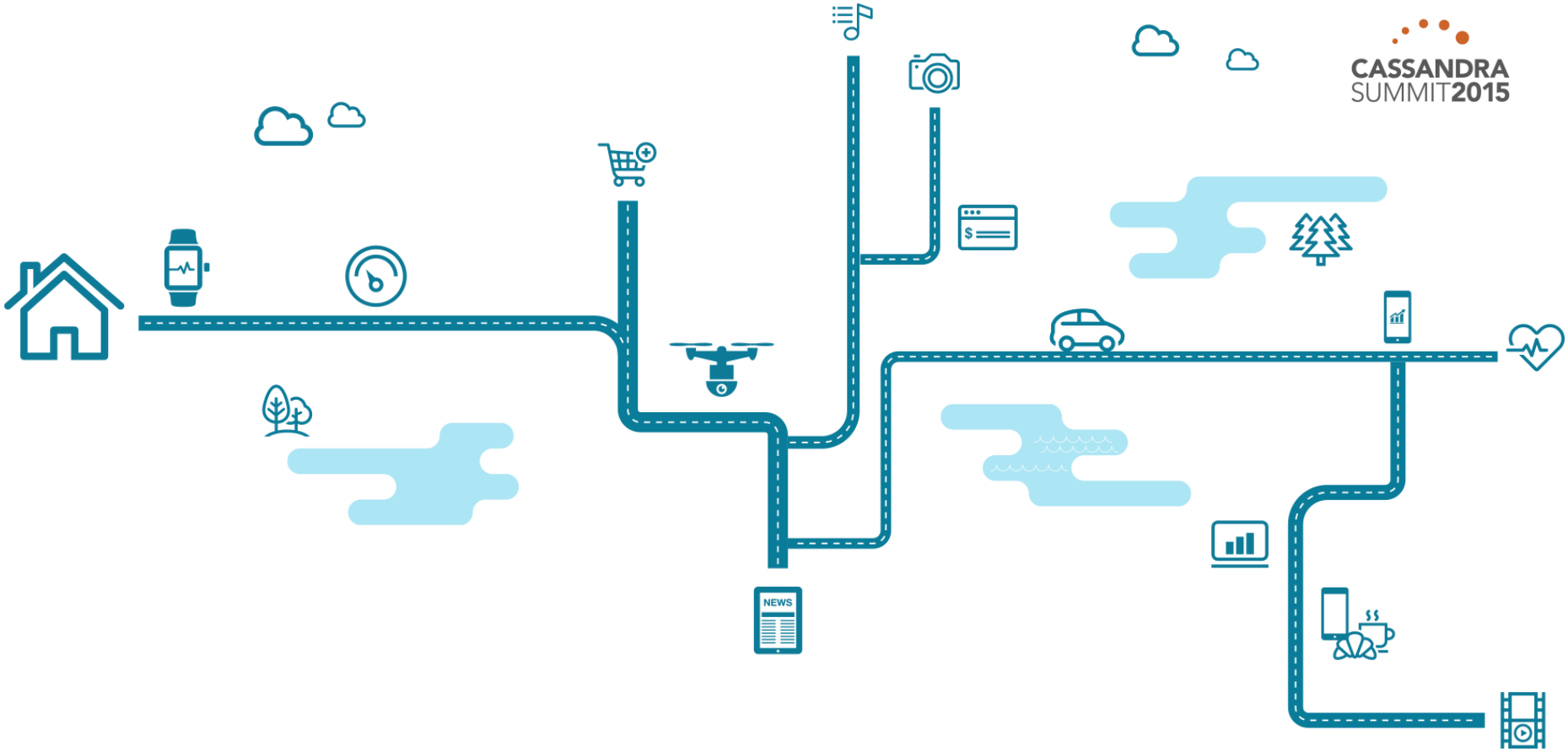


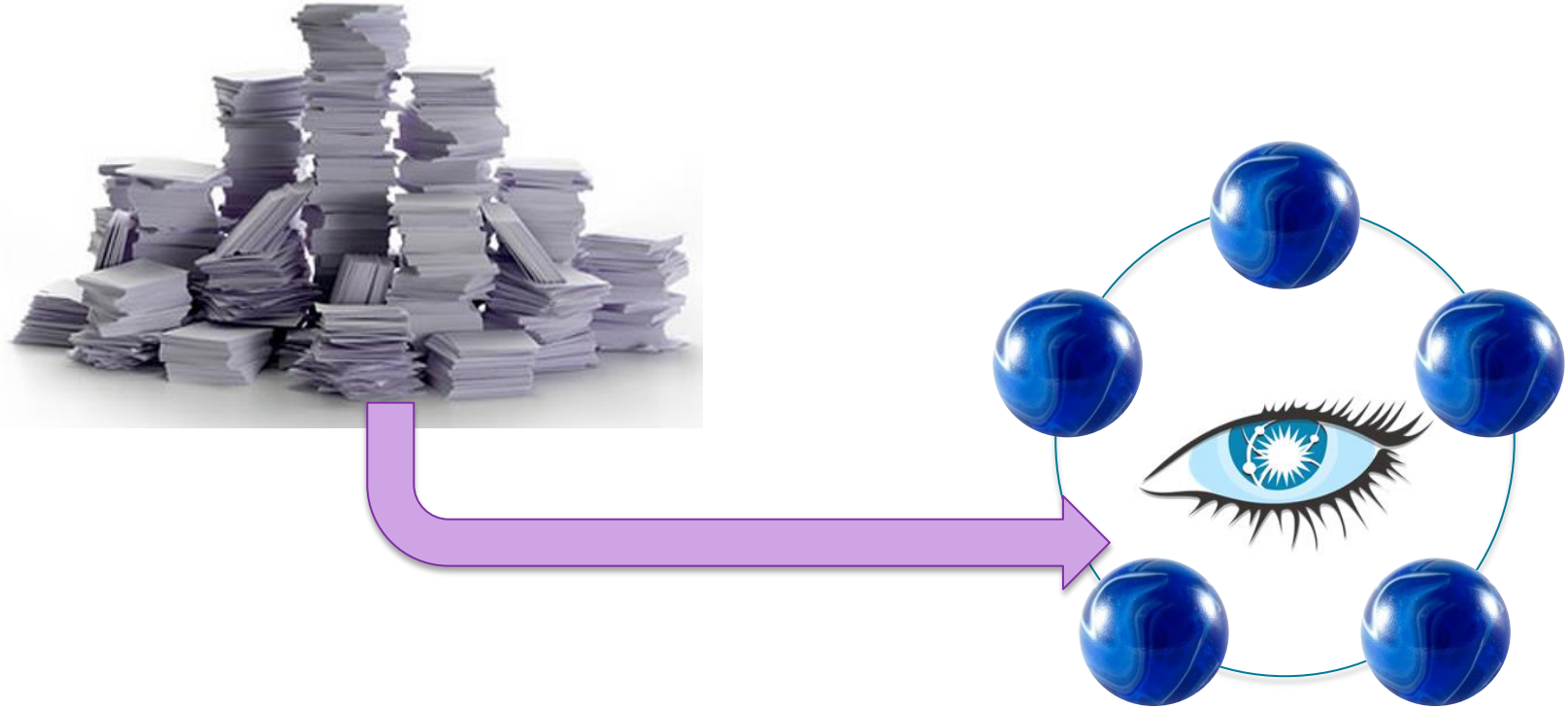
Bulk Loading into Cassandra

What are we talking about today?

