"S" + = Spark-Bench
Simulate, Test, Compare, Exercise, and Yes, Even Benchmark!

Emily May Curtin  @emilymaycurtin
The Weather Company  @ecurtin
Atlanta, GA  #SparkBench
Who Am I

- Artist, former documentary film editor
- 2017 Spark Summit East – Spark + Parquet
- Tabs, not spaces (at least for Scala) (where 1 tab = 2 spaces)
- Software Engineer, The Weather Company, an IBM Business
- Current lead developer of CODAIIIT/spark-bench
Atlanta!!!!
Spark-Bench

• History & Rewrite Motivation

• How Spark-Bench Works

• Detailed Example: Parquet vs. CSV Benchmarking

• More Examples
  – Cluster Hammer
  – Multiple Notebook Users
  – Varying over Spark parameters
2015 A.D.
2015 A.D.
2015 A.D.
2015 A.D.
Spark Benchmarking circa 2015 A.D.
Spark-Bench 2015

"SparkBench: A Comprehensive Benchmarking Suite For In Memory Data Analytics Platform Spark"
Min Li, Jian Tan, Yandong Wang, Li Zhang, Valentina Salapura IBM TJ Watson Research Center - APC 2015
Spark-Bench 2015

"SparkBench: A Comprehensive Benchmarking Suite For In Memory Data Analytics Platform Spark"
Min Li, Jian Tan, Yandong Wang, Li Zhang, Valentina Salapura IBM TJ Watson Research Center - APC 2015
2017 A.D.
2017 A.D.
2017 A.D.

LEMONADE

#SparkBench
Watson Data Platform

Data Science Experience

IBM Analytics Engine (Beta)

Watson Machine Learning

AND MUCH, MUCH MORE

#SparkBench
Hi-Bench, etc.
Spark vs. other big data engines

Too high level

Juuuust right, sorta

databricks/spark-perf
Spark vs. Spark, cluster performance

Too low level

Profilers, Tracers
Fine-grain details of individual Spark jobs and setups
As a researcher, I want to run a set of standard benchmarks with generated data against one instance of Spark, one time.

--The user story that CODAIT/spark-bench version 1.0 fulfilled
Good start, let’s keep going!
The Vision
As a Spark developer, I want to run regression tests with my changes in order to provide some authoritative performance stats with my PR.
As a Spark developer, I want to simulate multiple notebook users hitting the same cluster so that I can test my changes to the dynamic allocation algorithm.
As an enterprise user of Spark, I want to make my daily batch ingest job run faster by tuning parameters.
As a data scientist, I want to run several machine learning algorithms over the same set of data in order to compare results.
As anybody who has ever had to make a graph about Spark, I want to get numbers easily and quickly so I can focus on making beautiful graphs.
How It Works
## Structural Details

<table>
<thead>
<tr>
<th></th>
<th>Old Version</th>
<th>✦ New Version ✦</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Project Design</strong></td>
<td>Series of shell scripts calling individually built jars</td>
<td>Scala project built using SBT</td>
</tr>
<tr>
<td><strong>Build/Release</strong></td>
<td>Manually/Manually</td>
<td>SBT, TravisCI, auto-release to Github Releases on PR merge</td>
</tr>
<tr>
<td><strong>Configuration</strong></td>
<td>Scattered in shell script variables</td>
<td>Centralized in one config file with many new capabilities</td>
</tr>
<tr>
<td><strong>Parallelism</strong></td>
<td>None</td>
<td>Three different levels!</td>
</tr>
<tr>
<td><strong>Custom Workloads</strong></td>
<td>Requires writing a new subproject with all accompanying bash</td>
<td>Implement our abstract class, Bring Your Own Jar</td>
</tr>
</tbody>
</table>

#SparkBench
The Config File (a first glance)

```json
spark-bench = {
    spark-submit-config = [
        {
            workload-suites = [
                {
                    descr = "One run of SparkPi and that's it!"
                    benchmark-output = "console"
                    workloads = [
                        {
                            name = "sparkpi"
                            slices = 10
                        }
                    ]
                }
            ]
        }
    ]
}
```

#SparkBench
Workloads

- Atomic unit of spark-bench
- Optionally read from disk
- Optionally write results to disk
- All stages of the workload are timed
- Infinitely composable
Workloads

• Machine Learning
  – Kmeans
  – Logistic Regression
  – Other workloads ported from old version

• SQL queries

• Oddballs
  – SparkPi
  – Sleep
  – Cache Test
Workloads: Data Generators

• Graph data generator
• Kmeans data generator
• Linear Regression data generator
• More coming!
What About Other Workloads?

