WIFI SSID:Spark+AI-Summit | Password: UnifiedDataAnalytics
Performance Troubleshooting Using Apache Spark Metrics

Luca Canali, CERN

#UnifiedDataAnalytics #SparkAISummit
About Luca

• Data Engineer at CERN
  – Hadoop and Spark service, database services
  – 19+ years of experience with data engineering

• Sharing and community
  – Blog, notes, tools, contributions to Apache Spark

@LucaCanaliDB – http://cern.ch/canali
CERN: founded in 1954: 12 European States
Science for Peace and Development
Today: 23 Member States

Member States: Austria, Belgium, Bulgaria, Czech Republic, Denmark, Finland, France, Germany, Greece, Hungary, Israel, Italy, Netherlands, Norway, Poland, Portugal, Romania, Serbia, Slovak Republic, Spain, Sweden, Switzerland and United Kingdom

Associate Members in the Pre-Stage to Membership: Cyprus, Slovenia

Associate Member States: India, Lithuania, Pakistan, Turkey, Ukraine

Applications for Membership or Associate Membership: Brazil, Croatia, Estonia

Observers to Council: Japan, Russia, United States of America; European Union, JINR and UNESCO

~ 2600 staff
~ 1800 other paid personnel
~ 14000 scientific users
Budget (2019) ~ 1200 MCHF
Data at the Large Hadron Collider

LHC experiments data: >300 PB
Computing jobs on the WLCG Grid: using ~1M cores
Analytics Platform @CERN

- “Big Data” open source components
- Integrated with domain-specific software and existing infrastructure
- Users in: Physics, Accelerators, IT

HEP software

Experiments storage
HDFS
Personal storage

SPARK+AI SUMMIT 2019
Hadoop and Spark Clusters at CERN

• Spark running on clusters:
  – YARN/Hadoop
  – Spark on Kubernetes

<table>
<thead>
<tr>
<th>Accelerator logging (part of LHC infrastructure)</th>
<th>Hadoop - YARN - 30 nodes (Cores - 1200, Mem - 13 TB, Storage – 7.5 PB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Purpose</td>
<td>Hadoop - YARN, 65 nodes (Cores – 2.2k, Mem – 20 TB, Storage – 12.5 PB)</td>
</tr>
<tr>
<td>Cloud containers</td>
<td>Kubernetes on Openstack VMs, Cores - 250, Mem – 2 TB Storage: remote HDFS or EOS (for physics data)</td>
</tr>
</tbody>
</table>
Do the heavy lifting in spark and collect aggregated view to panda DF

```python
In [11]: df_loadavg_pandas = spark.sql("SELECT submitter_host, 
                          avg(body.LoadAvg) as avg, 
                          hour(from_unixtime(timestamp / 1000, 'yyy-MM-dd HH:mm:ss')) as hr 
                          FROM loadavg 
                          WHERE submitter_hostgroup = 'hadoop/itdb/datanode' 
                          AND dayofmonth(from_unixtime(timestamp / 1000, 'yyy-MM-dd HH:mm:ss')) = 15 
                          GROUP BY hour(from_unixtime(timestamp / 1000, 'yyy-MM-dd HH:mm:ss')), submitter_host")
                          .toPandas()
```

Visualize with seaborn

```python
In [10]: fig, ax = sns.heatmap(df_loadavg_pandas.pivot(index='submitter_host', columns='hr', values='avg'), cmap='Blues')
```

Out[10]: `Heatmap of loadAvg`

Sparkmonitor -> Jupyter extension for Spark monitoring, developed as a GSoC project with CERN. https://medium.com/@krishnanr/sparkmonitor-big-data-tools-for-physics-analysis-bbcdef68b35a
Performance Troubleshooting

Goals:
- Improving productivity
- Reducing resource usage and cost
- **Metrics**: latency, throughput, cost

How:
- Practice and methodologies
- **Gather** performance and workload data
Performance Methodologies and Anti-Patterns

