
# SparkOscope: Enabling Apache Spark Optimization Through Cross-Stack Monitoring and Visualization

Yiannis Gkoufas

IBM Research Dublin,Ireland

High Performance Systems

# whoami

- Research Software Engineer in **IBM Research, Ireland** since 2012
- Work on **Analytics Foundations Middleware**
  - Distributed Frameworks, Anything Java/Scala based, Web-based POCs
- **High Performance Systems Group**: Kostas, Andrea, Dimitris, Khalid, Michael, Michele, Mustafa, Pierre, Sri

# Spark Experience

- We love developing in Spark our analytical workloads and **fully embraced** it since the early 1.0.x versions
- Last few years, used it to run jobs on large volume of **energy-related sensor data**

# Jobs on Daily Basis

- Once we managed to develop the needed jobs, they were executed in a **recurring fashion**
- We were receiving a new batch of **data every day**

# Fighting Bugs

- When there was a bug on our code, it was very easy to discover it the **Spark Web UI**
- We could easily retrieve information about the **job, stage and line number** in our source code

# Fighting bottlenecks

- However we couldn't easily spot which **jobs and stages** were causing a **slow down**
- **What was the part of our code that was the bottleneck?**

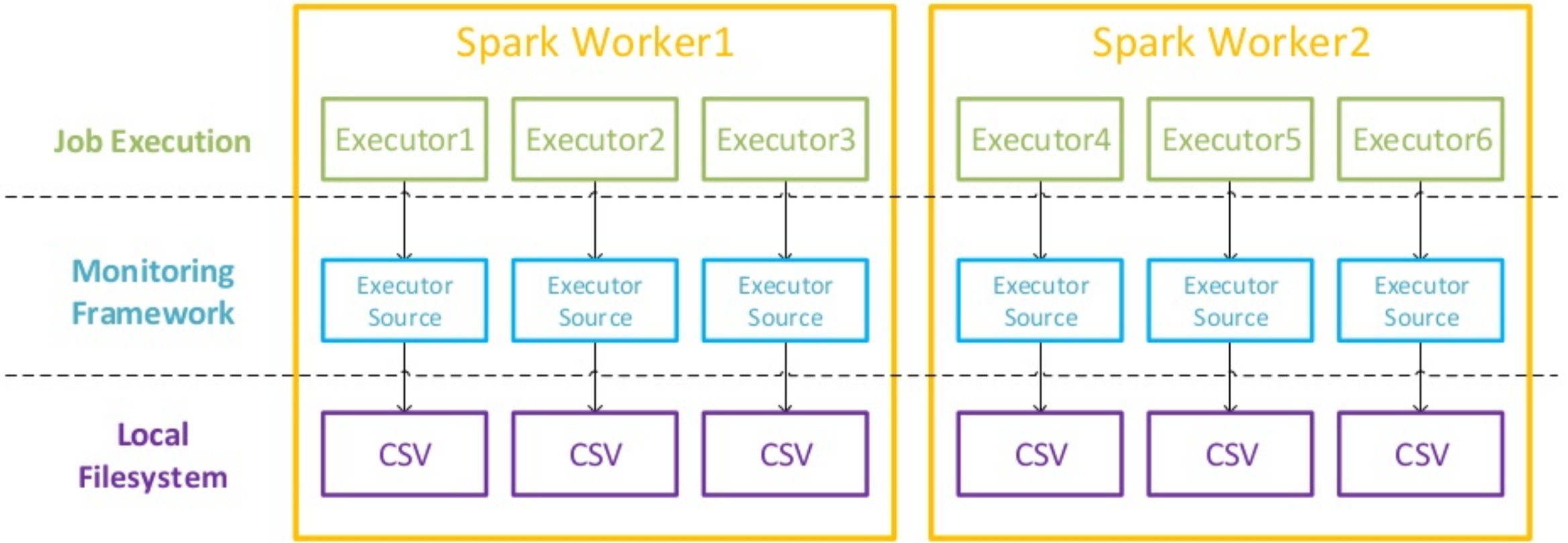
# Ganglia Extension

- We had the option to use the **Ganglia Extension** to export the metrics but:
  - We need to maintain/configure **yet another external system**
  - There is **no association** with the Spark jobs/stages/source code

