Databricks

Building and Operating a Big Data Service Based on Apache Spark

Ali Ghodsi <ali@databricks.com>
Cloud Computing and Big Data

• Three major trends
  – Computers not getting any faster
  – More people connected to the Internet
  – More devices collecting data

• Computation moving to the cloud
The Dawn of Big Data

• Most companies collect lots of data
  – Cheap storage (hardware, software)

• Everyone is hoping to extract *insights*
  – Great examples (Netflix, Uber, Ebay)

• Big Data is Hard!

*Working with Big Data is Hard*

“Through 2017, 60% of big-data projects will fail to go beyond piloting and experimentation and will be abandoned.”

GARTNER
Big Data is Hard

• Compute the average of 1,000 integers
• Compute the average of 10 terabyte of integers
Goal: Make Big Data Simple
The Challenges of Data Science

Building a cluster

Import and explore data with different tools

ETL

Data Exploration

Advanced Analytics

Data Warehousing

Dashboards & Reports

Production Deployment

Build and deploy data applications

Import and explore data with different tools

Dashboards & Reports

Production Deployment

Build and deploy data applications

Import and explore data with different tools

Dashboards & Reports

Production Deployment

Build and deploy data applications
Databricks is an End-to-End Solution

Single tool for Ingest, Exploration, Advanced Analytics, Production, Visualization

- Automatically Managed Clusters
- ETL: Diverse data source connectors
- Data Exploration: Notebooks & visualization
- Advanced Analytics: Built-in libraries
- Data Warehousing
- Dashboards & Reports: Real-time query engine, Dashboards, 3rd party apps

Production Deployment: Job scheduler

3rd party apps: Qlik, Tableau
Databricks in a nutshell

Talk outline

• Apache Spark
  – ETL, interactive queries, streaming, machine learning

• Cluster and Cloud Management
  – Operating thousands of machines in the cloud

• Interactive Workspace
  – Notebook environment, Collaboration, Visualization, Versioning, ACLs

• Lessons
  – Lessons in building a large scale distributed system in the cloud
PART I: Apache Spark

What we added to Spark
Apache Spark

- Resilient Distributed Datasets (RDDs) as core abstraction
  - Collection of objects
  - Like a LinkedList `<MyObjects>`

- Spark RDDs are distributed
  - RDD collections are partitioned
  - RDD partitions can be cached
  - RDD partitions can be recomputed
RDDS continued

• RDDS can be composed
  – All RDDS initially derived from data source
  – RDDS can be created from other RDDS
  – Two basic operations: map & reduce
  – Many other operators: join, filter, union etc

```scala
val text = sc.textFile("s3://my-bucket/wikipedia")
val words = text.flatMap(line => line.split(" "))
val pairs = words.map(word => (word, 1))
val result = pairs.reduceByKey((a, b) => a + b)
```
Spark Libraries on top of RDDs

- **SQL (Spark SQL)**
  - Full Hive SQL support with UDF, UDAFs, etc
  - how: Internally keep RDDs of row objects (or RDD of column segments)

- **Machine Learning (MLlib)**
  - Library of machine learning algorithms
  - how: Cache an RDD, repeatedly iterate it

- **Streaming (Spark Streaming)**
  - Streaming of real-time data
  - how: Series of RDDs, each containing seconds of real-time data

- **Graph Processing (GraphX)**
  - Iterative computation on graphs (e.g. social network)
  - how: RDD of Tuple<Vertex, Edge, Vertex> and perform self joins
Unifying Libraries

• Early user feedback
  – Different use cases for R, Python, Scala, Java, SQL
  – How to intermix and go across these?

• Explosion of R Data Frames and Python Pandas
  – DataFrame is a table
  – Many procedural operations
  – Ideal for dealing with semi-structured data

• Problem
  – Not declarative, hard to optimize
  – Eagerly executes command by command
  – Language specific (R dataframes, Pandas)
Unifying Libraries

• Early user feedback
  – Different use cases for R, Python, Scala, Java, SQL
  – How to intermix and go across these?

Common performance problem in Spark

```scala
val pairs = words.map(word => (word, 1))
val grouped = pairs.groupByKey()
val counts = grouped.map((key, values) => (key, values.sum))
```

• Problem
  – Not declarative, hard to optimize
  – Eagerly executes command by command
  – Language specific (R dataframes, Pandas)

 edadabricks
Spark Data Frames

• Procedural DataFrames vs declarative SQL
  – Two different approaches

• Developed DataFrames for Spark
  – DataFrames situated above the SQL optimizer
  – DataFrame operations available in R, Python, Scala, Java
  – SQL operations return DataFrames

users = context.sql("select * from users")  # SQL
young = users.filter(users.age < 21)  # Python
young.groupBy("gender").count()

tokenizer = Tokenizer(inputCol="name", outputCol="words")  # ML
hashingTF = HashingTF(inputCol="words", outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])
model = pipeline.fit(young)  # model
Proliferation of Data Solutions

• Customers already run a slew of data management systems
  – MySQL category, Cassandra category, S3 category, HDFS category
  – ETL all data over to Databricks?

