NSF/IUCRC CAC PROJECT

# INTEGRATED VISUALIZING, MONITORING, AND MANAGING HPC SYSTEMS

Jie Li
Doctoral Student, TTU

11/20/2020

Advisors:
Mr. Jon Hass, SW Architect, Dell Inc.
Dr. Alan Sill, Managing Director, HPCC, TTU
Dr. Yong Chen, Associate Professor, CS Dept, TTU
Dr. Tommy Dang, Assistant Professor, CS Dept, TTU

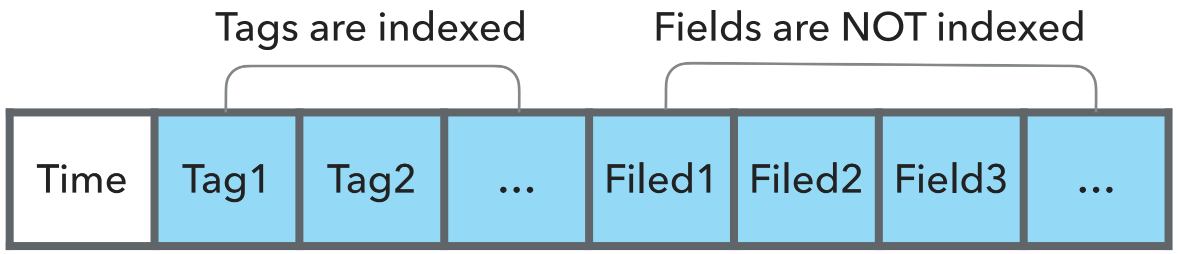
# INFLUXDB VS TIMESCALE

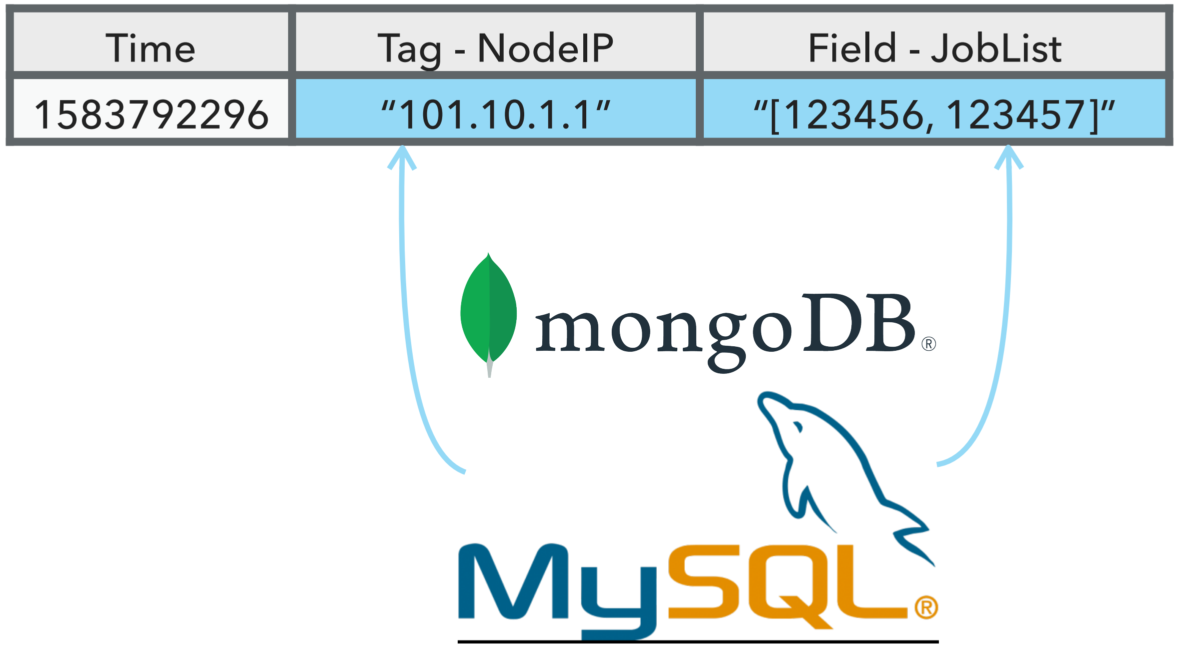
- InfluxDB vs TimescaleDB
  - Data model
  - Stability
  - Performance

Ref: https://blog.timescale.com/blog/timescaledb-vs-influxdb-for-time-series-data-timescale-influx-sql-nosql-36489299877/.