Typical benchmark graph
- Just a simple measurement
- No root-cause analysis
- Guesses and generalization

Sound methodologies:
http://www.brendangregg.com/methodology.html
Workload and Performance Data

- You want data to find answers to questions like
  - What is my workload doing?
  - Where is it spending time?
  - What are the bottlenecks (CPU, I/O)?
  - How are systems resources used?
  - Why do I measure the {latency/throughput} that I measure?
    - Why is not 10x better?
Data + Context => Insights

Workload monitoring data + Spark architecture knowledge

Application

Info on application architecture

Info on computing environment

Agent takes produces: insights + actions
Measuring Spark

- Distributed system, parallel architecture
  - Many components, complexity increases when running at scale
  - Execution hierarchy: SQL -> Jobs -> Stages -> Tasks
  - Interaction with clusters and storage
Spark Instrumentation - WebUI

WebUI and History server: standard instrumentation

- Details on jobs, stages, tasks
- Default: http://driver_host:4040
- Details on SQL execution and execution plans
- https://github.com/apache/spark/blob/master/docs/web-ui.md
Task metrics:

• Instrument resource usage by executor tasks:
  — Time spent executing tasks,
  — CPU used, I/O metrics,
  — Shuffle read/write details, ..
  — SPARK-25170: 

  https://spark.apache.org/docs/latest/monitoring.html

SQL metrics:

• DataFrame/SQL operations. Mostly used by Web UI SQL tab.
  See SPARK-28935 + Web-UI documentation
How to Gather Spark Task Metrics

- Web UI exposes REST API
  Example: http://localhost:4040/api/v1/applications
  History server reads from Event Log (JSON file)
  - spark.eventLog.enabled=true
  - spark.eventLog.dir = <path>

- Programmatic interface via “Spark Listeners”
  sparkMeasure -> a tool and working example code of how to collect metrics with Spark Listeners
Spark Metrics in REST API

```
  "stageId" : 1,
  "attemptId" : 0,
  "numTasks" : 8,
  "numActiveTasks" : 0,
  "numCompleteTasks" : 8,
  "numFailedTasks" : 0,
  "numKilledTasks" : 0,
  "numCompletedIndices" : 8,
  "submissionTime" : "2019-10-07T15:18:55.405GMT",
  "firstTaskLaunchedTime" : "2019-10-07T15:18:55.530GMT",
  "executorDeserializeTime" : 245,
  "executorDeserializeCpuTime" : 78331585,
  "executorRunTime" : 1136181,
  "executorCpuTime" : 1126929606671,
  "resultSize" : 22387,
  "jvmGcTime" : 232,
  "resultSerializationTime" : 1,
  "memoryBytesSpilled" : 0,
  "diskBytesSpilled" : 0,
  "peakExecutionMemory" : 0,
  "inputBytes" : 0,
  "inputRecords" : 10000,
..."
Task Metrics in the Event Log

val df = spark.read.json("/var/log/spark-history/application_1567507314781_").filter("Event='SparkListenerTaskEnd'").select("Task Metrics.*").printSchema

 |-- Disk Bytes Spilled: long (nullable = true)
 |-- Executor CPU Time: long (nullable = true)
 |-- Executor Deserialize CPU Time: long (nullable = true)
 |-- Executor Deserialize Time: long (nullable = true)
 |-- Executor Run Time: long (nullable = true)
 |-- Input Metrics: struct (nullable = true)
 |   |-- Bytes Read: long (nullable = true)
 |   |-- Records Read: long (nullable = true)
 |-- JVM GC Time: long (nullable = true)
 |-- Memory Bytes Spilled: long (nullable = true)
 |-- Output Metrics: struct (nullable = true)
 |   |-- Bytes Written: long (nullable = true)
 |   |-- Records Written: long (nullable = true)
 |-- Result Serialization Time: long (nullable = true)
 |-- Result Size: long (nullable = true)
 |-- Shuffle Read Metrics: struct (nullable = true)
 |   |-- Fetch Wait Time: long (nullable = true)
 |   |-- Local Blocks Fetched: long (nullable = true)
 |   |-- Local Bytes Read: long (nullable = true)
 |   |-- Remote Blocks Fetched: long (nullable = true)
 |   |-- Remote Bytes Read: long (nullable = true)
 |   |-- Remote Bytes Read To Disk: long (nullable = true)
 |   |-- Total Records Read: long (nullable = true)
 |-- Shuffle Write Metrics: struct (nullable = true)
 |   |-- Shuffle Bytes Written: long (nullable = true)
 |   |-- Shuffle Records Written: long (nullable = true)
 |   |-- Shuffle Write Time: long (nullable = true)
 |-- Updated Blocks: array (nullable = true)
 |   |-- element: string (containsNull = true)

Spark Internal Task metrics:
Provide info on *executors' activity*:
Run time, CPU time used, I/O metrics, JVM Garbage Collection, Shuffle activity, etc.
Spark Listeners, @DeveloperApi

• Custom class, extends SparkListener

