Building an Apache Spark Performance Lab: Tools and Techniques for Optimization

Luca Canali
CERN
April 2024

About Luca

- Data Engineer at CERN
  - Data analytics and Spark service, database services
  - 20+ years with databases and data engineering
  - Passionate about performance engineering

- Repos, blogs, presentations

@LucaCanaliDB
https://github.com/lucacanali
http://cern.ch/canali

CERN

Motivations and Scope

- Apache Spark is great at large-scale data processing
  - Distributed computing is hard
  - Getting optimal execution plans is hard
- Data-driven troubleshooting and tuning of Spark jobs
  - Beyond just measuring execution time
  - Collect and analyze Spark Metrics, Spark-Dashboard
  - Workload generator instrumented with sparkMeasure: TPCDS-PySpark
- Not a goal
  - A guide to Spark performance troubleshooting and tuning

Build a Lab!

- The key idea of this presentation:
  - Build a Spark Performance Lab
  - Run Spark jobs at scale
    - Start small (GB) and scale to TBs
  - Use instrumentation
    - Monitor the workload execution
    - Understand using data: use Spark metrics
    - Learn by running experiments: change configuration, scale, SW version, HW, etc.

Tools

1. Workload Generation

- Start by deploying TPCDS_PySpark.
  - It's a workload generator that runs TPC-DS queries on Spark, allowing for performance studies across different configurations and Spark versions.
  - Execute TPC-DS queries, a well-known suite of complex SQL, representative of many OLAP environments
  - Run at scale, from local mode and a few GB to cluster and 10s of TBs
  - Collect and analyze Spark performance metrics thanks to the integration with sparkMeasure (see discussion on sparkMeasure).

TPCDS_PySpark – getting started

- Run this getting-started example from the command line:

```
# Install the tool and dependencies
pip install pyspark
pip install sparkmeasure
pip install tpcds_pyspark

# Download the test data
wget
https://sparkdltrigger.web.cern.ch/sparkdltrigger/TPCDS/tpcds_10.zip
unzip -q tpcds_10.zip

# 1. Run the tool for a minimal test
tpcds_pyspark_run.py -d tpcds_10 -n 1 -r 1 --queries q1,q2

# 2. run all queries with default options
tpcds_pyspark_run.py -d tpcds_10
```

2. Explore: WebUI

- Spark WebUI is the official Spark instrumentation a starting point for performance investigations. Use it to explore the SQL, jobs, and configurations while the workload is running.
- https://spark.apache.org/docs/latest/web-ui.html

3. Detailed Analysis with sparkMeasure

- For a more detailed analysis, integrate SparkMeasure into your applications
- This tool is invaluable for identifying performance bottlenecks, understanding resource utilization, and comparing different Spark configurations or code changes.
- Modes of operation
  - Interactive use, analyze the metrics as you go. Use it with Notebooks or CLI.
  - Batch mode for post-execution analysis. Use it to instrument your code and/or use it for CI/CD jobs.
  - Flight-recorder mode: collect metrics without any code change.