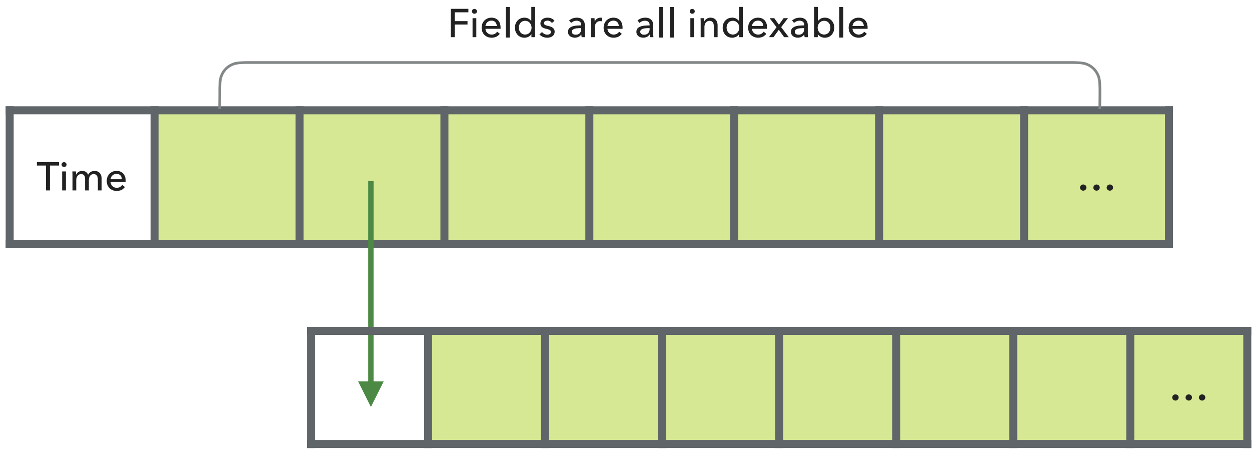
# DATA MODEL

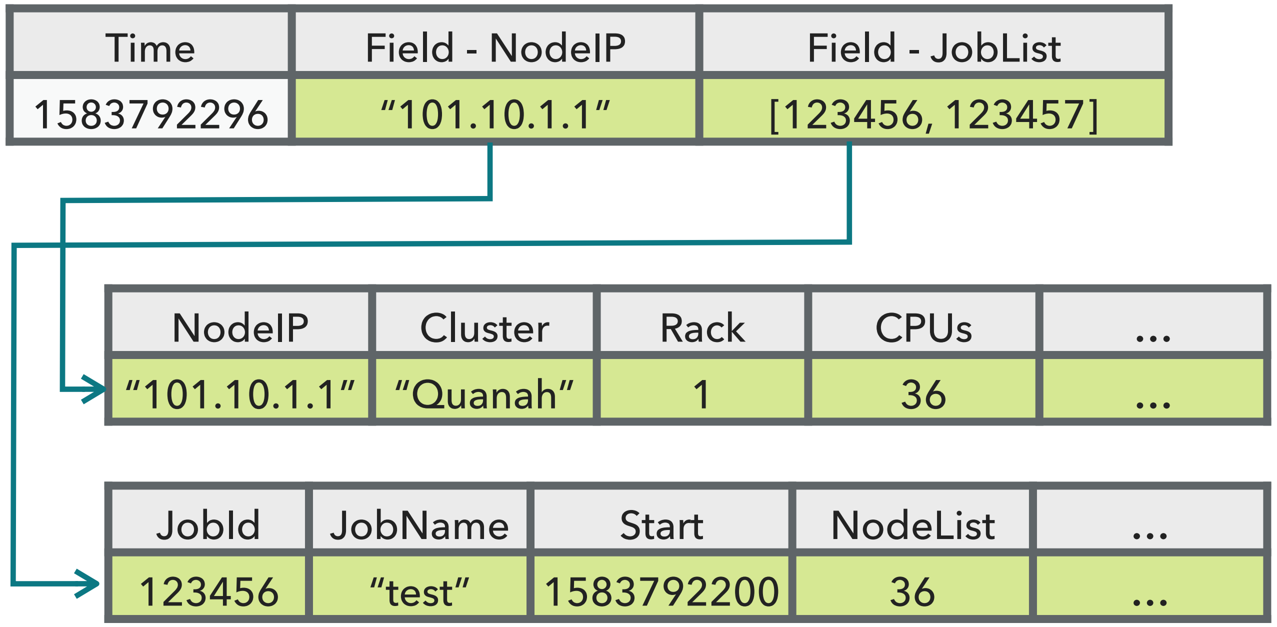
| InfluxDB | TimescaleDB |
|---|---|
| Non-relational DB (built from scratch in Go) | Relational DB (built on PostgreSQL) |
| floats, ints, strings, and booleans | floats, ints, strings, booleans, arrays, JSON, etc. |
| Only tags values are indexed | Indexable on all fields |





