CASSANDRA SUMMIT 2016

Micro-batching: High performance writes

Adam Zegelin, Instaclustr

instaclustr

Me

Adam Zegelin

- Co-founder of Instaclustr

Instaclustr

- Managed Apache Cassandra & DSE + Spark/Zeppelin in the ☁️
- Amazon Web Services, Azure, IBM Softlayer
- 24×7×365 Support

- Cassandra & Spark Consulting
- Enterprise support-only contracts

Problem Background

- Metrics — 2,000+ metrics (events) per-node, collected every 20 seconds
  - ~ 50k events *per-second*

- Streamed via RabbitMQ to Riemann
- Riemann by *Aphyr*
  - Event stream processor framework — Clojure (JVM)
  - Analyses/combines/filters events
  - Rules — events matching, over threshold, etc.
  - Actions — PagerDuty, email, Slack, etc.
  - Forwards *everything* to Cassandra

Problem Background cont'd

- Riemann:
  - Initially, single instance
  - Fairly complicated to scale (+HA)
    See my blog post *Post 500 Nodes: High Availability & Scalability with Riemann*
  - Profiling: Writing to Cassandra = expensive (CPU intensive) *on the client*
  - Less CPU time to process events = backlog of events = 🔥

- End goal: reduce CPU-time spent writing to Cassandra

- Batching an applicable solution for our use-case

Micro-batching

- Insert data using small batches of `Statements`

- Improves throughput
- Reduces network overhead

- Less is more

Partition-aware Batching

- Batches of `Statement`s, where each batch contains statements for the same partition

- `LoadBalancingPolicy.newQueryPlan(…)`
  - Returns `Iterable<Host>`, in order of preference
  - For `TokenAwarePolicy`, returns the replicas responsible for the Statement’s partition key

