A look ahead at Spark’s development

Reynold Xin @rxin
Spark Summit EU, Amsterdam
Oct 29th, 2015
Spark stack diagram

SQL  |  Streaming  |  MLlib  |  GraphX

Spark Core (RDD)
Spark stack diagram
(a different take)

Frontend
(user facing APIs)

Backend
(execution)
Spark stack diagram
(a different take)

Frontend
(RDD, DataFrame, ML pipelines, …)

Backend
(scheduler, shuffle, operators, …)
Last 12 months of Spark evolution

Frontend
DataFrames
Data sources
R
Machine learning pipelines
...

Backend
Project Tungsten
Sort-based shuffle
Netty-based network
...

---
databricks
Last 12 months of Spark evolution

**Frontend**
- DataFrames
- Data sources
- R
- Machine learning pipelines
- ...

**Backend**
- Project Tungsten
- Sort-based shuffle
- Netty-based network
- ...

---

![Dataproc logo](image_url)
DataFrame: A Frontend Perspective
Spark DataFrame

Scalable data frame for Java, Python, R, Scala

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

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

*dataricks*
Spark RDD Execution

Java/Scala frontend

opaque closures (user-defined functions)

Java/Scala backend

Python frontend

Python backend
Spark DataFrame Execution

1. DataFrame frontend
2. Logical Plan
3. Catalyst optimizer
4. Physical execution

Intermediate representation for computation
Spark DataFrame 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

i.e. much easier to add new language bindings (Julia, Clojure, …)
Performance

[Bar chart showing runtime for an example aggregation workload for Python and Java/Scala within RDD]
Benefit of Logical Plan: Performance Parity Across Languages

Runtime for an example aggregation workload (secs)
Tungsten: A Backend Perspective
Hardware Trends

Storage

Network

CPU
<table>
<thead>
<tr>
<th>Component</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage</td>
<td>50+MB/s (HDD)</td>
</tr>
<tr>
<td>Network</td>
<td>1Gbps</td>
</tr>
<tr>
<td>CPU</td>
<td>~3GHz</td>
</tr>
</tbody>
</table>

Hardware Trends

2010
## Hardware Trends

<table>
<thead>
<tr>
<th></th>
<th>2010</th>
<th>2015</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Storage</strong></td>
<td>50+MB/s (HDD)</td>
<td>500+MB/s (SSD)</td>
</tr>
<tr>
<td><strong>Network</strong></td>
<td>1Gbps</td>
<td>10Gbps</td>
</tr>
<tr>
<td><strong>CPU</strong></td>
<td>~3GHz</td>
<td>~3GHz</td>
</tr>
</tbody>
</table>
## Hardware Trends

<table>
<thead>
<tr>
<th>Component</th>
<th>2010</th>
<th>2015</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage</td>
<td>50+MB/s (HDD)</td>
<td>500+MB/s (SSD)</td>
<td>10X</td>
</tr>
<tr>
<td>Network</td>
<td>1Gbps</td>
<td>10Gbps</td>
<td>10X</td>
</tr>
<tr>
<td>CPU</td>
<td>~3GHz</td>
<td>~3GHz</td>
<td>🥺</td>
</tr>
</tbody>
</table>
Project Tungsten

Substantially speed up execution by optimizing CPU efficiency, via:

1. Runtime code generation
2. Exploiting cache locality
3. Off-heap memory management
From DataFrame to Tungsten

Initial phase in Spark 1.5
More work coming in 2016
3 Things to Look Forward To
Dataset API in Spark 1.6

Typed interface over DataFrames / Tungsten

case class Person(name: String, age: Int)

val dataframe = read.json("people.json")
val ds: Dataset[Person] = dataframe.as[Person]

ds.filter(p => p.name.startsWith("M"))
  .groupBy("name")
  .avg("age")
“Encoder” to specify type information so Spark can translate it into DataFrame and generate optimized memory layouts.

Checkout SPARK-9999
Streaming DataFrames

Easier-to-use APIs (batch, streaming, and interactive)

And optimizations:
- Tungsten backends
- native support for out-of-order data
- data sources and sinks

val stream = read.kafka("...")
stream.window(5 mins, 10 secs)
  .agg(sum("sales"))
  .write.jdbc("mysql://...")
Largest VM in the Cloud

THURSDAY, JANUARY 8, 2015

DREW MCDANIEL
Principal Program Manager, Azure

G-Series Size Details

<table>
<thead>
<tr>
<th>VM Size</th>
<th>Cores</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard_G1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Standard_G2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Standard_G3</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>Standard_G4</td>
<td>16</td>
<td>22</td>
</tr>
<tr>
<td>Standard_G5</td>
<td>32</td>
<td>448 GiB</td>
</tr>
</tbody>
</table>
3D XPoint

- DRAM latency
- SSD capacity
- Byte addressable
Unified API, One Engine, Automatically Optimized

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

Tungsten backend
- JVM
- LLVM
- SIMD
- 3D XPoint
- ...

Dataframe Logical Plan
Tungsten Execution

Python

R

Streaming

Advanced Analytics

DataFrame (Dataset)

Tungsten Execution
## Office Hours Today @ Databricks booth

<table>
<thead>
<tr>
<th>Time</th>
<th>Topic Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>10:30 – 11:30</td>
<td>Spark general (Reynold)</td>
</tr>
<tr>
<td>13:00 – 14:00</td>
<td>R and data science (Hossein)</td>
</tr>
<tr>
<td>13:30 – 14:30</td>
<td>machine learning (Joseph)</td>
</tr>
<tr>
<td>14:00 – 15:00</td>
<td>Spark, YARN, etc (Andrew)</td>
</tr>
</tbody>
</table>