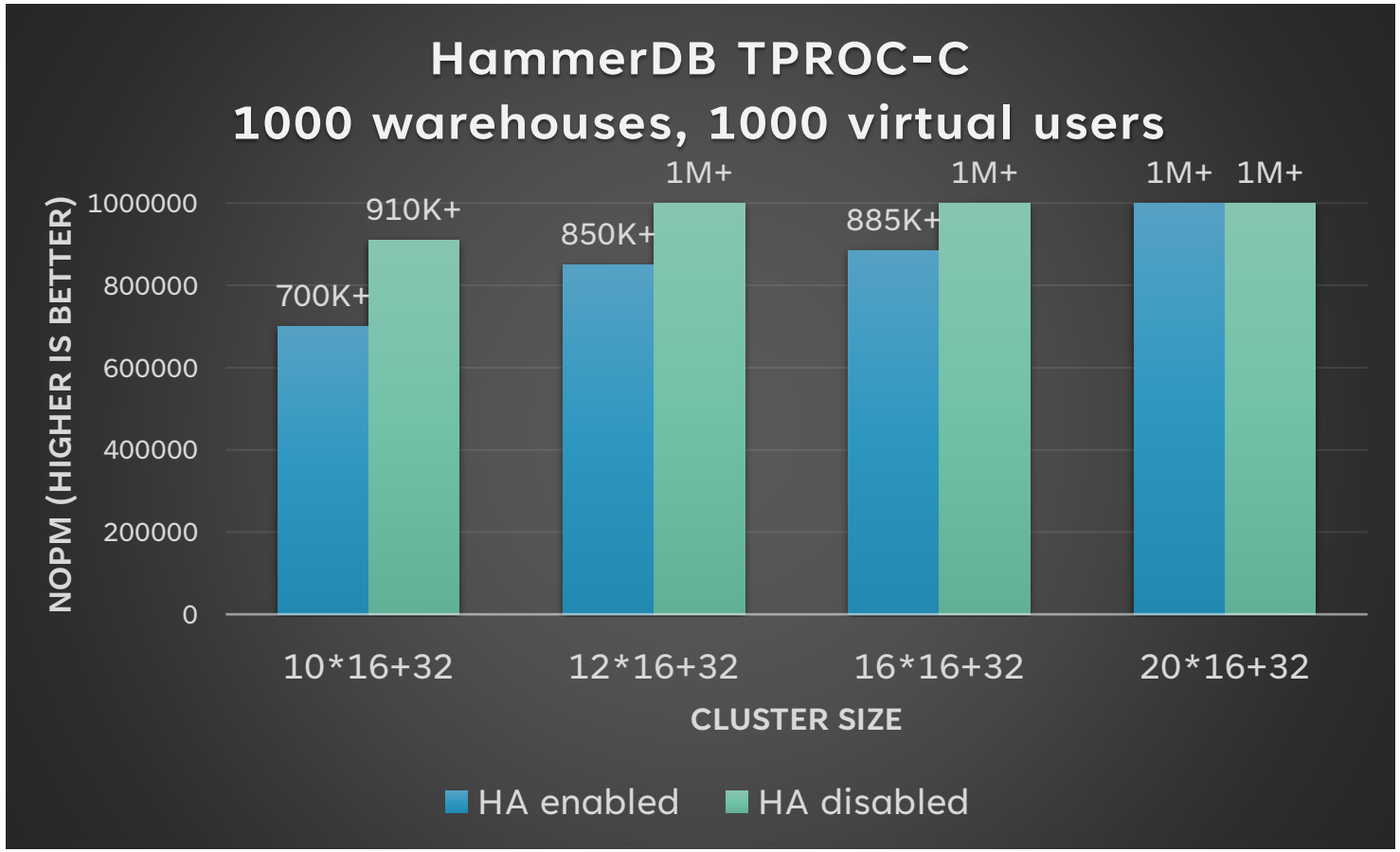
- Lowest number:
  - 700K NOPM
  - Still quite a good result
- Minimum cluster size for 1M+
  - 12 workers
  - 224 vcpus total

Interpreting results



- HA enabled: overhead
- Extra latency at the end of each transaction: make sure the backup server has the write changes.
- HA overhead decreases as we increase the number of workers/cores:
  - 10w – 23% penalty
  - 12w – 16% penalty
  - 16w – 13% penalty
  - 20w – <1% penalty