| Time | Tag - NodeIP | Field - JobList |
|---|---|---|
| 1583792296 | “101.10.1.1” | “[123456, 123457]” |





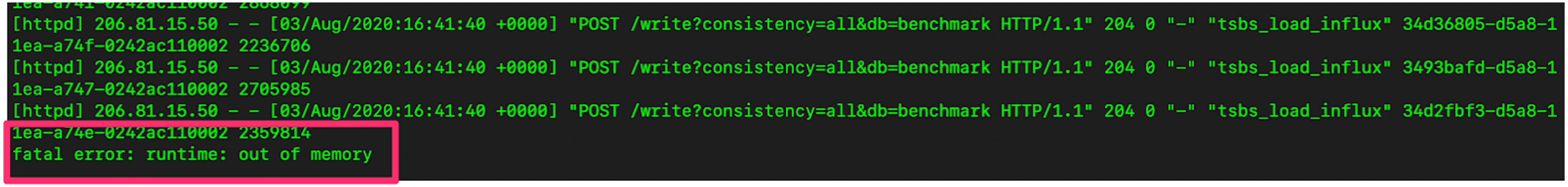
| Time | Field - NodeIP | Field - JobList |
|---|---|---|
| 1583792296 | “101.10.1.1” | [123456, 123457] |

| NodeIP | Cluster | Rack | CPUs | ... |
|---|---|---|---|---|
| “101.10.1.1” | “Quanah” | 1 | 36 | ... |

| JobId | JobName | Start | NodeList | ... |
|---|---|---|---|---|
| 123456 | “test” | 1583792200 | 36 | ... |