- Group by the `first()` (most-preferable) host of each statement’s query plan

  -
    ```
    Multimap<Host, Statement> groups = Multimaps.index(statements,
        s -> lbp.newQueryPlan(s.getKeyspace(), s).next()
    );
    ```
    eqiv. to `Map<Host, List<Statement>>`

CASSANDRA SUMMIT 2016

Benchmark

Insert *1 million* rows

instaclustr

Benchmark Overview

- Write 1 million rows via a variety of strategies:
  - Individual statements, batches & host-aware batches
  - Batch sizes: 10, 50, 100, 250, 500 individual `INSERT`s
  - Consistency levels: `ALL`, `LOCAL_QUORUM`, `LOCAL_ONE`
- `UNLOGGED` batches
- Each strategy executed 10 times — average + std. dev.
- Determine:
  - Fastest
  - Lowest client CPU usage
  - Lowest cluster CPU usage

Write Strategies

Individual Writes

Each `PreparedStatement` is submitted to Cassandra via `executeAsync(…)`

Batch Writes

Groups of *n* statements are combined into `BatchStatement`s, one per group.
Each `BatchStatement` is submitted via `executeAsync(…)`

Partition Aware Batch Writes

Groups of *n* statements, where every statement in the group shares the same partition key, are combined into `BatchStatement`s, one per group. Submitted via `executeAsync(…)`

Timing

The stopwatch is stopped once `ResultSetFuture.get()` unblocks for all futures returned by `executeAsync(…)`

Runtime

Wall-clock time — `System.nanoTime()` via Guava's `Stopwatch`

CPU Time

Total CPU time — `OperatingSystemMXBean.getProcessCpuTime()`
Sum of the CPU time used by all threads in the JVM. Does not include I/O wait time.

Cluster Average CPU Time

Average of the Total CPU time across all nodes in the cluster — collected via JMX

Benchmark Setup

Cluster

- Apache Cassandra 3.7
- 9 node cluster (3 racks, 3 nodes per rack)
  - m4.large — 250GB of EBS SSD, 2 vCPUs, 8 GB RAM
- `NetworkTopologyStrategy`, RF = 3
- `cassandra.yaml`:
  - Increased `batch_size_error_threshold_in_kb` (and `batch_size_warn_threshold_in_kb` to reduce log noise)