Interpreting results



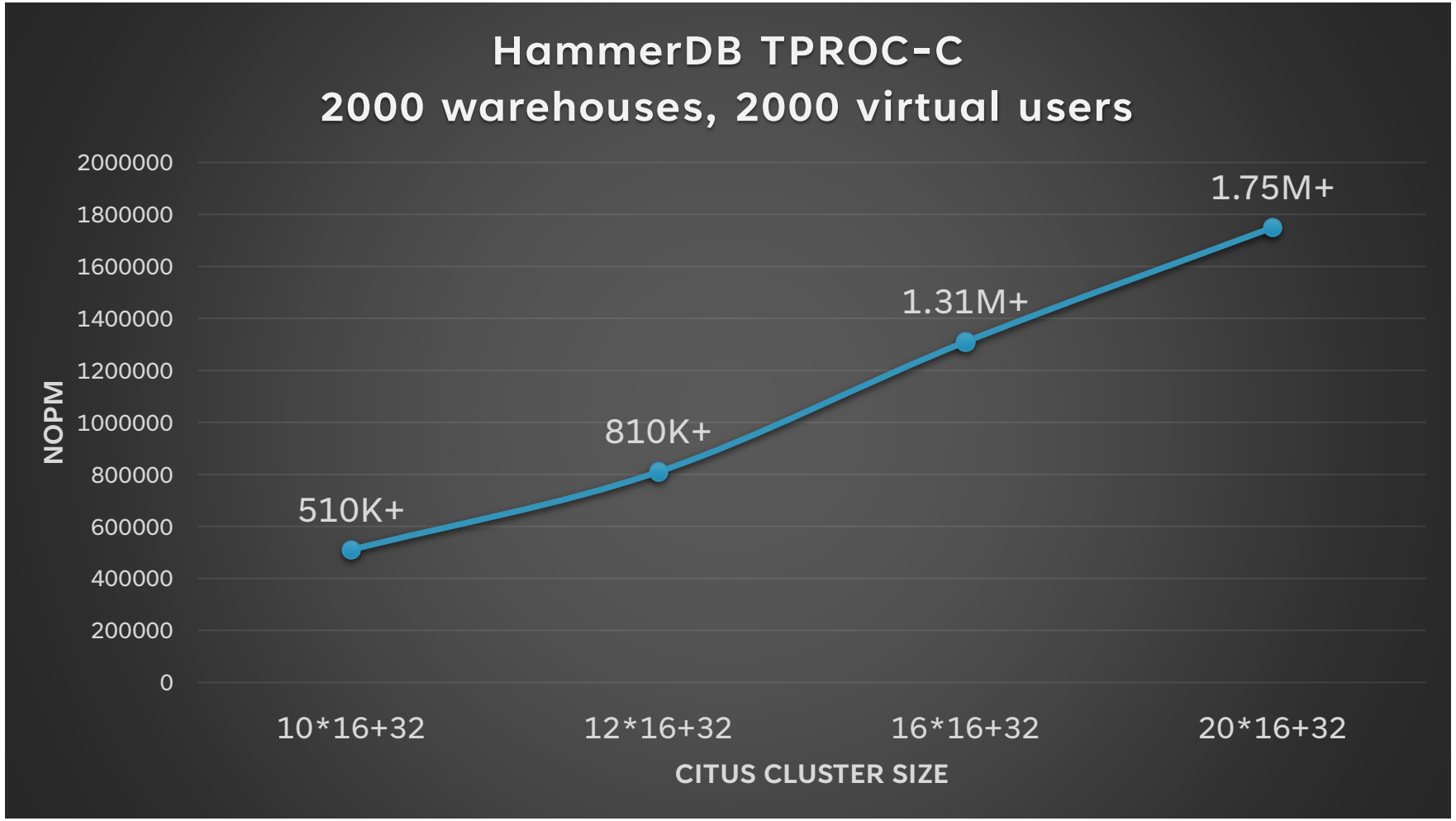
- NOPM improving from 10 to 12 workers
- NOPM is the same from 12 to 20 workers
- The reason is that 12 workers is enough number of cores to handle 1000 connections.
- ***To utilize more cores we need more virtual users***