1. Active development, more coming!
2. Contributions welcome 😊

3. Bring Your Own Workload!

```java
{
    name = "custom"
    class = "com.seriouscompany.HelloString"
    str = ["Hello", "Hi"]
}
```
Custom Workloads

```json
{
  name = "custom"
  class = "com.seriouscompany.OurGiganticIngestDailyIngestJob"
  date = "Tues Oct 26 18:58:25 EDT 2017"
  source = "s3://buncha-junk/2017-10-26/"
}

{
  name = "custom"
  class = "com.example.WordGenerator"
  output = "console"
  rows = 10
  cols = 3
  word = "Cool stuff!!"
}
```

See our documentation on custom workloads!
Workload

Could be Kmeans, SparkPi, SQL, Sleep, etc.

```json
{
    name = "sparkpi"
    slices = 500000000
}
```
SparkPi

Borrowing from the classic Spark example, this workload computes an approximation of pi. From the Spark examples page: https://spark.apache.org/examples.html

Spark can also be used for compute-intensive tasks. This code estimates π by "throwing darts" at a circle. We pick random points in the unit square (0, 0) to (1,1)) and see how many fall in the unit circle. The fraction should be π / 4, so we use this to get our estimate.

SparkPi is particularly useful for exercising the computing power of Spark without the consideration of heavy I/O from data-reliant workloads.

The timing result of SparkPi will include the estimate of Pi that was generated.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>yes</td>
<td>–</td>
<td>“sparkpi”</td>
</tr>
<tr>
<td>slices</td>
<td>no</td>
<td>2</td>
<td>Number of partitions that will be spawned</td>
</tr>
</tbody>
</table>

Examples

```
Workload
Could be Kmeans, SparkPi, SQL, Sleep, etc.

```
{
    name = "sparkpi"
    slices = 500000000
}
```

Data Generation Workload

```
{
    name = "data-generation-kmeans"
    rows = 1000
    cols = 24
    output = "/tmp/kmeans-data.parquet"
    k = 45
}
```
Data Generation, Small Set

```json
{
  name = "data-generation-kmeans"
  rows = 100
  cols = 24
  output = "/tmp/small-data.parquet"
}
```

Data Generation, Large Set

```json
{
  name = "data-generation-kmeans"
  rows = 100000000
  cols = 24
  output = "/tmp/large-data.parquet"
}
```
Workloads running serially
Workloads running in parallel
Workload Suites

- Logical group of workloads
- Run workloads serially or in parallel
- Control the repetition of workloads
- Control the output of benchmark results
- Can be composed with other workload suites to run serially or in parallel
One workload suite with three workloads running serially

```
workload-suites = [ {
  descr = "Lol"
  parallel = false
  repeat = 1
  benchmark-output = "console"
  workloads = [
    {
      // Three workloads here
    }
  ]
}
]`
One workload suite with three workloads running in parallel.

```python
workload-suites = [ {
    descr = "Lol"
    parallel = true
    repeat = 1
    benchmark-output = "console"
    workloads = [
        { // Three workloads here
            ...
        }
    ]
}]
```

#SparkBench
Three workload suites running **serially**. Two have **workloads** running **serially**, one has **workloads** running in **parallel**.

*(config file is abbreviated for space)*

```python
suites-parallel = false
data_list = [
    { descr = "One" parallel = false },
    { descr = "Two" parallel = false },
    { descr = "Three" parallel = true }
]
```

#SparkBench
Three workload suites running **in parallel**.
Two have workloads running serially, one has workloads running in parallel.

**(config file is abbreviated for space)**

```python
suites-parallel = true
workload Suites = [
    {
        descr = "One"
        parallel = false
    },
    {
        descr = "Two"
        parallel = false
    },
    {
        descr = "Three"
        parallel = true
    }
]
```

#SparkBench
./bin/spark-bench.sh my-amazing-config-file.conf
Spark-Submit-Config

- Control any parameter present in a spark-submit script
- Produce and launch multiple spark-submit scripts
- Vary over parameters like executor-mem
- Run against different clusters or builds of Spark
- Can run serially or in parallel
One spark-submit with one workload suite with three workloads running serially

```
spark-bench = {
    spark-submit-config = [{
        spark-args = {
            master = "yarn"
        }
        workload-suites = [{
            descr = "One"
            benchmark-output = "console"
            workloads = [
                // . . .
            ]
        }]
    }]
}
```
Two spark-submits running in parallel, each with one workload suite, each with three workloads running \textit{serially}
Two spark-submits running **serially**, each with one workload suite, one with three workloads running **serially**, one with three workloads running **in parallel**.
One spark-submit with two workload suites running serially.

The first workload suite generates data in parallel.

The second workload suite runs different workloads serially.

The second workload suite repeats.
spark-bench = {
    spark-submit-parallel = false
    spark-submit-config = [{
        spark-args = { // . . . }
        suites-parallel = false
        workload-suites = [{
            descr = "Data Generation"
            parallel = true
            repeat = 1 // generate once and done!
            workloads = { // . . . }
        },
        workload-suites = [{
            descr = "Actual workloads that we want to benchmark"
            parallel = false
            repeat = 100 // repeat for statistical validity!
            workloads = { // . . . }
        }]
    }]
}
Detailed Example: Parquet vs. CSV
Example: Parquet vs. CSV

