Update on Apache Spark

Ion Stoica
UC Berkeley

http://www.istc-cc.cmu.edu/
Apache Spark

2009: Started at AMPLab, UC Berkeley by Databricks co-founders
• Fault tolerance, in-memory storage
• Powerful programming API
• Unified computation engine

2010: Open Source

2013: Apache Project

Today: widely popular
Adoption

1000+ companies

[Logos: Goldman Sachs, Toyota, Capital One, Novartis, Verizon, Thomson Reuters, Salesforce, Adobe, eBay, Uber, Autodesk, Airbnb, Telefonica, NBCUniversal, Palantir, Microsoft, Netflix, Databricks, Cloudera, Intel, Hortonworks, MapR, IBM, Oracle, SAP, Qlik, DataStax, TIBCO, Amazon Web Services, Tableau, Alteryx, Blue Data, Informatica, MicroStrategy, ...]
Growth is Accelerating

• Number of total contributors increased **3x** since last year
  ▫ Most active Apache project since 2014
    ▫ **255** (June 2014) → **730** (June 2015)

• Number of meetup members increased **2x** since beginning of this year
  ▫ **12K** (Jan 2015) → **26K** (July 2015)

• Number of Spark Summit attendees increased **2x** last year; **3x** this year
Spark Packages

A community index of packages for Apache Spark.

109 packages

All (109)  Core (4)  Data Sources (18)  Machine Learning (25)  Streaming (18)  Graph (2)  PySpark (1)  Applications (4)

Deployment (5)  Examples (6)  Tools (10)

spark-avro
Integration utilities for using Spark with Apache Avro data
from: @databricks / owner: @pwendell / Latest release: 1.0.0 (04/10/15) / Apache-2.0 / ★★★★★ (7)

spark-redshift
Spark and Redshift integration
from: @databricks / owner: @pwendell / Latest release: 0.4.0-hadoop2 (05/20/15) / Apache-2.0 / ★★★★★ (2)
Spark based MOOCs (EdX)

“Intro to Big Data with Apache Spark”
  • Anthony Joseph, UC Berkeley
  • June 1\textsuperscript{st}, 5 weeks
  • 78,000+ registrations, 12\% completion

“Scalable Machine Learning with Spark”
  • Ameet Talwalkar, UCLA
  • June 22\textsuperscript{nd}, 5 weeks
  • 50,000+ registrations, 14\% completion
Our Goal for Spark

Unified engine across data workloads and platforms

Streaming  SQL  ML  Graph  Batch  ...

Spark

hadop  amazon web services  cassandra  openstack  Google Compute Engine  MySQL  ...
Past 2 Years

Fast growth in libraries and integration points
  ▫ 10x growth of ML library
  ▫ Pluggable data source API
  ▫ R language

Result: very diverse use of Spark
  ▫ Only 40% of users on Hadoop YARN
  ▫ Most users use at least 2 of Spark’s built-in libraries
  ▫ 98% of Databricks customers use SQL, 60% use Python
New Directions

Dataframes API: raising the level of abstraction

Project Tungsten: bringing Sparks’ performance closer to the bare metal

Some of the largest changes to Spark (Core) since it has been created
Single-node tabular data structure, with API for

- relational algebra (filter, join, ...)
- math and stats
- input/output (CSV, JSON, ...)
- ...

Google Trends for “dataframe”
# Data frame: lingua franca for “small data”

head(flights)

```r
#> Source: local data frame [6 x 16]
#>
#>   year  month  day dep_time dep_delay arr_time arr_delay carrier tailnum
#> 1  2013     1     1      517        2     830         11    UA   N14228
#> 2  2013     1     1      533        4     850         20    UA   N24211
#> 3  2013     1     1      542        2     923         33    AA   N619AA
#> 4  2013     1     1      544       -1    1004        -18    B6   N804JB
#> ...
```

Spark DataFrame

Distributed data frame for Java, Python, R, Scala

Similar APIs as single-node tools (Pandas), i.e. easy to learn

> head(filter(df, df$waiting < 50))  # an example in R
## eruptions waiting
##1 1.750 47
##2 1.750 47
##3 1.867 48
Scaling