Increasing concurrent connections

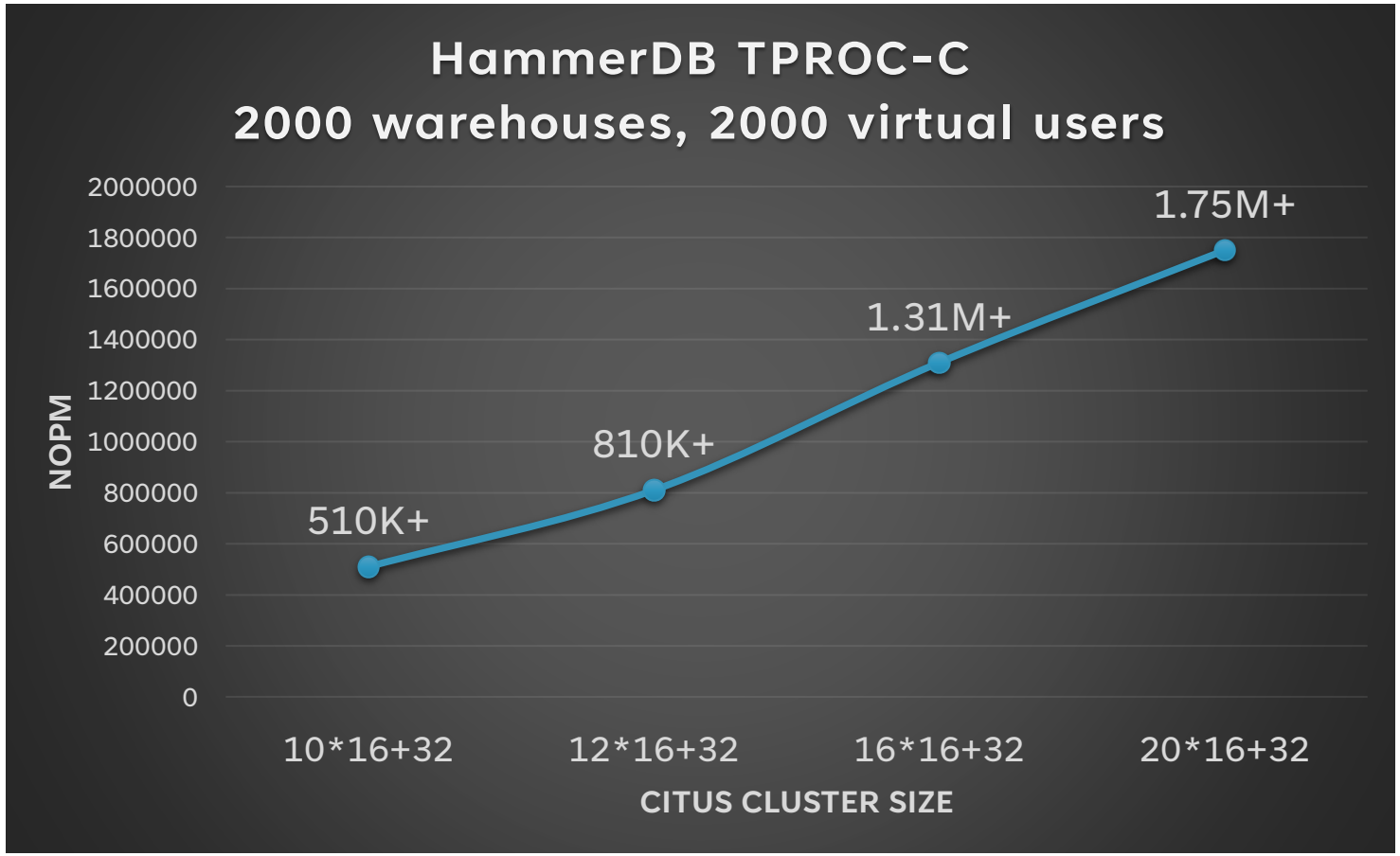
- We need to keep the powerful Citus cluster constantly busy
- Thoroughly utilize cores to maximize NOPM throughput
- As mentioned earlier, Azure CosmosDB For PostgreSQL has a limit of 1000 client connections.
- To change it, simply reach out to Azure support for an increase in the maximum number of client connections
- Let's see some exciting results for

2000 virtual users, 2000 warehouses (~200GB)