```scala
class StageInfoRecorderListener extends SparkListener {

• Methods react on events to collect data, example:

```scala
override def onStageCompleted(stageCompleted: SparkListenerStageCompleted): Unit = {
  val stageInfo = stageCompleted.stageInfo
  val taskMetrics = stageInfo.taskMetrics
  val jobId = StageIdtoJobId(stageInfo.stageId)
```

• Attach custom Lister class to Spark Session

  --conf spark.extraListeners=..
Spark Task Metrics, the Listener Bus and SparkMeasure Architecture

SparkListener Bus
- EventLogging Listener
- AppStatus and other Spark internal Listeners

Custom Listener in the Spark Context:
sparkMeasure
- StageInfoRecorder
- TaskInfoRecorder
- FlightRecorder modes:
  - Filesystem output
  - InfluxDBSink

Use cases and environments
- Interactive:
  Scala, Python, Jupyter notebook
- Code instrumentation

Metrics Output
- Online monitoring, workload analysis
SparkMeasure – Getting Started

```
bin/spark-shell --packages ch.cern.sparkmeasure:spark-measure_2.11:0.15

val stageMetrics = ch.cern.sparkmeasure.StageMetrics(spark)

val myQuery = "select count(*) from range(1000) cross join range(1000) cross join range(1000)"

stageMetrics.runAndMeasure(spark.sql(myQuery).show())
```
Scheduling mode = FIFO
Spark Context default degree of parallelism = 8
Aggregated Spark stage metrics:
numStages => 3
sum(numTasks) => 17
elapsedTime => 9103 (9 s)
sum(stageDuration) => 9027 (9 s)
sum(executorRunTime) => 69238 (1.2 min)
sum(executorCpuTime) => 68004 (1.1 min)
sum(executorDeserializeTime) => 1031 (1 s)
sum(executorDeserializeCpuTime) => 151 (0.2 s)
sum(resultSerializationTime) => 5 (5 ms)
sum(jvmGCTime) => 64 (64 ms)
sum(shuffleFetchWaitTime) => 0 (0 ms)
sum(shuffleWriteTime) => 26 (26 ms)
max(resultSize) => 17934 (17.0 KB)
sum(numUpdatedBlockStatuses) => 0
sum(diskBytesSpilled) => 0 (0 Bytes)
sum(memoryBytesSpilled) => 0 (0 Bytes)
max(peakExecutionMemory) => 0
sum(recordsRead) => 2000
sum(bytesRead) => 0 (0 Bytes)
sum(recordsWritten) => 0
sum(bytesWritten) => 0 (0 Bytes)
sum(shuffleTotalBytesRead) => 472 (472 Bytes)
sum(shuffleTotalBlocksFetched) => 8
sum(shuffleLocalBlocksFetched) => 8
sum(shuffleRemoteBlocksFetched) => 0
sum(shuffleBytesWritten) => 472 (472 Bytes)
sum(shuffleRecordsWritten) => 8
SparkMeasure, Usage Modes

