Microsoft

Performance measurement and tuning of Cassandra 5.0 transactions on Cloud infrastructure

German Eichberger, Principal Engineering Manager Microsoft Azure Data&AI
Pallavi Iyengar, Senior Software Engineer Microsoft Azure Data&AI

Hello

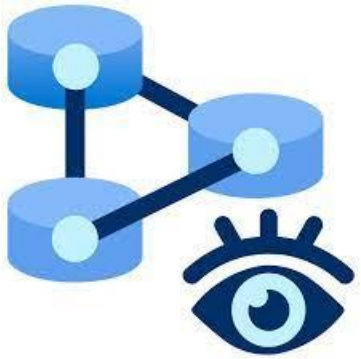
German Eichberger

- Attended my first ApacheCon 2005 with a SUN Java Community Project scholarship
- Set up a production Cassandra Cluster for HP Connected in 2013
- Principal Software Engineering Manager on the Azure Cassandra Team @ Microsoft
- Developer of cassandra-proxy and contributed some bug fixes to Apache Cassandra

Pallavi Iyengar

- Senior Software Engineer @Microsoft working with Apache Cassandra team
- Implemented write through cache functionality leveraging local NVMe disks for Azure Cassandra
- Contributed bug fixes to open source LDAP plugin for Apache Cassandra

Our Services



Azure Managed Instances for Apache Cassandra

- We run Apache Cassandra 3.11, 4.0, 4.1 (and soon 5.0) as-a-Service in Microsoft Azure
- We have a wide variety of customers, some large ones with mission critical work loads

Azure Cosmos DB for Apache Cassandra

- We run an Apache Cassandra CQL compatible wire protocol to store data in Azure Cosmos DB
- We have a wide variety of customers who value request based pricing, elasticity, and scalability

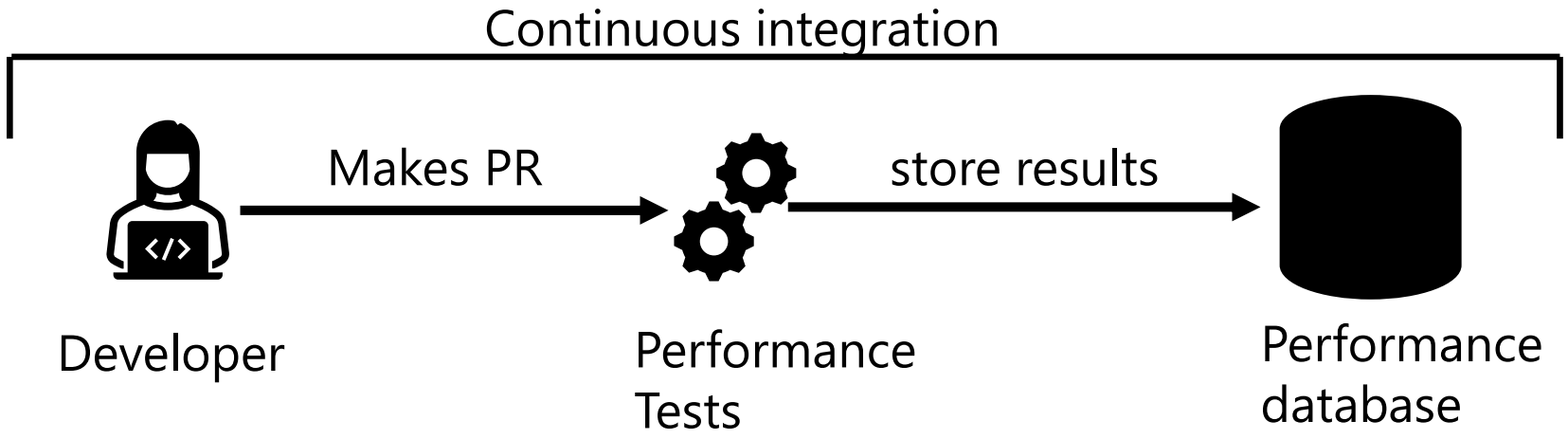
Agenda

- Our performance setup
- C* 3.11, 4.1, and 5.0 on various Azure SKUs
- Write through cache and performance
- The trouble with measuring NoSQL transaction performance
- ~~Early results with NoSQL Bench's small economy workload~~

What is Cassandra?

