Maximum Overdrive: Tuning the Spark Cassandra Connector

Russell Spitzer, Datastax

Who is this guy and why should I listen to him?

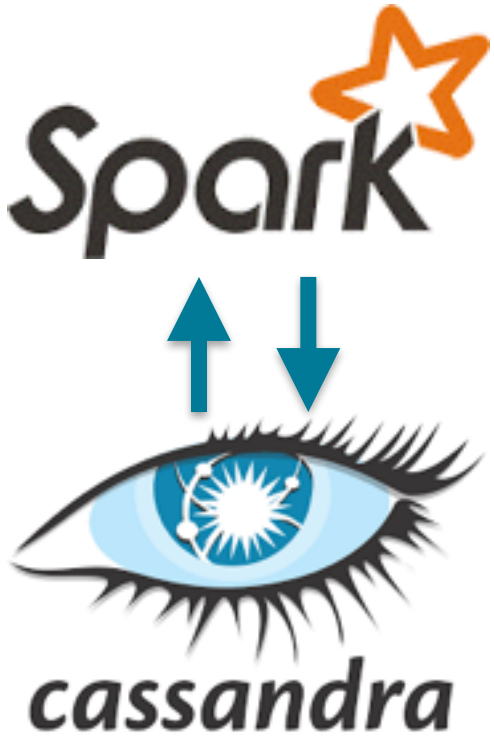
Russell Spitzer, Passing Software Engineer

- Been working at DataStax since 2013
- Worked in Test Engineering and now Analytics Dev
- Working with Spark since 0.9
- Working with Cassandra since 1.2
- Main focus: the Spark Cassandra Connector
- Surgically grafted to the Spark Cassandra Connector Mailing List

The Spark Cassandra Connector Connects Spark to Cassandra

It's all there in the name

- Provides a DataSource for Datasets/Data Frames
- Provides methods for Writing DataSets/Data Frames
- Reading and Writing RDD
- Connection Pooling
- Type Conversions and Mapping
- Data Locality
- Open Source Software!