• Interactive: use from shell or notebooks
  — Works with Jupyter notebooks, Azure, Colab, Databricks, etc.
• Use to instrument your code
• Flight recorder mode
  — No changes needed to the code
  — For Troubleshooting, for CI/CD pipelines, …
• Use with Scala, Python, Java

https://github.com/LucaCanali/sparkMeasure
from sparkmeasure import StageMetrics
stagemetrics = StageMetrics(spark)

stagemetrics.begin()
spark.sql("select count(*) from range(1000) cross join range(1000) cross join range(1000)").show()
stagemetrics.end()

# print report to standard output
stagemetrics.print_report()

# save session metrics data in json format (default)
df = stagemetrics.create_stagemetrics_DF("PerfStageMetrics")
stagemetrics.save_data(df.orderBy("jobId", "stageId"), "/tmp/stagemetrics_test1")
aggregatedDF = stagemetrics.aggregate_stagemetrics_DF("PerfStageMetrics")
stagemetrics.save_data(aggregatedDF, "/tmp/stagemetrics_report_test2")

https://github.com/LucaCanali/sparkMeasure/blob/master/docs/Instrument_Python_code.md
SparkMeasure on Notebooks: Local Jupyter and Cloud Services

```python
In [ ]:
1 spark = SparkSession
2 .builder
3 .master("local[*]")
4 .appName("Test sparkmeasure instrumentation of Python/PySpark code")
5 .config("spark.jars.packages";
6 .getOrCreate()

In [1]:
1 # Install the Python wrapper package
2 !pip install sparkmeasure

In [2]:
1 # Load the Python API for sparkmeasure package
2 # and attach the sparkMeasure Listener for stagemetrics to the active Spark session
3 from sparkmeasure import StageMetrics
4 stagemetrics = StageMetrics(spark)
```

https://github.com/LucaCanali/sparkMeasure/tree/master/examples
SparkMeasure on Notebooks:

Jupyter Magic: `%%sparkmeasure`

```python
In [3]:
# Define cell and line magic to wrap the instrumentation
from IPython.core.magic import (register_line_magic, register_cell_magic, register_line_cell_magic)

@register_line_cell_magic
def sparkmeasure(line, cell=None):
    """run and measure spark workload. Use: %sparkmeasure or %%sparkmeasure"
    val = cell if cell is not None else line
    stagemetrics.begin()
    eval(val)
    stagemetrics.end()
    stagemetrics.print_report()

In [4]: `%%sparkmeasure`
spark.sql("select count(*) from range(1000) cross join range(1000) cross join range(1000)").show()
```

Scheduling mode = FIFO
Spark Context default degree of parallelism = 8
Aggregated Spark stage metrics:
numStages => 3
sum(numTasks) => 17
elapsedTime => 13962 (14 s)
sum(stageDuration) => 13928 (14 s)
sum(executorRunTime) => 110345 (1.8 min)
sum(executorCpuTime) => 109816 (1.8 min)
... (note, output truncated to fit in slide)
SparkMeasure as Flight Recorder

Capture metrics and **write to files** when finished:

```bash
bin/spark-submit --master local[*] --packages ch.cern.sparkmeasure:spark-measure_2.11:0.15 \
--class org.apache.spark.examples.SparkPi \
--conf spark.extralListeners=ch.cern.sparkmeasure.FlightRecorderStageMetrics \
--conf spark.sparkmeasure.printToStdout=true \
--conf spark.sparkmeasure.outputFilename="/tmp/myoutput_$(date +%s).json" \
examples/jars/spark-examples_2.11-2.4.3.jar 10
```

Monitoring option: write to **InfluxDB** on the fly:

```bash
bin/spark-shell --master local[*] --packages ch.cern.sparkmeasure:spark-measure_2.11:0.15 \
--conf spark.sparkmeasure.influxdbURL="http://myInfluxDB:8086" \
--conf spark.extralListeners=ch.cern.sparkmeasure.InfluxDBSink
```
Spark Metrics System