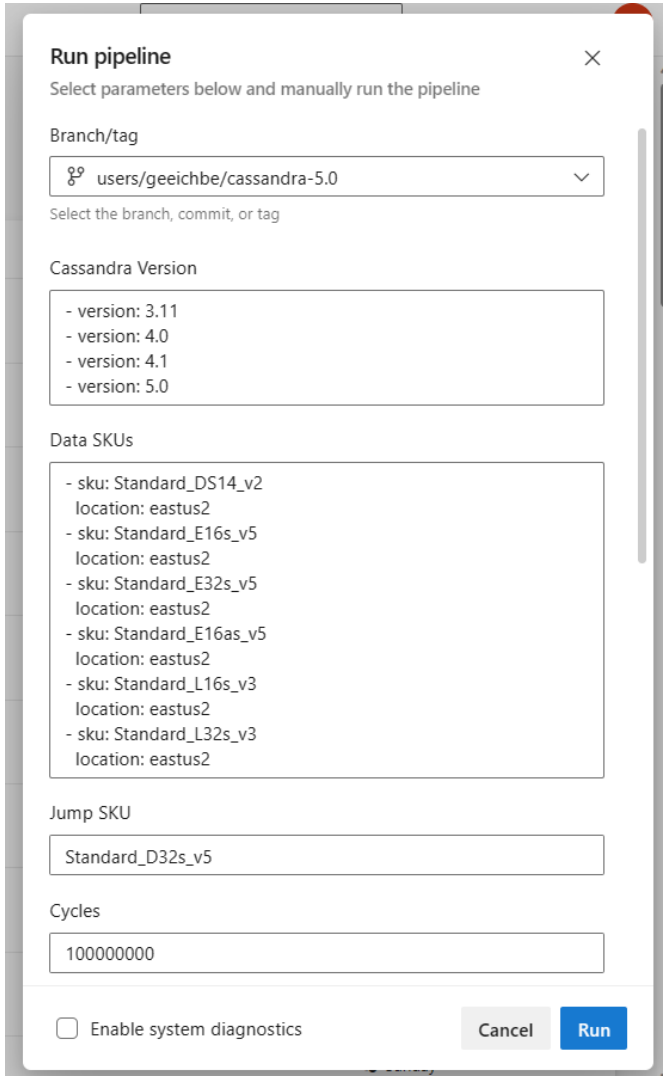
- Cassandra is a NoSQL distributed Database
- It’s linear scalable
- Fault tolerant
- …

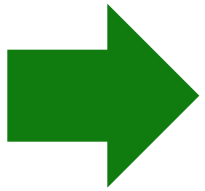
Our performance test setup



- Script which deploys a three node cluster with parameterized SKU and Cassandra version
- Runs NoSQLBench with a workload of our choosing
- Reports results into our log analytics database

Our performance setup





```
ManagedCassandraPerformance
| where PreciseTimeStamp > ago(7d)
| extend latency_max_s = latency_max / 1000000000
| project-reorder PreciseTimeStamp
| order by sku, throughput desc
```

PreciseTimeStamp	TIMESTAMP	cassandra_version	sku	disks	nodes	cycles	threads	lat
2023-10-02 01:44:08.0000000	2023-10-02 01:40:00.0000000	3.11	Standard_L32s_v3	4	3	100000000	160	53
2023-10-03 10:21:57.0000000	2023-10-03 10:20:00.0000000	3.11	Standard_L32s_v3	4	3	100000000	160	52
2023-10-01 23:05:28.0000000	2023-10-01 23:05:00.0000000	3.11	Standard_L32s_v3	4	3	100000000	160	54
2023-10-02 00:36:20.0000000	2023-10-02 00:35:00.0000000	3.11	Standard_L32s_v3	4	3	100000000	160	51
2023-10-02 10:20:52.0000000	2023-10-02 10:20:00.0000000	3.11	Standard_L32s_v3	4	3	100000000	160	66
2023-09-29 10:41:18.0000000	2023-09-29 10:40:00.0000000	4.0	Standard_L32s_v3	4	3	100000000	160	56
2023-09-26 21:43:56.0000000	2023-09-26 21:40:00.0000000	3.11	Standard_L32s_v3	4	3	100000000	160	71
2023-10-03 10:36:55.0000000	2023-10-03 10:35:00.0000000	4.0	Standard_L32s_v3	4	3	100000000	160	41
2023-10-02 03:56:44.0000000	2023-10-02 03:55:00.0000000	3.11	Standard_L32s_v3	4	3	100000000	160	6
2023-10-01 20:39:03.0000000	2023-10-01 20:35:00.0000000	3.11	Standard_L32s_v3	4	3	100000000	160	55
2023-09-30 10:38:31.0000000	2023-09-30 10:35:00.0000000	4.0	Standard_L32s_v3	4	3	100000000	160	53
2023-09-26 17:36:19.0000000	2023-09-26 17:35:00.0000000	3.11	Standard_L32s_v3	4	3	100000000	160	77
2023-10-01 10:39:22.0000000	2023-10-01 10:35:00.0000000	4.0	Standard_L32s_v3	4	3	100000000	160	56
2023-09-27 10:40:46.0000000	2023-09-27 10:40:00.0000000	4.0	Standard_L32s_v3	4	3	100000000	160	54
2023-09-28 10:27:12.0000000	2023-09-28 10:25:00.0000000	3.11	Standard_L32s_v3	4	3	100000000	160	59
2023-09-27 02:03:21.0000000	2023-09-27 02:00:00.0000000	3.11	Standard_L32s_v3	4	3	100000000	160	53
2023-09-30 21:43:25.0000000	2023-09-30 21:40:00.0000000	3.11	Standard_L32s_v3	4	3	100000000	160	52
2023-09-30 20:24:19.0000000	2023-09-30 20:20:00.0000000	5.0	Standard_L32s_v3	4	3	100000000	160	43
2023-09-30 19:25:29.0000000	2023-09-30 19:25:00.0000000	3.11	Standard_L32s_v3	4	3	100000000	160	67
2023-09-30 10:14:07.0000000	2023-09-30 10:10:00.0000000	3.11	Standard_L32s_v3	4	3	100000000	160	53