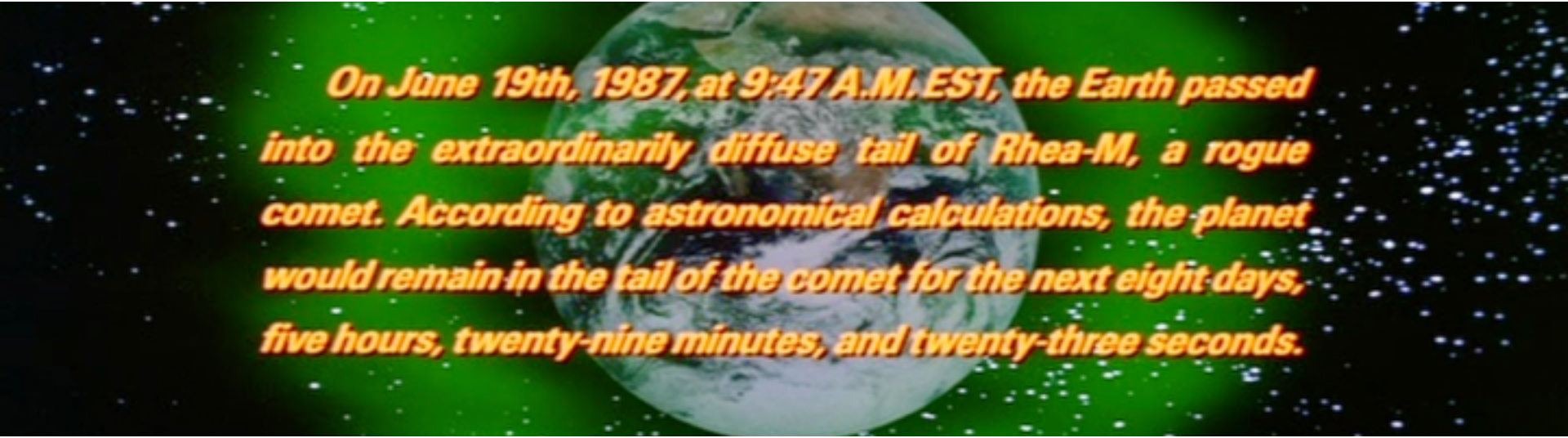


https://github.com/datastax/spark-cassandra-connector

Tuning the Spark Cassandra Connector

1	Lots of Write Tuning
2	A Bit of Read Tuning

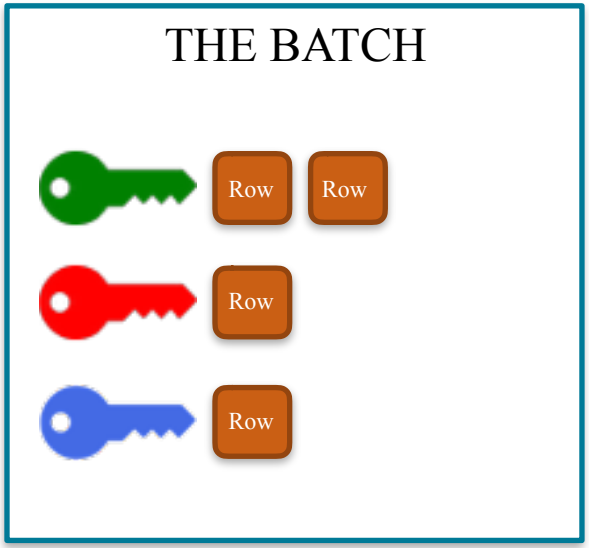
Context is Very Important



Knowing your Data is Key for Maximum Performance

Write Tuning in the SCC Is all about Batching

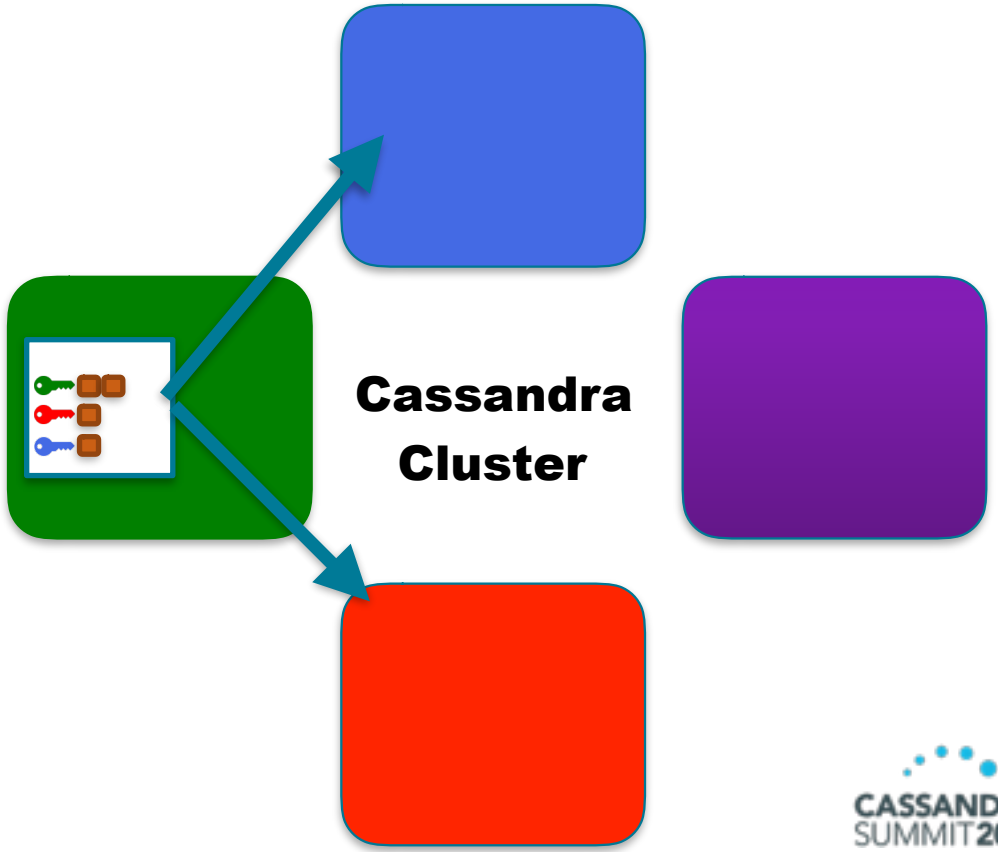
Multi-Partition Key Batches put load on the Coordinator



Multi-Partition Key Batches put load on the Coordinator

A batch moves as a single entity to the Coordinator for that write

This batch has to sit there until all the portions of it get confirmed at their set consistency level

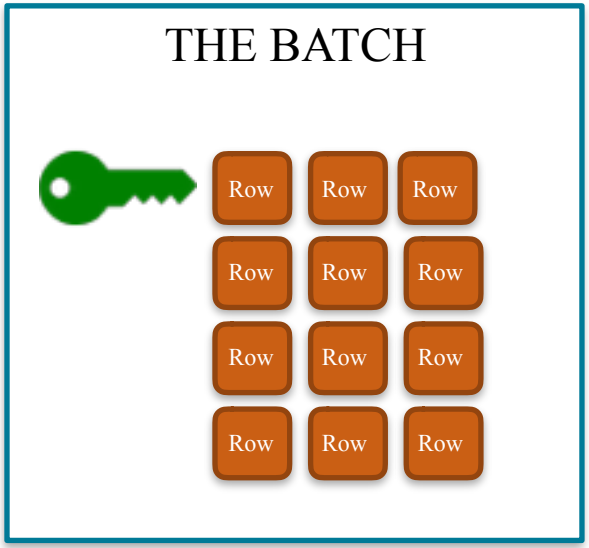


Multi-Partition Key Batches put load on the Coordinator

Even when some portions of the batch finish early we have to wait until the entire thing is done before we can respond to the client.

We end up with a lot of rows just sitting around in memory waiting for others to get out of the way

Single Partition Batches are Treated as A Single Mutation in Cassandra



Single Partition Batches are Treated as A Single Mutation in Cassandra

Now the entire batch can be treated as a single mutation. We also only have to wait for one set of replicas

When all of the Rows are Going to the Same Place Writing to Cassandra is Fast

The Connector Will Automatically Batch Writes

RDD

```
rdd.saveToCassandra("bestkeyspace", "besttable")
```

DataFrame

```
df.write
  .format("org.apache.spark.sql.cassandra")
  .options(Map("table" -> "besttable", "keyspace" -> "bestkeyspace"))
  .save()
```

```
import org.apache.spark.sql.cassandra._

df.write
  .cassandraFormat("besttable", "bestkeyspace")
  .save()
```

By default batching happens on Identical Partition Key

Change it as a SparkConf or DataFrame Parameter

`output.batch.grouping.key`	Partition	Determines how insert statements are grouped into batches. Available values are • `none` : a batch may contain any statements • `replica_set` : a batch may contain only statements to be written to the same replica set • `partition` : a batch may contain only statements for rows sharing the same partition key value

Or directly pass in a WriteConf

```
WriteConf(batchGroupingKey= ?)
```

https://github.com/datastax/spark-cassandra-connector/blob/master/doc/reference.md#write-tuning-parameters

Batches are Placed in Holding Until Certain Thresholds are hit

https://github.com/datastax/spark-cassandra-connector/blob/master/doc/reference.md#write-tuning-parameters

Batches are Placed in Holding Until Certain Thresholds are hit

output.batch.grouping.buffer.size
output.batch.size.bytes / output.batch.size.rows
output.concurrent.writes
output.consistency.level

https://github.com/datastax/spark-cassandra-connector/blob/master/doc/reference.md#write-tuning-parameters

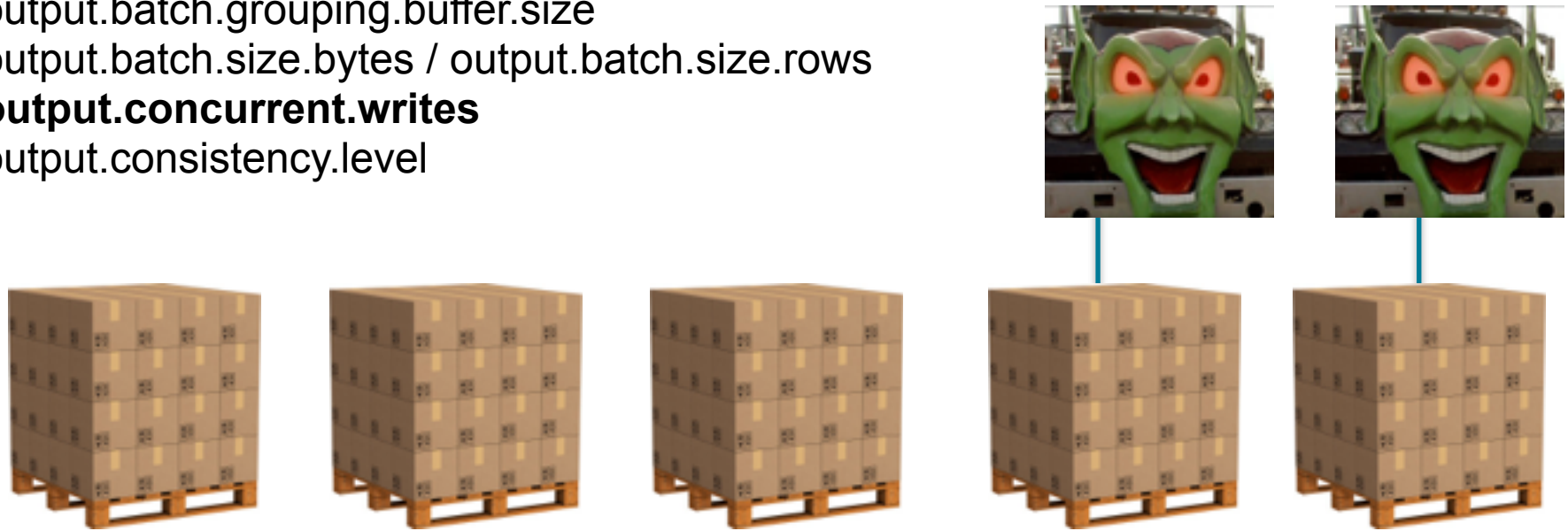
Batches are Placed in Holding Until Certain Thresholds are hit

output.batch.grouping.buffer.size
output.batch.size.bytes / output.batch.size.rows
output.concurrent.writes
output.consistency.level

https://github.com/datastax/spark-cassandra-connector/blob/master/doc/reference.md#write-tuning-parameters

