Streaming Big Data with Spark, Kafka, Cassandra, Akka & Scala

Helena Edelson
@helenaedelson

1	***Delivering Meaning In Near-Real Time At High Velocity***
2	***Overview Of Spark Streaming, Kafka and Akka***
3	***Cassandra and the Spark Cassandra Connector***
4	***Integration In Big Data Applications***

Who Is This Person?

DATASTAX

- Spark Cassandra Connector Committer
- Akka Contributor (2 features in Cluster)
- Scala & Big Data Conference Speaker
-  @helenaedelson
-  https://github.com/helena
- Senior Software Engineer, Analytics @ DataStax
- Previously Senior Cloud Engineer at VMware & others

Use Case: Hadoop + Scalding

DATASTAX

```
/** Reads SequenceFile data from S3 buckets, computes then persists to Cassandra. */
class TopSearches(args: Args) extends TopKDailyJob[MyDataType](args) with Cassandra {

 PailSource.source[Search](rootpath, structure, directories).read
  .mapTo('pailItem -> 'engines) { e: Search ⇒ results(e) }
  .filter('engines) { e: String ⇒ e.nonEmpty }
  .groupBy('engines) { _.size('count).sortBy('engines) }
  .groupBy('engines) { _.sortedReverseTake[(String, Long)](('engines, 'count) -> 'tcount, k) }
  .flatMapTo('tcount -> ('key, 'engine, 'topCount)) { t: List[(String, Long)] ⇒
      t map { case (k, v) ⇒ (jobKey, k, v) }}
  .write(CassandraSource(connection, "top_searches", Scheme(‘key, ('engine, ‘topCount))))

}
```

Use Case: Spark?

DATASTAX

```
20  object SparkWordCount extends WordCountBlueprint {
21
22    sc.textFile("./src/main/resources/data/words")
23      .flatMap(_.split("\\s+"))
24      .map(word => (clean(word), 1))
25      .reduceByKey(_ + _)
26      .saveAsTextFile(s"./wordcount-$timestamp")
27  }
```

Delivering Meaning In Near-Real Time At High Velocity At Massive Scale

Fault-tolerance

by @jrecursive

Strategies

DATASTAX

- Partition For Scale
- Replicate For Resiliency
- Share Nothing
- Asynchronous Message Passing
- Parallelism
- Isolation
- Location Transparency

What We Need

DATASTAX

- Fault Tolerant
- Failure Detection
- Fast - low latency, distributed, data locality
- Masterless, Decentralized Cluster Membership
- Span Racks and DataCenters
  - Hashes The Node Ring
  - Partition-Aware
- Elasticity
- Asynchronous - message-passing system
- Parallelism
- Network Topology Aware

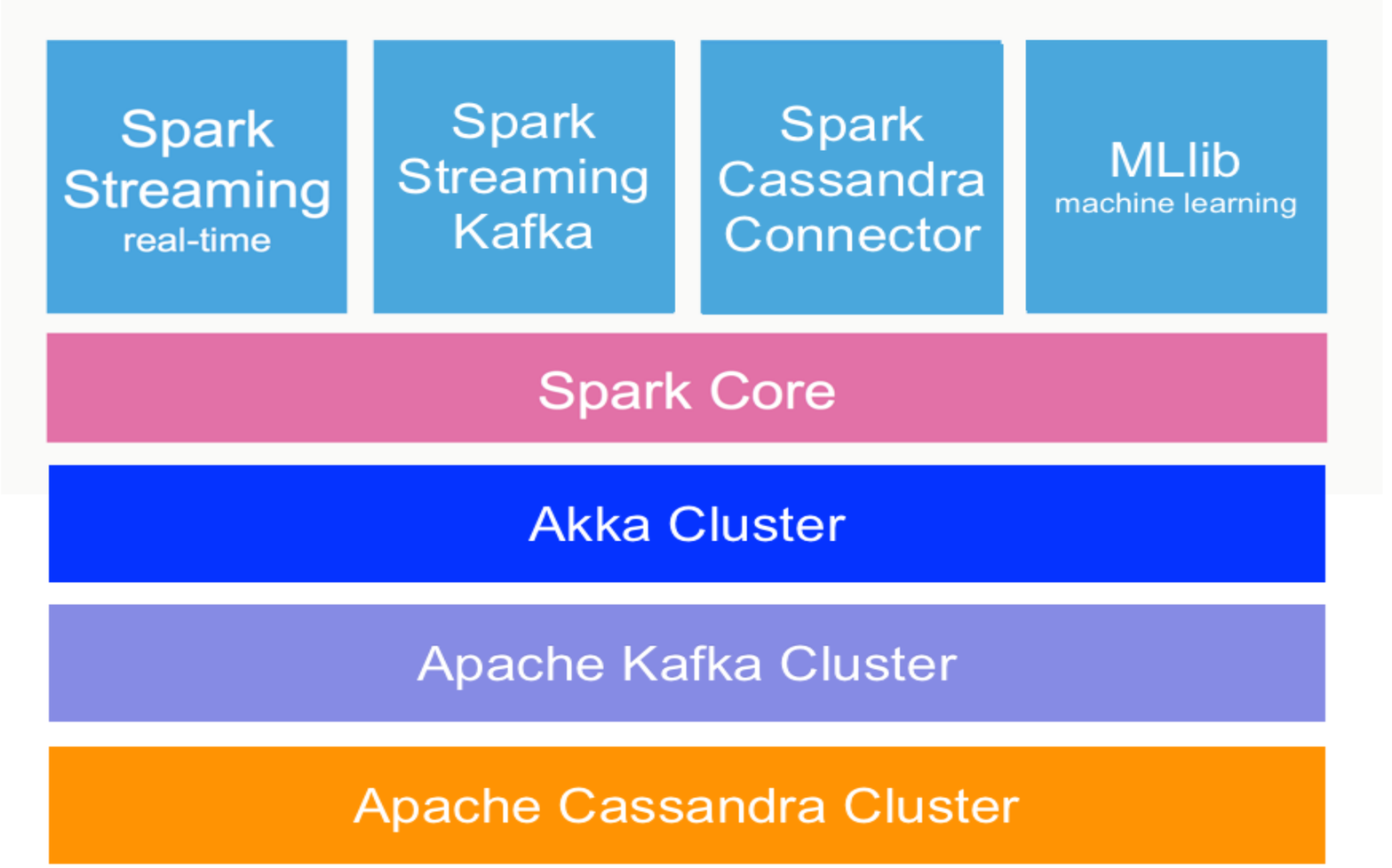
Lambda Architecture with Spark, Kafka, Cassandra and Akka (Scala!)

DATASTAX

Lambda Architecture - is a data-processing architecture designed to handle massive quantities of data by taking advantage of both batch and stream processing methods.

- Spark - one of the few, if not the only, data processing framework that allows you to have both batch and stream processing of terabytes of data in the same application.
- Storm does not

KillrWeather Architecture



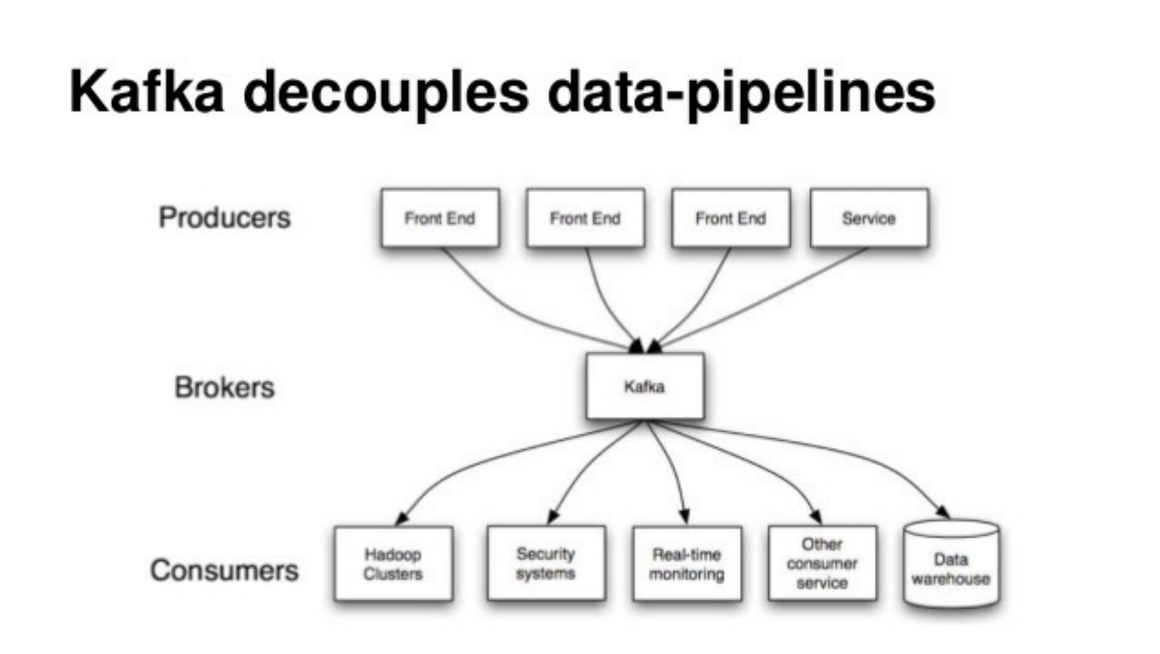
Why Akka Rocks

DATASTAX

- location transparency
- fault tolerant
- async message passing
- non-deterministic
- share nothing
- actor atomicity (w/in actor)