CASSANDRA SUMMIT2015

- Problem statement
- Possible Solutions
  - cqlsh COPY FROM
  - Custom code using SSTable formatted files
  - Java CQL INSERTs
- Test Results
- Unloading considerations

The problem is simple…



Load a pile of files into Cassandra

CASSANDRA SUMMIT2015

- Where do the files start out?
  - "On my Laptop/Server's local file system"
    - The focus today!
  - "In HDFS (or another DFS)
    - Consider using Spark – not the topic today
  - "In an NFS mount"
    - Consider using Spark – not the topic today

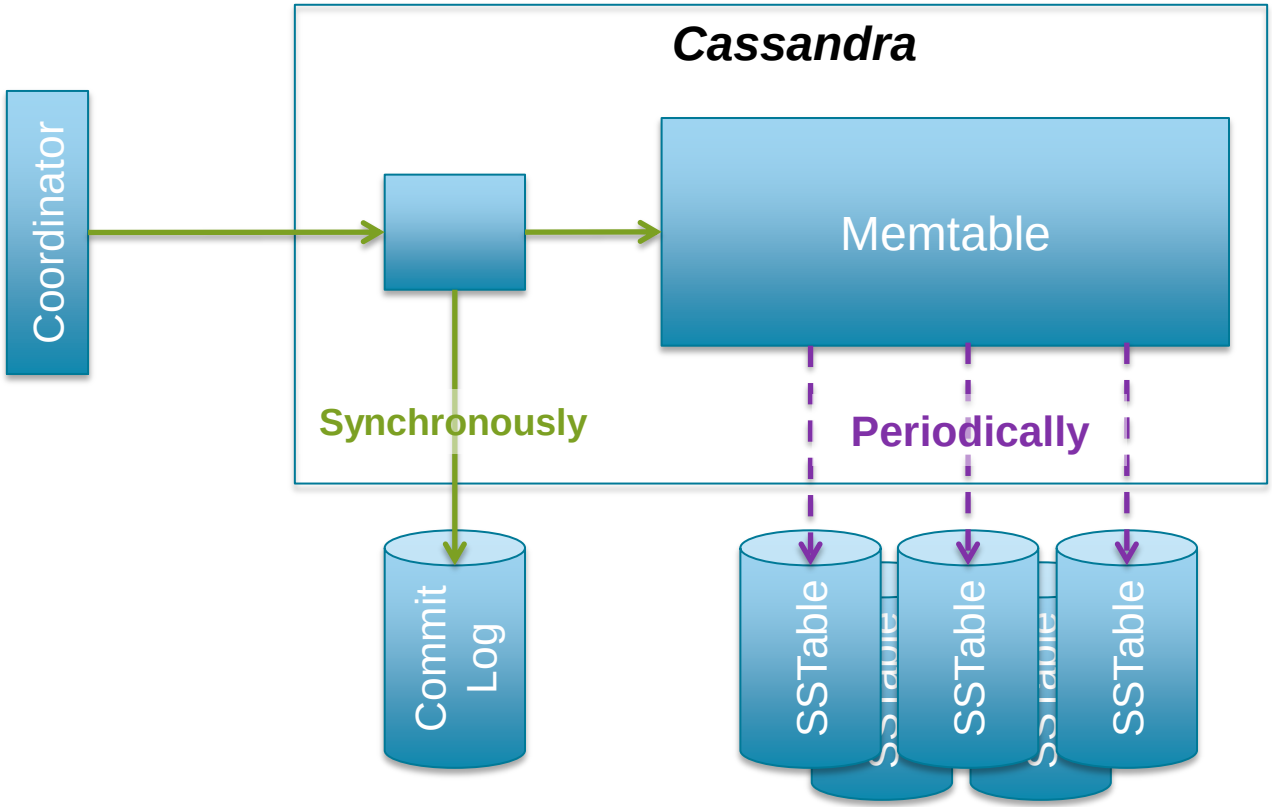
The Options

CASSANDRA SUMMIT2015

- The “Front Door”
  - Cqlsh **`COPY FROM`**
  - Java program loading via INSERT statements and executeAsync()
    - Or the language of your choice: C/C++, Python, C#, etc.
- The “Side Door”
  - Leverage “streaming” via **`sstableloader`**
  - Need to create SSTables via Java and CQLSStableLoader
    - No other language choice