# Spark Monitoring Framework

- We could use the built-in **Spark Monitoring Framework** but:
  - **Collecting CSVs** from the worker nodes and **aggregating** them seems **cumbersome**
  - Again we couldn't easily **extract associations with our source code** of the job

# Current Monitoring Architecture

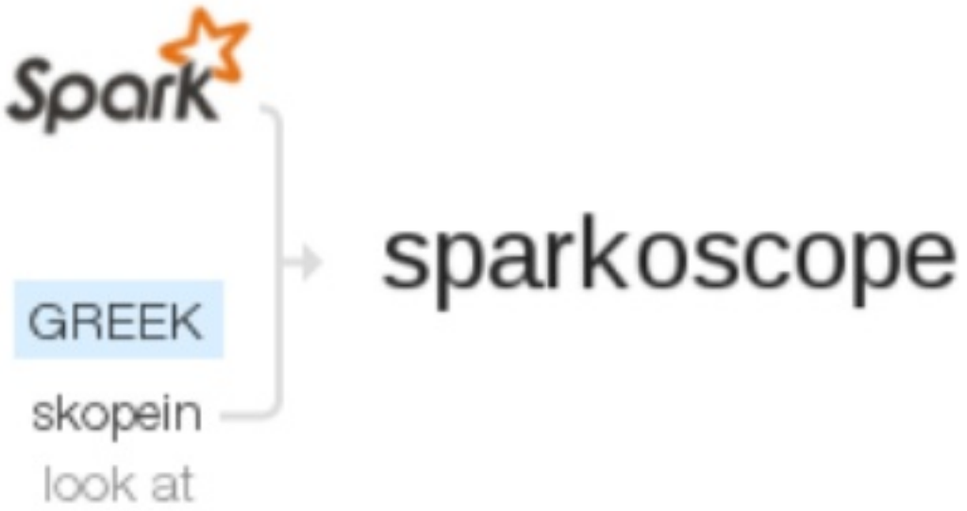


# Enter SparkOscope

sparkoscope

/spɑːkəskəʊp/

Origin



# SparkOscope Overview

- Extension to enrich Spark's Monitoring Framework with **OS-level Metrics**
- Enhancement of the Web UI to **plot all the available metrics** + the newly developed OS-level metrics