Apache Kafka

DATASTAX



From Joe Stein

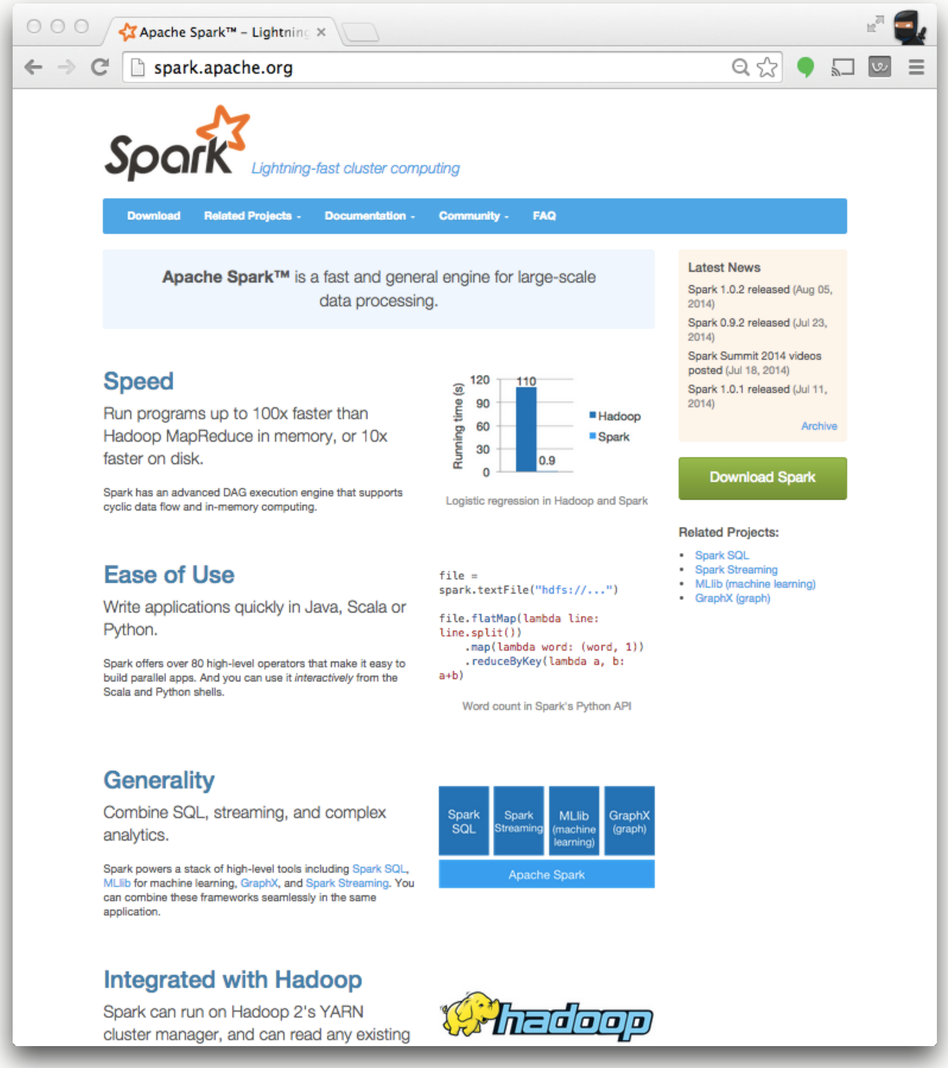
Apache Spark and Spark Streaming

The Drive-By Version

What Is Apache Spark

DATASTAX

- Fast, distributed, scalable and fault tolerant cluster compute system
- Enables Low-latency with complex analytics
- Developed in 2009 at UC Berkeley AMPLab, open sourced in 2010, and became a top-level Apache project in February, 2014



Most Active OSS In Big Data

DATASTAX

apache / spark
mirrored from git://git.apache.org/spark.git

Watch 427 | Star 2,285 | Fork 2,044

November 11, 2014 – December 11, 2014

Period: **1 month**

Overview

154 Active Pull Requests

0 Active Issues

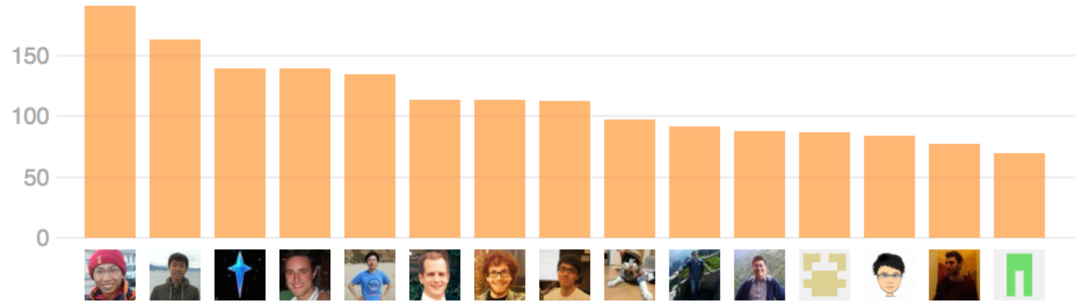
0 Merged Pull Requests

154 Proposed Pull Requests

0 Closed Issues

0 New Issues

Excluding merges, **317 authors** have pushed **297 commits** to master and **3,546 commits** to all branches. On master, **702 files** have changed and there have been **38,458 additions** and **18,021 deletions**.



154 Pull requests proposed by **82** people

Apache Spark - Easy to Use & Fast

DATASTAX

- 10x faster on disk,100x faster in memory than Hadoop MR
- Write, test and maintain 2 - 5x less code
- Fault Tolerant Distributed Datasets
- Batch, iterative and streaming analysis
- In Memory Storage and Disk
- Integrates with Most File and Storage Options

Easy API

DATASTAX

- Abstracts complex algorithms to high level functions
- Collections API over large datasets
- Uses a functional programming model - clean
- Scala, Java and Python APIs
- Use from a REPL
- Easy integration with SQL, streaming, ML, Graphing, R…

Resilient Distributed Dataset (RDD)

DATASTAX

- Fault Tolerant: can recalculate from any point of failure
- Created through transformations on data (map,filter..) or other RDDs
- Immutable
- Partitioned
  - Indicate which RDD elements to partition across machines based on a key in each record
- Can be reused