The Front Door: CQL INSERT

Cassandra Write Path



Cassandra Clients

CASSANDRA SUMMIT2015

- Load Balancing
  - Prepared Statements
  - Token-aware routing
  - Round-robin
- Connections per Cassandra host
  - The driver connects to every Cassandra host in the “local” data center
- Synchronous / Asyncrhonous Execution
  - How many “in-flight queries”?
- Consistency Level
- Does not require all nodes to be online
  - Standard Cassandra rules apply – hinted handoff, etc

Cqlsh COPY FROM

CASSANDRA SUMMIT2015

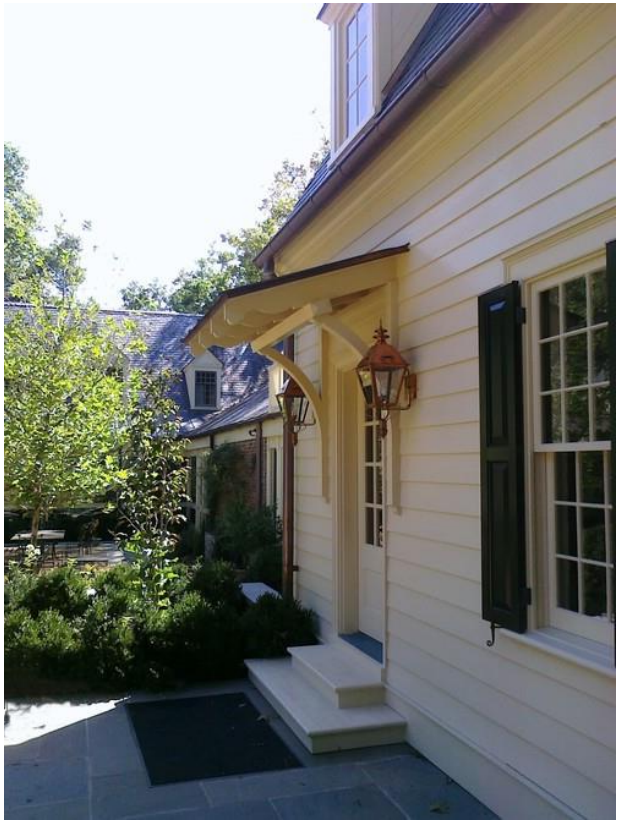
- Command-line CQL tool
- Ships with Cassandra
  - Can be run from a client machine (versions must match)
- Built in Python
  - In 2.1 leverages the Python driver
  - No “token aware routing” yet
- Only makes connection to one coordinator
  - Does not round-robin
- Executes CQL INSERTs asynchronously

Java client – e.g., cassandra-loader
(https://github.com/brianmhess/cassandra-loader)

CASSANDRA SUMMIT2015

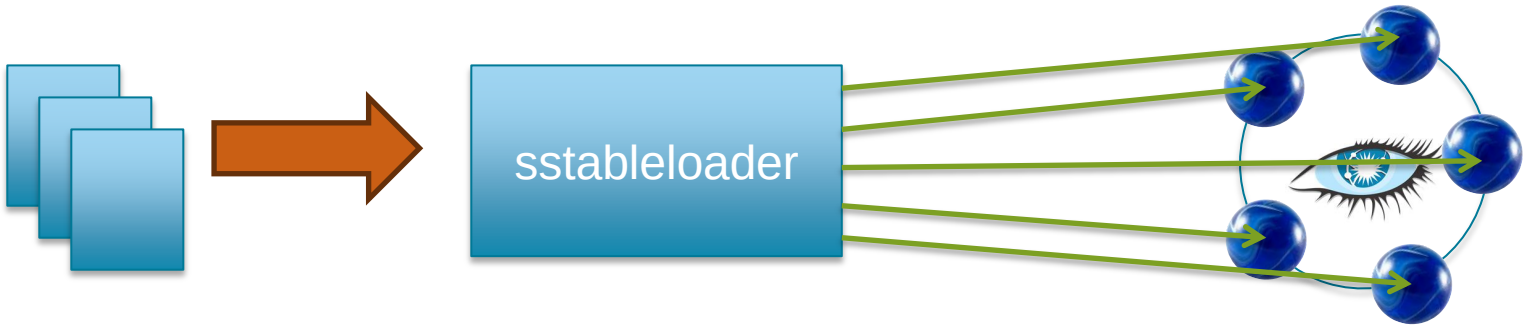
- Java program leveraging the Java CQL driver
  - Java driver (and others) provided by DataStax
- Connects to every node in the cluster
  - Potentially multiple times per node
- Variety of driver options
  - Load balancing – TokenAwarePolicy, DCAwareRoundRobinPolicy, etc
  - Connections per host
  - Consistency Level, etc
- Asynchronous execution
  - Or Synchronous – e.g., for “DDL” operations
  - Aside: cassandra-loader uses asynchronous execution (no DDL)