Batches are Placed in Holding Until Certain Thresholds are hit

output.batch.grouping.buffer.size
output.batch.size.bytes / output.batch.size.rows
output.concurrent.writes
output.consistency.level



https://github.com/datastax/spark-cassandra-connector/blob/master/doc/reference.md#write-tuning-parameters

Batches are Placed in Holding Until Certain Thresholds are hit

output.batch.grouping.buffer.size
output.batch.size.bytes / output.batch.size.rows
output.concurrent.writes
output.consistency.level





https://github.com/datastax/spark-cassandra-connector/blob/master/doc/reference.md#write-tuning-parameters

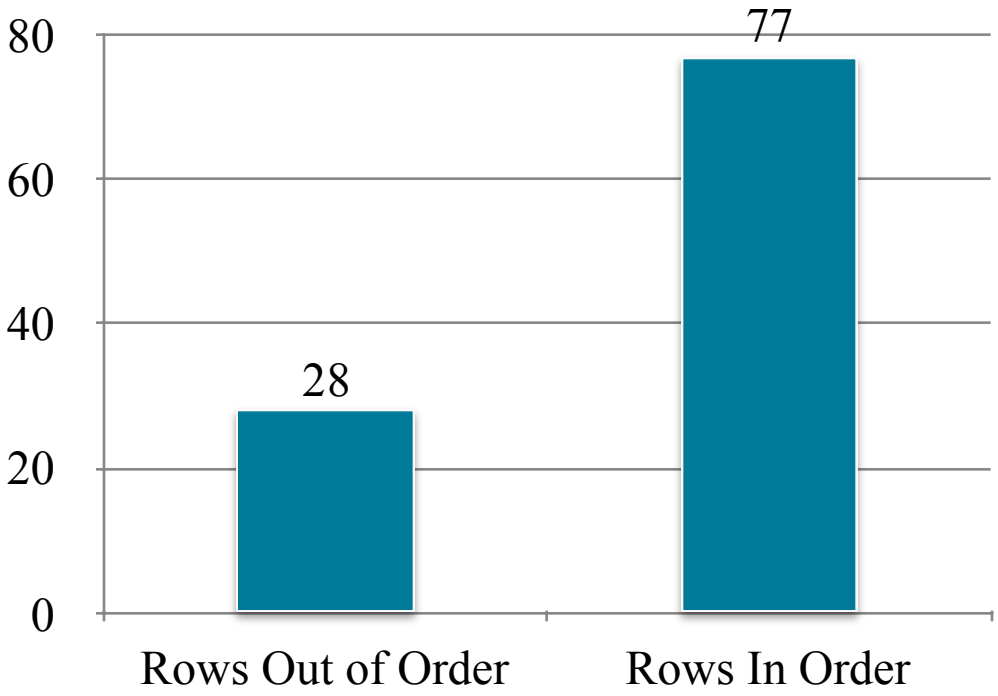
Spark Cassandra Stress for Running Basic Benchmarks

https://github.com/datastax/spark-cassandra-stress

Running Benchmarks on a bunch of AWS Machines,
5 M3.2XLarge
DSE 5.0.1
Spark 1.6.1
Spark CC 1.6.0
RF = 3
2 M Writes/ 100K C* Partitions and 400 Spark Partitions

Caveat: Don't benchmark exactly like this
I'm making some bad decisions to to make some broad points

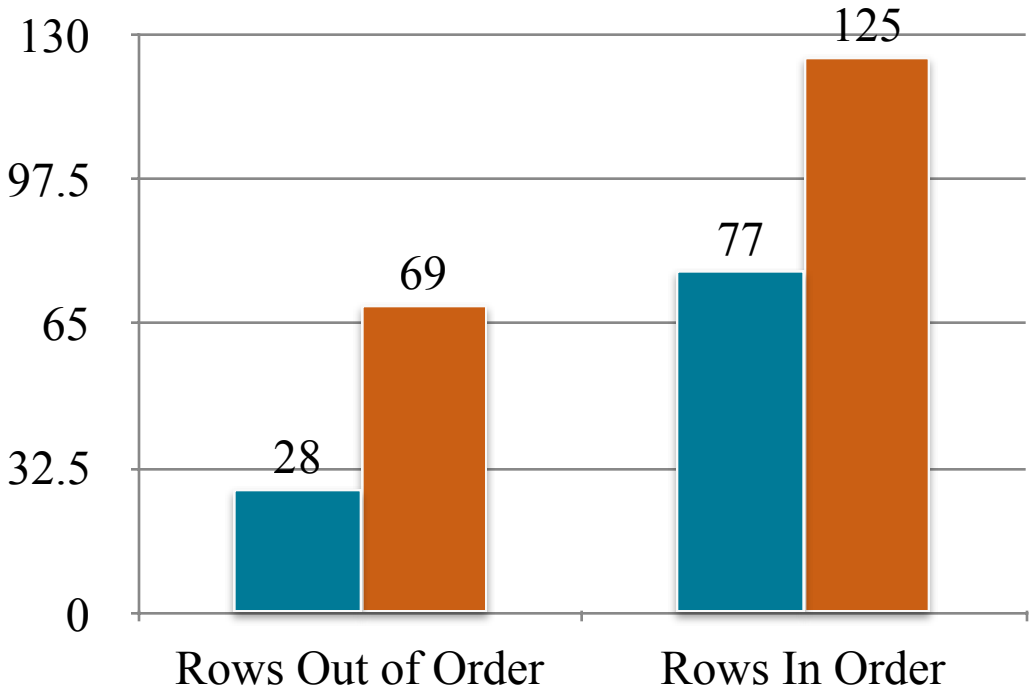
Depending on your use case Sorting within Partitions can Greatly Increase Write Performance



Grouping on Partition Key
The Safest Thing You Can Do

https://github.com/datastax/spark-cassandra-connector/blob/master/doc/reference.md#write-tuning-parameters