  - Increased `write_request_timeout_in_ms`
- Disabled compactions (`nodetool disableautocompaction`) & auto-snapshots
  Reduce variance between benchmark runs due to background tasks.

Benchmark Setup cont’d

Client

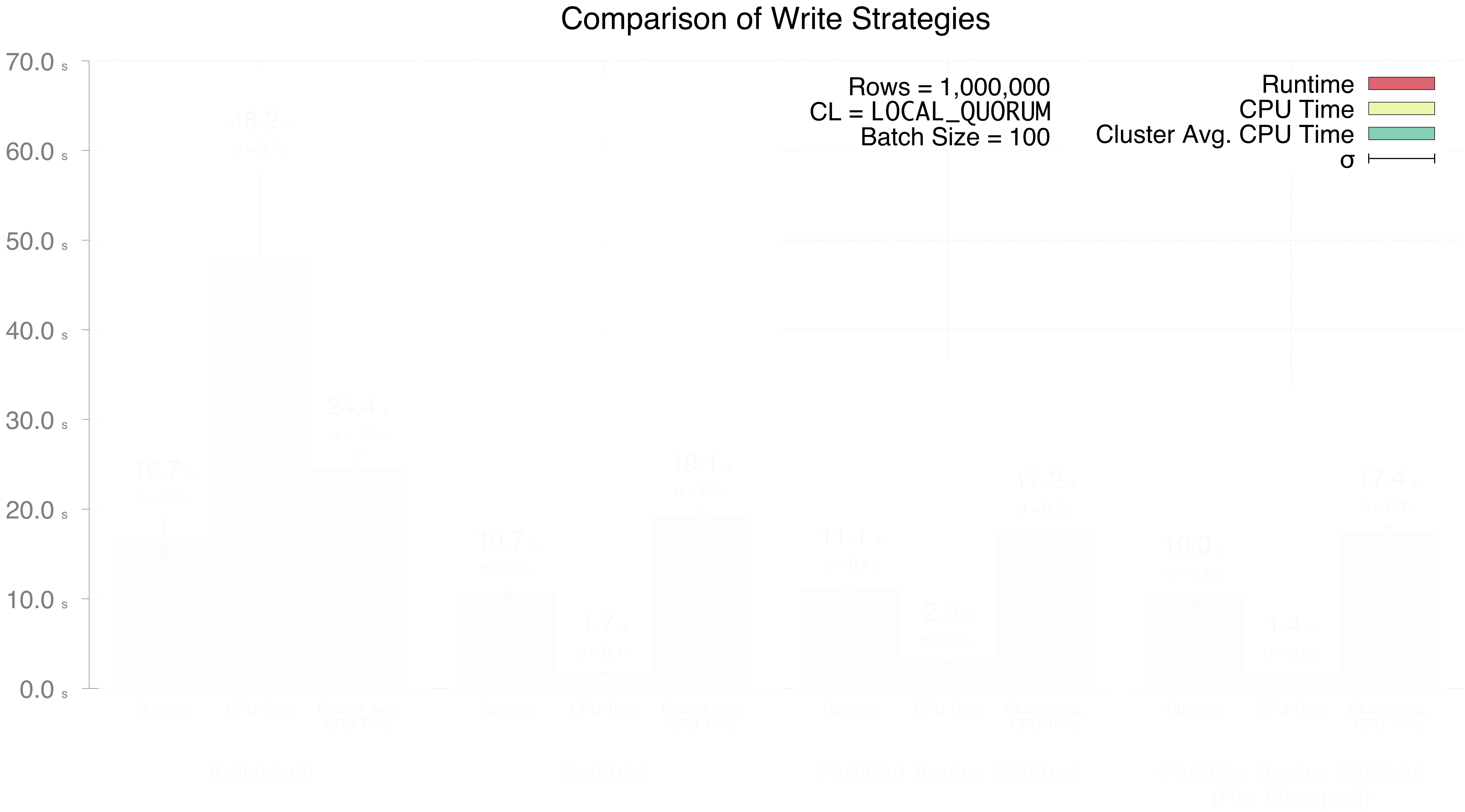
- *Single* c4.xlarge — 7.5GB RAM, 4 vCPUs
- OpenJDK 1.8, DataStax Cassandra Java Driver 3.1
- `TokenAwarePolicy` + `DCAwareRoundRobinPolicy`
- Tweaked connection pool & socket options:
  - Max requests per connection: 32,000 for `LOCAL` & `REMOTE`
  - Max connections per host: 20 for `LOCAL` & `REMOTE`
  - Pool timeout: 50 sec
  - Socket read timeout: 50 sec
- Upfront `prepare(…)` & `bind(…)` all 1 million statements

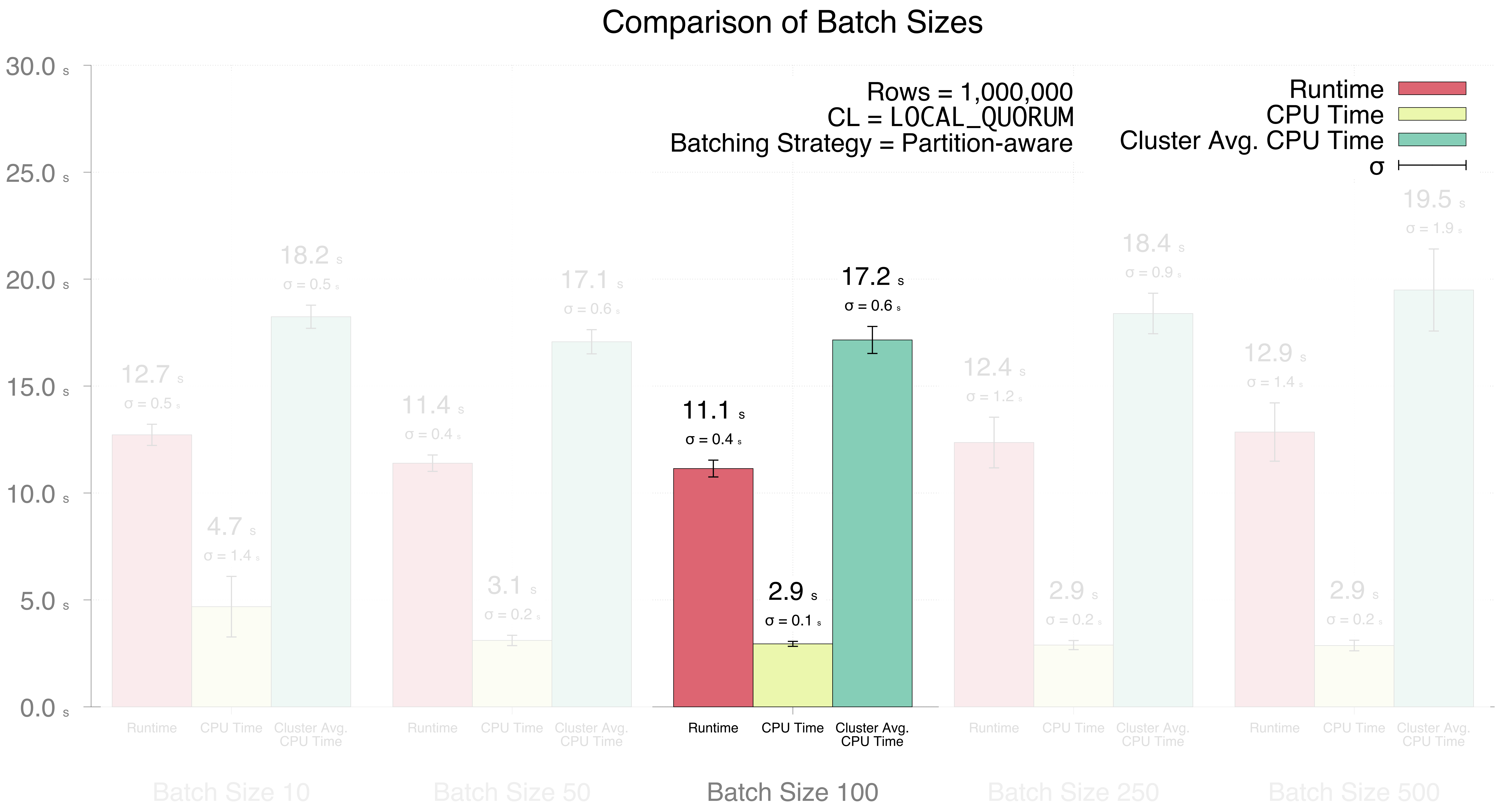
Benchmark Setup cont'd

Table/CF & Data

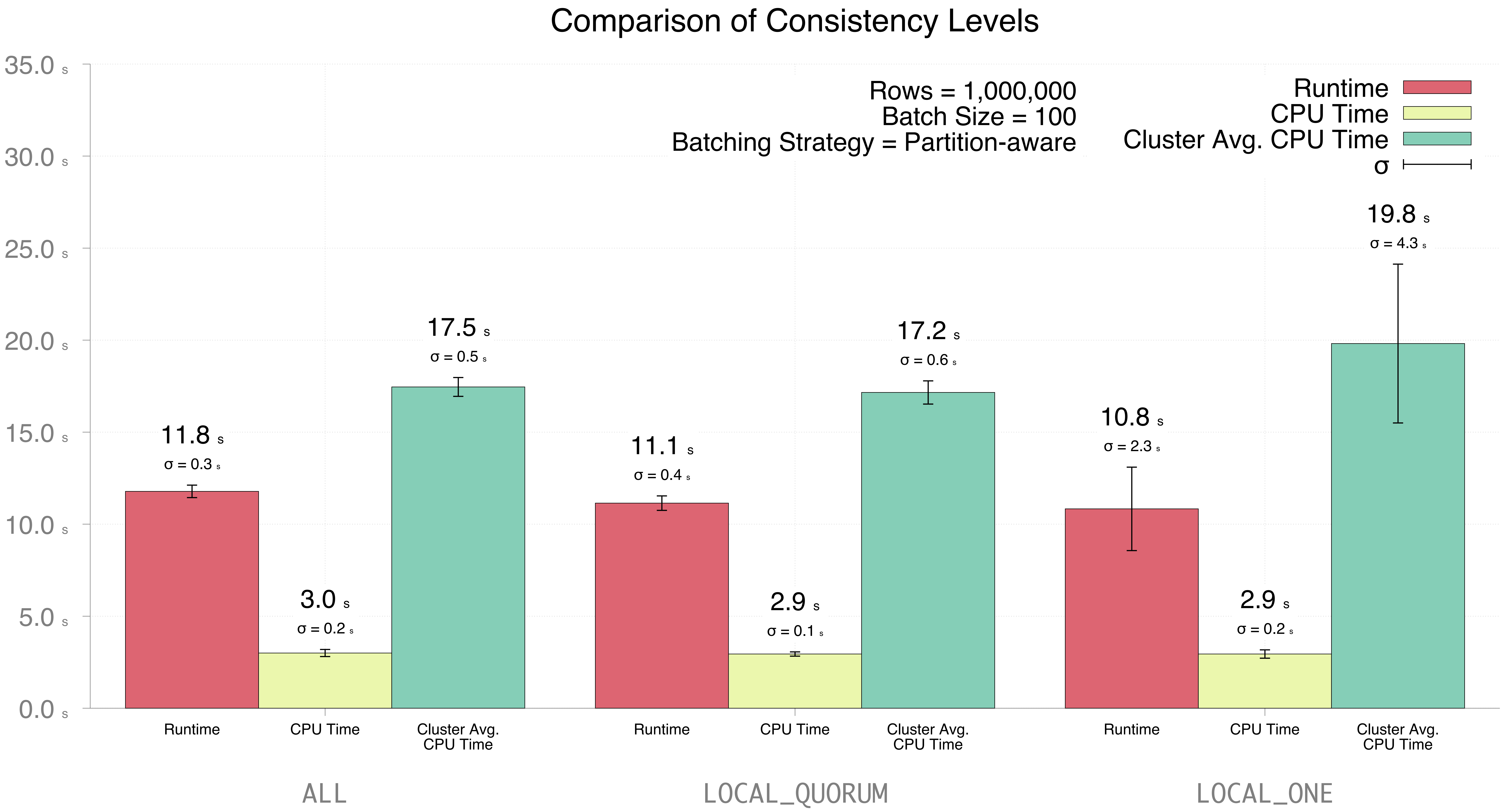
- Resembles our Riemann schema
- Keyspace DROPed and re-CREATEd every benchmark run
- Generate:
  - 1000 random hosts (UUIDs) × 1000 random services (UUIDs)
  - Static values for bucket_time, time, metric & state

