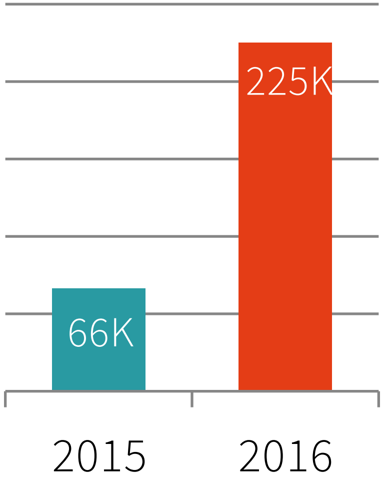
Simplifying Big Data in Apache Spark 2.0

Matei Zaharia
@matei_zaharia

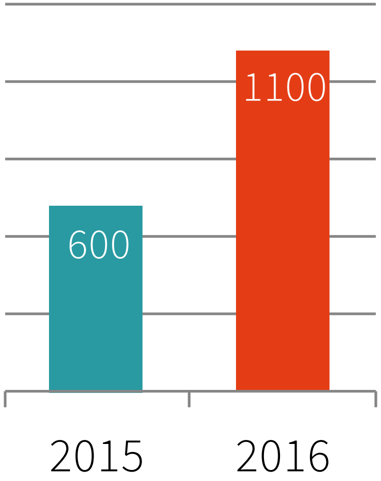
databricks™

A Great Year for Apache Spark

Meetup Members



Developers Contributing



New Major Version #

APACHE Spark™ 2.0

About Spark 2.0

Remains highly compatible with 1.x

Builds on key lessons and simplifies API

2000 patches from 280 contributors

What's Hard About Big Data?

Complex combination of processing tasks, storage systems & modes

- ETL, aggregation, machine learning, streaming, etc

Hard to get both productivity and performance

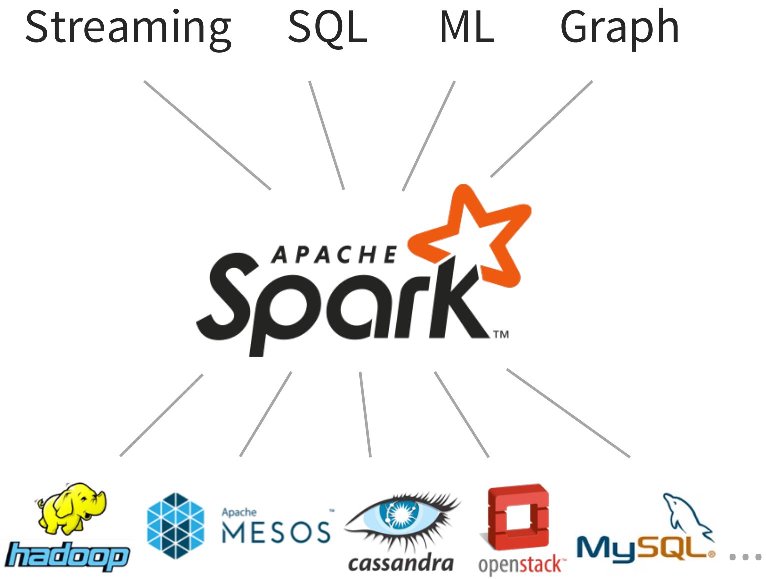
Apache Spark's Approach

Unified engine

- Express entire workflow in one API
- Connect existing libraries & storage

High-level APIs with space to optimize

- RDDs, DataFrames, ML pipelines



New in 2.0

Structured API improvements
(DataFrame, Dataset, SQL)

Whole-stage code generation

Structured Streaming

Simpler setup (SparkSession)

SQL 2003 support

MLlib model persistence

MLlib R bindings

SparkR user-defined functions

...