SparkMeasure – getting started

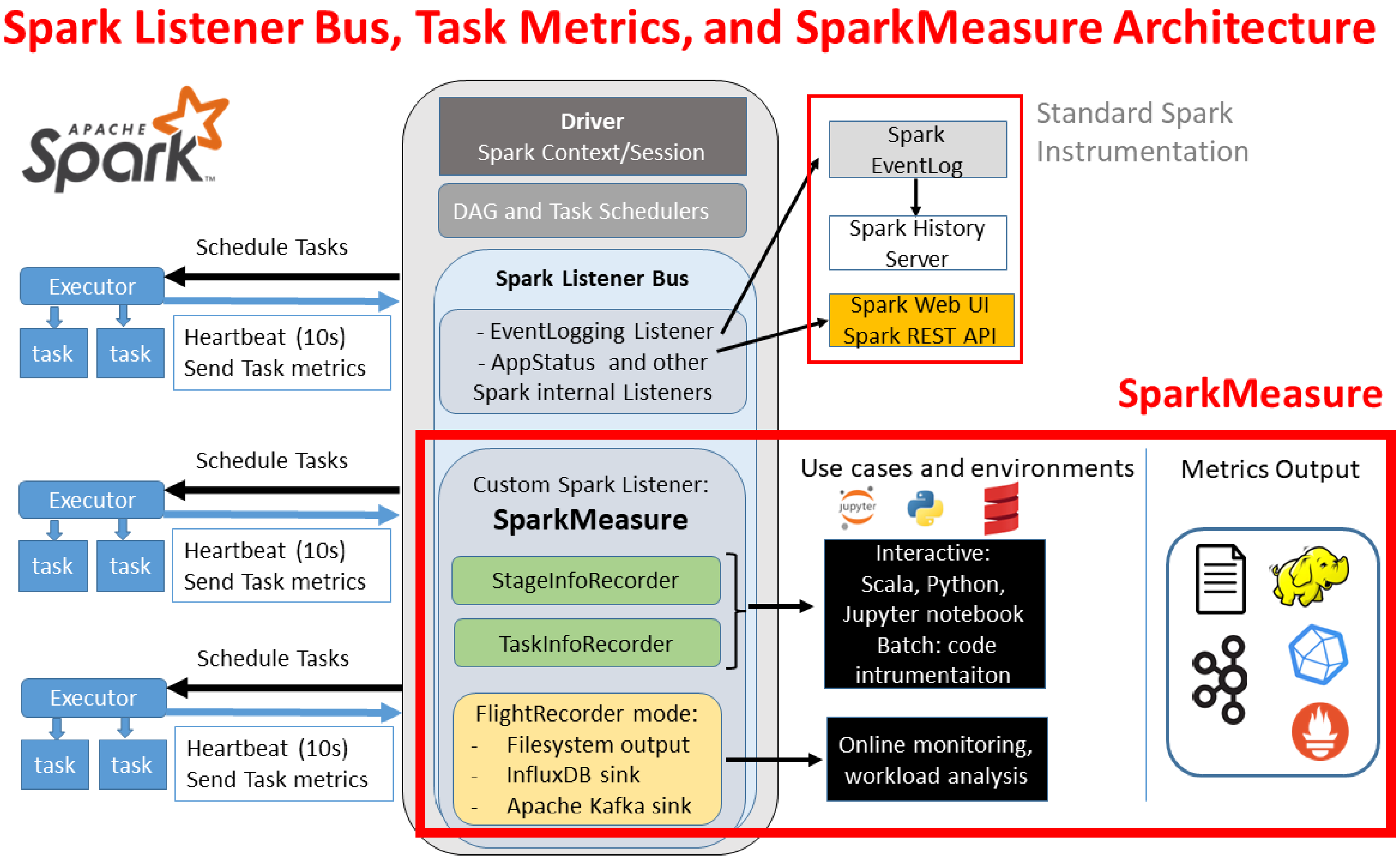
- Run these getting started examples from the command line:

```
# Scala CLI
spark-shell --packages ch.cern.sparkmeasure:spark-measure_2.12:0.24

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())
-----------------
# Python CLI
# pip install pyspark
pip install sparkmeasure
