# External Table

Luca's blog on databases, data platforms, performance.

Friday, August 24, 2018

## SparkMeasure, a tool for performance troubleshooting of Apache Spark workloads

### SparkMeasure

SparkMeasure simplifies the collection and analysis of **Spark task metrics data**. It is also intended as a working example of how to use Spark listeners for collecting and processing Spark performance metrics.
The work on sparkMeasure has been previously presented in this blog with examples. Recently, an **updated** version of sparkMeasure (version 0.13) introduces additional integration for the **PySpark** and **Jupyter** environments, improved **documentation** and additional features provided by the community via PRs (many thanks to the contributors).
At CERN Spark and Hadoop service we have been using sparkMeasure in a few occasions and found it useful for understanding the performance characteristics of Spark workloads and for performance troubleshooting.

### Download and deploy with examples

You can find sparkMeasure, its documentation with examples in the **sparkMeasure development repository** on GitHub and/or in its mirror **cerndb repo**.
You can deploy **sparkMeasure from Maven Central** or build with "`sbt package`". PySpark users can find the Python wrapper API on PyPI: "`pip install sparkmeasure`".

### Use sparkMeasure for measuring interactive and batch workloads

- Interactive: measure and analyze performance from shell or notebooks: using spark-shell (Scala), PySpark (Python) or Jupyter notebooks.
- Code instrumentation: add calls in your code to deploy sparkMeasure custom Spark listeners and/or use the classes StageMetrics/TaskMetrics and related APIs for collecting, analyzing and saving metrics data.
- "Flight Recorder" mode: this records all performance metrics automatically and saves data for later processing.

### Documentation and examples

- **Scala shell and notebooks**
- **PySpark and Jupyter notebooks**
- **Instrument Scala code**
- **Instrument Python code**
- **Flight Recorder mode**
- Notes on implementation and APIs
- Notes on metrics analysis
- TODO list and known issues

### Architecture diagram

About Me

Luca Canali
Geneva, Switzerland
@LucaCanaliDB

View my complete profile

Links

- Luca's talks, articles and blog posts list
- Blog of the database services at CERN
- Luca's Twitter
- Luca's GitHub

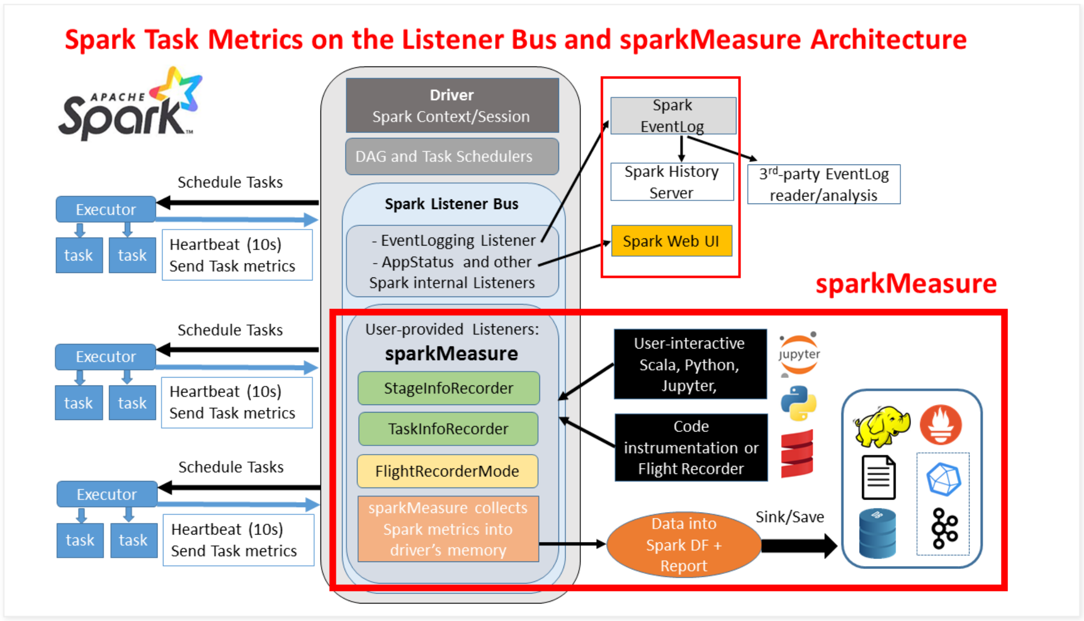
Blog Archive

- ▼ 2018 (1)
  - ▼ August 2018 (1)
    - SparkMeasure, a tool for performance troubleshooti...
- ► 2017 (4)
- ► 2016 (9)
- ► 2015 (10)
- ► 2014 (11)
- ► 2013 (10)
- ► 2012 (15)

Follow by Email

Email address... Submit

Member of the Oak Table Network


Spark Task Metrics on the Listener Bus and sparkMeasure Architecture
Driver
Spark Context/Session
DAG and Task Schedulers
Spark Listener Bus
- EventLogging Listener
- AppStatus and other Spark internal Listeners
User-provided Listeners:
sparkMeasure
StageInfoRecorder
TaskInfoRecorder
FlightRecorderMode
sparkMeasure collects Spark metrics into driver's memory
Schedule Tasks
Executor
task
Heartbeat (10s)
Send Task metrics
Spark EventLog
Spark History Server
3rd-party EventLog reader/analysis
Spark Web UI
sparkMeasure
User-interactive Scala, Python, Jupyter,
Code instrumentation or Flight Recorder
Data into Spark DF + Report
Sink/Save

## Main concepts underlying sparkMeasure

- The tool is based on the Spark Listener interface. Listeners transport Spark executor Task Metrics data from the executor to the driver. They are a standard part of Spark instrumentation, used by the Spark Web UI and History Server for example.
- Metrics can be collected using sparkMeasure at the granularity of stage completion and/or task completion (configurable)
- Metrics are flattened and collected into local memory structures in the driver (ListBuffer of a custom case class).
- Spark DataFrame and SQL are used to further process metrics data for example to generate reports.
- Metrics data and reports can be saved for offline analysis.