Original Spark API

Arbitrary Java functions on Java objects

```
val lines = sc.textFile(“s3://...”)
val points = lines.map(line  => new Point(line))
```

+ Can organize your app using functions, classes and types

– Difficult for the engine to optimize

- Inefficient in-memory format
- Hard to do cross-operator optimizations

Structured APIs

New APIs for data with a fixed schema (table-like)

- Efficient storage taking advantage of schema (e.g. columnar)
- Operators take expressions in a special DSL that Spark can optimize

DataFrames (untyped), Datasets (typed), and SQL

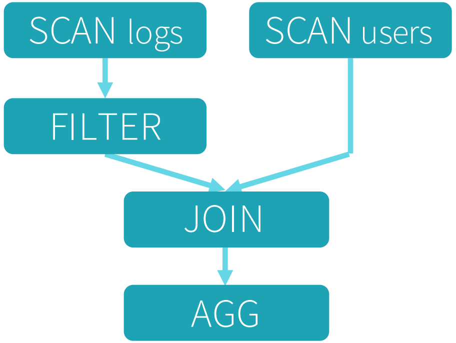
Structured API Example

```
events =
 sc.read.json(“/logs”)

stats =
 events.join(users)
  .groupBy(“loc”,“status”)
  .avg(“duration”)

errors = stats.where(
 stats.status == “ERR”)
```

DataFrame API



Optimized Plan

```
while(logs.hasNext) {
  e = logs.next
  if(e.status == “ERR”) {
    u = users.get(e.uid)
    key = (u.loc, e.status)
    sum(key) += e.duration
    count(key) += 1
  }
}
...
```

Specialized Code

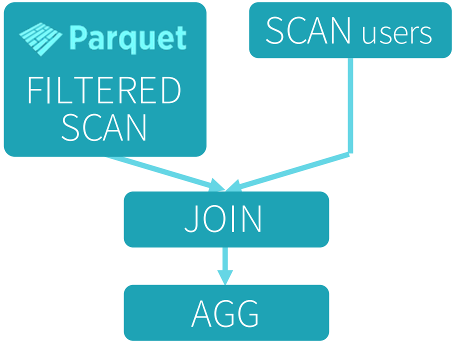
Structured API Example

```
events =
 sc.read.json(“/logs”)

stats =
 events.join(users)
  .groupBy(“loc”,“status”)
  .avg(“duration”)

errors = stats.where(
 stats.status  == “ERR”)
```

DataFrame API



Optimized Plan

```
while(logs.hasNext)  {
  e = logs.next
  if(e.status  == “ERR”) {
    u = users.get(e.uid)
    key = (u.loc, e.status)
    sum(key) += e.duration
    count(key) += 1
  }
}
...
```