# STABILITY

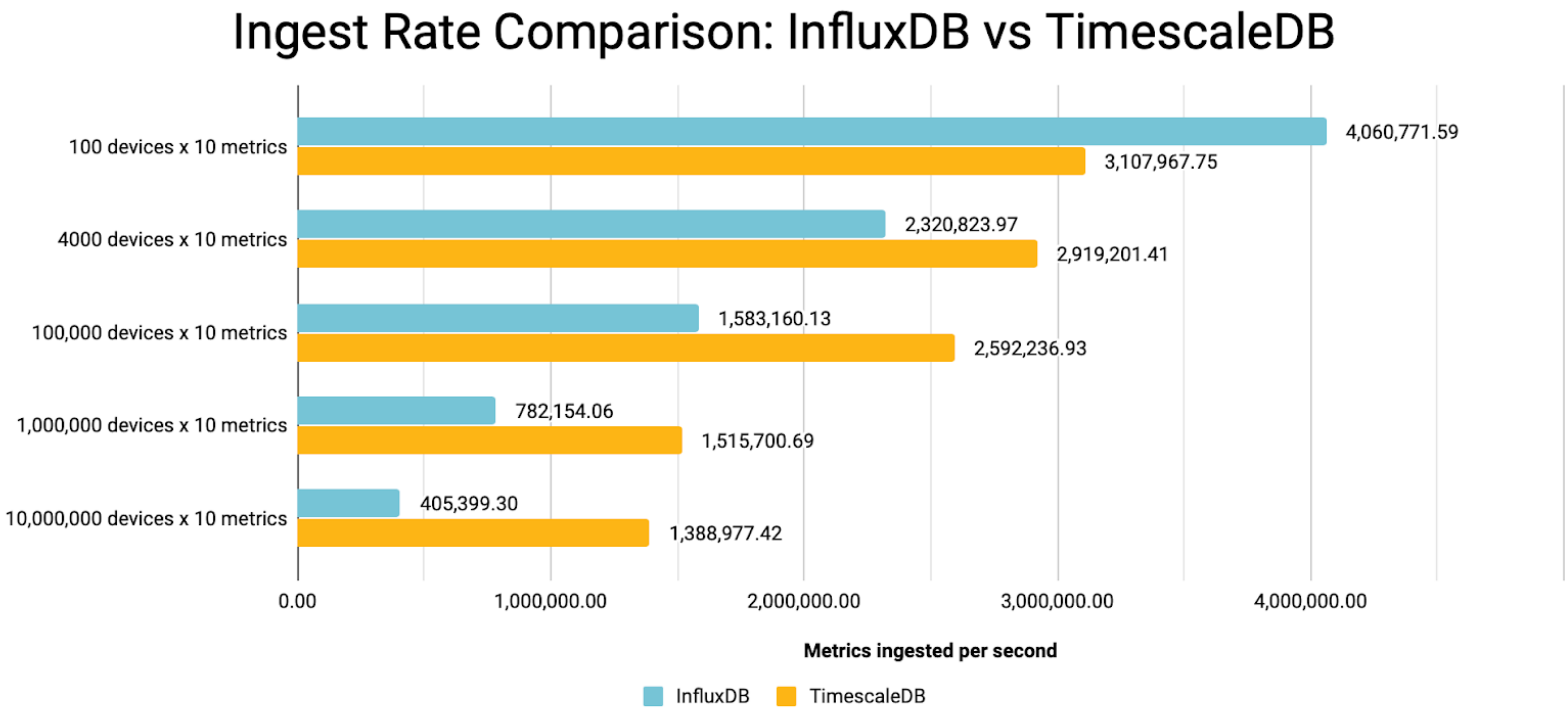
- Inserting batches into InfluxDB
  - Inserting batches of 10k into InfluxDB at high cardinalities will have write errors caused by timeouts, exceeding the maximum cache memory size, fatal out of memory errors.
  - Increasing maximum cache size and decreasing the batch size could solve these errors.
- Reading queries on InfluxDB
  - InfluxDB at high cardinalities could consume all available memory to run the query and crashed with an Out of Memory error.
- Writing large batches and reading queries on TimescaleDB do not have such issues. PostgreSQL limits system memory usage with settings like shared_buffers and work_mem.



```
[httpd] 206.81.15.50 - - [03/Aug/2020:16:41:40 +0000] "POST /write?consistency=all&db=benchmark HTTP/1.1" 204 0 "-" "tsbs_load_influx" 34d36805-d5a8-1
1ea-a74f-0242ac110002 2236706
[httpd] 206.81.15.50 - - [03/Aug/2020:16:41:40 +0000] "POST /write?consistency=all&db=benchmark HTTP/1.1" 204 0 "-" "tsbs_load_influx" 3493bafd-d5a8-1
1ea-a747-0242ac110002 2705985
[httpd] 206.81.15.50 - - [03/Aug/2020:16:41:40 +0000] "POST /write?consistency=all&db=benchmark HTTP/1.1" 204 0 "-" "tsbs_load_influx" 34d2fbf3-d5a8-1
1ea-a74e-0242ac110002 2359814
fatal error: runtime: out of memory
```

Ref: https://blog.timescale.com/blog/timescaledb-vs-influxdb-for-time-series-data-timescale-influx-sql-nosql-36489299877/.