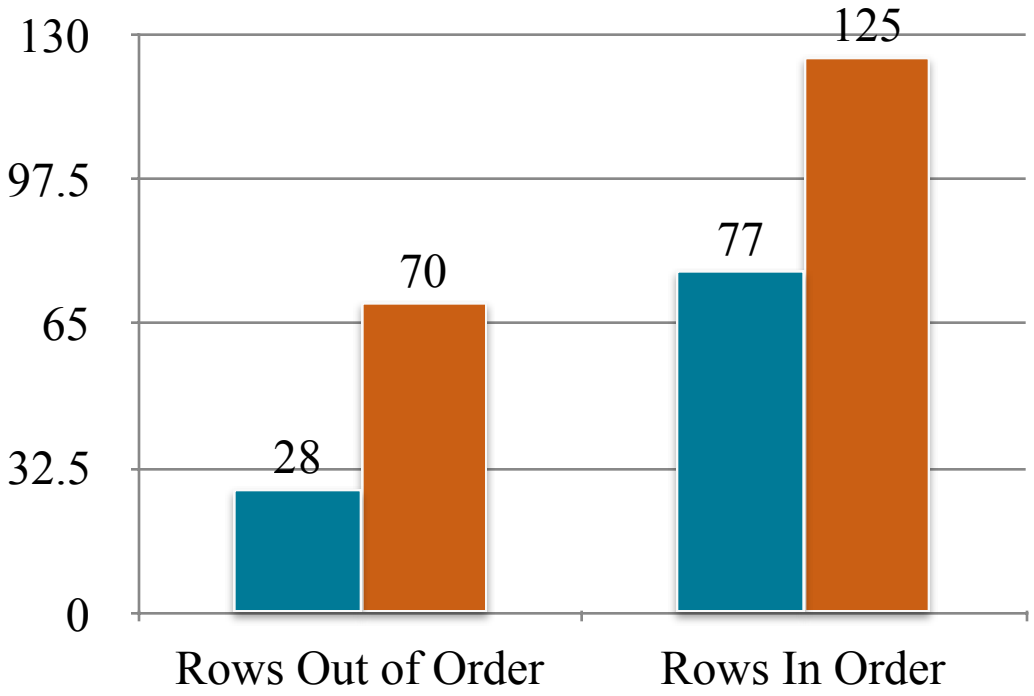
Including everything in the Batch



May be Safe For Short Durations
BUT WILL LEAD TO SYSTEM INSTABILITY

https://github.com/datastax/spark-cassandra-connector/blob/master/doc/reference.md#write-tuning-parameters

Grouping on Replica Set



Safer, But still will put
extra load on the Coordinator

https://github.com/datastax/spark-cassandra-connector/blob/master/doc/reference.md#write-tuning-parameters

Remember the Tortoise vs the Hare

Overwhelming Cassandra will slow you down

Limit the amount of writes per executor : output.throughput_mb_per_sec
Limit maximum executor cores : spark.max.cores
Lower concurrency : output.concurrent.writes

DEPENDING ON DISK PERFORMANCE YOUR INITIAL SPEEDS IN BENCHMARKING MAY NOT BE SUSTAINABLE

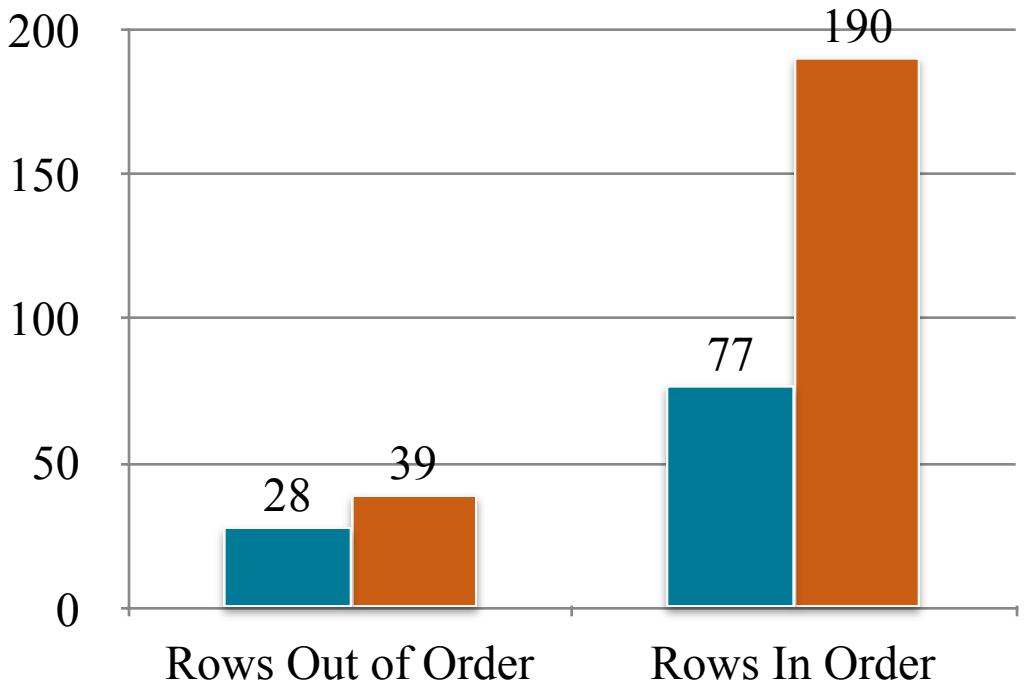
For Example Lets run with Batch Key None for a Longer Test (20M writes)

```
[Stage 0:=========================>                 (191 + 15) / 400]WARN  2016-08-19 21:11:55,817
org.apache.spark.scheduler.TaskSetManager: Lost task 192.0 in stage 0.0 (TID 193, ip-172-31-13-127.us-west-1.compute.internal):
java.io.IOException: Failed to write statements to ks.tab.
        at com.datastax.spark.connector.writer.TableWriter$$anonfun$write$1.apply(TableWriter.scala:166)
        at com.datastax.spark.connector.writer.TableWriter$$anonfun$write$1.apply(TableWriter.scala:134)
        at com.datastax.spark.connector.cql.CassandraConnector$$anonfun$withSessionDo$1.apply(CassandraConnector.scala:110)
        at com.datastax.spark.connector.cql.CassandraConnector$$anonfun$withSessionDo$1.apply(CassandraConnector.scala:109)
        at com.datastax.spark.connector.cql.CassandraConnector.closeResourceAfterUse(CassandraConnector.scala:139)
        at com.datastax.spark.connector.cql.CassandraConnector.withSessionDo(CassandraConnector.scala:109)
        at com.datastax.spark.connector.writer.TableWriter.write(TableWriter.scala:134)
        at com.datastax.spark.connector.RDDFunctions$$anonfun$saveToCassandra$1.apply(RDDFunctions.scala:37)
        at com.datastax.spark.connector.RDDFunctions$$anonfun$saveToCassandra$1.apply(RDDFunctions.scala:37)
        at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:66)
        at org.apache.spark.scheduler.Task.run(Task.scala:89)
        at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:214)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
        at java.lang.Thread.run(Thread.java:745)
```