Specialized Code

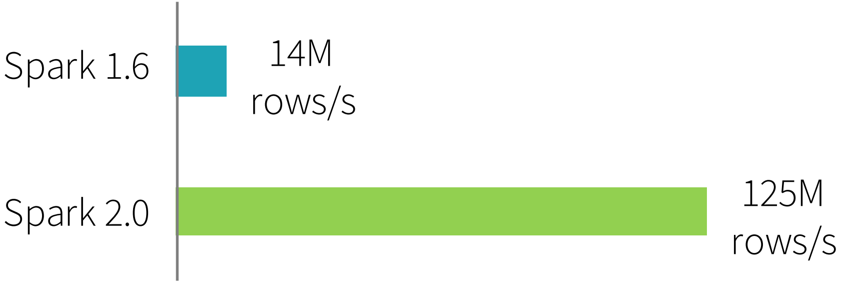
New in 2.0

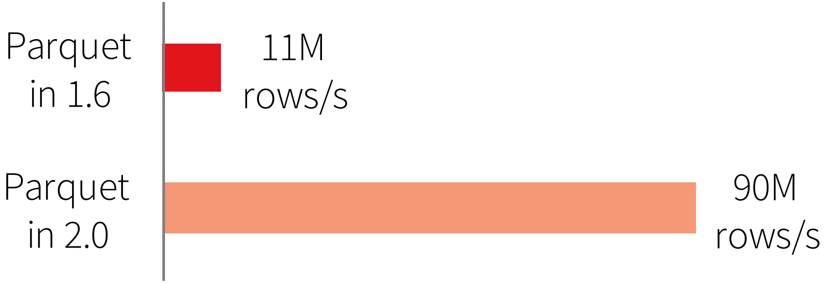
Merging DataFrame & Dataset

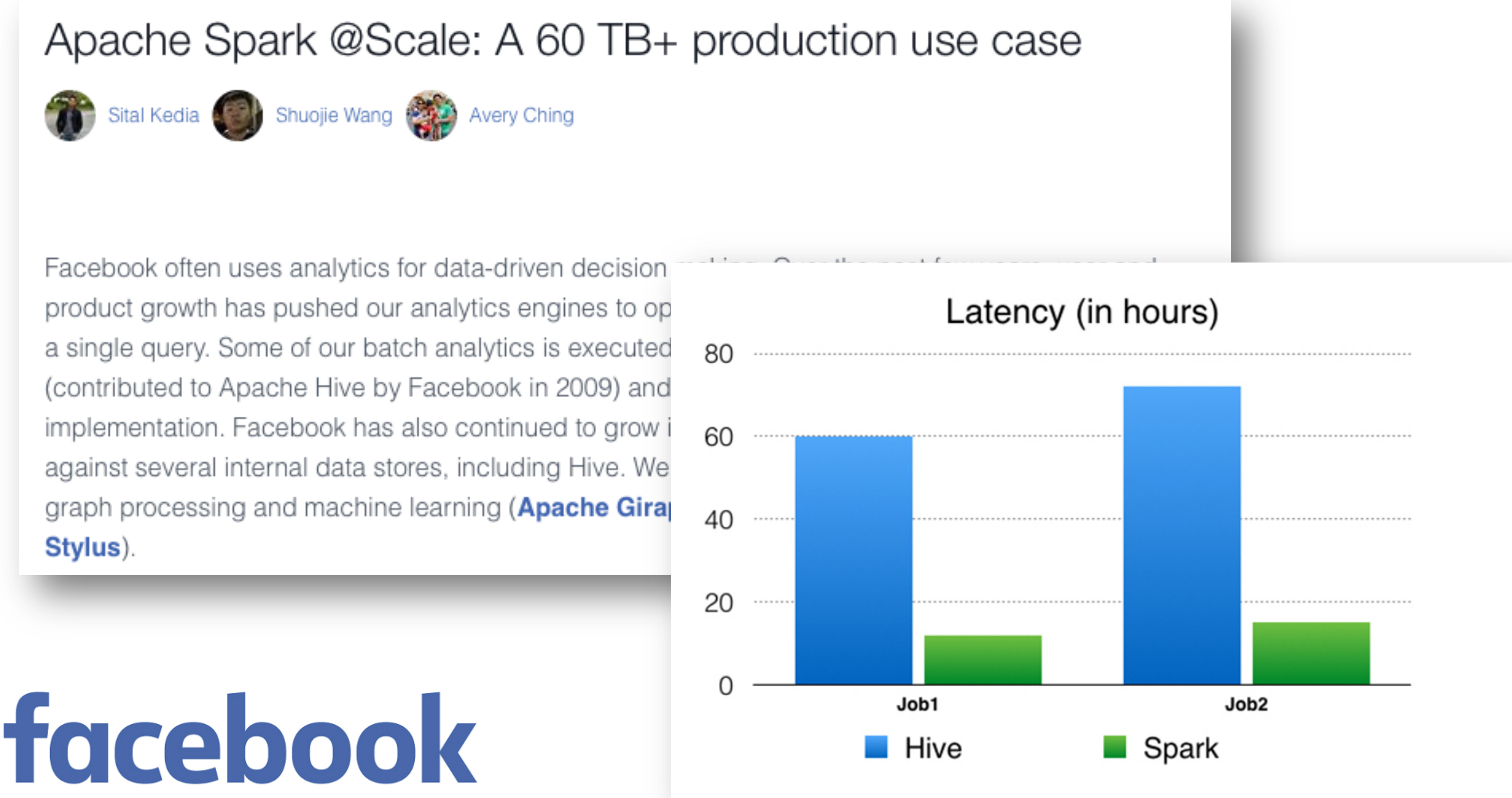
- DataFrame = Dataset[Row]