Interpreting results

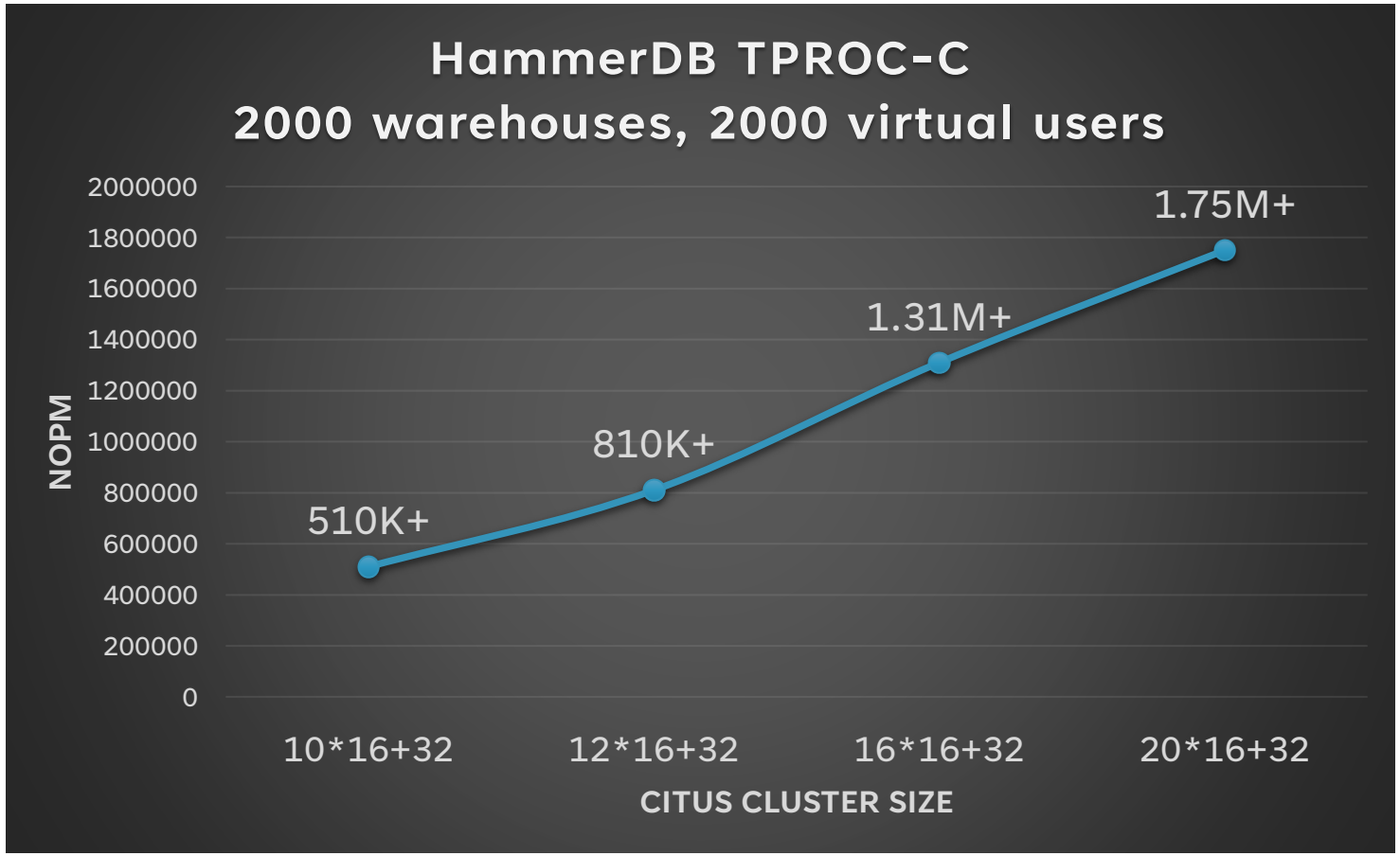


Interpreting results



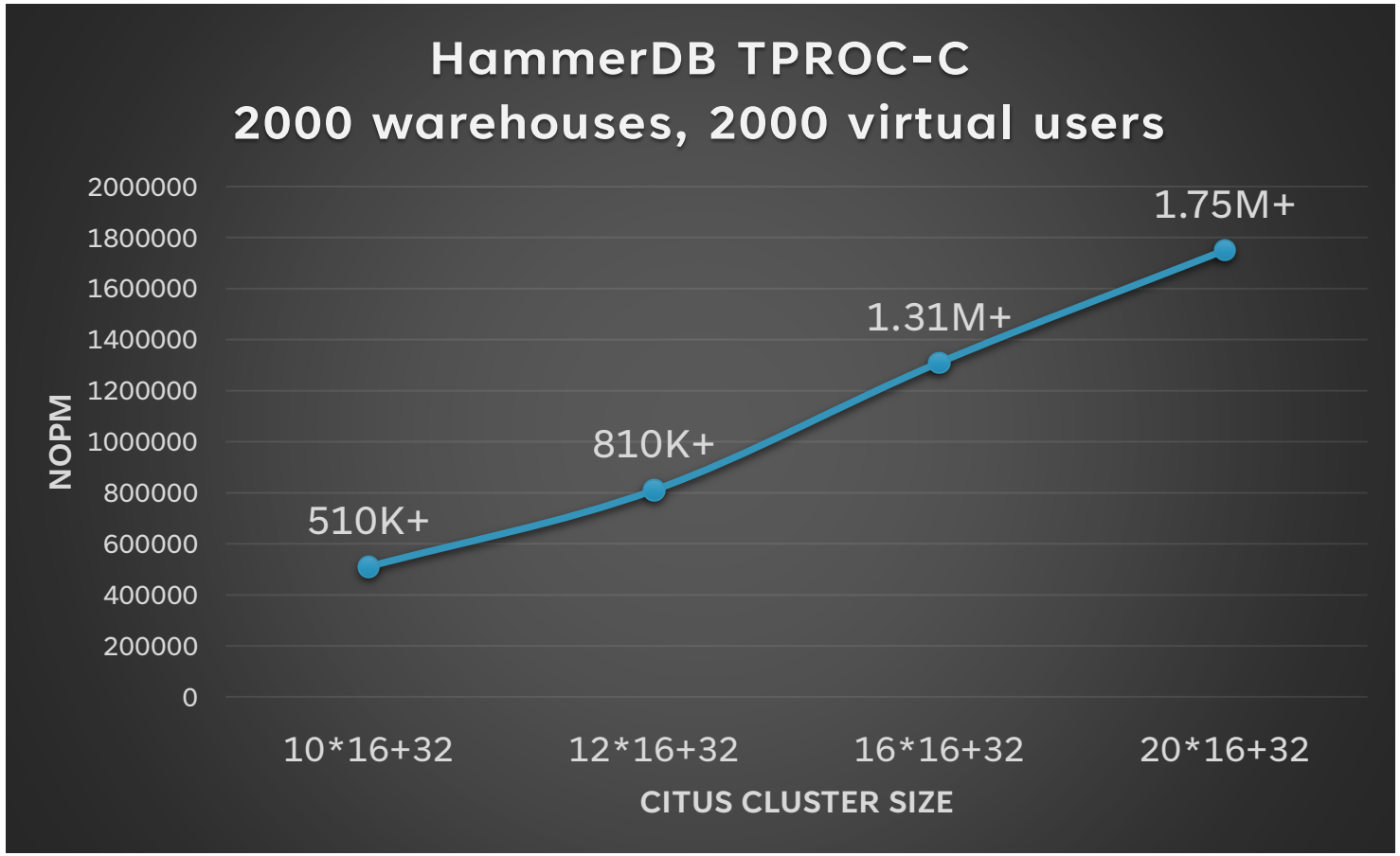
- Clear rising trend in NOPM as the cluster size increases
- Incredible 1.75M+ NOPM with 20*16+32 = 352 cores.
- Note that all clusters have HA disabled

Interpreting results



- From the previous benchmark, 10/12 workers had already utilized their resources with 1000 virtual users and warehouses
  - 10*16+32 910K+ NOPM
  - 12*16+32 1M+ NOPM
- They don't perform well with this high benchmark load
  - 10*16+32 510K+
  - 12*16+32 810K+

Interpreting results



- For 16 workers, CPU utilization is ~90% in all workers. The load is very-well distributed.
- For 20 workers, CPU utilization is ~75% in all workers. The load is very well-distributed as well.