For Example Lets run with Batch Key None for a Longer Test (20M writes)

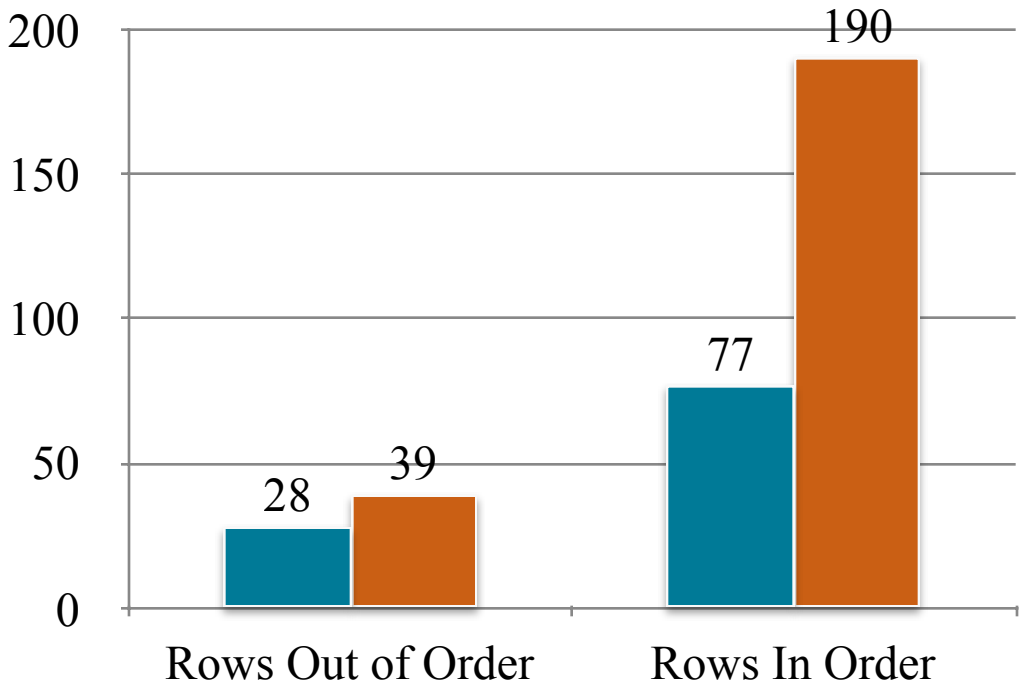
[Stage 0:=========================> (191 + 15) / 400]WARN 2016-08-19 21:11:55,817
org.apache.spark.scheduler.TaskSetManager: Lost task 192.0 in stage 0.0 (TID 193, ip-172-31-13-127.us-west-1.compute.internal):
java.io.IOException: Failed to write
at com.datastax.spa a:166)
at com.datastax.spa a:134)
at com.datastax.spa y(CassandraConnector.scala:110)
at com.datastax.spa y(CassandraConnector.scala:109)
at com.datastax.spa aConnector.scala:139)
at com.datastax.spa tor.scala:109)
at com.datastax.spa
at com.datastax.spa Functions.scala:37)
at com.datastax.spa Functions.scala:37)
at org.apache.spark
at org.apache.spark
at org.apache.spark
at java.util.concurre
at java.util.concurre
at java.lang.Thread.

Back to Default PartitionKey Batching



So why are we doing so much better over a longer run?

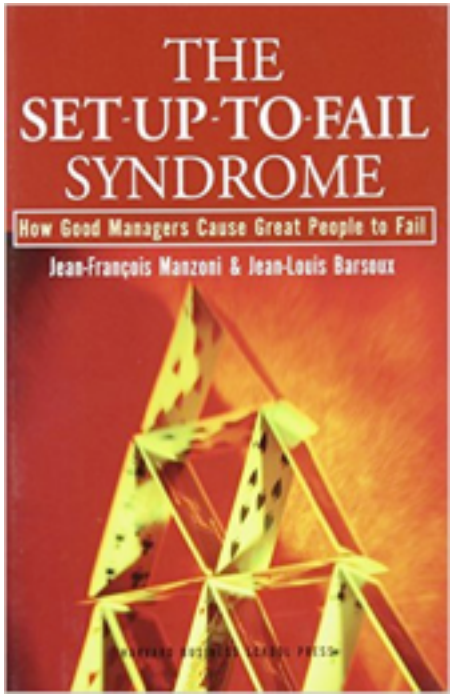
Back to Default PartitionKey Batching



400 Spark Partitions in Both Cases
2M/ 400 = 5000
20M / 400 = 50000

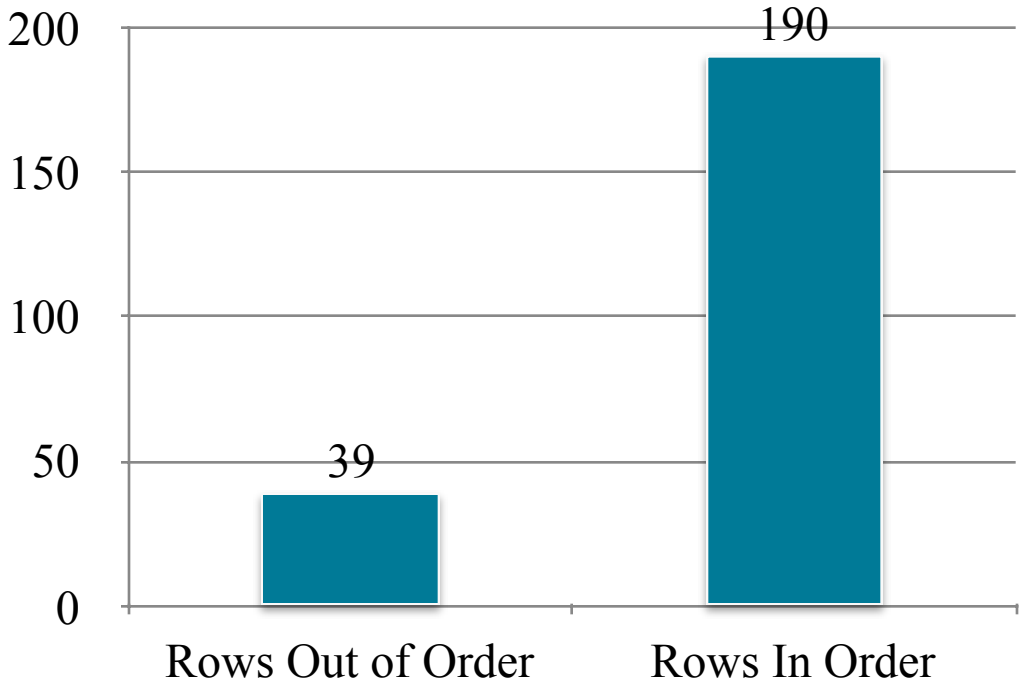
Initial Run kOps/s
10X Length Run

Having Too Many Partitions will Slow Down your Writes



Every task has Setup and Teardown and we can only build up good batches if there are enough elements to build them from