Whole-stage code generation

- Fuse across multiple operators
- Optimized Parquet I/O







Apache Spark @Scale: A 60 TB+ production use case

Sital Kedia Shuojie Wang Avery Ching

Facebook often uses analytics for data-driven decision
product growth has pushed our analytics engines to op
a single query. Some of our batch analytics is executed
(contributed to Apache Hive by Facebook in 2009) and
implementation. Facebook has also continued to grow i
against several internal data stores, including Hive. We
graph processing and machine learning (**Apache Gira**
Stylus).

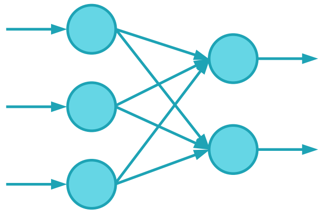
facebook

Beyond Batch & Interactive: Higher-Level API for Streaming

What's Hard In Using Streaming?

Complex semantics

- What possible results can the program give?
- What happens if a node runs slowly? If one fails?



Integration into a complete application

- Serve real-time queries on result of stream
- Give consistent results with batch jobs

Structured Streaming

High-level streaming API based on DataFrames / Datasets

- Same semantics & results as batch APIs
- Event time, windowing, sessions, transactional I/O

Rich integration with complete Apache Spark apps

- Memory sink for ad-hoc queries
- Joins with static data
- Change queries at runtime

Not just streaming, but "continuous applications"

Structured Streaming API

Incrementalize an existing DataFrame/Dataset/SQL query

Example
batch job:

```
logs = ctx.read.format(“json”).open(“hdfs://logs”)

logs.groupBy(“userid”, “hour”).avg(“latency”)
    .write.format(”parquet”)
    .save(“s3://...”)
```

Structured Streaming API

Incrementalize an existing DataFrame/Dataset/SQL query

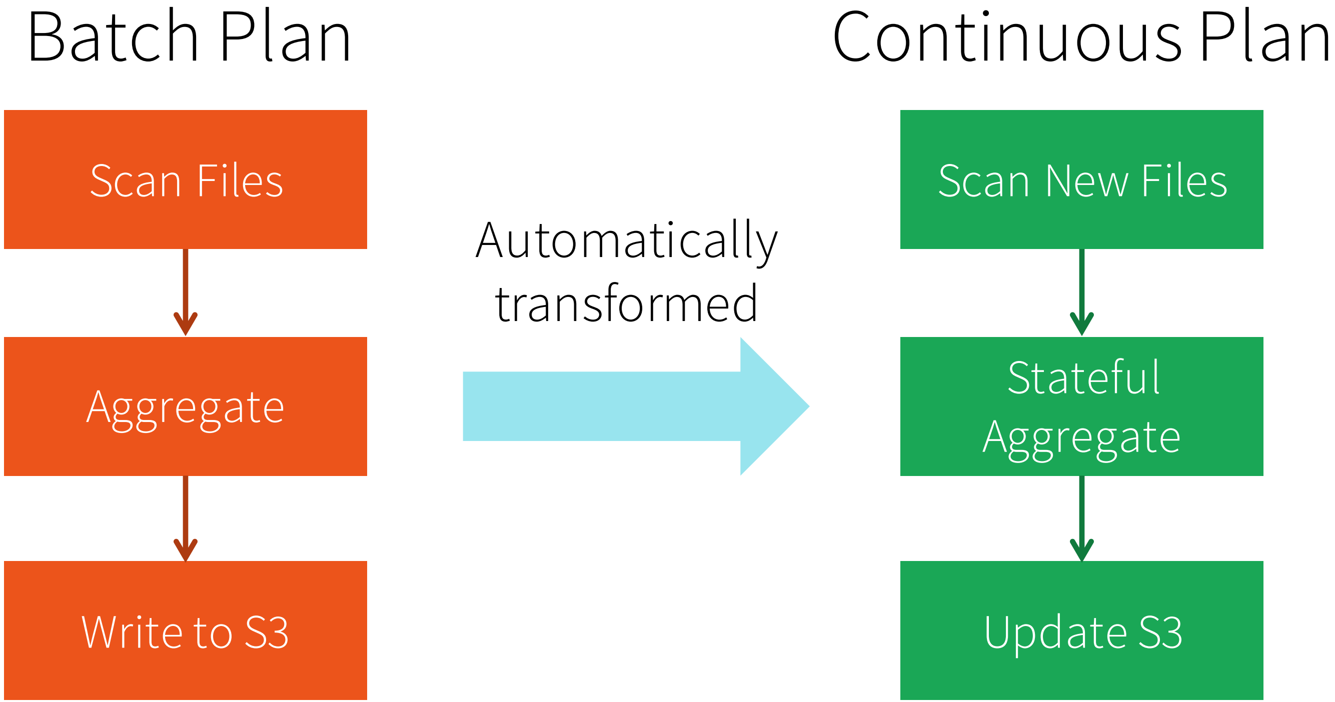
Example as streaming:

```
logs = ctx.readStream.format(“json”).load(“hdfs://logs”)

logs.groupBy(“userid”, “hour”).avg(“latency”)
    .writeStream.format(”parquet")
    .start(“s3://...”)
```