## Getting started examples of sparkMeasure usage

1. Link to an example Python_Jupyter Notebook
2. Example notebooks on the Databricks platform (community edition): example Scala notebook on Databricks, example Python notebook on Databricks
3. An example using Scala REPL/spark-shell:

```
bin/spark-shell --packages ch.cern.sparkmeasure:spark-measure_2.11:0.13

val stageMetrics = ch.cern.sparkmeasure.StageMetrics(spark)
stageMetrics.runAndMeasure(spark.sql("select count(*) from range(1000) cross join range(1000) cross join range(1000)").show())
```

The output should look like this:

```
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
```

## FAQ

- Why measuring performance with workload metrics instrumentation rather than just using time?
  - Measuring elapsed time, treats your workload as "a black box" and most often does not allow you to understand the root cause of the performance. With workload metrics you can (attempt to) go further in understanding and root cause analysis, bottleneck identification, resource usage measurement.
- What are Apache Spark tasks metrics and what can I use them for?
  - Apache Spark measures several details of each task execution, including run time, CPU time, information on garbage collection time, shuffle metrics and on task I/O. See also this short description of the Spark Task Metrics
- How is sparkMeasure different from Web UI/Spark History Server and EventLog?
  - sparkMeasure uses the same ListenerBus infrastructure used to collect data for the Web UI and Spark EventLog.
    - Spark collects metrics and other execution details and exposes them via the Web UI.
    - Notably Task execution metrics are also available through the REST API
    - In addition Spark writes all details of the task execution in the EventLog file (see config of `spark.eventlog.enabled` and `spark.eventLog.dir`)
    - The EventLog is used by the Spark History server + other tools and programs can read and parse the EventLog file(s) for workload analysis and performance troubleshooting, see a proof-of-concept example of reading the EventLog with Spark SQL
  - There are key differences that motivate this development:
    - sparkmeasure can collect data at the stage completion-level, which is more lightweight than measuring all the tasks, in case you only need to compute aggregated performance metrics. When needed, sparkMeasure can also collect data at the task granularity level.
    - sparkmeasure has an API that makes it simple to add instrumention/performance measurements in notebooks and application code.
    - sparkmeasure collects data in a flat structure, which makes it natural to use Spark SQL for workload data processing, which provides a simple and powerful interface
    - limitations: sparkMeasure does not collect all the data available in the EventLog, sparkMeasure buffers data in the driver memory, see also the TODO and issues doc
- What are known limitations and gotchas?
  - The currently available Spark task metrics can give you precious quantitative information on resources used by the executors, however there do not allow to fully perform time-based analysis of the workload performance, notably they do not expose the time spent doing I/O or network traffic.
  - Metrics are collected on the driver, which can be quickly become a bottleneck. This is true in general for ListenerBus instrumentation, in addition sparkMeasure in the current version buffers all data in the driver memory.
  - Task metrics values collected by sparkMeasure are only for successfully executed tasks. Note that resources used by failed tasks are not collected in the current version.
  - Task metrics are collected by Spark executors running on the JVM, resources utilized outside the JVM are currently not directly accounted for (notably the resources used when running Python code inside the python.daemon in the case of PySpark).
- When should I use stage metrics and when should I use task metrics?
  - Use stage metrics whenever possible as they are much more lightweight. Collect metrics at the task granularity if you need the extra information, for example if you want to study effects of skew, long tails and task stragglers.
- What are accumulables?
  - Metrics are first collected into accumulators that are sent from the executors to the driver. Many metrics of interest are exposed via [[TaskMetrics]] others are only available in StageInfo/TaskInfo accumulables (notably SQL Metrics, such as "scan time")
- How can I save/sink the collected metrics?
  - You can print metrics data and reports to standard output or save them to files (local or on HDFS). Additionally you can sink metrics to external systems (such as Prometheus, other sinks like InfluxDB or Kafka may be implemented in future versions).
- How can I process metrics data?
  - You can use Spark to read the saved metrics data and perform further post-processing and analysis. See the also Notes on metrics analysis.
- How can I contribute to sparkMeasure?
  - SparkMeasure has already profited from PR contributions. Additional contributions are welcome. See the TODO_and_issues list for a list of known issues and ideas on what you can contribute.

## Acknowledgements and additional links

This work has been developed in the context of the CERN Hadoop, Spark and Streaming services and of the CERN openlab project on data analytics. Credits go to my colleagues for collaboration. Many thanks also to the Apache Spark community, in particular to the contributors of PRs and reporters of issues. Additional links on this topic are:

- Link to 2017 blog post on sparkMeasure

- Presentation at Spark Summit Europe 2017: "Apache Spark Performance Troubleshooting at Scale, Challenges, Tools, and Methodologies"

Posted by Luca Canali at 8:51 AM

Labels: performance, Spark, tools

## No comments:

## Post a Comment

Enter your comment...

Comment as: dimoula.ioanr ▼ Sign out

Publish Preview ☐ Notify me

## Links to this post

Create a Link

Home Older Post

Subscribe to: Post Comments (Atom)