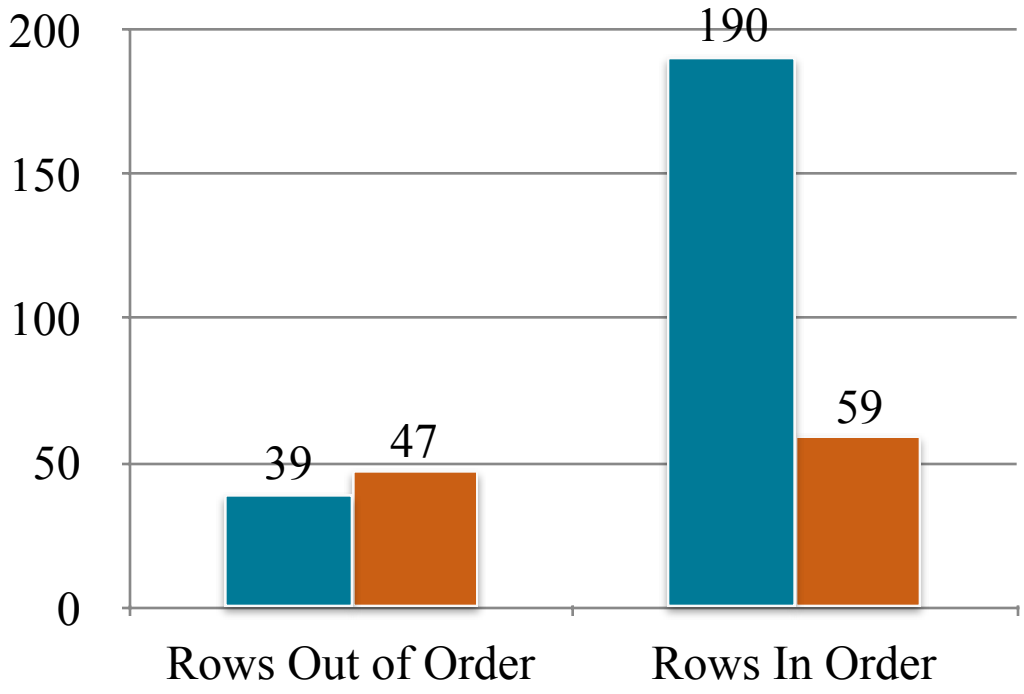
Depending on your use case Sorting within Partitions can Greatly Increase Write Performance



A spark sort on partition key *may* speed up your total operation by several fold

https://github.com/datastax/spark-cassandra-connector/blob/master/doc/reference.md#write-tuning-parameters

Maximizing performance for out of Order Writes or No Clustering Keys



Turn Off Batching

```
spark.cassandra.output.batch.size.rows 1
```

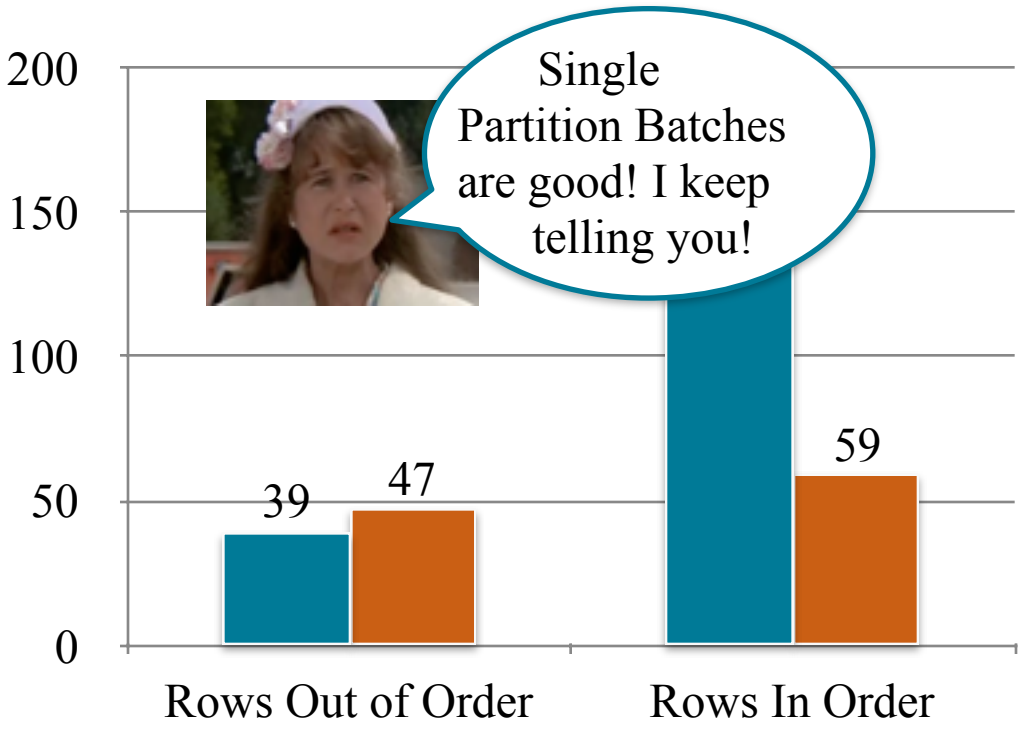
Increase Concurrency

```
spark.cassandra.output.concurrent.writes 2000
```

10X Length Run
Modified Conf kOps/s

https://github.com/datastax/spark-cassandra-connector/blob/master/doc/reference.md#write-tuning-parameters