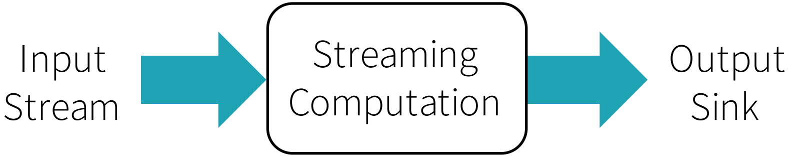
Results always same as a batch job on a prefix of the data

Under the Hood

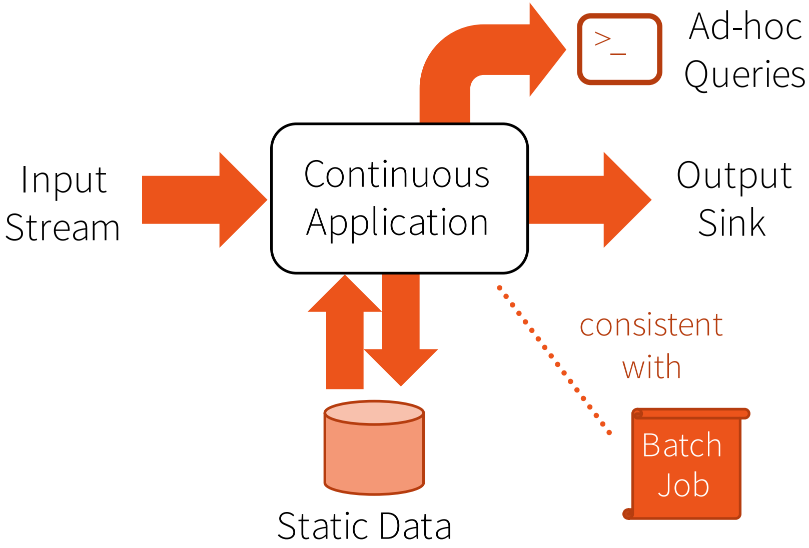


End Goal: Full Continuous Apps

Pure Streaming System



Continuous Application



databricks

Development Status

2.0.1: supports ETL workloads from file systems and S3

2.0.2: Kafka input source, monitoring metrics

2.1.0: event time aggregation workloads & watermarks

Demo

Greg Owen

databricks™