# PERFORMANCE

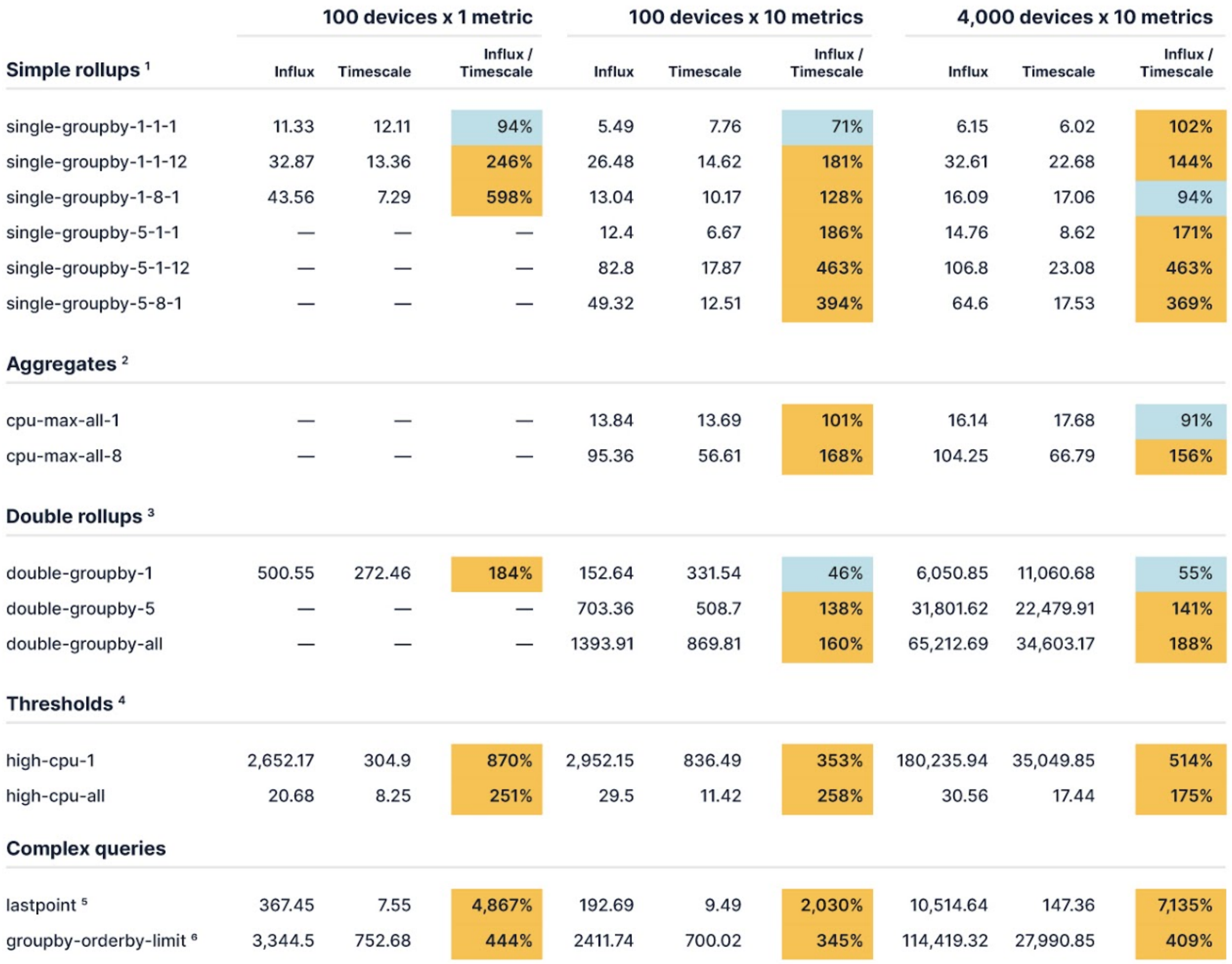


- InfluxDB outperforms TimescaleDB for workloads with low cardinality
- InfluxDB insert performance drops off dramatically as cardinality increases.
- TimescaleDB has ~3.5x the insert performance as InfluxDB

Ref: https://blog.timescale.com/blog/timescaledb-vs-influxdb-for-time-series-data-timescale-influx-sql-nosql-36489299877/.