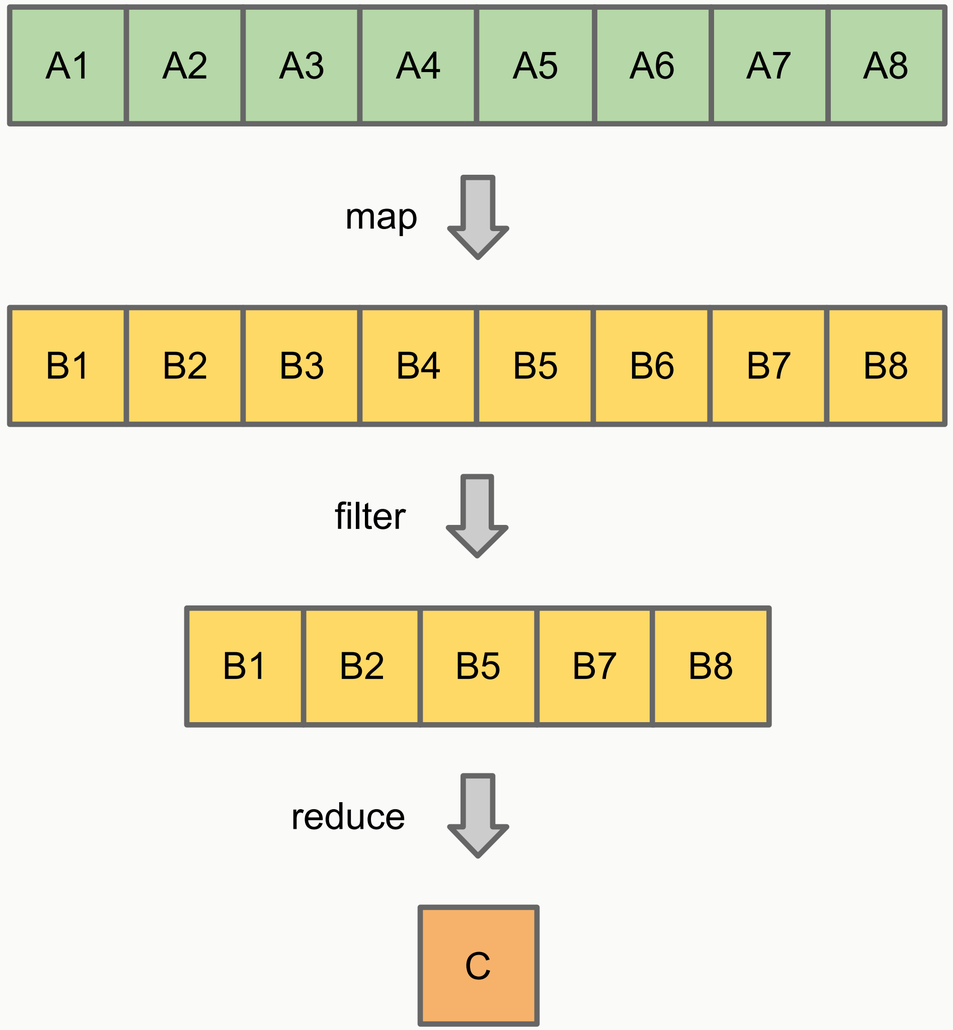
Spark Data Model

DATASTAX

Resilient **D**istributed **D**ataset

A collection:

- immutable
- iterable
- serializable
- distributed
- parallel
- lazy



RDD Operations

DATASTAX

```
16  object SparkWordCount extends WordCountBlueprint {
17
18    sc.textFile("./src/main/resources/data/words")
19      .flatMap(_.split("\\s+"))
20      .map(word => (clean(word), 1))
21      .reduceByKey(_ + _)
22      .collect foreach println
23  }
```

Transformation

Action

Collection To RDD

DATASTAX

```
scala> val data = Array(1, 2, 3, 4, 5)
data: Array[Int] = Array(1, 2, 3, 4, 5)

scala> val distData = sc.parallelize(data)
distData: spark.RDD[Int] =
spark.ParallelCollection@10d13e3e
```

Spark Basic Word Count

DATASTAX

```
val conf = new SparkConf()
  .setMaster("local[*]")
  .setAppName("Simple Word Count")

val sc = new SparkContext(conf)

sc.textFile(words)
   .flatMap(_.split("\\s+"))
   .map(word => (word.toLowerCase, 1))
   .reduceByKey(_ + _)
   .collect foreach println
```

Apache Spark - Easy to Use API

DATASTAX

```
/** Returns the top (k) highest temps for any location in the `year`. */
val k = 20

def topK(aggregate: Seq[Double]): Seq[Double] =
  sc.parallelize(aggregate).top(k)


/** Returns the top (k) highest temps … in a Future */
def topK(aggregate: Seq[Double]): Future[Seq[Double]] =
  sc.parallelize(aggregate).top(k).collectAsync
```

Not Just MapReduce

DATASTAX

```
def aggregate[U](zeroValue: U)(seqOp: (U, T) ⇒ U, combOp
```
Aggregate the elements of each partition, and then the results for all th

```
def cache(): RDD.this.type
```
Persist this RDD with the default storage level (MEMORY_ONLY).

```
def cartesian[U](other: RDD[U])(implicit arg0: ClassTag[
```
Return the Cartesian product of this RDD and another one, that is, the

```
def checkpoint(): Unit
```
Mark this RDD for checkpointing.

```
def coalesce(numPartitions: Int, shuffle: Boolean = fals
```
Return a new RDD that is reduced into numPartitions partitions.

```
def collect[U](f: PartialFunction[T, U])(implicit arg0:
```
Return an RDD that contains all matching values by applying f.

```
def collect(): Array[T]
```
Return an array that contains all of the elements in this RDD.

```
def context: SparkContext
```
The org.apache.spark.SparkContext that this RDD was created on.

```
def count(): Long
```
Return the number of elements in the RDD.

```
def countApprox(timeout: Long, confidence: Double = 0.95
```
Approximate version of count() that returns a potentially incomplete res

```
def countApproxDistinct(relativeSD: Double = 0.05): Long
```
Return approximate number of distinct elements in the RDD.

```
def countApproxDistinct(p: Int, sp: Int): Long
```
Return approximate number of distinct elements in the RDD.

```
def countByValue()(implicit ord: Ordering[T] = null): Ma
```
Return the count of each unique value in this RDD as a map of (value,

```
def countByValueApprox(timeout: Long, confidence: Double
```
Approximate version of countByValue().

```
def dependencies: Seq[Dependency[_]]
```
Get the list of dependencies of this RDD, taking into account whether t

```
def distinct(): RDD[T]
```
Return a new RDD containing the distinct elements in this RDD.

```
def glom(): RDD[Array[T]]
```
Return an RDD created by coalescing all elements within each parti

```
def groupBy[K](f: (T) ⇒ K, p: Partitioner)(implicit kt
```
Return an RDD of grouped items.

```
def groupBy[K](f: (T) ⇒ K, numPartitions: Int)(implici
```
Return an RDD of grouped elements.

```
def groupBy[K](f: (T) ⇒ K)(implicit kt: ClassTag[K]):
```
Return an RDD of grouped items.

```
val id: Int
```
A unique ID for this RDD (within its SparkContext).

```
def intersection(other: RDD[T], numPartitions: Int): R
```
Return the intersection of this RDD and another one.

```
def intersection(other: RDD[T], partitioner: Partition
```
Return the intersection of this RDD and another one.

```
def intersection(other: RDD[T]): RDD[T]
```
Return the intersection of this RDD and another one.

```
def isCheckpointed: Boolean
```
Return whether this RDD has been checkpointed or not

```
def iterator(split: Partition, context: TaskContext):
```
Internal method to this RDD; will read from cache if applicable, or otl

```
def keyBy[K](f: (T) ⇒ K): RDD[(K, T)]
```
Creates tuples of the elements in this RDD by applying f.

```
def map[U](f: (T) ⇒ U)(implicit arg0: ClassTag[U]): RD
```
Return a new RDD by applying a function to all elements of this RDD

```
def mapPartitions[U](f: (Iterator[T]) ⇒ Iterator[U], p
```
Return a new RDD by applying a function to each partition of this RD

```
def mapPartitionsWithContext[U](f: (TaskContext, Itera
```
Return a new RDD by applying a function to each partition of this RD

```
def mapPartitionsWithIndex[U](f: (Int, Iterator[T]) ⇒
```
Return a new RDD by applying a function to each partition of this RD

```
def max()(implicit ord: Ordering[T]): T
```
Returns the max of this RDD as defined by the implicit Ordering[T].

```
def min()(implicit ord: Ordering[T]): T
```
Returns the min of this RDD as defined by the implicit Ordering[T].

```
var name: String
```
A friendly name for this RDD

```
val partitioner: Option[Partitioner]
```
Optionally overridden by subclasses to specify how they are partitioned.

```
def partitions: Array[Partition]
```
Get the array of partitions of this RDD, taking into account whether the RDD is chec

```
def persist(): RDD.this.type
```
Persist this RDD with the default storage level (MEMORY_ONLY).

```
def persist(newLevel: StorageLevel): RDD.this.type
```
Set this RDD's storage level to persist its values across operations after the first tim

```
def pipe(command: Seq[String], env: Map[String, String] = Map(), p
```
Return an RDD created by piping elements to a forked external process.

```
def pipe(command: String, env: Map[String, String]): RDD[String]
```
Return an RDD created by piping elements to a forked external process.

```
def pipe(command: String): RDD[String]
```
Return an RDD created by piping elements to a forked external process.

```
def preferredLocations(split: Partition): Seq[String]
```
Get the preferred locations of a partition (as hostnames), taking into account whethe

```
def randomSplit(weights: Array[Double], seed: Long = Utils.random.
```
Randomly splits this RDD with the provided weights.

```
def reduce(f: (T, T) ⇒ T): T
```
Reduces the elements of this RDD using the specified commutative and associative