Our workloads

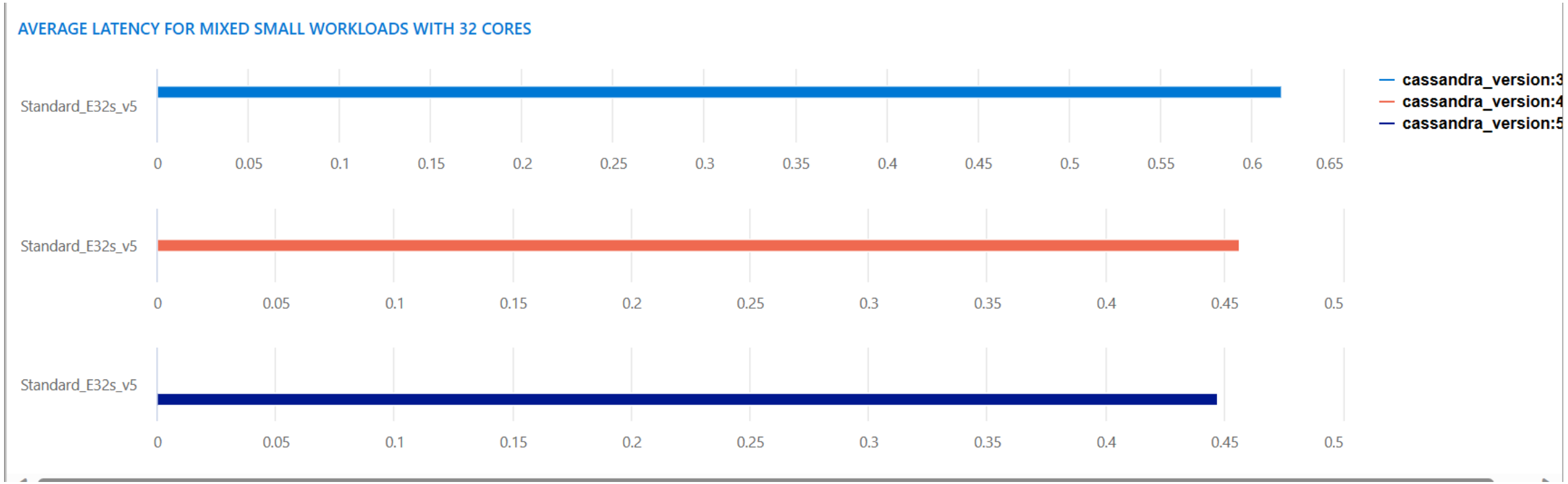
- Small: small data 10 byte of value for text fields
- Bulky: 4K-5K sized text to bust system/page cache to measure disk read/write performance

- We can run them read, write, and mixed (both read and write)