# PERFORMANCE

## Query Performance (measured in milliseconds)



| | 100 devices x 1 metric | | | 100 devices x 10 metrics | | | 4,000 devices x 10 metrics | | |
|---|---|---|---|---|---|---|---|---|---|
| **Simple rollups** [1] | Influx | Timescale | Influx / Timescale | Influx | Timescale | Influx / Timescale | Influx | Timescale | Influx / Timescale |
| single-groupby-1-1-1 | 11.33 | 12.11 | 94% | 5.49 | 7.76 | 71% | 6.15 | 6.02 | 102% |
| single-groupby-1-1-12 | 32.87 | 13.36 | 246% | 26.48 | 14.62 | 181% | 32.61 | 22.68 | 144% |
| single-groupby-1-8-1 | 43.56 | 7.29 | 598% | 13.04 | 10.17 | 128% | 16.09 | 17.06 | 94% |
| single-groupby-5-1-1 | — | — | — | 12.4 | 6.67 | 186% | 14.76 | 8.62 | 171% |
| single-groupby-5-1-12 | — | — | — | 82.8 | 17.87 | 463% | 106.8 | 23.08 | 463% |
| single-groupby-5-8-1 | — | — | — | 49.32 | 12.51 | 394% | 64.6 | 17.53 | 369% |
| **Aggregates** [2] | | | | | | | | | |
| cpu-max-all-1 | — | — | — | 13.84 | 13.69 | 101% | 16.14 | 17.68 | 91% |
| cpu-max-all-8 | — | — | — | 95.36 | 56.61 | 168% | 104.25 | 66.79 | 156% |
| **Double rollups** [3] | | | | | | | | | |
| double-groupby-1 | 500.55 | 272.46 | 184% | 152.64 | 331.54 | 46% | 6,050.85 | 11,060.68 | 55% |
| double-groupby-5 | — | — | — | 703.36 | 508.7 | 138% | 31,801.62 | 22,479.91 | 141% |
| double-groupby-all | — | — | — | 1393.91 | 869.81 | 160% | 65,212.69 | 34,603.17 | 188% |
| **Thresholds** [4] | | | | | | | | | |
| high-cpu-1 | 2,652.17 | 304.9 | 870% | 2,952.15 | 836.49 | 353% | 180,235.94 | 35,049.85 | 514% |
| high-cpu-all | 20.68 | 8.25 | 251% | 29.5 | 11.42 | 258% | 30.56 | 17.44 | 175% |
| **Complex queries** | | | | | | | | | |
| lastpoint [5] | 367.45 | 7.55 | 4,867% | 192.69 | 9.49 | 2,030% | 10,514.64 | 147.36 | 7,135% |
| groupby-orderby-limit [6] | 3,344.5 | 752.68 | 444% | 2411.74 | 700.02 | 345% | 114,419.32 | 27,990.85 | 409% |

- Generally, Timescale outperforms InfluxDB.
- When simply rolling up a single metric, InfluxDB can sometimes outperform TimescaleDB
- TimescaleDB vastly outperforms InfluxDB for complex queries.

🟦 **InfluxDB outperforms**

🟨 **TimescaleDB outperforms**

Ref: https://blog.timescale.com/blog/timescaledb-vs-influxdb-for-time-series-data-timescale-influx-sql-nosql-36489299877/.

# CONCLUSION

- We do NOT need to use a non-TSDB to store static data (job data) if using TimescaleDB to store the HPC monitoring data.
- TimescaleDB is much more stable when writing and reading high-cardinality datasets.
- TimescaleDB performs better on writing and reading high-cardinality datasets.

We may use TimescaleDB as the main storage solution for monitoring the RedRaider cluster.

# QUESTIONS?/COMMENTS?