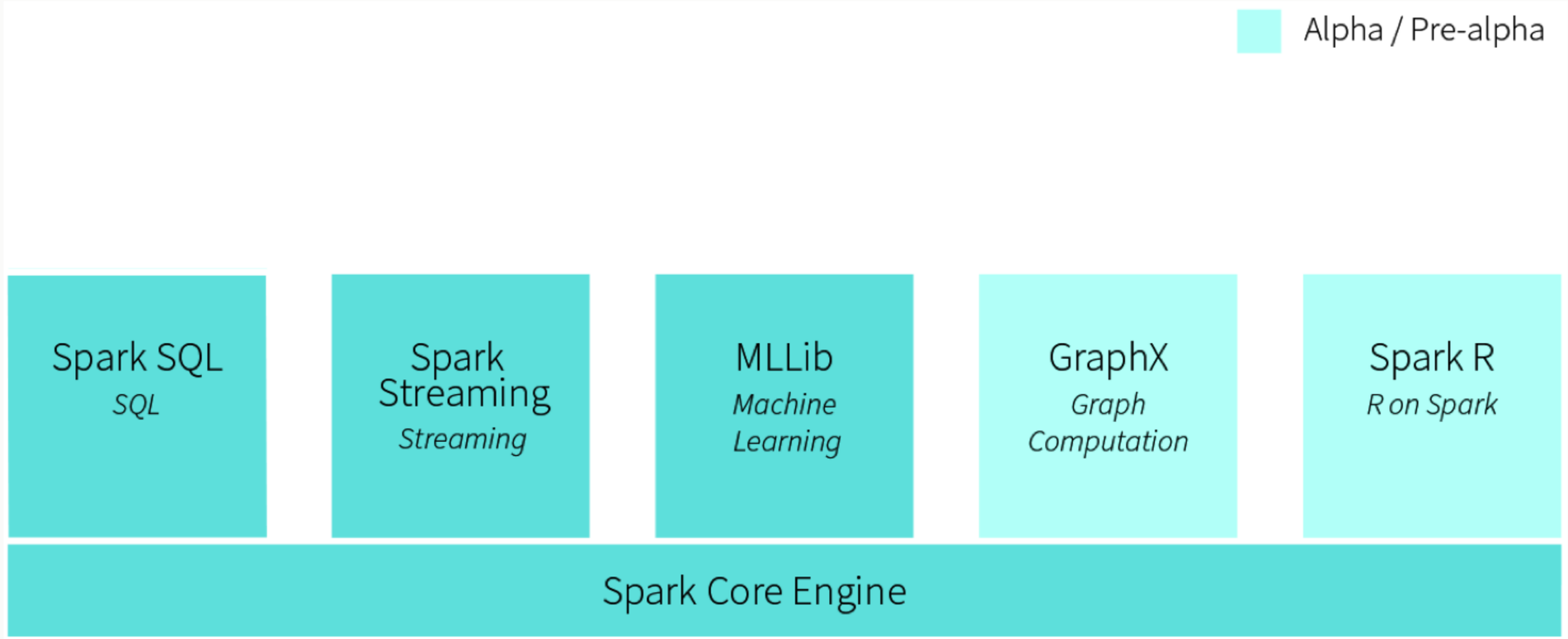
```
def repartition(numPartitions: Int)(implicit ord: Ordering[T] = nu
```
Return a new RDD that has exactly numPartitions partitions.

```
def sample(withReplacement: Boolean, fraction: Double, seed: Long
```
Return a sampled subset of this RDD.

```
def saveAsObjectFile(path: String): Unit
```
Save this RDD as a SequenceFile of serialized objects.

```
def saveAsTextFile(path: String, codec: Class[_ <: CompressionCode
```
Save this RDD as a compressed text file, using string representations of elements.

Spark Components



Why Scala?

DATASTAX

http://apache-spark-user-list.1001560.n3.nabble.com/Why-Scala-tp6536p6538.html

- Functional
- On the JVM
- Capture functions and ship them across the network
- Static typing - easier to control performance
- Leverage REPL Spark REPL

When Batch Is Not Enough

DATASTAX

- Scheduled batch only gets you so far
- I want results continuously in the event stream
- I want to run computations in my even-driven asynchronous apps

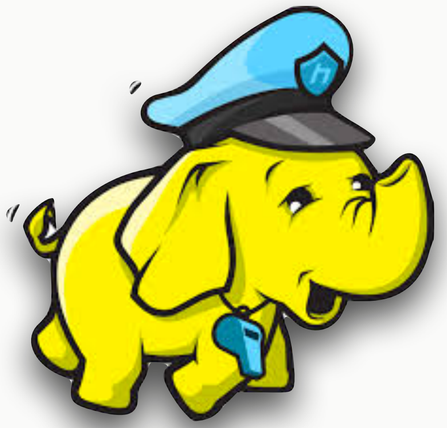
Spark Streaming

zillions of bytes

gigabytes per second

Input & Output Sources

DATASTAX



Spark Streaming

DATASTAX

Common Use Cases

DATASTAX

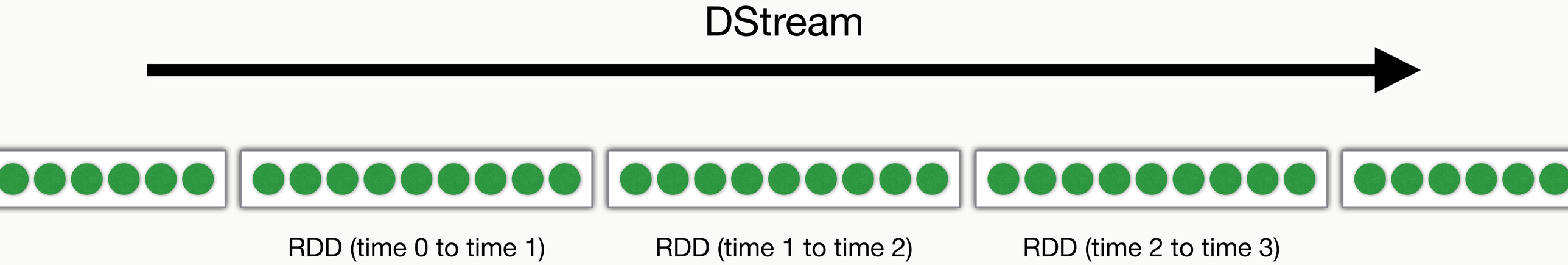
applications	sensors	web	mobile phones
intrusion detection	malfunction detection	site analytics	network metrics analysis
fraud detection	dynamic process optimisation	recommendations	location based ads
log processing	supply chain planning	sentiment analysis	...

DStream (Discretized Stream)

DATASTAX

Continuous stream of micro batches

- Complex processing models with minimal effort
- Streaming computations on small time intervals



A transformation on a DStream = transformations on its RDDs

InputDStreams and Receivers

DATASTAX

DStreams - the stream of raw data received from streaming sources:

- Basic Source - in the StreamingContext API
- Advanced Source - in external modules and separate Spark artifacts

Receivers

- Reliable Receivers - for data sources supporting acks (like Kafka)
- Unreliable Receivers - for data sources not supporting acks

Spark Streaming Modules

DATASTAX

GroupId	ArtifactId	Latest Version	
org.apache.spark	spark-streaming-kinesis-asl_2.10	1.1.0	
org.apache.spark	spark-streaming-mqtt_2.10	1.1.0	all (7)
org.apache.spark	spark-streaming-zeromq_2.10	1.1.0	all (7)
org.apache.spark	spark-streaming-flume_2.10	1.1.0	all (7)
org.apache.spark	spark-streaming-flume-sink_2.10	1.1.0	
org.apache.spark	spark-streaming-kafka_2.10	1.1.0	all (7)
org.apache.spark	spark-streaming-twitter_2.10	1.1.0	all (7)

Spark Streaming Setup

DATASTAX

```
val conf = new SparkConf().setMaster(SparkMaster).setAppName(AppName)
  .set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
  .set("spark.kryo.registrator", "com.datastax.killrweather.KillrKryoRegistrator")
  .set("spark.cleaner.ttl", SparkCleanerTtl)

val streamingContext = new StreamingContext(conf, Milliseconds(500))

// Do work in the stream

ssc.checkpoint(checkpointDir)

ssc.start()
ssc.awaitTermination
```

Checkpointing

DATASTAX

Allows saving enough of information to a fault-tolerant storage to allow the RDDs

- Metadata - the information defining the streaming computation
- Data (RDDs)

Usage

- With updateStateByKey, reduceByKeyAndWindow - stateful transformations
- To recover from failures in Spark Streaming apps

Can affect performance, depending on

- The data and or batch sizes.
- The speed of the file system that is being used for checkpointing

Basic Streaming: FileInputDStream

DATASTAX

```
// Creates new DStreams
ssc.textFileStream("s3n://raw_data_bucket/")
   .flatMap(_.split("\\s+"))
   .map(_.toLowerCase, 1))
   .countByValue()
   .saveAsObjectFile("s3n://analytics_bucket/")
```

ReceiverInputDStreams

DATASTAX

```
val stream = KafkaUtils.createStream[String, String, StringDecoder,
StringDecoder](
  ssc, kafkaParams, Map(KafkaTopicRaw -> 10), StorageLevel.DISK_ONLY_2)
  .map { case (_, line) => line.split(",")}
  .map(RawWeatherData(_))

/** Saves the raw data to Cassandra - raw table. */
stream.saveToCassandra(keyspace, rawDataTable)

stream.map { data =>
  (data.wsid, data.year, data.month, data.day, data.oneHourPrecip)
}.saveToCassandra(keyspace, dailyPrecipitationTable)

ssc.start()
ssc.awaitTermination
```