Tweaking parameters

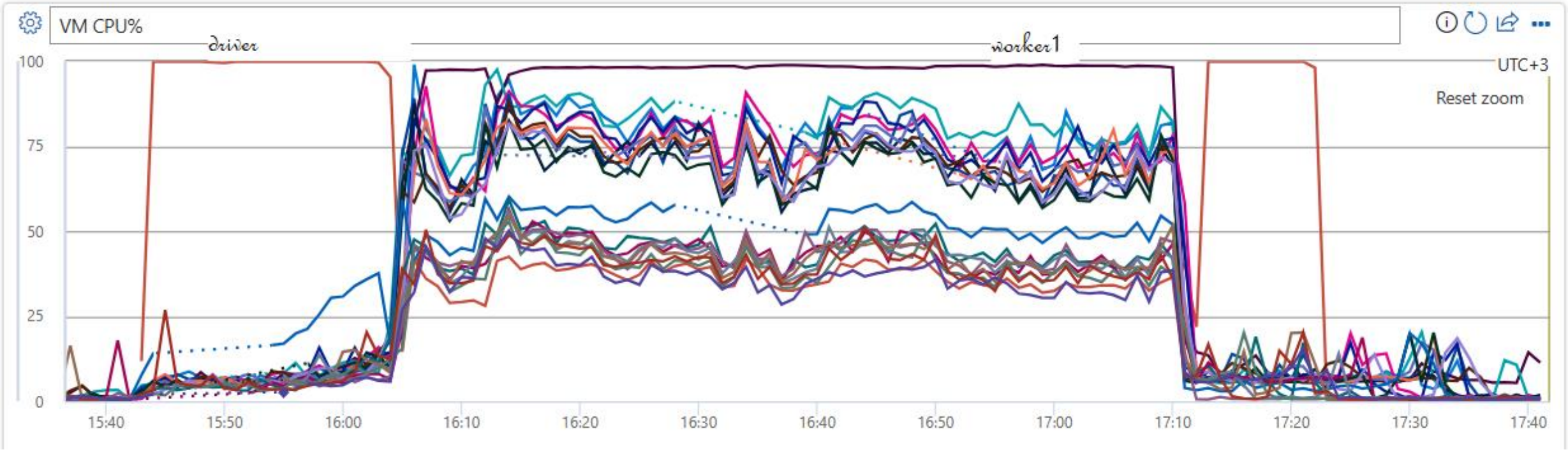
- Mentioned CPU and Disk IOPS in the previous results
- Tweaked parameters based on the utilization graphs

IF low CPU usage, THEN increase benchmark load ☑
IF unbalanced CPU usage, THEN balance data on cluster
IF high Disk IOPS, THEN reuse some warehouses

Tweaking parameters: shard_count

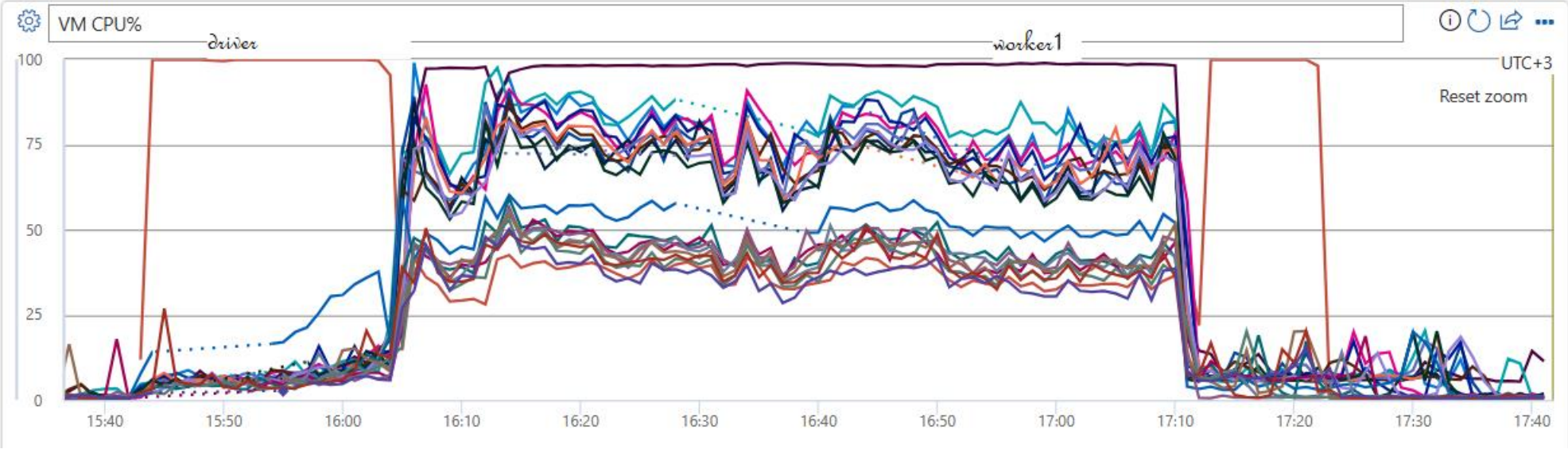
- Default shard count in Azure CosmosDB for PostgreSQL is 32.
- To divide data load evenly among workers, shard count should be a multiple of the number of workers.