- Spark is also instrumented using the Dropwizard/Codahale metrics library
- Multiple sources (data providers)
  - Various instrumentation points in Spark code
  - Including task metrics, scheduler, etc
  - Instrumentation from the JVM
- Multiple sinks
  - Graphite (InfluxDB), JMX, HTTP, CSV, etc
Ingredients for a Spark Performance Dashboard

• **Architecture**
  — Know how the “Dropwizard metrics system” works
  — Which Spark components are instrumented

• **Configure backend components**
  — InfluxDB and Grafana

• **Relevant Spark configuration parameters**

• **Dashboard graphs**
  — familiarize with available metrics
  — InfluxDB query building for dashboard graphs
Apache Spark Metrics System + InfluxDB + Grafana => Dashboard

Source: Spark components instrumented with dropwizard library metrics

Sink: collect and store the metrics using InfluxDB

Visible: using Grafana dashboards

Available sink endpoints:
- Graphite
- CSV
- Console
- JMX
- MetricsServlet
- Siif4
- StatsD

Components send metrics updates to the sink

Dashboard: queries pull data from InfluxDB
Send Spark Metrics to InfluxDB

- Edit $SPARK_HOME/conf/metrics.properties
- Alternative: use the config parameters spark.metrics.conf.*

```bash
$ SPARK_HOME/bin/spark-shell \
--conf "spark.metrics.conf.driver.sink.graphite.class"="org.apache.spark.metrics.sink.GraphiteSink" \
--conf "spark.metrics.conf.executor.sink.graphite.class"="org.apache.spark.metrics.sink.GraphiteSink" \
--conf "spark.metrics.conf.*.sink.graphite.host"="graphiteEndPoint_influxDB_hostName>" \
--conf "spark.metrics.conf.*.sink.graphite.port"=<graphite_listening_port> \
--conf "spark.metrics.conf.*.sink.graphite.period"=10 \
--conf "spark.metrics.conf.*.sink.graphite.unit"=seconds \
--conf "spark.metrics.conf.*.sink.graphite.prefix"="lucatest" \
--conf "spark.metrics.conf.*.source.jvm.class"="org.apache.spark.metrics.source.JvmSource"
```
Assemble Dashboard Components

Metrics written from Spark to InfluxDB
— Configuration of a Graphite endpoint in influxdb.conf
— Templates: how to ingest Spark metrics into InfluxDB series
  https://github.com/LucaCanali/Miscellaneous/tree/master/Spark_Dashboard

Grafana graphs built using data queried from InfluxDB
— Get started: Import an example dashboard definition

Kubernetes users: a helm chart to automate config at:
— https://github.com/cerndb/spark-dashboard
Grafana Dashboard

- Summaries
- Key metrics
- Graphs for drill-down analysis
Spark Dashboard - Examples

Graph: “number of active tasks” vs. time
  • Is Spark using all the available resources/cores?
  • Are there time ranges with significant gaps?

Identify possible issues:
  • Long tails
  • Stragglers
  • Data skew
Dashboard – I/O metrics

Graph: “HDFS Read Throughput” vs. time
Dashboard – Memory

Graphs of JVM memory usage

- Heap
- Off-heap
- Executors and driver
Dashboard – Executor CPU Utilization

Graph: “CPU utilization by executors’ JVM” vs. time

- Total JVM CPU:
- CPU used by tasks
- CPU used by GC
Task Time Drill Down, by Activity

Graph: Task total run time + drill down by component:
• CPU, Wait time, Garbage collection, etc

Investigation:
• CPU bound?
• Impact of GC
• I/O time?
• Other time?
Graph Annotations

Improvement:
• Mark SQL/job/stage begin and end timestamps

Implementation:
• SparkMeasure collects and writes query/jobs begin and end timestamps data to InfluxDB
• Grafana implements annotations
Spark Dashboard, Lessons Learned

- Very useful to search for bottlenecks
  - Many instrumented components
  - Drilldown where time is spent
  - Time evolution details
    - Time series of N# active tasks, CPU, I/O, memory, etc
- Effort: you have to understand the root causes
  - Use data to make and prove or disprove models
  - The instrumentation is still evolving
    - example: I/O time is not measured directly, Python UDF, etc
WIP: How to Measure I/O Time?