Streaming Window Operations

DATASTAX

```
// where pairs are (word,count)
pairsStream
.flatMap { case (k,v) => (k,v.value) }
.reduceByKeyAndWindow((a:Int,b:Int) => (a + b),
     Seconds(30), Seconds(10))
.saveToCassandra(keyspace,table)
```

Streaming Window Operations

DATASTAX

- window(Duration, Duration)
- countByWindow(Duration, Duration)
- reduceByWindow(Duration, Duration)
- countByValueAndWindow(Duration, Duration)
- groupByKeyAndWindow(Duration, Duration)
- reduceByKeyAndWindow((V, V) => V, Duration, Duration)

Ten Things About Cassandra

You always wanted to know but were afraid to ask

Apache Cassandra

DATASTAX

- Elasticity - scale to as many nodes as you need, when you need
- Always On - No single point of failure, Continuous availability
- Masterless peer to peer architecture
- Designed for Replication
- Flexible Data Storage
- Read and write to any node syncs across the cluster
- Operational simplicity - with all nodes in a cluster being the same, there is no complex configuration to manage

Easy to use

DATASTAX

- CQL - familiar syntax
- Friendly to programmers
- Paxos for locking

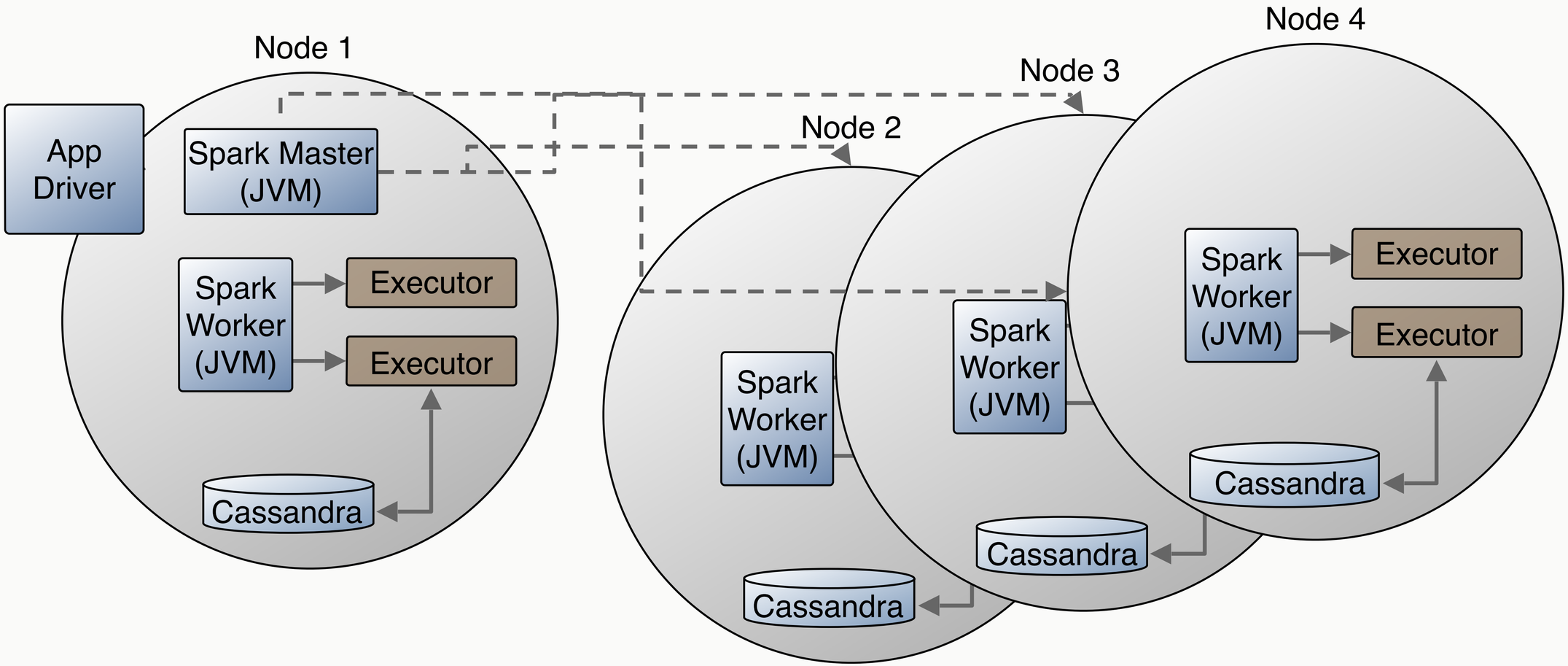
```
CREATE TABLE users (
   username varchar,
   firstname varchar,
   lastname varchar,
   email list<varchar>,
   password varchar,
   created_date timestamp,
   PRIMARY KEY (username)
);
```

```
INSERT INTO users (username, firstname, lastname,
 email, password, created_date)
VALUES ('hedelson','Helena','Edelson',
 [‘helena.edelson@datastax.com'],'ba27e03fd95e507daf2937c937d499ab','2014-11-15 13:50:00');
```

```
INSERT INTO users (username, firstname,
  lastname, email, password, created_date)
VALUES ('pmcfadin','Patrick','McFadin',
  ['patrick@datastax.com'],
  'ba27e03fd95e507daf2937c937d499ab',
  '2011-06-20 13:50:00')
IF NOT EXISTS;
```