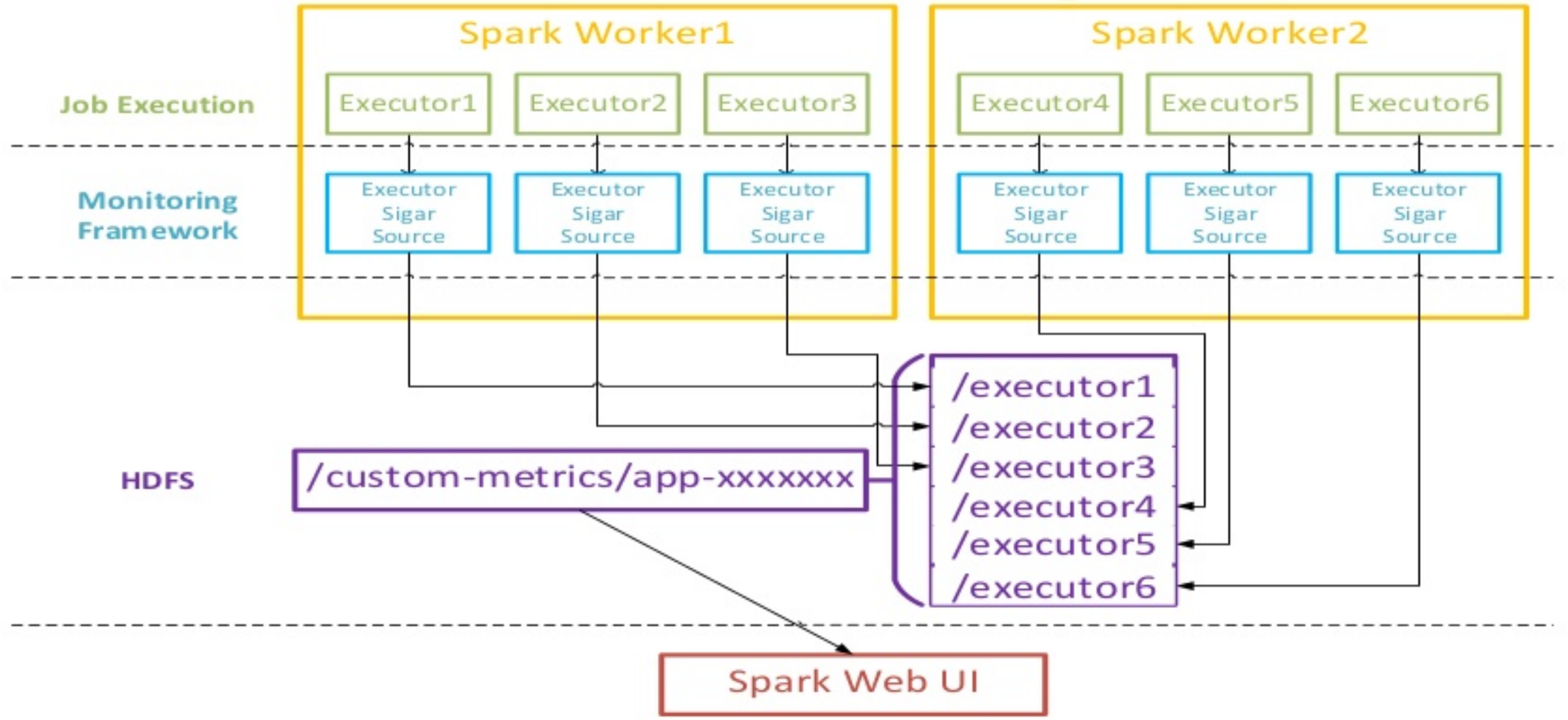
# SparkOscope Modules

- **SigarSource:** Attached to the executor, leveraging Hyperic Sigar library to get OS-Level Metrics
- **HDFSSink:** Exports all available metrics to an HDFS directory
- **MQTTSink:** Publishes all available metrics on an MQTT Topic
- **Modified Web UI:** Modified Spark Web UI to plot historical and realtime plots, generated from the modules