The Side Door: “Streaming”



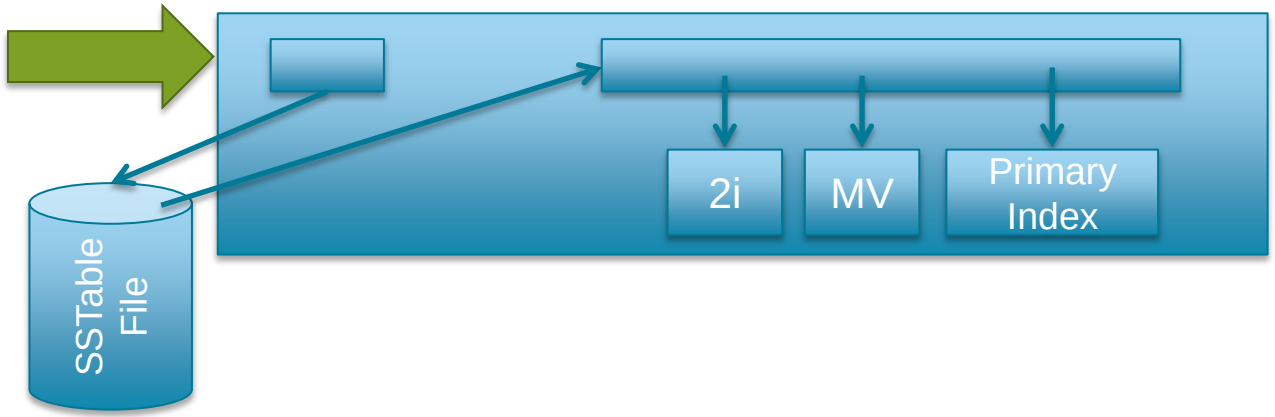
Streaming – the Client

- A connection to each Cassandra node
  - Along with token range information
- For each file
  - Read records
  - Determine which nodes own the token range for this record
  - Send the record to those nodes



Streaming – the Cluster

- Receive records from the client
  - First write out to SSTable file
  - Read file back in to create various Cassandra objects
    - The “Primary Index” – in memory index for “shortcuts”
    - Any secondary indices defined on this table
    - Any materialized views defined on this table (in 3.0)
  - Move on to next SSTable file and repeat



Streaming

CASSANDRA SUMMIT2015

- Streaming requires all nodes to be online
  - Because sstableloader will connect to each node
- Can “blacklist” nodes to skip
  - sstableloader will not stream to those nodes
  - Must know which nodes up front – via **`nodetool status`**, say
  - SSTables won’t be streamed later to offline nodes when they come online
    - No “streaming hints”
- To get data to offline nodes, you must repair

- Streaming also requires SSTables to start with
  - Use CQLSSTableWriter Java class to create SSTables

The test

CASSANDRA SUMMIT2015

- Delimited files
  - Different size rows: 100 bytes, 1KB, 10KB, 1MB
  - Same “schema”: 12-byte TEXT, 8-byte BIGINT, rest in a TEXT
    - 12-byte TEXT is the partition key, BIGINT is unique and the clustering column
  - Each file is 1GB – 20 files to load
    - Larger rows means fewer rows-per-file
  - Parallel execution of commands is allowed – use all the cores
- Hardware/Software
  - 8x i2.2xlarge nodes for Cassandra running DSE 4.7.3
  - 1x r3.xlarge node as the client