Goal:
• How much of the workload time is spent doing I/O (reading)?

Apache Spark does not instrument I/O time
• Apache Hadoop Filesystem API does not measure I/O time

Experimenting
• Added I/O read time instrumentation for HDFS and S3A to sandbox Hadoop fork
• Exported the metrics using Spark Executor Plugins SPARK-28091
Executor Plugins Extend Metrics

- User-defined executor metrics, SPARK-28091, target Spark 3.0.0
  - Example: add I/O metrics for s3a filesystem:

  ```
  /bin/spark-shell --jars <path>/sparkexecutorplugins_2.12-0.1.jar \
  --conf spark.executor.plugins=ch.cern.ExecutorPluginScala.S3AMetrics27
  https://github.com/cerndb/SparkExecutorPlugins
  ```

```scala
class S3AMetrics27 extends org.apache.spark.ExecutorPlugin {
  
  override def init(myContext: ExecutorPluginContext) = {
    
  val metricRegistry = myContext.metricRegistry

  metricRegistry.register(MetricRegistry.name("s3BytesRead"), new Gauge[Long] {
    override def getValue: Long = {
      val hdfsStats = FileSystem.getAllStatistics().asScala.find(s => s.getScheme.equals("s3a"))
      hdfsStats.map(_.getBytesRead).getOrElse(0L)
    }
  })
```
Metrics from OS Monitoring

• Very useful also to collect OS-based metrics
  – Hadoop: dashboard with HDFS and YARN metrics
  – OS host metrics: Collectd, Ganglia
  – Kubernetes: Prometheus-based monitoring and dashboard
Notable JIRAs about Metrics

- **Documentation improvements “Spark monitoring”**
  - Master, SPARK-26890, Add Dropwizard metrics list and configuration details
  - Spark 2.4.0, SPARK-25170: Add Task Metrics description to the documentation

- **Master, SPARK-23206 Additional Memory Tuning Metrics**
  - Master, SPARK-29064 Add Prometheus endpoint for executor metrics
  - WIP, SPARK-27189 Add memory usage metrics to the metrics system

- **Master, SPARK-28091 Extend Spark metrics system with user-defined metrics using executor plugins**

- **Master, SPARK-28475 Add regex MetricFilter to GraphiteSink**

- **CPU time used by JVM:**
  - Spark 2.4.0: SPARK-25228 Add executor CPU Time metric
  - Master: SPARK-26928, Add driver CPU Time to the metrics system,

- **Spark 2.3.0: SPARK-22190 Add Spark executor task metrics to Dropwizard metrics**
Conclusions

• **Performance troubleshooting by understanding**
  - Spark architecture + Spark instrumentation, Web UI
  - Spark Task metrics + Listeners, tool: sparkMeasure
  - Spark Metrics System, tool: Grafana dashboard

• **Contribute, adopt, share**
  - Instrumentation in Spark ecosystem keeps improving
  - Solid methodologies and tools are key
  - Share your results, tools, issues, dashboards..

#UnifiedDataAnalytics #SparkAISummit
Acknowledgements

• Colleagues at CERN
  — Hadoop and Spark service, in particular Prasanth Kothuri and Riccardo Castellotti

• Thanks to Apache Spark committers and community
  — Help with JIRAs and PRs

• References:
  — https://github.com/LucaCanali/sparkmeasure

#UnifiedDataAnalytics #SparkAIsummit
DON’T FORGET TO RATE AND REVIEW THE SESSIONS

SEARCH SPARK + AI SUMMIT

![Download on the App Store](apple.png)  ![GET IT ON Google Play](android.png)