Disclaimers

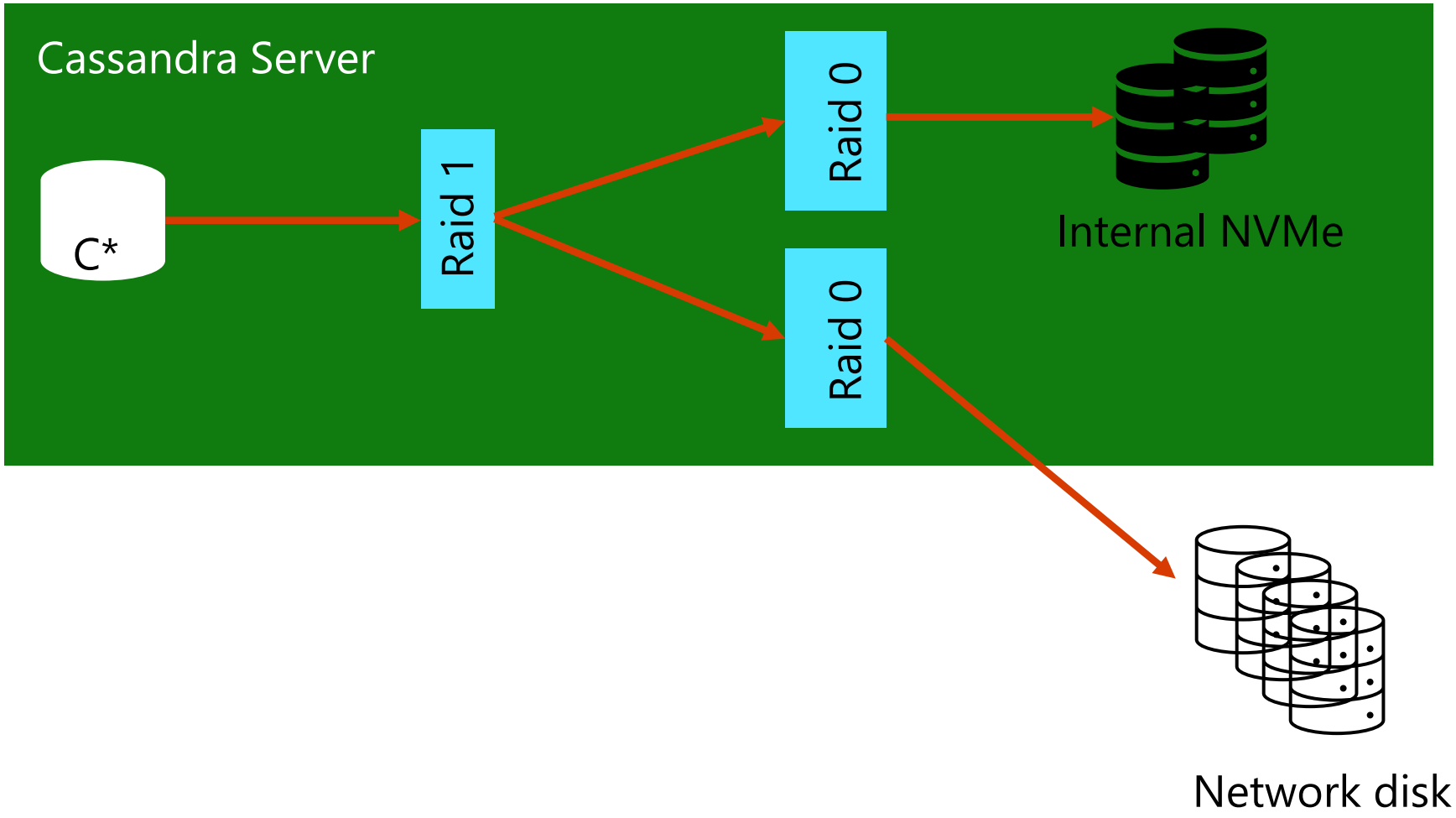
- C* 5.0 is an alpha version so performance numbers will definitely be different with the final release
- Our setup is not for serious nor official benchmarking and as such numbers are for informational purpose only and are not official Azure numbers
  - Our standard for official benchmarking is published (not proprietary) workloads and done by independent researchers
- We only were able to perform **one** benchmark run due to time constraints and bugs in 5.0. Ordinarily, we aim for multiple to minimize cloud introduced tail latencies (see our talk from last years) and outliers

Results for various SKUs



Write Through Cache

Write Through Cache



Source: https://discord.com/blog/how-discord-supercharges-network-disks-for-extreme-low-latency

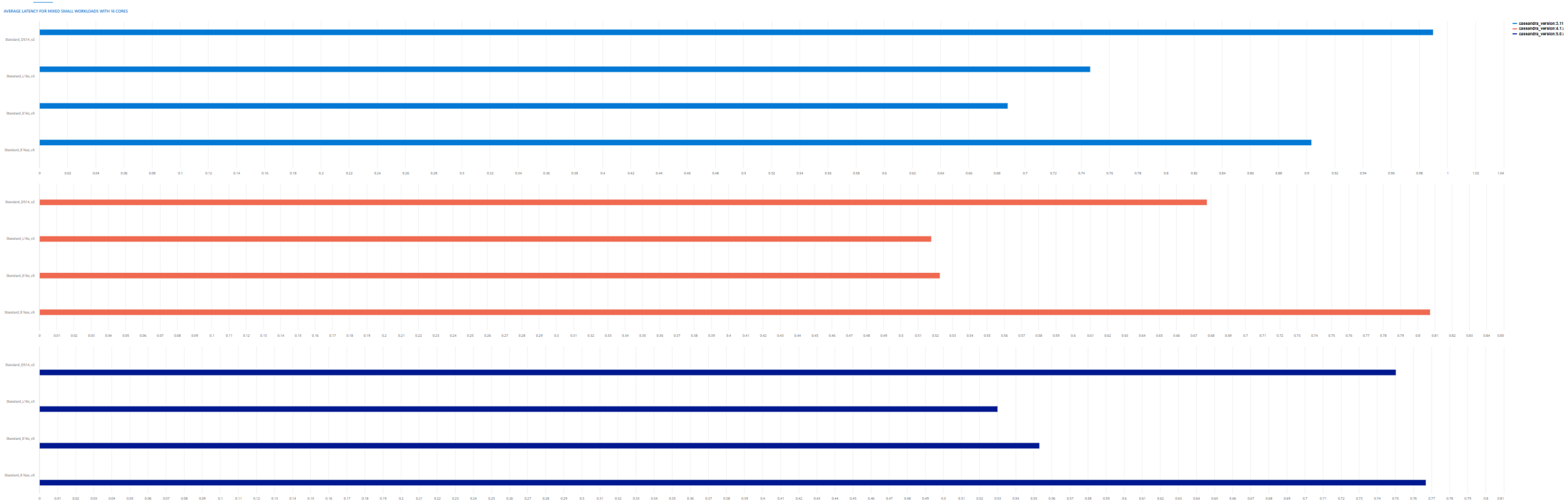
Why do we need a write through cache?