The Contenders

CASSANDRA SUMMIT2015

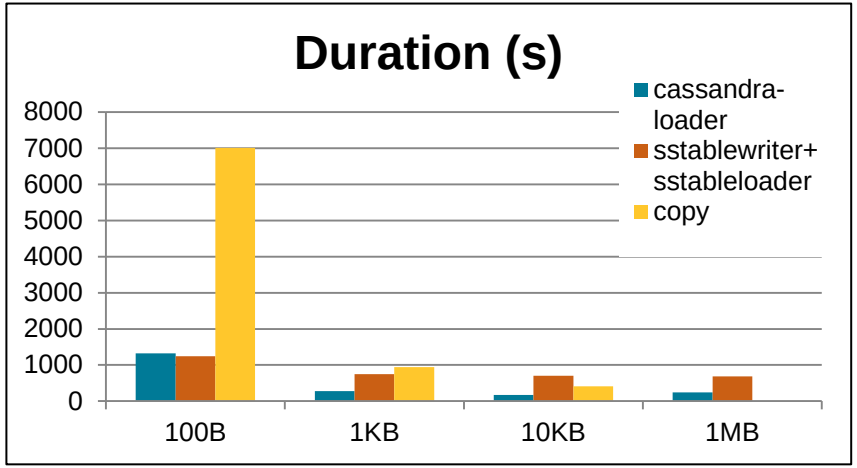
1. CQLSSTableWriter + sstableloader
   - Wrote a Java program to take delimited files to SSTables
   - Use command-line sstableloader to load
   - Need to combine the times of both
2. Cqlsh COPY FROM
   - Use DSE 4.7.3 (not started) on the client
3. cassandra-loader
   - https://github.com/brianmhess/cassandra-loader
   - Java CQL driver client