pyspark --packages ch.cern.sparkmeasure:spark-measure_2.12:0.24

from sparkmeasure import StageMetrics
stagemetrics = StageMetrics(spark)
stagemetrics.runandmeasure(globals(), 'spark.sql("select count(*) from
range(1000) cross join range(1000) cross join range(1000)").show()')
```

SparkMeasures' architecture



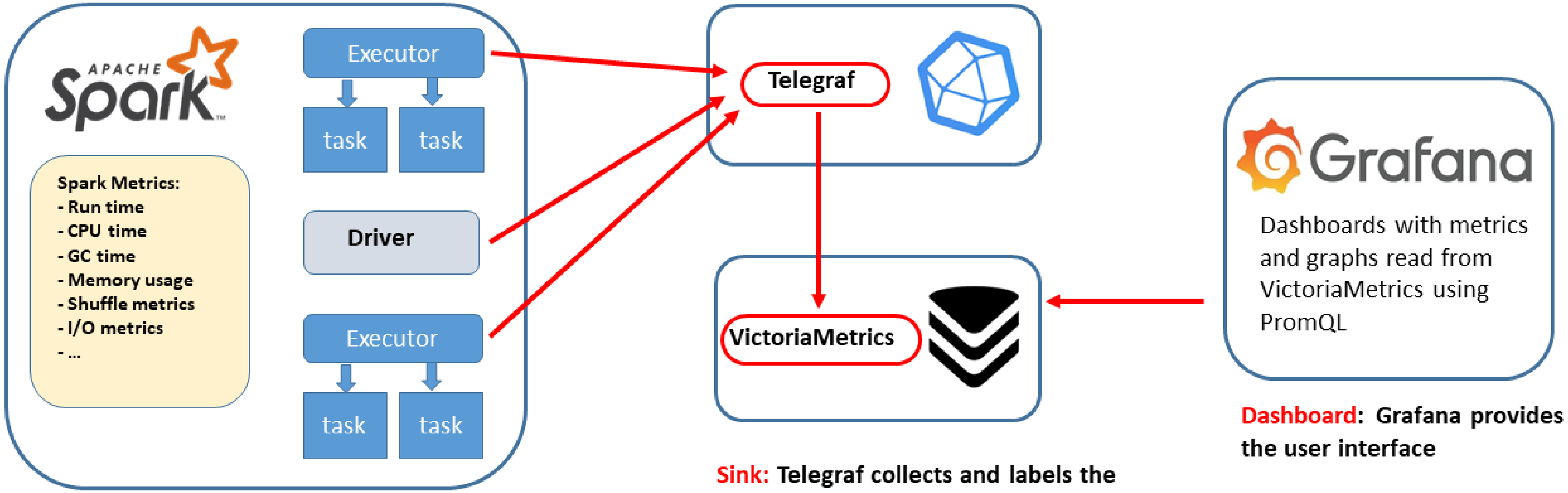
4. Monitor execution with Spark-Dashboard