- Our current setup allows only 5,000 IOPS per network disk and we need more for read heavy workloads
- Azure compute is phasing out dedicated write-through cache in later SKU versions so we needed our own implementation
- Network disk tail latencies can heavily influence performance

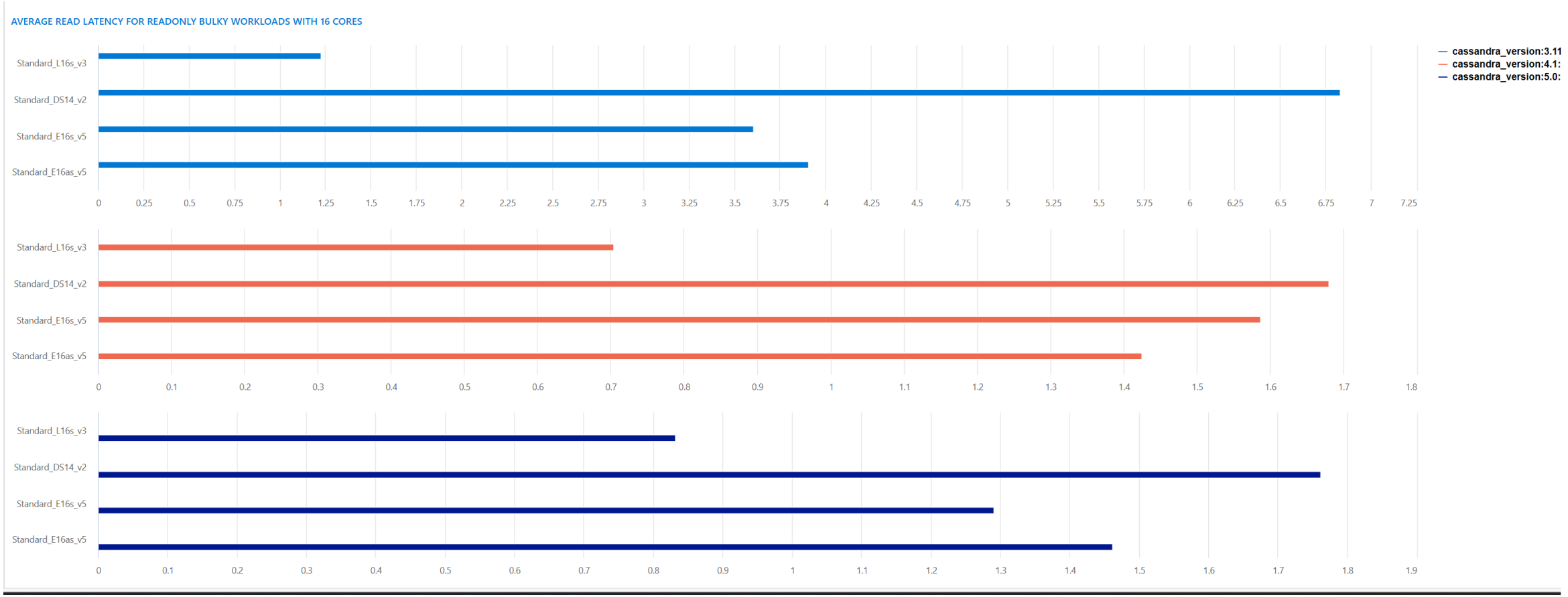
Caveats & Alternatives

- Why can’t you just run on local disk?
  - Azure will loose the disk if the vm gets migrated to a new host
  - There is no guarantee there can’t be a regional outage and you will loose all disks in that region and potentially data
  - This allows for the app to work right after an outage though with higher latency
- Why can’t you just use better network disks?
  - Cost
- Why can’t you use bcache or others?
  - Indeterministic performance because of warming up the cache. Our solution will have predictable latency once the local NVMe disk is rebuild
- We don’t want the latency hit during rebuild
  - Take the node out of rotation during rebuild