Maximizing performance for out of Order Writes or No Clustering Keys



Turn Off Batching

```
spark.cassandra.output.batch.size.rows 1
```

Increase Concurrency

```
spark.cassandra.output.concurrent.writes 2000
```

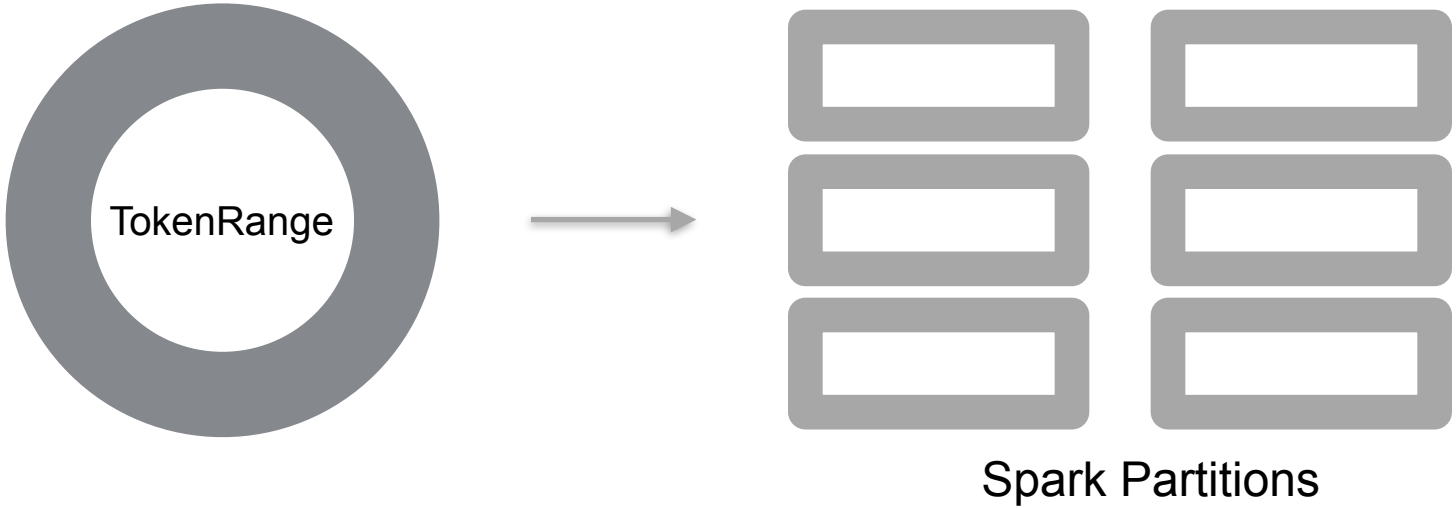
https://github.com/datastax/spark-cassandra-connector/blob/master/doc/reference.md#write-tuning-parameters

This turns the connector into a Multi-Machine Cassandra Loader (Basically just executeAsync as fast as possible)

https://github.com/brianmhess/cassandra-loader

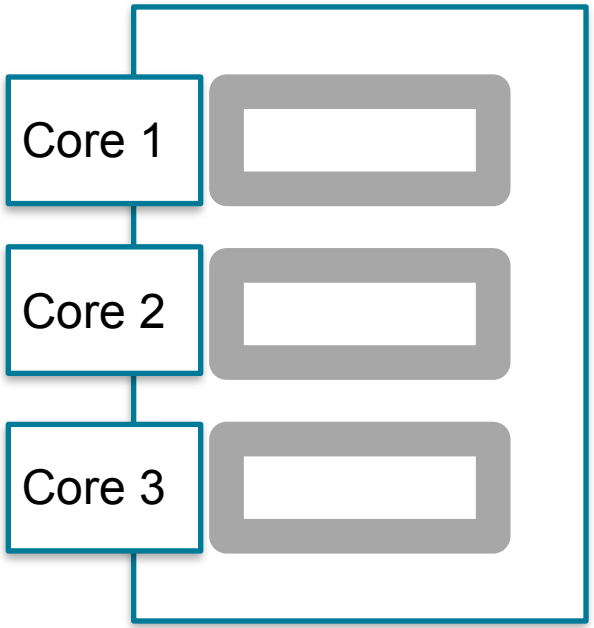
Now Let's Talk About Reading!

Read Tuning mostly About Partitioning



- RDDs are a large Dataset Broken Into Bits,
- These bits are call Partitions
- Cassandra Partitions != Spark Partitions
- Spark Partitions are sized based on the estimated data size of the underlying C* table
- input.split.size_in_mb

OOMs Caused by Spark Partitions Holding Too Much Data



Executor JVM Heap

As a general rule of thumb your Executor should be set to hold

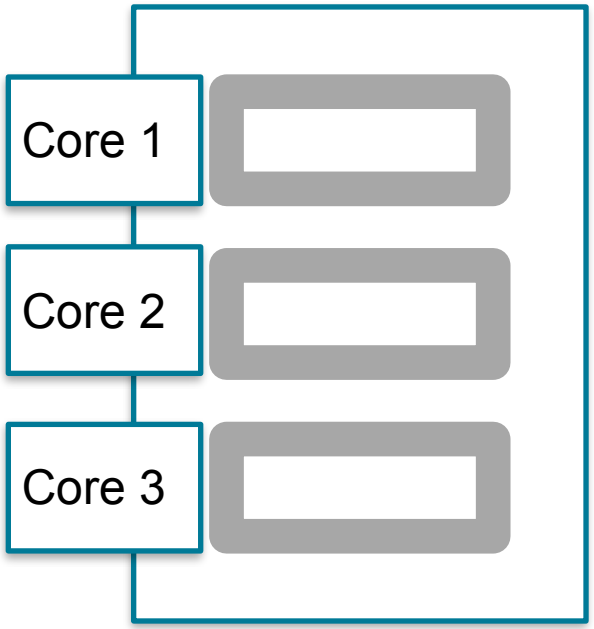
Number of Cores * Size of Partition * 1.2

See a lot of GC? OOM? Increase the amount of partitions

Some Caveats

- We don't know the actual partition size until runtime
- Cassandra on disk memory usage != in memory size

OOMs Caused by Spark Partitions Holding Too Much Data



Executor JVM Heap

```
input.split.size_in_mb    64
Approx amount of data to be fetched into a
Spark partition. Minimum number of resulting
Spark partitions is
1 + 2 * SparkContext.defaultParallelism
```

split.size_in_mb compares uses the system table size_esitmates to determine how many Cassandra Partitions should be in a Spark Partition.

Due to Compression and Inflation, the actual in memory size can be much larger

Certain Queries can't be broken Up

- Hot Spots Make a Spark Partition OOM
  - Full C* Partition in Spark Partiton

Certain Queries can't be broken Up

- Hot Spots Make a Spark Partition OOM
  - Full C* Partition in Spark Partiton
- Single Partition Lookups
  - Can't do anything about this
  - Don't know how partition is distributed

Certain Queries can't be broken Up

- Hot Spots Make a Spark Partition OOM
  - Full C* Partition in Spark Partiton
- Single Partition Lookups
  - Can't do anything about this
  - Don't know how partition is distributed
- IN clauses
  - Replace with JoinWithCassandraTable

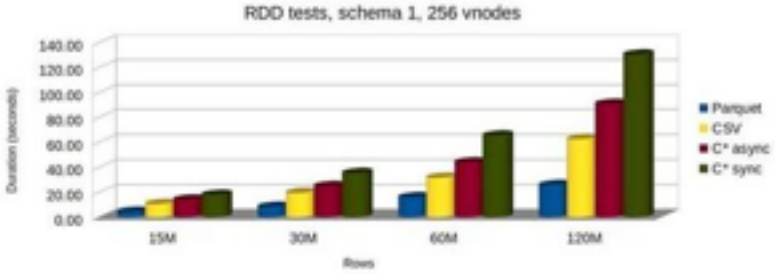
- If all else fails use CassandraConnector