Tweaking parameters: shard_count



CPU utilization graph during 2000 virtual users benchmark for 20*16+32 shard_count = 32

Tweaking parameters: shard_count

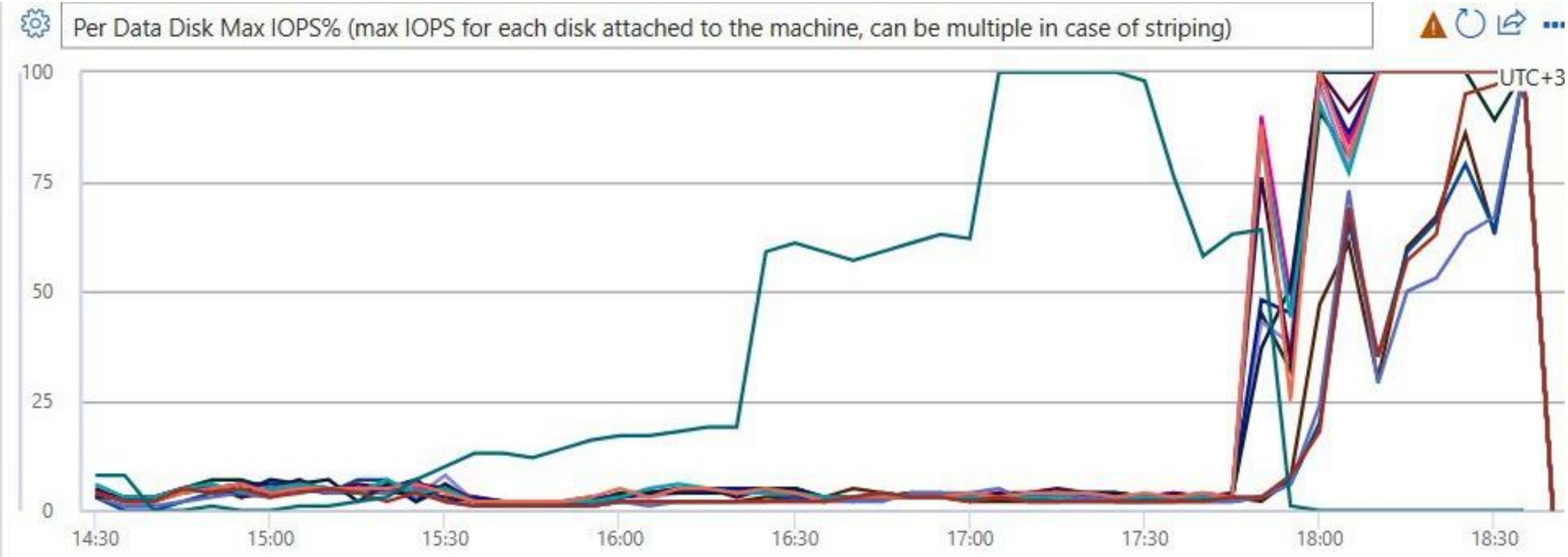


- 32 shards in 20 workers means 12 workers get 2 shards each, 8 workers get 1 shard each.
- **Shard_count = 32 yielded 1.22M NOPM.** Clearly, 12 of the workers are more loaded than the remaining 8.
- After changing **shard_count to 40 (divisible by 20)**, the result went up to **1.75M NOPM**.

Tweaking parameters: all_warehouses

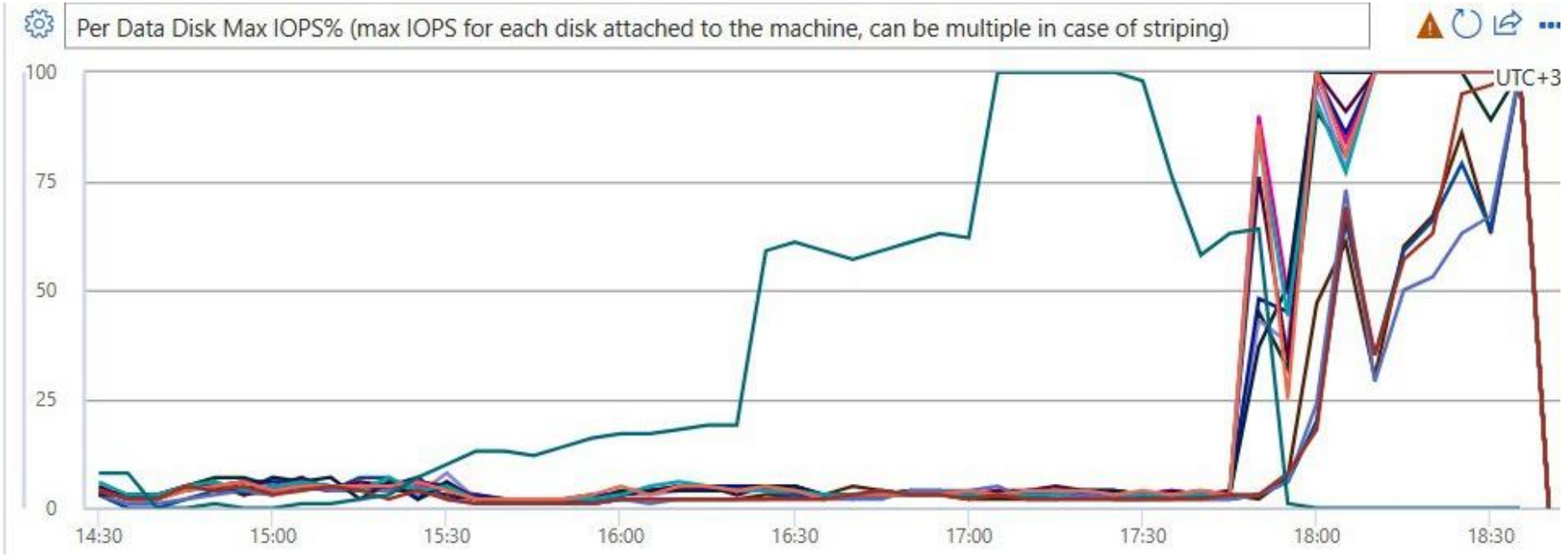
- By default each Virtual User selects a warehouse at random.
- This doesn't ensure all warehouses will be used.
- Setting all_warehouses to true means that the Virtual Users select a new warehouse for each transaction from an available list divided between all Virtual Users
- This ensures all warehouses will be used.
- However, I/O activity will be higher!