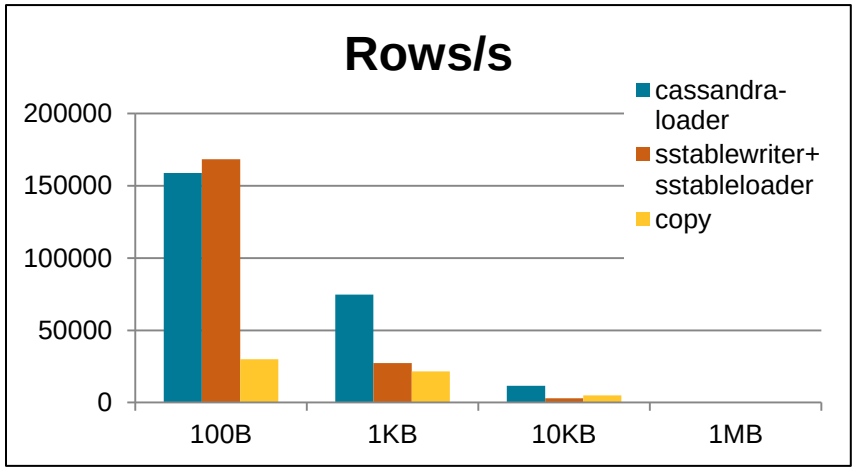
Experiment Execution Details

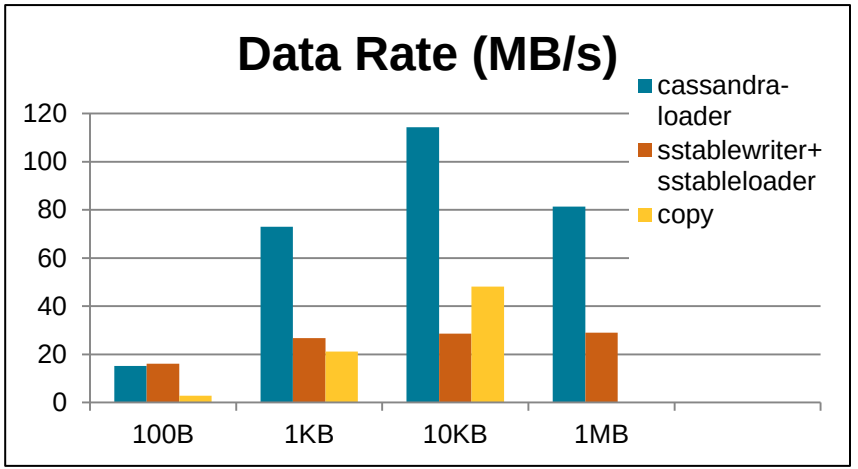
CASSANDRA SUMMIT2015

- CQLSSTableWriter + sstableloader
  - Leverage “make -j 8” to run 8 at a time
  - Time CQLSSTableWriter and then time sstableloader
- Cqlsh COPY FROM
  - Leverage “make -j 2” to run 2 at a time
  - Running more than 2 caused timeouts and errors
- cassandra-loader
  - 8 threads (“-numThreads 8”)
  - unlogged batches of size 4 (“-batchSize 4”)
  - 10000 queries in flight (“-numFutures 10000”)
  - Exception for 100-byte test
    - 10 threads
    - unlogged batches of size 20
    - 50000 queries in flight

Results







Observations

CASSANDRA SUMMIT2015

- Java executeAsync() was faster in all tests
  - Except the 100-byte test, where it was a close second
  - cassandra-loader means no custom code
- CQLSSTableWriter+sstableloader works better for smaller records
  - Performance eroded as record size increased
  - Custom Java program for each format
- Cqlsh COPY FROM was never the winner
  - Second place in the 10KB/row test
  - Could not handle the 1MB/row test – ERROR

To Batch or Not To Batch

CASSANDRA SUMMIT2015

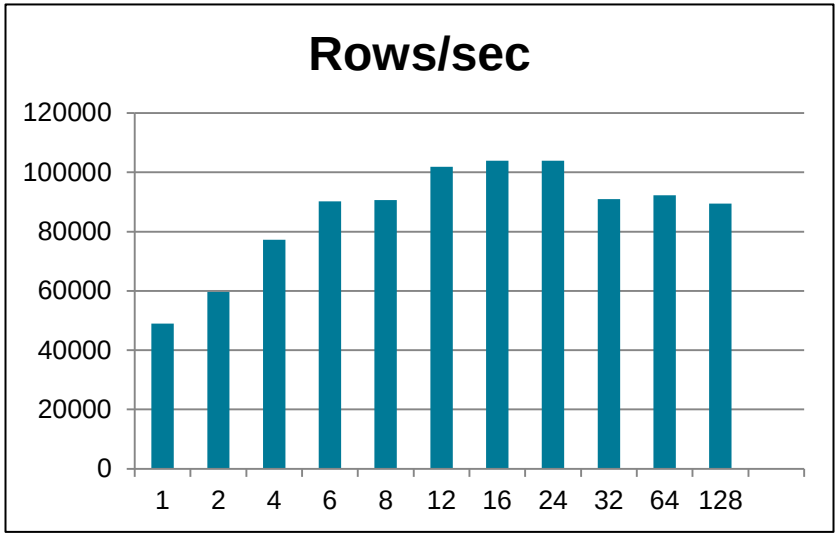
- Varying opinions on unlogged batches
- Batching puts more load on the coordinator
  - The coordinator gets the list of INSERTs and executes each one
  - Not all INSERTs will be “owned” by the coordinator
  - Essentially, the client offloads work to the coordinator
- Batching means fewer queries to the cluster
  - Since the INSERTs are bundled into one query