- Use Spark-Dashboard to monitor the Spark jobs execution in real time.
  - This involves collecting metrics from your Spark jobs and visualizing them with a Grafana dashboard.
- The dashboard displays metric related to CPU usage, I/O, Shuffle, Memory usage.
  - Time-series to follow the evolution and find bottlenecks
- The setup process is straightforward, thanks to pre-configured Docker container images.

Spark-Dashboard's architecture

Spark-Dashboard, a Monitoring Pipeline

Spark Metrics System + Telegraf + VictoriaMetrics + Grafana = Monitoring



Dashboard: Grafana provides the user interface

Sink: Telegraf collects and labels the measurements

Time series DB: VictoriaMetrics stores the data

Source: Spark executors and driver emit metrics measurements directly to a sink

Spark-Dashboard – getting started

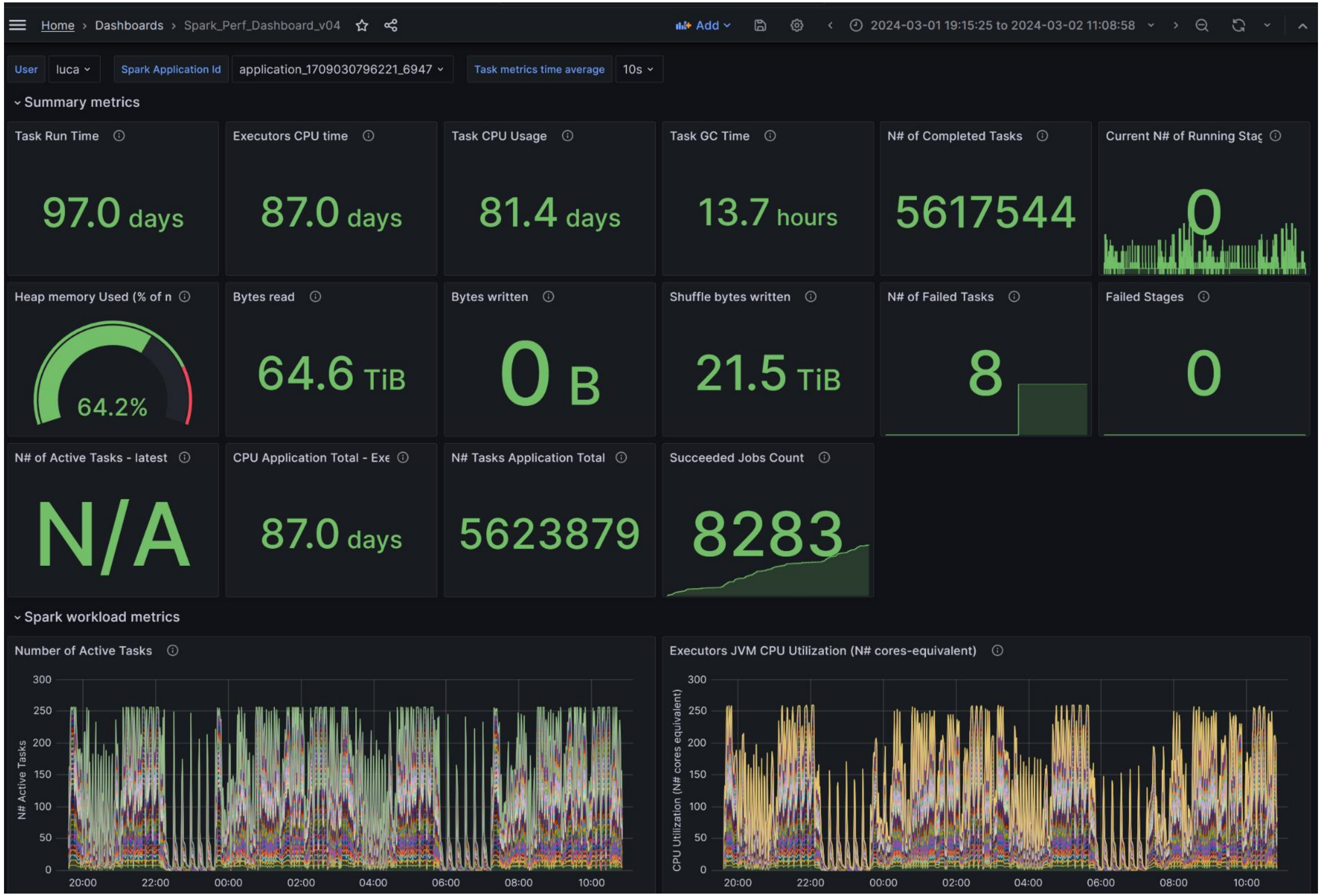
```
# 1. Start the container image
docker run -p 3000:3000 -p 2003:2003 -d lucacanali/spark-dashboard

# 2. Run Spark
bin/spark-shell (or spark-submit or pyspark)
--conf "spark.metrics.conf.*.sink.graphite.class"="org.apache.spark.metrics.sink.GraphiteSink" \
--conf "spark.metrics.conf.*.sink.graphite.host"="localhost" \
--conf "spark.metrics.conf.*.sink.graphite.port"=2003 \
--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" \
--conf "spark.metrics.staticSources.enabled"=true \
--conf "spark.metrics.appStatusSource.enabled"=true

# 3. Go to the dashboard: http://localhost:3000
```