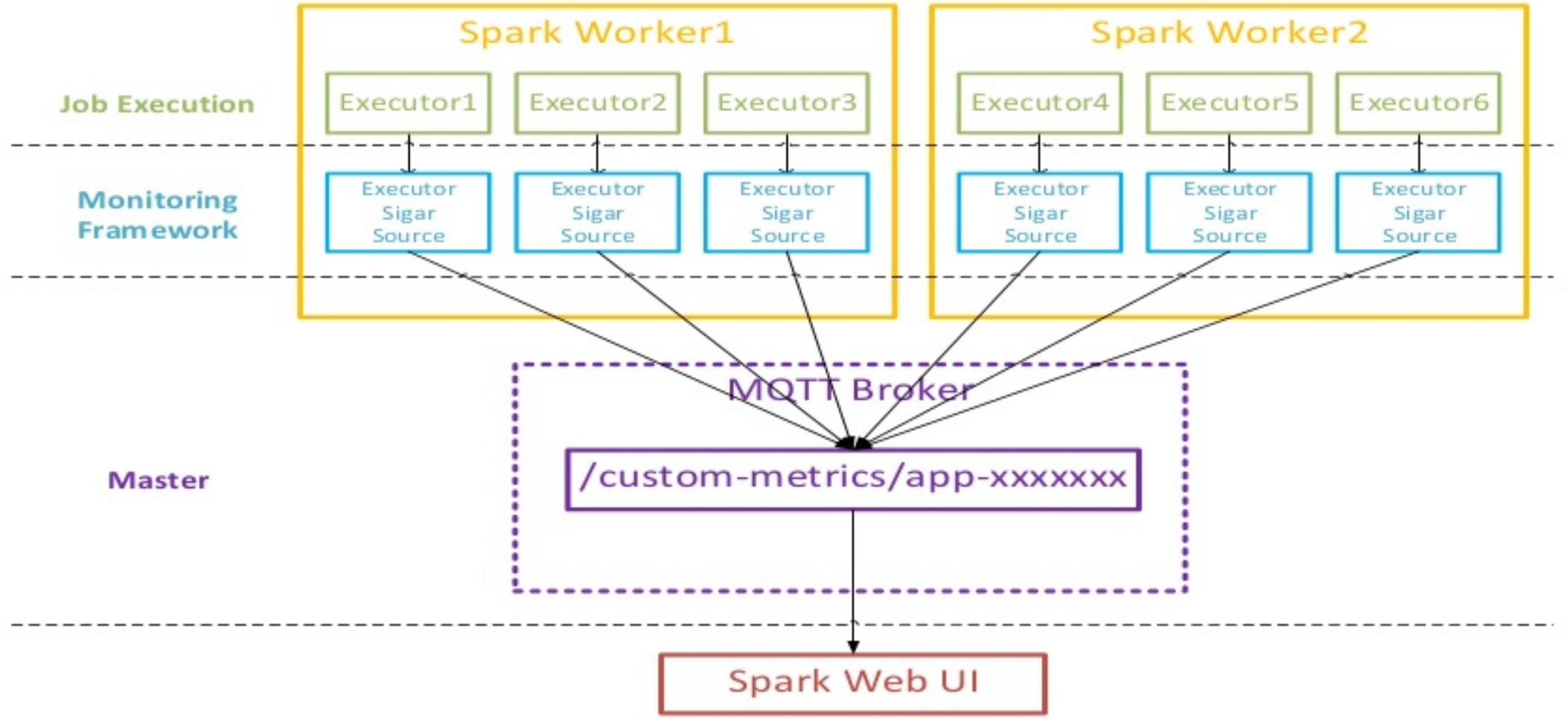
# SparkOscope Flavors

- **Historical Plots:** View metrics on the UI after the job has finished
- **Realtime Plots:** View metrics on the UI in realtime as the job is being executed
- **Headless:** Use SigarSource, HDFSSink, MQTTSink without viewing the plots on the UI
  - https://github.com/ibm-research-ireland/sparkoscope-headless

# SparkOscope High-level Architecture - Historical plots



# SparkOscope High-level Architecture - Realtime plots



# SparkOscope Basic Installation

- Clone the git repo: https://github.com/ibm-research-ireland/sparkoscope
- Build Spark
- Modify the configuration files:

*metrics.properties*

```
executor.sink.hdfs.class=org.apache.spark.metrics.sink.HDFSSink

executor.sink.hdfs.pollPeriod = 20

executor.sink.hdfs.dir = hdfs://localhost:9000/custom-metrics

executor.sink.hdfs.unit = seconds
```

*spark-defaults.conf*

```
spark.hdfs.metrics.dir          hdfs://127.0.0.1:9000/custom-metrics

spark.eventLog.enabled          true

spark.eventLog.dir              hdfs://127.0.0.1:9000/spark-logs
```

# SparkOscope OS-level Metrics

- Download the Hyperic Sigar library to all the slave nodes
- Extract it anywhere in the system
- Modify the configuration files

*metrics.properties*

```
executor.source.jvm.class=org.apache.spark.metrics.source.SigarSource
```

*spark-env.sh*

```
HADOOP_CONF_DIR=/path/to/hadoop/etc/hadoop
```

```
LD_LIBRARY_PATH=/path/to/hyperic-sigar-1.6.4/sigar-bin/lib/:$LD_LIBRARY_PATH
```

# SparkOscope Realtime Plots

- Modify the configuration files

*metrics.properties*

```
executor.sink.mqtt.class=org.apache.spark.metrics.sink.MQTTSink
executor.sink.mqtt.pollPeriod = 1
executor.sink.mqtt.host = masterIP
executor.sink.mqtt.port = 1883
executor.sink.mqtt.unit = seconds
```