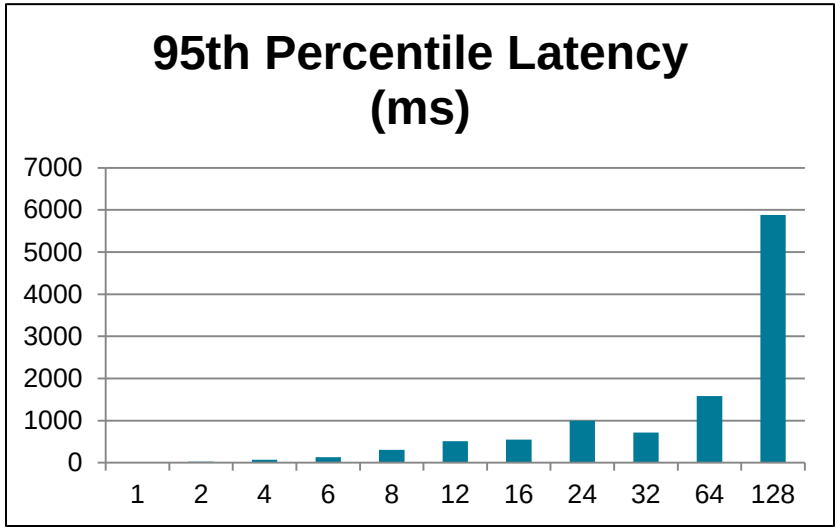
Batch test

CASSANDRA SUMMIT2015

- 10 delimited files
  - 10 BIGINT columns – one is partition key, one is clustering column
- Use cassandra-loader
  - Vary the `-batchSize` argument
- Measure
  - Throughput – Rows/sec
  - Latency – 95th Percentile

Results





- Observation: Increasing batch size
  - Increases throughput (to a point)
  - Increases latency
- Neither is surprising…

Bulk Unloading

CASSANDRA SUMMIT2015

- The Problem
  - Get all that data from Table X out to file(s)

- Refinement
  - Where's the data going?
    - Local FS
    - Distributed FS (e.g., HDFS) – Use Spark (or Hadoop, if you have to)?

- The “Front Door”
  - CQL SELECT

- There is no “Side Door”

CASSANDRA SUMMIT2015

Parallel unload

CASSANDRA SUMMIT2015

- Split token range into pieces
  - Need the set of splits to cover and not overlap
    - Cassandra drivers provide that
  - Need each split to be completely within one node
    - So each extract is able to talk only to one Cassandra node
    - Optimization step – not *necessary* ☺
  - Same approach as Spark and Hadoop (and others)
- Connection / Query per “split”
  - Export to a different file
- Optimize paging size
  - Reduce overhead for decompression
- Consistency Level

Available Tools

CASSANDRA SUMMIT2015

- Cqlsh COPY TO
  - Built into Cassandra command-line tool **`cqlsh`**
  - Leverages the Python driver
  - Recent improvements: CASSANDRA-9304
    - Parallel export, etc
- cassandra-unloader
  - Part of the https://github.com/brianmhess/cassandra-loader project
  - Delimited file options, just like cassandra-loader
  - Parallel export

Performance

CASSANDRA SUMMIT2015

- From the CASSANDRA-9304 ticket:

  “A small benchmark was done on a table of 10M rows inside of a Vagrant box with 8 cores. The table was created using the following command `tools/bin/cassandra-stress write n=10M -rate threads=50`.

  The original single proc version took about **30 minutes** to export the table. The multi proc version takes about **7 minutes**.
  Brian Hess's cassandra-unloader takes a little over **2 minutes**.”

- Summary:
  - Pre-9304 COPY TO: 30 minutes
  - Post-9304 COPY TO: 7 minutes
  - cassandra-unloader: 2 minutes

Summary

CASSANDRA SUMMIT2015

- Bulk Loading
  - CQL asynchronous INSERTs are your best bet
    - Simplicity, performance (almost always), configurability, low/no coding
  - CQLSSTableWriter requires a custom Java application
  - sstableloader requires all nodes to be online
    - Operational consideration
- Batching
  - Can improve throughput at the cost of latency
- Bulk Unloading
  - Parallel export via splitting token range
  - Use CQL, there is no “side door”

Thank you