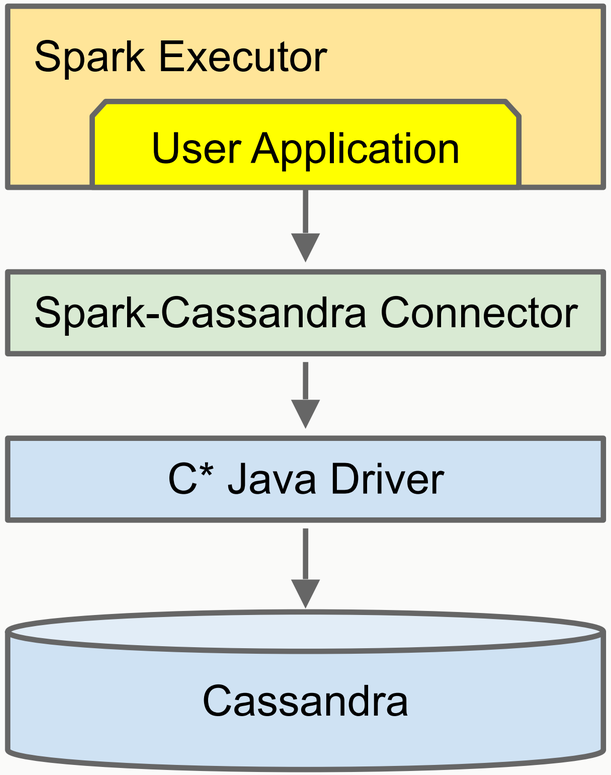
Spark Cassandra Integration

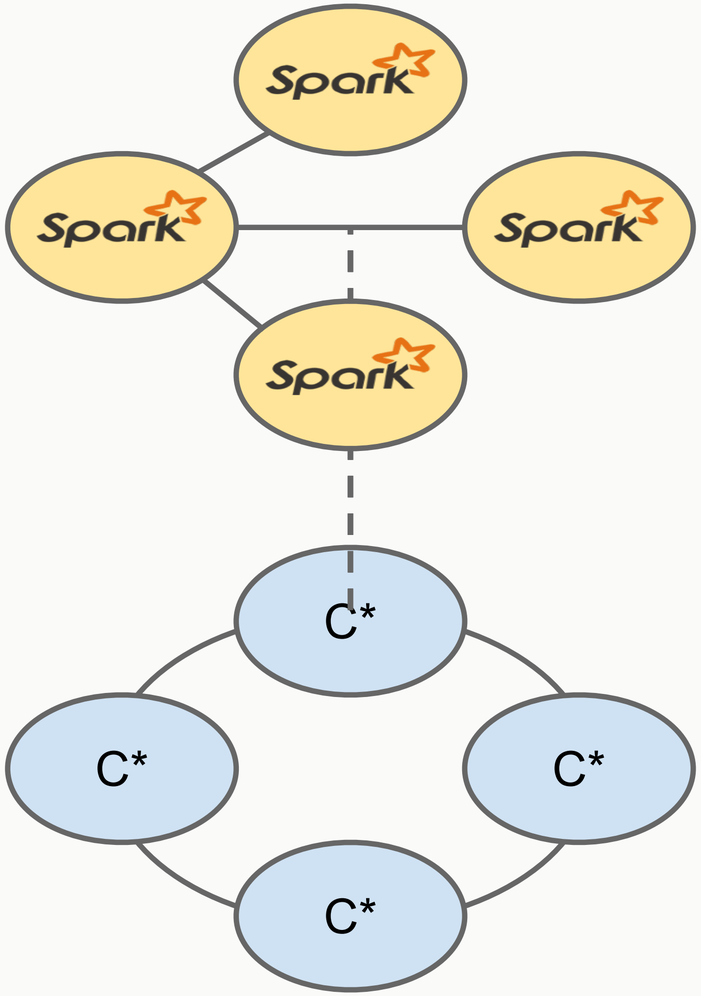
Deployment Architecture



Spark Cassandra Connector

DATASTAX





Spark Cassandra Connector

DATASTAX

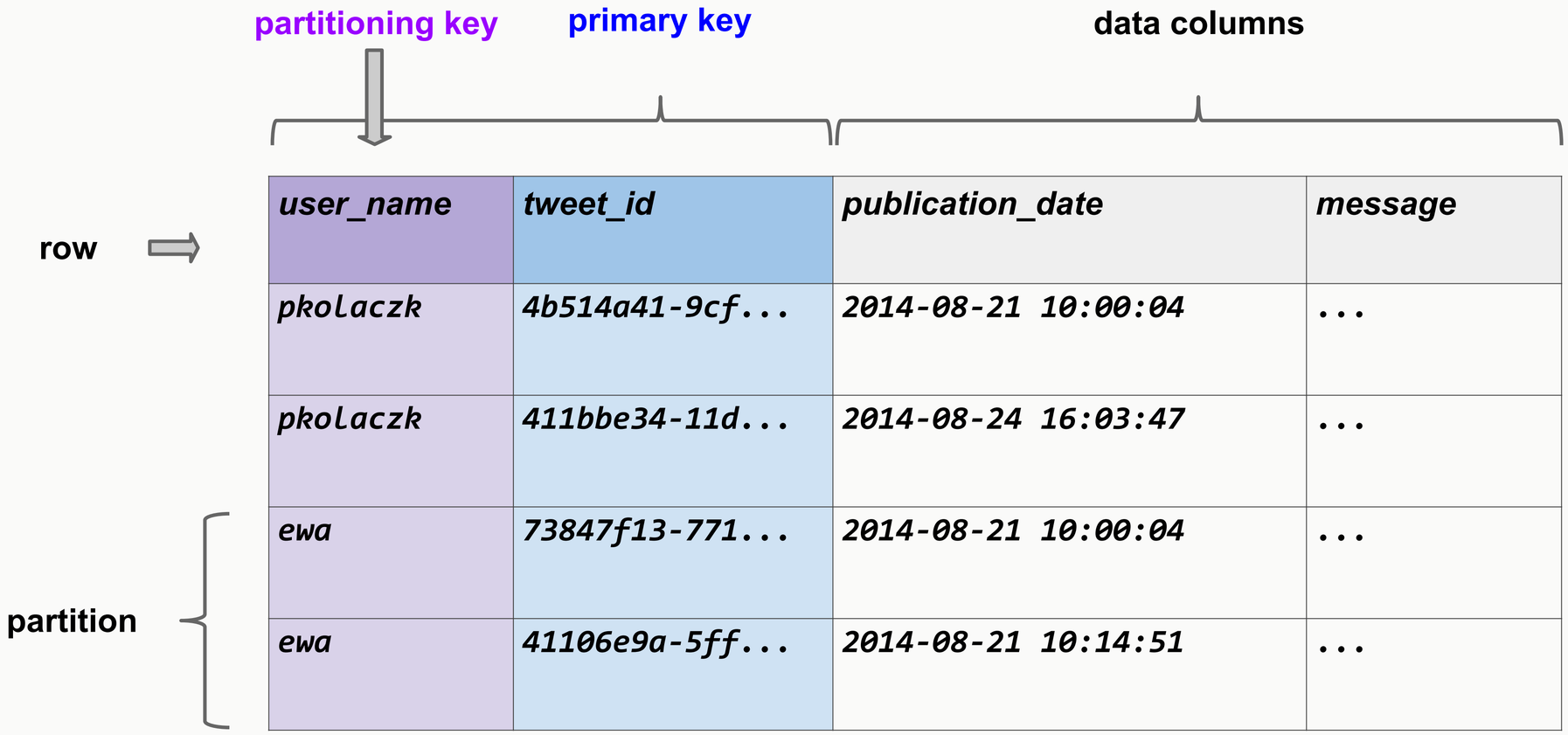
https://github.com/datastax/spark-cassandra-connector

- Loads data from Cassandra to Spark
- Writes data from Spark to Cassandra
- Handles type conversions
- Offers an object mapper

- Implemented in Scala
- Scala and Java APIs
- Open Source

Cassandra Data Model

DATASTAX



user_name	*tweet_id*	*publication_date*	*message*
pkolaczk	*4b514a41-9cf...*	*2014-08-21 10:00:04*	*...*
pkolaczk	*411bbe34-11d...*	*2014-08-24 16:03:47*	*...*
ewa	*73847f13-771...*	*2014-08-21 10:00:04*	*...*
ewa	*41106e9a-5ff...*	*2014-08-21 10:14:51*	*...*

Use Cases

DATASTAX

- Get data by weather station
- Get data for a single date and time
- Get data for a range of dates and times
- Compute, store and quickly retrieve daily, monthly and annual aggregations of data

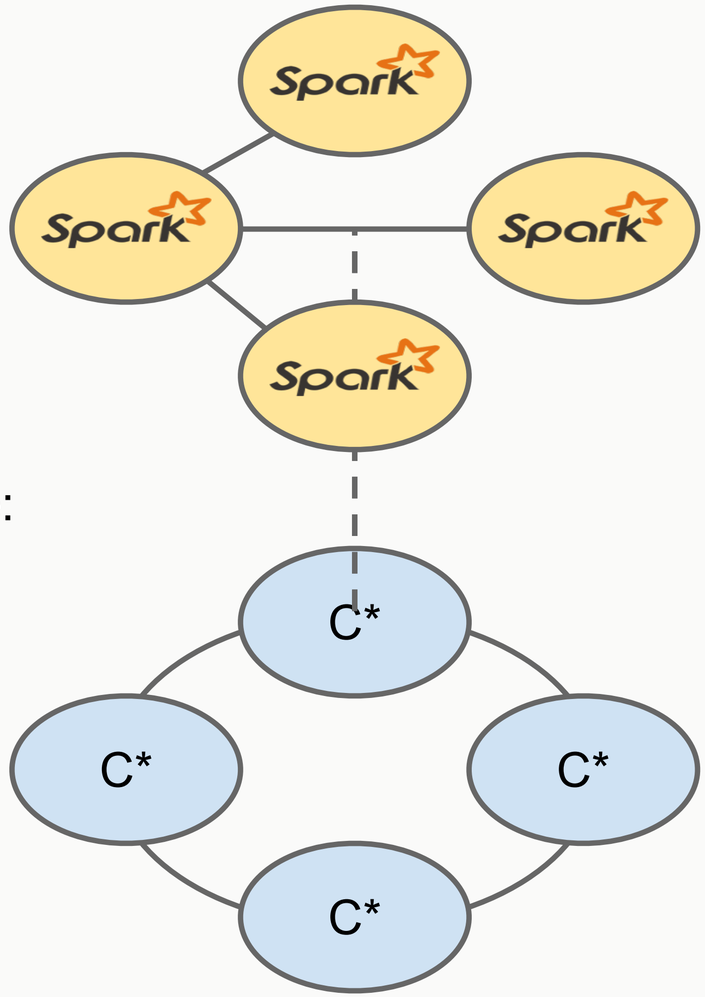
Design Data Model to support queries (speed, scale, distribution) vs contort queries to fit data model

- Store raw data per weather station
- Store time series in order: most recent to oldest
- Compute and store aggregate data in the stream
- Set TTLs on historic data

Data Locality

DATASTAX

- Spark asks an RDD for a list of its partitions (splits)
- Each split consists of one or more token-ranges
- For every partition
  - Spark asks RDD for a list of preferred nodes to process on
  - Spark creates a task and sends it to one of the nodes for execution