Tweaking parameters: all_warehouses



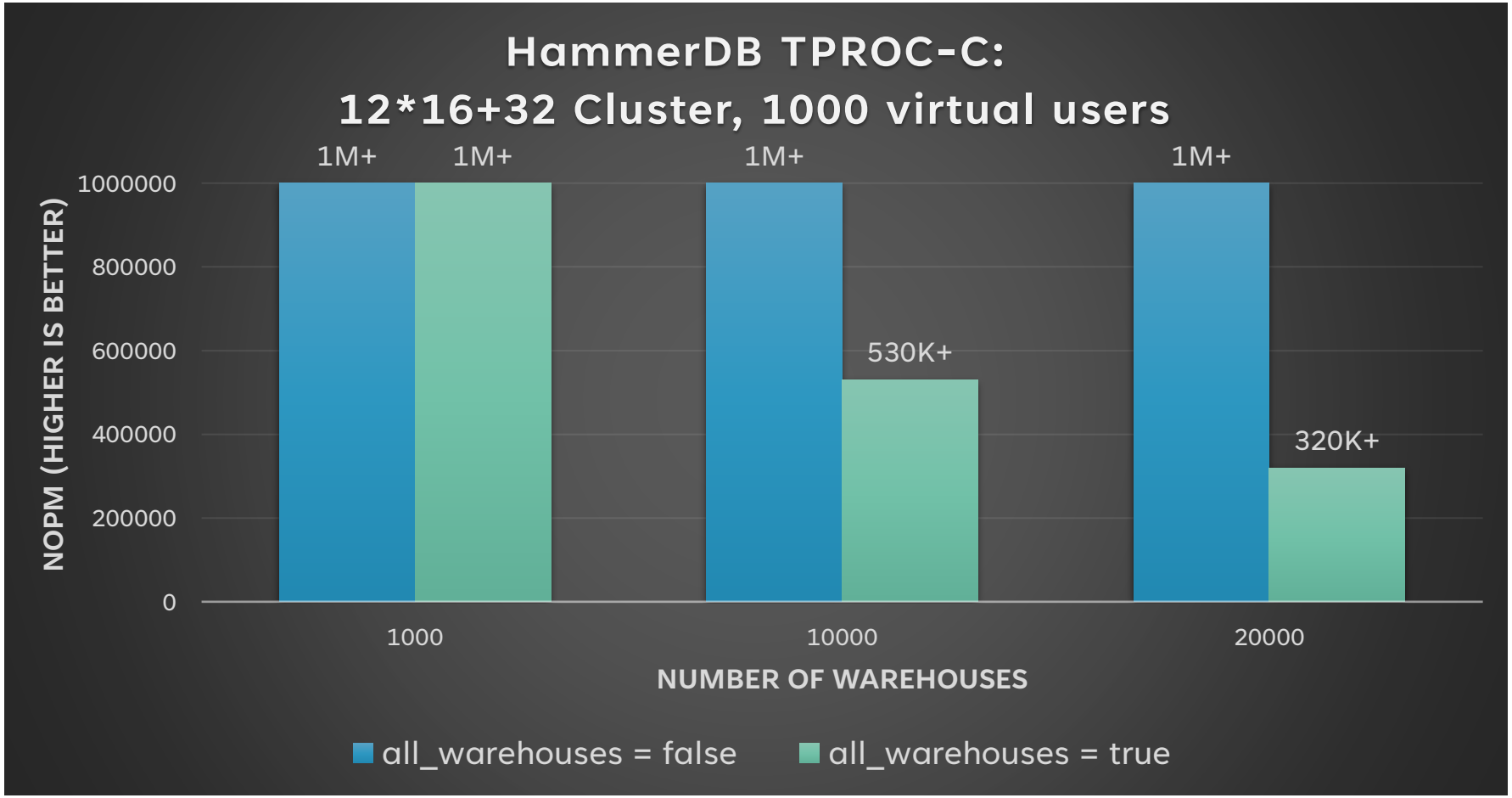
Per Data Disk Max IOPS% graph during 10K warehouses benchmark for 12*16+32 all_warehouses = true