Spark DataFrame

Existing Single-node Data Frames

KB  MB  GB  TB  PB

Data size
Spark and Python/R

- Spark DF
  - scalability
  - multi-core
  - multi-machines

- Python/R DF
  - wealth
  - of libraries

- Stats
- Machine Learning
- Viz
Spark RDD Execution

Java/Scala API

opaque closures (user-defined functions)

JVM Execution

Python API

Python Execution
Spark DataFrame Execution

DataFrame

Logical Plan

Catalyst optimizer

Physical Execution

Intermediate representation for computation
Spark DataFrame Execution

- **Python DF**
- **Java/Scala DF**
- **R DF**

Logical Plan

Catalyst optimizer

Physical Execution

Simple wrappers to create logical plan

Intermediate representation for computation
Benefit of Logical Plan: Simpler Frontend

Python: ~2000 line of code (built over a weekend)

R: ~1000 line of code

Much easier to add new language bindings (i.e., Julia, Clojure, ...
Runtime for an example aggregation workload
Benefits: Performance Parity Across Languages

Runtime for an example aggregation workload (secs)
Tungsten
Hardware Trends

Storage

Network

CPU
Hardware Trends

2010

Storage 50+MB/s (HDD)

Network 1Gbps

CPU ~3GHz
## Hardware Trends

<table>
<thead>
<tr>
<th>Category</th>
<th>2010</th>
<th>2015</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage</td>
<td>50+MB/s</td>
<td>500+MB/s</td>
</tr>
<tr>
<td>(HDD)</td>
<td></td>
<td>(SSD)</td>
</tr>
<tr>
<td>Network</td>
<td>1Gbps</td>
<td>10Gbps</td>
</tr>
<tr>
<td>CPU</td>
<td>~3GHz</td>
<td>~3GHz</td>
</tr>
<tr>
<td>Component</td>
<td>2010</td>
<td>2015</td>
</tr>
<tr>
<td>---------------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>Storage</td>
<td>50+MB/s (HDD)</td>
<td>500+MB/s (SSD)</td>
</tr>
<tr>
<td>Network</td>
<td>1Gbps</td>
<td>10Gbps</td>
</tr>
<tr>
<td>CPU</td>
<td>~3GHz</td>
<td>~3GHz</td>
</tr>
</tbody>
</table>
Tungsten: Preparing Spark for Next 5 Years

Substantially speed up execution by optimizing CPU efficiency, via:

(1) Off-heap memory management
(2) Runtime code generation
(3) Exploiting cache locality
1. Off-Heap Memory Management

Store data off-heap to avoid object overhead & GC
- For RDDs: fast serialization libraries
- For DataFrames & SQL: binary format we compute on directly

2-10x space saving, especially for strings, nested objects

Can use new RAM-like devices, e.g. flash, 3D XPoint
2. Runtime Code Generation

Generate Java code for DataFrame and SQL expressions requested by user

Avoids virtual calls and generics/boxing

Can do same in core, ML and graph
  - Code-gen serializers, fused functions, math expressions

Evaluating “SELECT a+a+a” (time in seconds)

- Interpreted Projection: 36.7
- Code gen: 9.4
- Handwritten: 9.3
3. Cache-Aware Algorithms

Use custom memory layout to better leverage CPU cache

Example: AlphaSort-style prefix sort
- Store prefixes of sort key inside pointer array
- Compare prefixes to avoid full record fetches + comparisons

Naïve layout

Cache friendly layout
Tungsten Performance Results

Run time (seconds)

Data set size (relative)

- Default
- Code Gen
- Tungsten onheap
- Tungsten offheap
Unified API, One Engine, Automatically Optimized

language frontend
- SQL
- Python
- Java/Scala
- R
- ...

DataFrame
Logical Plan

Tungsten backend
- JVM
- LLVM
- GPU
- NVRAM
- ...

Unified API, One Engine, Automatically Optimized
Spark New Architecture

SQL  Python  R  Streaming  Advanced Analytics (ML)

DataFrame

Tungsten Execution
Growing community

Some of the most exciting times ahead!

- Dataframes
- Project Tunsten: Memory and CPU efficiency
- Others
  - Network and disk I/O optimizations
  - Adaptive query execution

Intel a great collaborator along the years

- Major contributions to Streaming, ML