**Goal:** benchmark two different SQL queries over the same dataset stored in two different formats

1. Generate a big dataset in CSV format
2. Convert that dataset to Parquet
3. Run first query over both sets
4. Run second query over both sets
5. Make beautiful graphs
spark-bench = {
    spark-submit-config = [{
        spark-home = "/usr/iop/current/spark2-client/"
        spark-args = {
            master = "yarn"
            executor-memory = "28G"
            num-executors = 144
        }
    }
    conf = {
        "spark.dynamicAllocation.enabled" = "false"
        "spark.dynamicAllocation.monitor.enabled" = "false"
        "spark.shuffle.service.enabled" = "true"
    }
    suites-parallel = false

    // ...CONTINUED...
workload-suites = [
{
    descr = "Generate a dataset, then transform it to Parquet format"
    benchmark-output = "hdfs:///tmp/emily/results-data-gen.csv"
    parallel = false
    workloads = [
    {
        name = "data-generation-kmeans"
        rows = 10000000
        cols = 24
        output = "hdfs:///tmp/emily/kmeans-data.csv"
    },
    {
        name = "sql"
        query = "select * from input"
        input = "hdfs:///tmp/emily/kmeans-data.csv"
        output = "hdfs:///tmp/emily/kmeans-data.parquet"
    }
    ]
},

// ...CONTINUED...
{  descr = "Run two different SQL over two different formats"  benchmark-output = "hdfs:///tmp/emily/results-sql.csv"  parallel = false  repeat = 10  workloads = [
    {
      name = "sql"
      input = [
        "hdfs:///tmp/emily/kmeans-data.csv",
        "hdfs:///tmp/emily/kmeans-data.parquet"
      ]
      query = [
        "select * from input",
        "select `0`, `22` from input where `0` < -0.9"
      ]
      cache = false
    }
  ]  ]}

#SparkBench
## Parquet vs CSV Results

<table>
<thead>
<tr>
<th>name</th>
<th>timestamp</th>
<th>loadTime</th>
<th>queryTime</th>
<th>total_Runtime</th>
<th>run</th>
<th>cache</th>
<th>queryStr</th>
<th>input</th>
<th>spark.executor.instances</th>
<th>spark.submit.deployMode</th>
<th>spark.master</th>
<th>sp</th>
</tr>
</thead>
</table>
Examples: Cluster Hammer
Cluster Hammer
Cluster Hammer

#SparkBench
Cluster Hammer
Examples: Multi-User Notebooks
Multi-User Notebook

Data Generation, sample.csv and GIANT-SET.csv

Workloads, working with the sample.csv

Sleep Workload

```json
{
    name = "sleep",
    distribution = "poisson",
    mean = 30000
}
```

Same workloads, working with GIANT-SET.csv
Examples of Easy Extensions

• Add more "users" (workload suites)
• Have a workload suite that simulates heavy batch jobs while notebook users attempt to use the same cluster
• Toggle settings for dynamic allocation, etc.
• Benchmark a notebook user on a heavily stressed cluster (cluster hammer!)
Examples: Varying Spark Parameters
Different Versions of Spark

```
spark-bench = {
    spark-submit-parallel = false
    spark-submit-config = [{
        spark-home = [
            "/opt/official-spark-build/",
            "/opt/my-branch-of-spark-with-new-changes/"
        ]
        spark-args = {
            master = "yarn"
        }
        workload-suites = [{// workload suites}]
    }]
}
```

This will produce 2 spark-submits
Varying More Than One Param

```python
spark-bench = {
    spark-submit-parallel = false
    spark-submit-config = [{
        spark-home = [
            "/opt/official-spark-build/",
            "/opt/my-fork-of-spark-with-new-changes/"
        ]
        spark-args = {
            master = "yarn"
            executor-mem = ["2G", "4G", "8G", "16G"]
        }
        workload-suites = [{// workload suites}]
    }]}
}
```

This will produce 2 X 4 = 8 spark-submits
Letters in Winning Word of Scripps National Spelling Bee correlates with Number of people killed by venomous spiders
Final Thoughts
Installation

1. Grab the latest release from the releases page on Github

2. Unpack the tar

3. Set the environment variable for $SPARK_HOME.

4. Run the examples!

   ./bin/spark-bench.sh examples/minimal-example.conf
Future Work & Contributions

• More workloads and data generators
• TPC-DS (on top of existing SQL query benchmark)
• Streaming workloads
• Python

Contributions very welcome from all levels!

Check out our ZenHub Board for details
Shout-Out to My Team

Matthew Schauer @showermat

Craig Ingram @cin

Brad Kaiser @brad-kaiser

*not an actual photo of my team
Summary

Spark-Bench is a flexible, configurable framework for Testing, Benchmarking, Simulating, Comparing, and more!

https://codait.github.io/spark-bench/
That's all Folks!

#SparkBench
CODAIT/spark-bench

Emily May Curtin  @emilymaycurtin  #EUeco8
IBM Watson Data Platform  @ecurtin  #sparkbench
Atlanta, GA

Project
https://github.com/CODAIT/spark-bench

Documentation
https://CODAIT.github.io/spark-bench/

Come See Me At DevNexus 2018!
#SparkBench