Tweaking parameters: all_warehouses

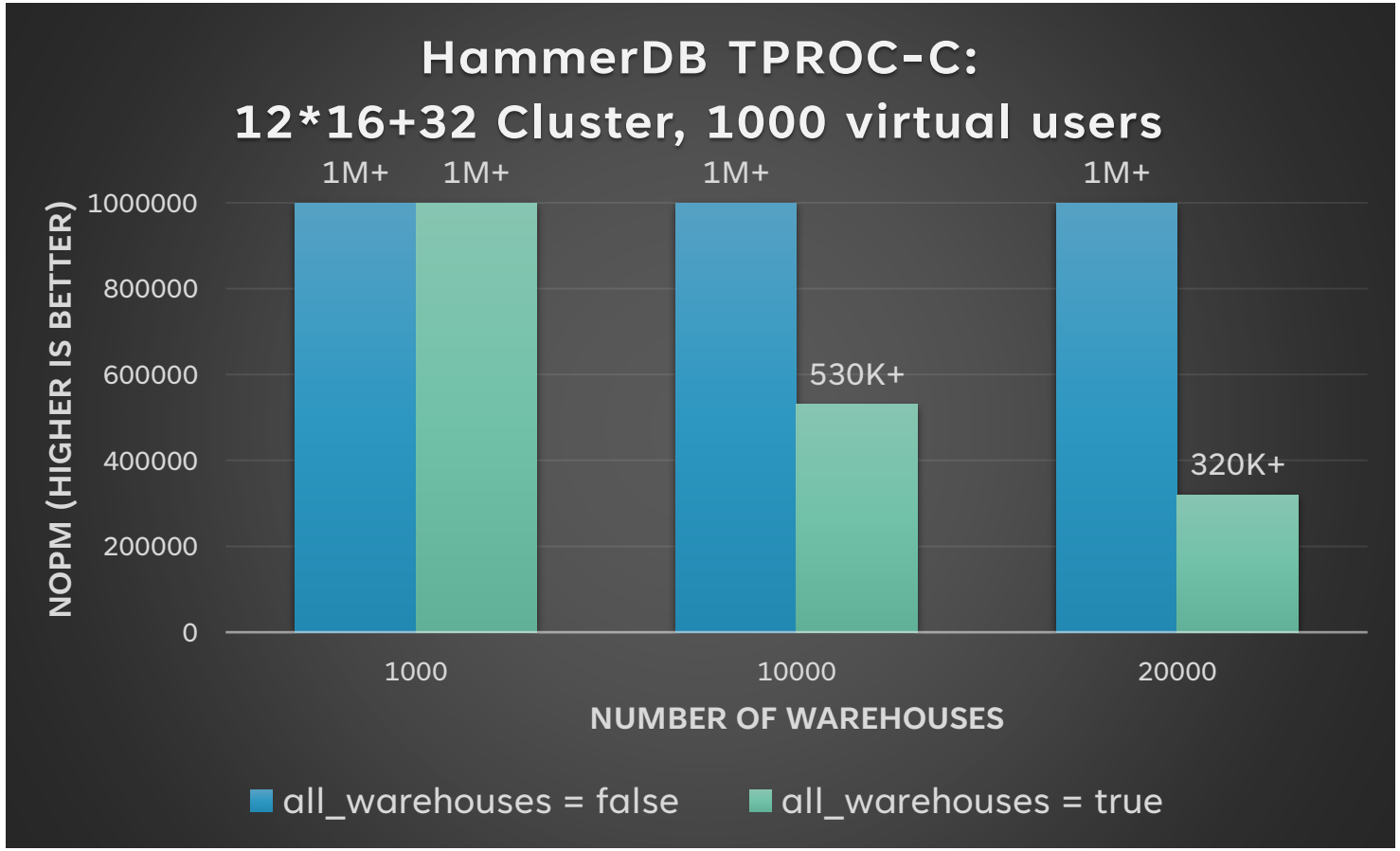


- All_warehouses was set to true, as it worked well for 1000 warehouses. But, clearly, for 10000 warehouses, disk IOPS is maxed out.
- After setting all_warehouses to false, I/O activity calmed down, allowing us to achieve 1M NOPM again.

Tweaking parameters: all_warehouses



Tweaking parameters: all_warehouses



- When we use all warehouses, as the number of warehouses increases, I/O activity is so high that we max out on disk IOPS and it negatively affects the benchmark result.
- Setting all_warehouses to false, however, doesn't affect our 1M+ NOPM at all.

Overall Reflections

- Read some HammerDB documentation and learn a bit about your Citus cluster capabilities
- Start somewhere meaningful, not perfect!
- Study results, change parameters based on the results to maximize cluster NOPM throughput
- It takes a lot of time to run these benchmarks, it's impossible to run all the combinations that you'd like ☺
- Personally, I have more to learn on Citus and HammerDB TPROC-C benchmarking – it’s a beautiful journey ☺

References

- HammerDB's official documentation
- Jelte Fennema's blog post on How to benchmark performance of Citus and Postgres with HammerDB on Azure

Thank you for your attention.