```
CREATE TABLE microbatching.example (
    host text,
    bucket_time timestamp,
    service text,
    time timestamp,
    metric double,
    state text,
    PRIMARY KEY ((host, bucket_time, service), time)
) WITH CLUSTERING ORDER BY (time DESC);
```


Comparison of Write Strategies
Rows = 1,000,000
CL = LOCAL_QUORUM
Batch Size = 100
Runtime
CPU Time
Cluster Avg. CPU Time
σ
70.0 s
60.0 s
50.0 s
40.0 s
30.0 s
20.0 s
10.0 s
0.0 s
16.7 s
σ = 2.3 s
48.2 s
σ = 9.7 s
24.4 s
σ = 1.6 s
10.7 s
σ = 0.4 s
1.7 s
σ = 0.1 s
19.1 s
σ = 0.6 s
11.1 s
σ = 0.4 s
2.9 s
σ = 0.1 s
17.2 s
σ = 0.6 s
10.0 s
σ = 0.6 s
1.4 s
σ = 0.2 s
17.4 s
σ = 0.7 s
Runtime
CPU Time
Cluster Avg. CPU Time
Individual
Batches
Partition Aware Batches
Partition Aware Batches (Pre Grouped)



Comparison of Batch Sizes

Comparison of Consistency Levels
Rows = 1,000,000
Batch Size = 100
Batching Strategy = Partition-aware
Runtime
CPU Time
Cluster Avg. CPU Time
σ
35.0 s
30.0 s
25.0 s
20.0 s
15.0 s
10.0 s
5.0 s
0.0 s
11.8 s
σ = 0.3 s
3.0 s
σ = 0.2 s
17.5 s
σ = 0.5 s
11.1 s
σ = 0.4 s
2.9 s
σ = 0.1 s
17.2 s
σ = 0.6 s
10.8 s
σ = 2.3 s
2.9 s
σ = 0.2 s
19.8 s
σ = 4.3 s
Runtime
CPU Time
Cluster Avg. CPU Time
Runtime
CPU Time
Cluster Avg. CPU Time
Runtime
CPU Time
Cluster Avg. CPU Time
ALL
LOCAL_QUORUM
LOCAL_ONE

Outcome

- More-performant
  - Shorter runtime
  - Lower client & cluster CPU load

- Possibly higher latency — more useful for bulk data processing

- Not a silver-bullet
- Standard `INSERT`s work well for most use-cases

- Cassandra benchmarking is hard

Source

The Java source code for this benchmark is available at:

bitbucket.org/adamzegelin/microbatching

CASSANDRA SUMMIT 2016

Thank You!

Questions?

instaclustr