Example Dashboard

Partial view of the dashboard



Tutorials and Demos

Demos and tutorials of the tools for a Spark Performance Lab.

- SparkMeasure - metrics collection
  - Watch the video sparkMeasure's getting started demo and tutorial

- TPCDS_PySpark - workload generator
  - Watch the video Watch TPCDS-PySpark demo and tutorial

- Spark-Dashboard - real-time dashboards
  - Watch the video Spark-Dashboard demo and tutorial

Metrics Drill Down

- Metrics provide insights
  - They take us beyond simple timing, revealing details about task execution, resource utilization, and bottlenecks.
- Execution Time is Not Enough
  - Measuring the execution time of a job is useful but it doesn’t show the whole picture.
  - Say the job ran in 10 seconds. It's crucial to understand why it took 10 seconds instead of 100 seconds or just 1 second. What was slowing things down? Was it the CPU, data input/output, or something else, like data shuffling?
  - This helps us identify the root causes of performance issues.

Make the Best of Spark Metrics

Documentation: Spark Task Metrics docs

Key Metrics to Collect and Monitor:

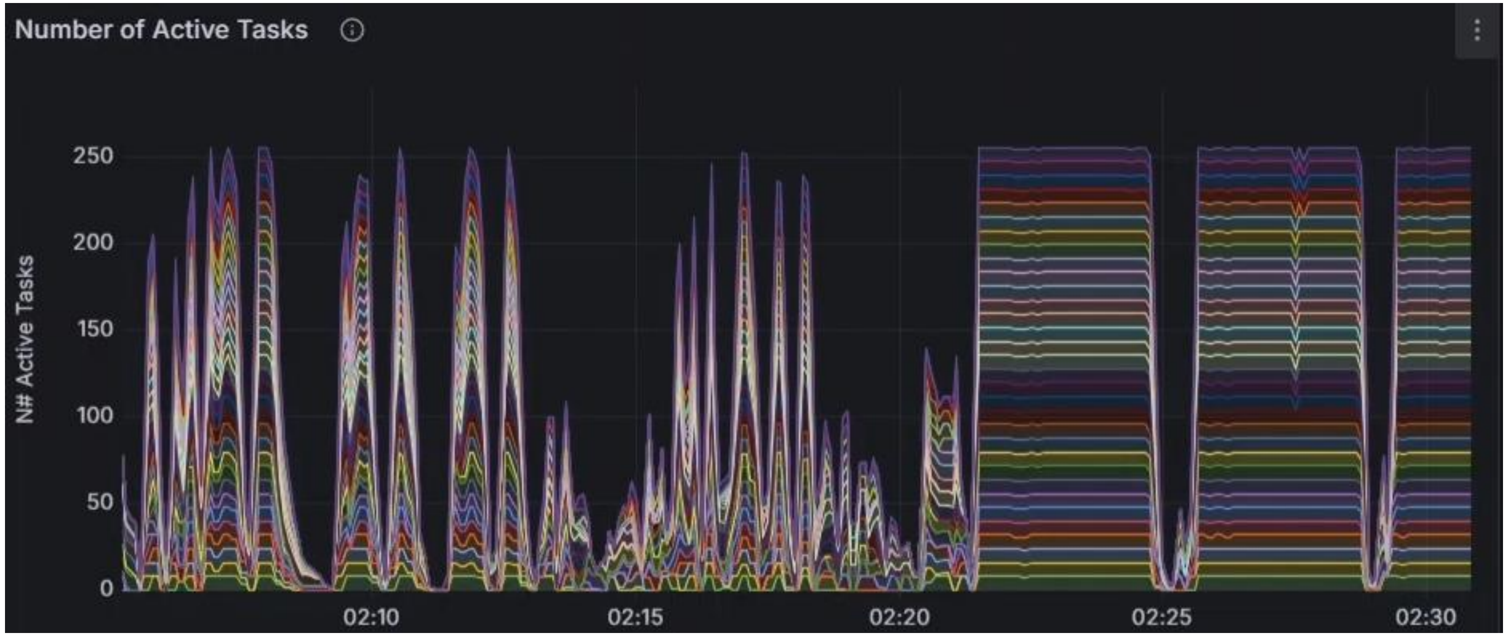
- **Executor Run Time:** Total time executors spend processing tasks.
- **Executor CPU Time:** Direct CPU time consumed by tasks.
- **JVM GC Time:** Time spent in garbage collection, affecting performance.
- **Shuffle and I/O Metrics:** Critical for understanding data movement and disk interactions.
- **Memory Metrics:** Key for performance and troubleshooting Out Of Memory errors

Metric Analysis, What to Look For

- Look for bottlenecks:
  - Are there resources that are the bottleneck? Are the jobs running mostly on CPU or waiting for I/O or Garbage Collection, or..?
- USE method:
  - Utilization Saturation and Errors (USE) Method
  - It is a methodology for analyzing the performance of any system.
  - The tools described here can help you to measure and understand Utilization and Saturation.

Cluster CPU Utilization

- Are you getting all allocated cores to work for you?
  - Check the number of active tasks vs. time
  - Figure: during TPCDS 10TB on a YARN cluster with 256 cores
  - Spikes and troughs. Drill down on root cause:
    - Resource allocation, partition skew, straggler tasks, stage boundaries, etc



Which Tools Should I Use?

- Start with using the Spark Web UI
- Instrument your jobs with sparkMesure.
  - This is recommended early in the application development, testing, and for Continuous Integration (CI) pipelines.
- Observe your Spark application execution with Spark-Dashboard
- Use OS-tools
  - See also Spark-Dashboard extended instrumentation: it collects and visualizes OS metrics (from cgroup statistics) like network stats, etc
- An example of “offline” Spark metrics analysis
  - TPCDS run at scale 10 TB

Lessons Learned

- **Collect, Analyze and Visualize Metrics:** Go beyond just measuring jobs' executions time, to troubleshoot and fine-tune Spark performance effectively.

- **Use the Right Tools:** Familiarize yourself with tools for performance measurement and monitoring.

- **Start Small, Scale Up:** Begin with smaller datasets and configurations, then gradually scale to test larger, more complex scenarios.

- **Tuning is an Iterative Process:** Experiment with different configurations, parallelism levels, and data partitioning strategies to find the best setup for your workload.

Conclusions

- Establishing a Spark Performance Lab is a fundamental step for any developer and data engineer aiming to master Spark's performance.
- By integrating tools like Web UI, TPCDS_PySpark, sparkMeasure, and Spark-Dashboard, developers and data engineers can gain unprecedented insights into Spark operations and optimizations.
- Learn by doing and experimentation!

Resources

- Blog: Building an Apache Spark Performance Lab: Tools and Techniques for Spark Optimization
- TPCDS_PySpark
- SparkMeasure
- Spark-Dashboard and Dashboard Notes
- Flame Graphs for Spark and Grafana Pyroscope with Spark
- Tools for OS performance monitoring

Acknowledgements: the teams behind the CERN data analytics, monitoring, and web notebook services, as well as the members of the ATLAS database group.