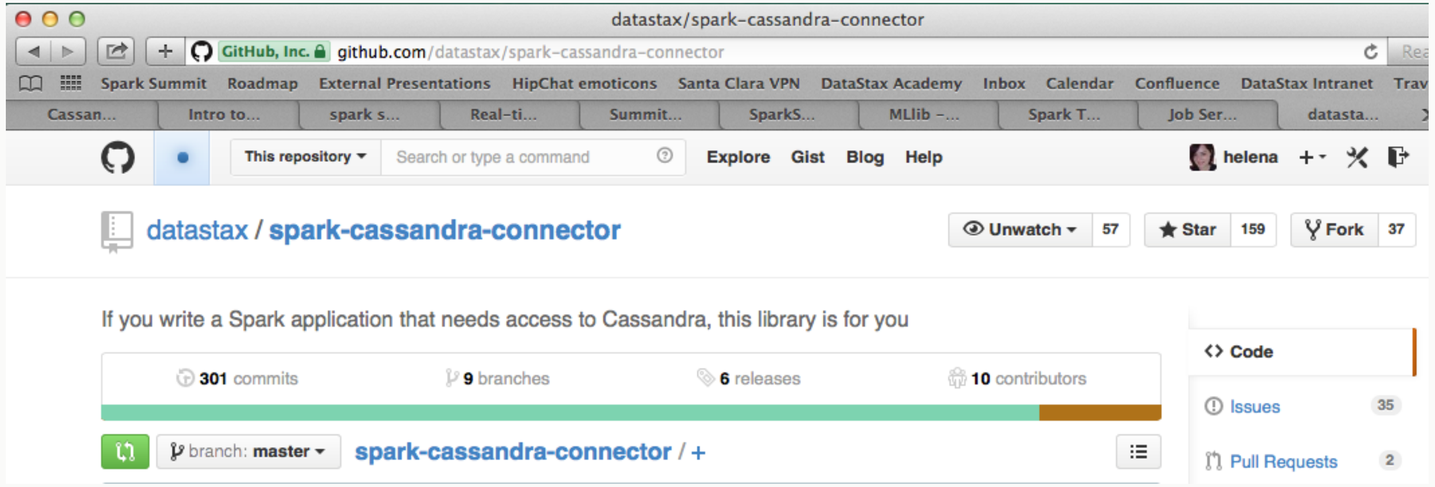


Every Spark task uses a CQL-like query to fetch data for the given token range:

```
SELECT "key", "value"
FROM "test"."kv"
WHERE
  token("key") >  59559742092113932 1 AND
  token("key") <= 59559743119420013 2
ALLOW FILTERING
```

Connector Code and Docs

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



Add It To Your Project:

```
"com.datastax.spark" %% "spark-cassandra-connector" % "1.1.0"
```

Using Spark With Cassandra

Reading Data: From C* To Spark

DATASTAX

row representation

keyspace

table

```
val table = sc
  .cassandraTable[CassandraRow]("db", "tweets")
  .select("user_name", "message")
  .where("user_name = ?", "ewa")
```

server side column and row selection

Spark Streaming, Kafka & Cassandra

DATASTAX

```
val conf = new SparkConf(true)
  .setMaster("local[*]")
  .setAppName(getClass.getSimpleName)
  .set("spark.executor.memory", "1g")
  .set("spark.cores.max", "1")
  .set("spark.cassandra.connection.host", "127.0.0.1")
val ssc = new StreamingContext(conf, Seconds(30))
```

Initialization

```
KafkaUtils.createStream[String, String, StringDecoder, StringDecoder](
 ssc, kafka.kafkaParams, Map(topic -> 1), StorageLevel.MEMORY_ONLY)
  .map(_._2)
  .countByValue()
  .saveToCassandra("mykeyspace", "wordcount")

ssc.start()
ssc.awaitTermination()
```

Transformations and Action

Spark Streaming, Twitter & Cassandra

DATASTAX

```
val stream = TwitterUtils.createStream(
    ssc, auth, filters, StorageLevel.MEMORY_ONLY_SER_2)

val stream = TwitterUtils.createStream(ssc, auth, Nil,
StorageLevel.MEMORY_ONLY_SER_2)

/** Note that Cassandra is doing the sorting for you here. */
stream.flatMap(_.getText.toLowerCase.split("""\s+"""))
  .filter(topics.contains(_))
  .countByValueAndWindow(Seconds(5), Seconds(5))
  .transform((rdd, time) =>
      rdd.map { case (term, count) => (term, count, now(time))})
  .saveToCassandra(keyspace, table)
```

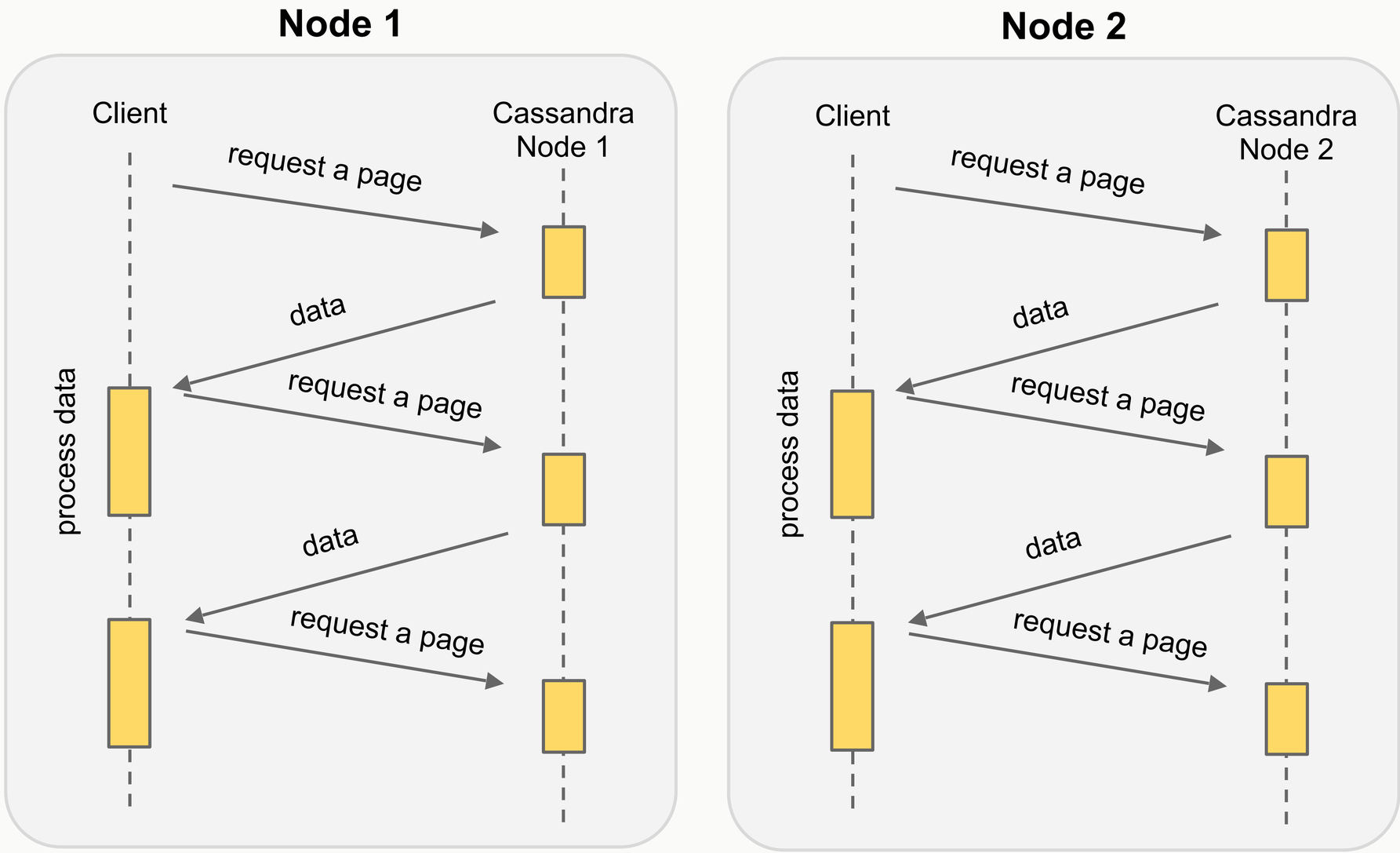
Paging Reads with .cassandraTable

DATASTAX

- Configurable Page Size
- Controls how many CQL rows to fetch at a time, when fetching a single partition
- Connector returns an `Iterator` for rows to Spark
- Spark iterates over this, lazily

ResultSet Paging and Pre-Fetching

DATASTAX



Converting Columns to a Type

DATASTAX

```
val table = sc.cassandraTable[CassandraRow]("db", “tweets")
val row: CassandraRow = table.first
val userName = row.getString("user_name")
val tweetId = row.getUUID(“row_id")
```

Converting Columns To a Type

DATASTAX

```
val row: CassandraRow = table.first

val date1 = row.get[java.util.Date]("publication_date")
val date2: org.joda.time.DateTime = row.get[org.joda.time.DateTime]("publication_date")
val date3: org.joda.time.DateTime = row.getDateTime(“publication_date”)
val date5: Option[DateTime] = row.getDateTimeOption(“publication_date”)

val list1 = row.get[Set[UUID]]("collection")
val list2 = row.get[List[UUID]]("collection")
val list3 = row.get[Vector[UUID]]("collection")
val list3 = row.get[Vector[String]]("collection")

val nullable = row.get[Option[String]]("nullable_column")
```

Converting Rows To Objects

DATASTAX

```
case class Tweet(
 userName: String,
 tweetId: UUID,
 publicationDate: Date,
 message: String)

val tweets = sc.cassandraTable[Tweet]("db", "tweets")
```

Scala	*Cassandra*
message	*message*
column1	*column_1*
userName	*user_name*

Scala	*Cassandra*
Message	*Message*
column1	*column1*
userName	*userName*

Converting Cassandra Rows To Tuples

DATASTAX

```
val tweets = sc
  .cassandraTable[(Int, Date, String)](“mykeyspace", “clustering”)
  .select(“cluster_id”, “time”, ”cluster_name”)
  .where("time > ? and time < ?", "2014-07-12 20:00:01", "2014-07-12 20:00:03")
```

When returning tuples, always use `select` to specify the column order.