• We added Spark Data Source API
  – Open APIs for implementing your own data source
  – Examples: CSV, JDBC, Parquet/Avro, ElasticSearch, RedShift, Cassandra

• Features
  – Pushdown of predicates, aggregations, column pruning
  – Locality information
  – User Defined Types (UDTs), e.g. vectors
Proliferation of Data Solutions

- Customers already run a slew of data management systems
  - MySQL category, Cassandra category, S3 category, HDFS category
  - ETL all data over to Databricks

- We added Spark Data Source API
  - Open APIs for implementing your own data source
  - Examples: CSV, JDBC, Parquet/Avro, ElasticSearch, RedShift, Cassandra

- Features
  - Pushdown of predicates, aggregations, column pruning
  - Locality information
  - User Defined Types (UDTs), e.g. vectors

```scala
class PointUDT extends UserDefinedType[Point]
{
    def dataType = StructType(Seq(
        StructField ("x", DoubleType),
        StructField ("y", DoubleType) ))

    def serialize(p: Point) = Row(p.x, p.y)

    def deserialize(r: Row) =
        Point(r.getDouble (0), r.getDouble (1))
}
```

- User Defined Types (UDTs), e.g. vectors
Modern Spark Architecture

Spark SQL  Spark Streaming  MLlib  GraphX
Modern Spark Architecture

Data Sources

Spark Core

DataFrames

- Spark SQL
- Spark Streaming
- MLlib
- GraphX

Languages:
- Scala
- Java
- Python
- R
Databricks as just-in-time Datawarehouse

- **Traditional datawarehouse**
  - Every night ETL all relevant data to a warehouse
  - Precompute cubes of fact tables
  - Slow, costly, poor recency

- **Spark JIT datawarehouse**
  - Switzerland of storage: NoSQL, SQL, cloud, …
  - Storage remains at source of truth
  - Spark used to directly read and cache date
PART II: Cluster Management
Spark as a Service in the Cloud

- Experience with Mesos, YARN, …
  - Use off-the-shelf cluster manager?

- Problems
  - Existing cluster managers were not cloud-aware
Cloud-Aware Cluster Management

• Instance manager
  – Responsible for acquiring machines from cloud provider

• Resource manager
  – Schedule and configure isolated containers on machine instances

• Spark cluster manager
  – Monitor and setup Spark clusters
Instance manager’s job is to manage machine instances

- Pluggable cloud providers
  - General interface that can be plugged in with AWS, ...
  - Availability management (AZ, 1h), configuration management (VPCs)

- Fault-handling
  - Terminated or slow instances, spot price hikes
  - Seamlessly replace machines

- Payment management
  - Bid for spot instances, monitor their price
  - Recording cluster usage for payment system
Databricks Resource Manager

Resource manager’s job is to multiplex tenants on instances

- Isolates tenants using container technology
  - Manages multiple versions of Spark
  - Configures firewall rules, filters traffic

- Provides fast SSD/in-memory caching across containers
  - ramdisk for a fast in-memory cache, mmap to access from Spark JVM
  - Bind-mount into containers for shared in-memory cache
Databricks Spark Cluster Manager

Spark CM’s job is to setup Spark clusters and multiplex REPLs

- Setting up Spark clusters
  - Currently using Standalone mode Spark
  - Dynamic resizing of clusters based on load (wip)

- Multiplexing of multiple REPLs
  - Many interactive REPLs/notebooks on the same Spark cluster
  - ClassLoader isolation and library management
PART III: Interactive Workspace
Collaborative Workspace

- Problem
  - Real time collaboration on notebooks
  - Version control of notebooks
  - Access control on notebooks
Pub/sub-based TreeStore

- **Web application server**
  - Stores an in-memory representation of Databricks workspace

- **TreeStore is a directory service + a pub-sub service**
  - In-memory tree structure representing: 
    *directories, notebooks, commands, results*
  - Browsers subscribe to subtrees and get notifications on updates
  - Special handler sends delta-updates over web sockets

- **Usage**
  - Subscribe to a notebook, see live edits of notebook
  - Used to create a collaborative environment
PART IV: Lessons
Lessons

• Loose coupling necessary but hard
  – Narrow well-defined APIs, backwards compatibility, upgrades

• State management very hard at scale
  – Legacy state: databases, configurations, machines, data formats…

• Cloud software development is superior
  – Two week sprints, two week releases, SCRUM …

• Testing is key for evolution and scale
  – Step-wise refinement for extension, testing pyramid 70/20/10

• Combine bottom-up with top-down approach
  – Top-down for quick results, bottom-up for modularity/reuse
Thank you & Questions

*Databricks is hiring, taking interns,* …

E-mail<ali@databricks.com>