Results for various SKUs



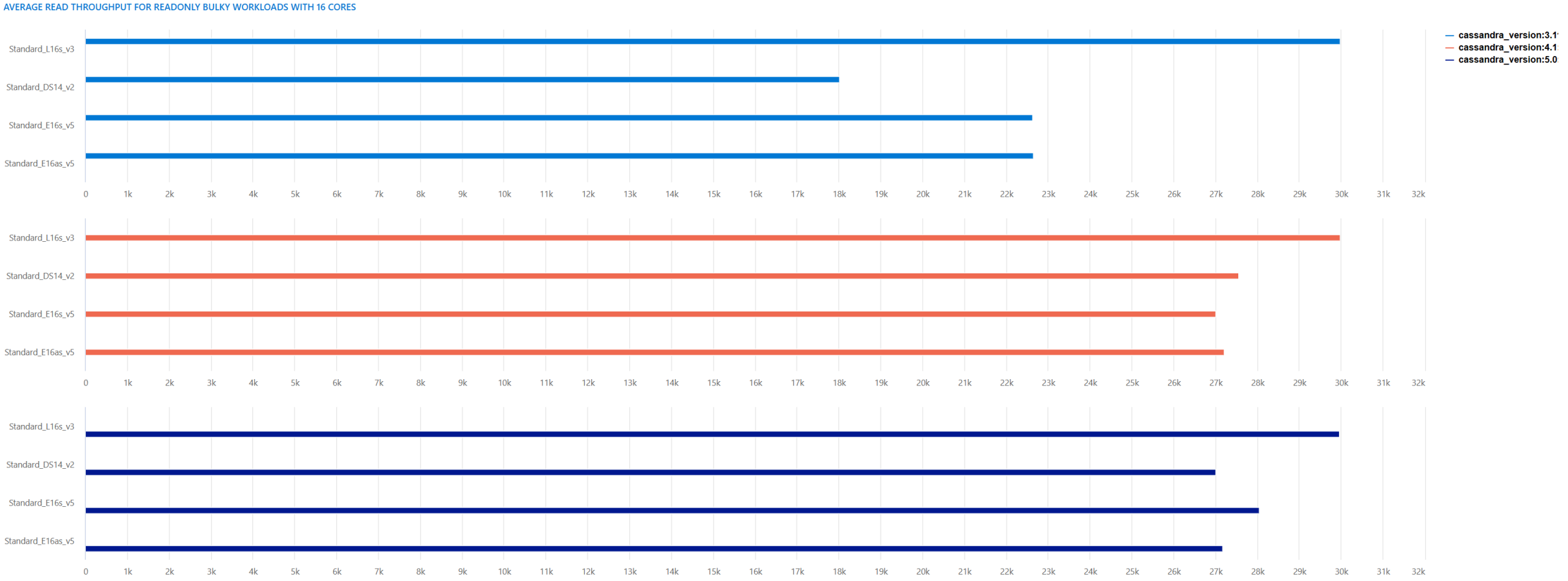
Results for various Azure SKUs



Performance Testing NoSQL Transactions

Subtitle

Results for various Azure SKUs

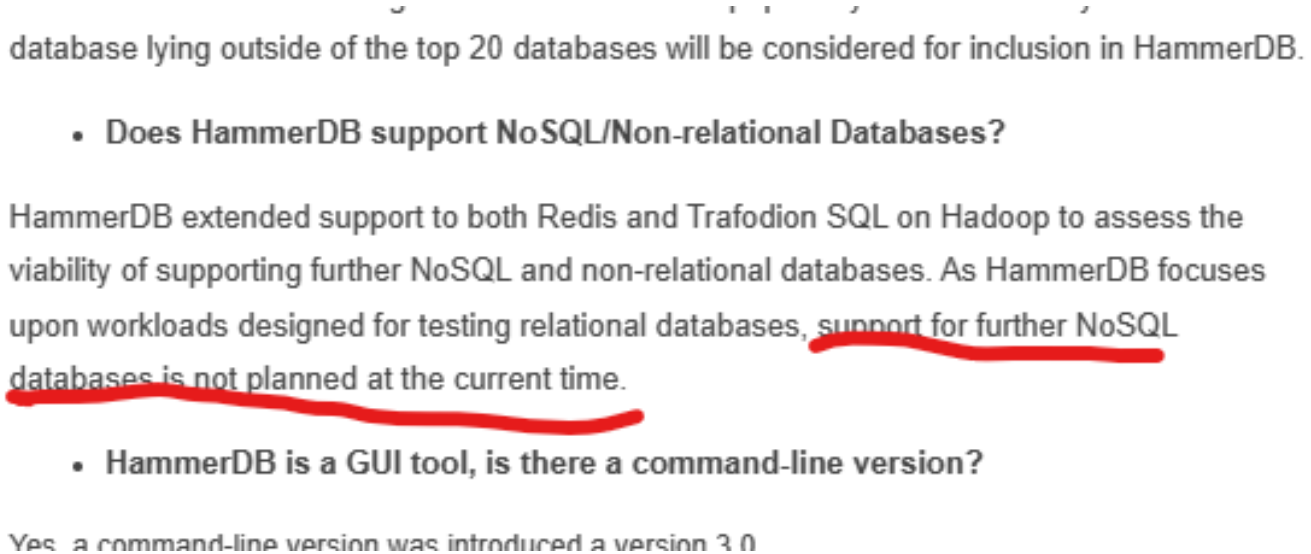


What are Transactions in Cassandra

- Accord protocol
- Cross partition, masterless
- One roundtrip fastpath
- Half the nodes can be down and it’s still on the Fast path

The state of NoSQL Benchmarking

- Database Transaction Unit or Cloud Database Benchmark (Microsoft)
  - "The DTU benchmark measures the performance of a mix of basic database operations that occur most frequently in online transaction processing (OLTP) workloads."
  - Reached out to the maintainer but they referred me to HammerDB
- Hammer DB (TPC Council)
  - "Gold standard" for transactional performance measurements simulating financial institution, flight reservations, etc.
  - But:



database lying outside of the top 20 databases will be considered for inclusion in HammerDB.

- **Does HammerDB support NoSQL/Non-relational Databases?**

HammerDB extended support to both Redis and Trafodion SQL on Hadoop to assess the viability of supporting further NoSQL and non-relational databases. As HammerDB focuses upon workloads designed for testing relational databases, support for further NoSQL databases is not planned at the current time.

- **HammerDB is a GUI tool, is there a command-line version?**

Yes, a command-line version was introduced a version 3.0

Source: https://learn.microsoft.com/en-us/azure/azure-sql/database/dtu-benchmark?view=azuresql and https://github.com/TPC-Council/HammerDB

The state of NoSQL Benchmarking

Are current benchmarks adequate to evaluate distributed transactional databases?

- Small Bank
  - Simulates a small banking system, where each customer has a pair of accounts, one for savings and the other for checking.
- Peak Bench
  - defines a package of workloads simulating intensive transactional processing requirements by designing a fine control in contention generation.
- YCSB+T