Converting Rows to Key-Value Pairs

DATASTAX

```
case class Key(...)
case class Value(...)

val rdd = sc.cassandraTable[(Key, Value)]("keyspace", "table")
```

Writing Data

DATASTAX

```
cqlsh> CREATE TABLE test.words(word TEXT PRIMARY KEY, count INT);
```

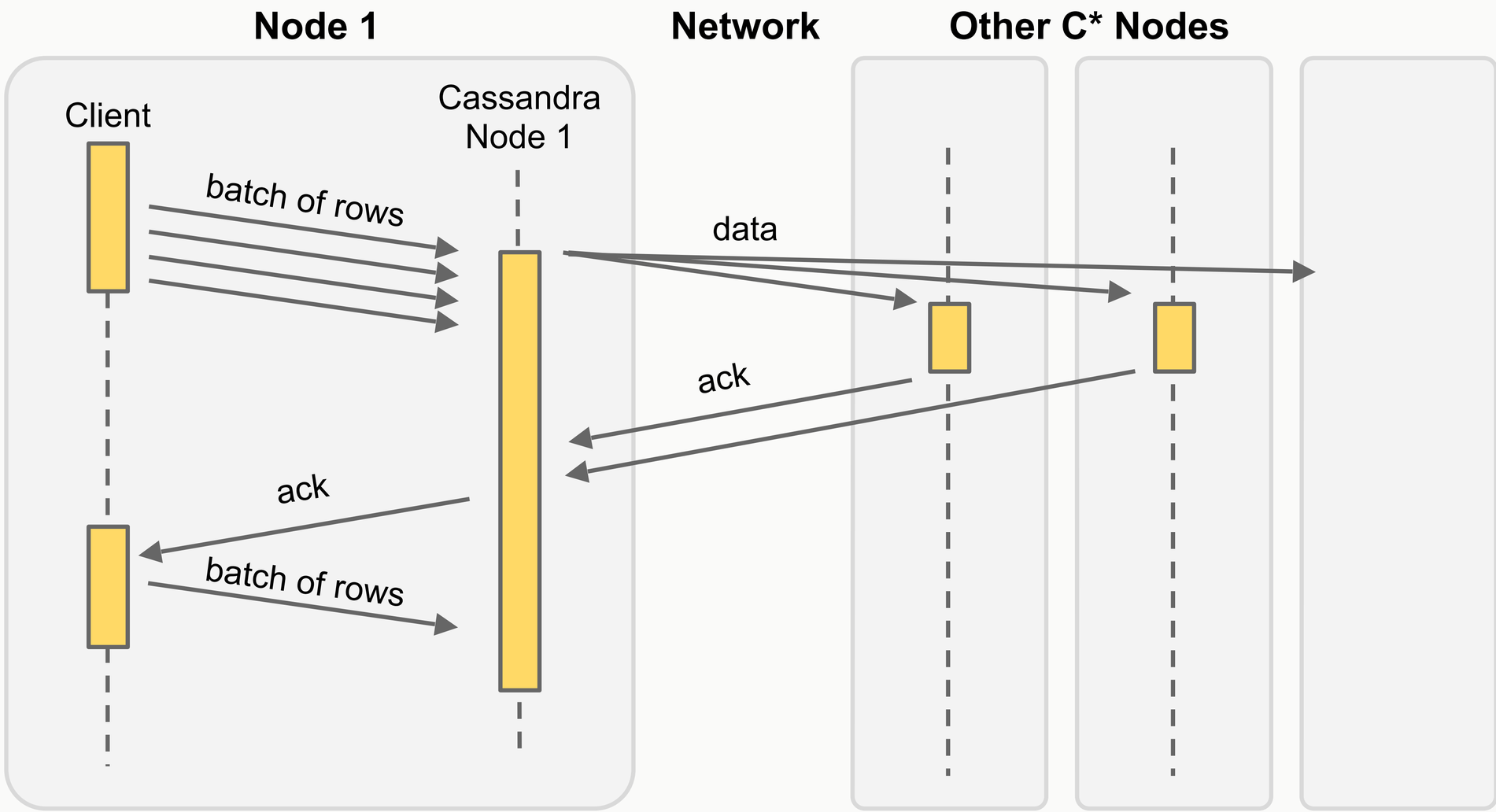
Scala:

```
sc.parallelize(Seq(("foo", 2), ("bar", 5)))
  .saveToCassandra("test", "words", SomeColumns("word", "count"))
```

```
cqlsh:test> select * from words;

 word | count
------+-------
  bar |     5
  foo |     2
```

How Writes Are Executed



Other Sweet Features

Spark SQL with Cassandra

DATASTAX

NOSQL Joins - Across the spark cluster, to Cassandra (multi-datacenter)

```
val sparkContext = new SparkContext(sparkConf)
val cc = new CassandraSQLContext(sparkContext)
cc.setKeyspace("nosql_joins")

cc.sql("""
  SELECT test1.a, test1.b, test1.c, test2.a
  FROM test1 AS test1
  JOIN
    test2 AS test2 ON test1.a = test2.a
    AND test1.b = test2.b
    AND test1.c = test2.c
“""").map(Data(_))
.saveToCassandra("nosql_joins","table3")
```

Spark SQL: Txt, Parquet, JSON Support

DATASTAX

```
import com.datastax.spark.connector._
import org.apache.spark.sql.{Row, SQLContext}

val sql = new SQLContext(sc)
val json = sc.parallelize(Seq(
"""{"user":"helena","commits":98, "month":12, “year":2014}""",
"""{"user":"pkolaczk", "commits":42, "month":12, "year":2014}"""))

sql.jsonRDD(json).map(MonthlyCommits(_)).flatMap(doSparkWork())
   .saveToCassandra("githubstats","monthly_commits")

sc.cassandraTable[MonthlyCommits]("githubstats","monthly_commits")
  .collect foreach println

sc.stop()
```

What's New In Spark?

DATASTAX

- Petabyte sort record
  - Myth busting:
    - Spark is in-memory. It doesn't work with big data
    - It's too expensive to buy a cluster with enough memory to fit our data
- Application Integration
  - Tableau, Trifacta, Talend, ElasticSearch, Cassandra
- Ongoing development for Spark 1.2
  - Python streaming, new MLib API, Yarn scaling…

Recent / Pending Spark Updates

DATASTAX

- Usability, Stability & Performance Improvements
  - Improved Lightning Fast Shuffle!
  - Monitoring the performance long running or complex jobs
- Public types API to allow users to create SchemaRDD's
- Support for registering Python, Scala, and Java lambda functions as UDF
- Dynamic bytecode generation significantly speeding up execution for queries that perform complex expression evaluation

Recent Spark Streaming Updates

DATASTAX

- Apache Flume: a new pull-based mode (simplifying deployment and providing high availability)
- The first of a set of streaming machine learning algorithms is introduced with streaming linear regression.
- Rate limiting has been added for streaming inputs

Recent MLib Updates

DATASTAX

- New library of statistical packages which provides exploratory analytic functions *stratified sampling, correlations, chi-squared tests, creating random datasets…)
- Utilities for feature extraction (Word2Vec and TF-IDF) and feature transformation (normalization and standard scaling).
- Support for nonnegative matrix factorization and SVD via Lanczos.
- Decision tree algorithm has been added in Python and Java.
- Tree aggregation primitive
- Performance improves across the board, with improvements of around

Recent/Pending Connector Updates

DATASTAX

- Spark SQL (came in 1.1)
- Python API
- Official Scala Driver for Cassandra
- Removing last traces of Thrift!!
- Performance Improvements
  - Token-aware data repartitioning
  - Token-aware saving
  - Wide-row support - no costly groupBy call

Resources

DATASTAX

- http://spark.apache.org
- https://github.com/datastax/spark-cassandra-connector
- http://cassandra.apache.org
- http://kafka.apache.org
- http://akka.io
- https://groups.google.com/a/lists.datastax.com/forum/#!forum/spark-connector-user

www.spark-summit.org

DATASTAX

Thanks for listening!

SEPTEMBER 10 - 11, 2014 | SAN FRANCISCO, CALIF. | THE WESTIN ST. FRANCIS HOTEL

Cassandra Summit

@helenaedelson

- Follow-Up Blog Post
- slideshare.net/helenaedelson