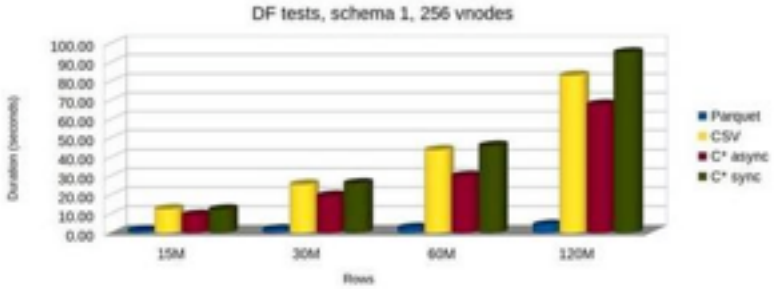
Read speed is mostly dictated by Cassandra's Paging Speed

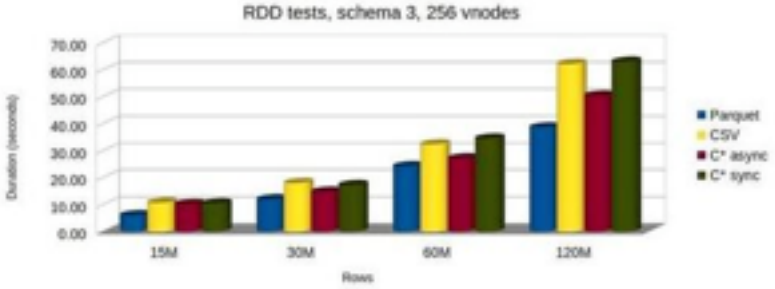
input.fetch.size_in_rows 1000 Number of CQL rows fetched per driver request

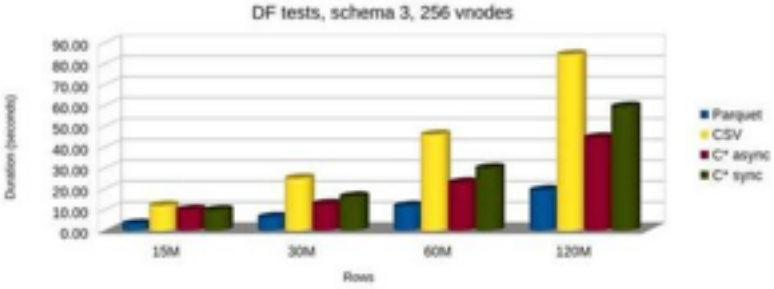
Cassandra of the Future, As Fast as CSV!?!

https://issues.apache.org/jira/browse/CASSANDRA-9259 :
Bulk Reading from Cassandra
Stefania Alborghetti







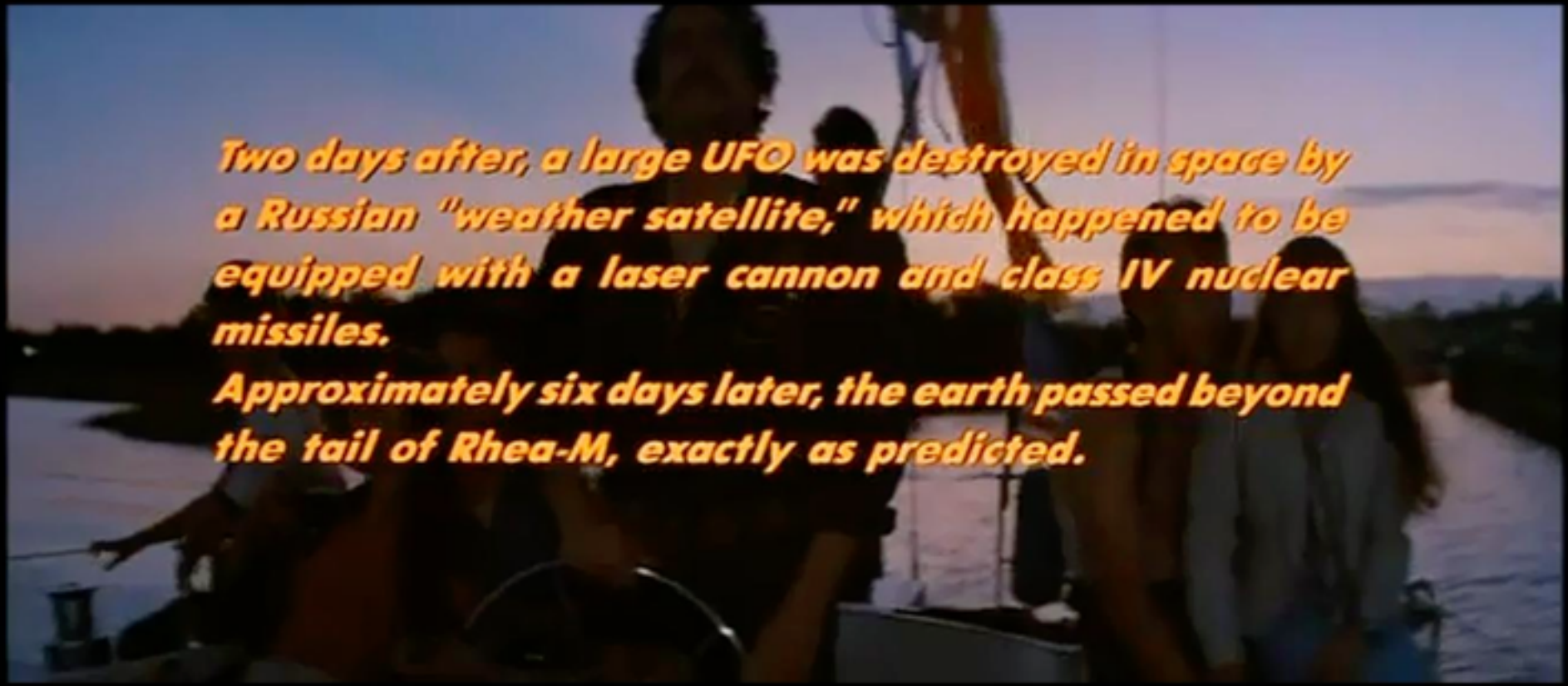


CASSANDRA SUMMIT 2016

In Summation, Know your Data

- Write Tuning
  - Batching Key
  - Sorting
  - Turning off Batching When Beneficial
  - Having enough Data in a Task
- Read Tuning
  - Number of Partitions
  - Some Queries can't be Broken Up
  - Changing paging from Cassandra
  - Future bulk read speedup!

The End


Two days after, a large UFO was destroyed in space by a Russian "weather satellite," which happened to be equipped with a laser cannon and class IV nuclear missiles.
Approximately six days later, the earth passed beyond the tail of Rhea-M, exactly as predicted.

Don't Let it End Like That!
Contribute to the Spark Cassandra Connector

- OSS Project that loves community involvement
- Bug Reports
- Feature Requests
- Write Code
- Doc Improvements
- Come join us!

https://github.com/datastax/spark-cassandra-connector

See you on the mailing list!

https://github.com/datastax/spark-cassandra-connector