  - Is just a PR on YCSB which isn't even merged?!

Conclusion:

"a new benchmark exploring all the choke points together with an easy-use support tool is imperative for promoting both development and fair benchmarking."

Source: https://www.sciencedirect.com/science/article/pii/S2772485922000187

What is YCSB+T

- YCSB+T aims to measure in addition to performance, scalability, availability, and replication:
  - **Transactional Overhead**: compare database operations wrapped in transactions and without
  - **Consistency**: detect consistency anomalies with a validation stage

Source: https://ieeexplore.ieee.org/document/6818330

What is YCSB+T

- The Small Economy model works as follow:
  - Load Phase
    - Set each account (identified by key account number) to an initial specified total_cash
  - Transactional Phase:
    - doTransactionInsert() creates a new account with an initial balance captured from doTransactionDelete() operation described below.
    - doTransactionRead() reads a set of account balances determined by the key generator.
    - doTransactionScan() scans the database given the start key and the number of records and fetches them from the database.
    - doTransactionUpdate() reads a record and add $1 from the balance captured from delete operations to it and write it back.
    - doTransactionDelete() reads an account record, add the amount to the captured the balance (capture used in doTransactionInsert()) and then deletes the record.
    - doTransactionReadModifyWrite() reads two records, subtracts 1 from the one of the two and adds 1 to the other before writing them both back.

Source: https://ieeexplore.ieee.org/document/6818330

What is YCSB+T

- The Small Economy model works as follow:
  - Validation Phase
    - Checks the sum iterates all the keys and adds up the account balance and validates the total against the total stored after the load stage using the total_cash
    - One would expect more anomalies to be introduced as operations are performed which leads to a simple anomalie score:

$$\gamma = \frac{|S_{initial} - S_{final}|}{n}$$

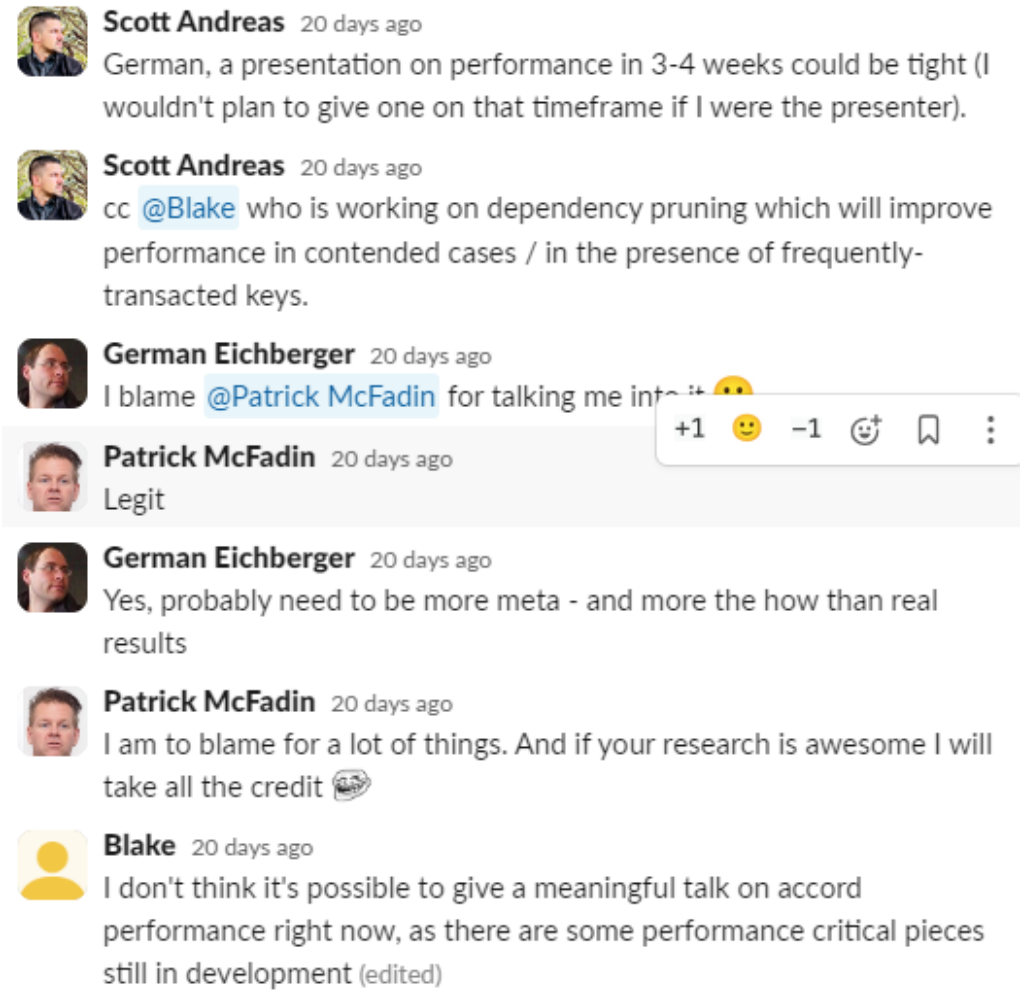
Source: https://ieeexplore.ieee.org/document/6818330

Adopt YCSB+T small economy model to NoSQLBench

- NoSQLBench is an OpenSource performance test solution and also used in our performance test setup
- Since YCSB+T never got merged we decided to port the workload to NoSQLBench
- We ran into several issues which might affect performance and need addressing in the NoSQLBench codebase or we need to tune our workload accordingly
- Cassandra's transactions are focused on batches and "read-before-write" – they are less useful for other operation **but** YCSB+T wants to wrap all operations in transactions to figure out the transactional overhead for "science"

Source: https://ieeexplore.ieee.org/document/6818330

Then there’s that



Scott Andreas 20 days ago
German, a presentation on performance in 3-4 weeks could be tight (I wouldn't plan to give one on that timeframe if I were the presenter).

Scott Andreas 20 days ago
cc @Blake who is working on dependency pruning which will improve performance in contended cases / in the presence of frequently-transacted keys.

German Eichberger 20 days ago
I blame @Patrick McFadin for talking me int

+1 −1

Patrick McFadin 20 days ago
Legit

German Eichberger 20 days ago
Yes, probably need to be more meta - and more the how than real results

Patrick McFadin 20 days ago
I am to blame for a lot of things. And if your research is awesome I will take all the credit

Blake 20 days ago
I don't think it's possible to give a meaningful talk on accord performance right now, as there are some performance critical pieces still in development (edited)

Results – or the lack thereof

- ACCORD had bugs which didn’t allow for a number of transactions to work but they should be fixed and were related to Transactional Metadata
- Latest version in branch doesn’t play nicely with k8ssandra management API we use in our setup in lieu of nodetool
- Result: I have got nothing…

Microsoft

Questions & Thank you

Acknowledgements:

- Quetzal Bradley for design of the performance system
- Uri Smiley for developing it
- Alexander Laye for compiling C* 5.0