*spark-defaults.conf*

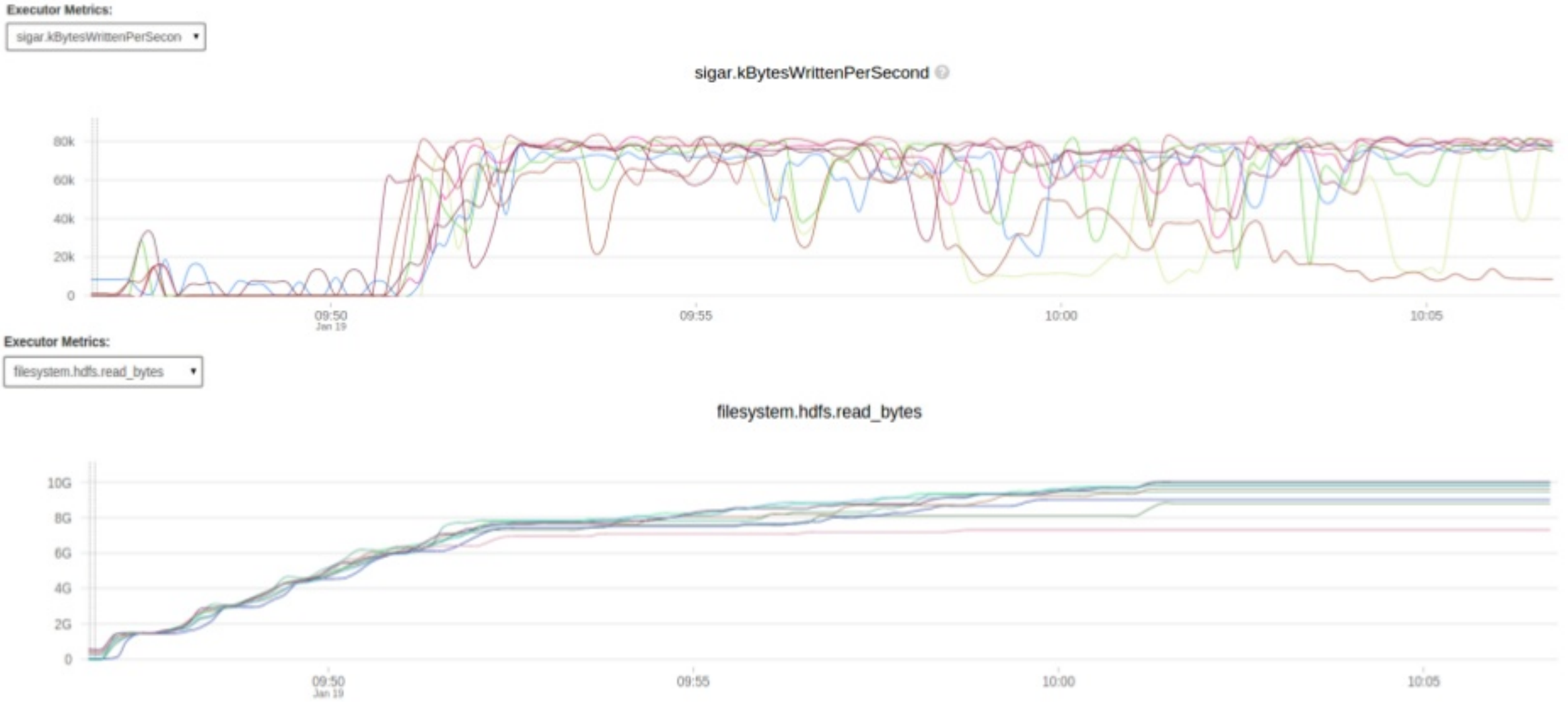
```
spark.moquette.port 1883
spark.moquette.websocket_port 8888
```

- Make sure that no service is currently running on ports specified on the Master
- Make sure that *executor.sink.mqtt.port* is the same as *spark.moquette.conf*

# SparkOscope Headless Installation

- Clone the git repo: https://github.com/ibm-research-ireland/sparkoscope-headless
- Build the maven project
- Modify the configuration files as described for SigarSource, HDFSSink, MQTTSink
- Additionally you need to append to *spark.executor.extraClassPath* the paths of the created jars
- **No need to have the patched Spark version, since the metrics are not displayed in the UI**

# Demo!


Executor Metrics:
sigar.kBytesWrittenPerSecon
sigar.kBytesWrittenPerSecond
80k
60k
40k
20k
0
09:50
Jan 19
09:55
10:00
10:05
Executor Metrics:
filesystem.hdfs.read_bytes
filesystem.hdfs.read_bytes
10G
8G
6G
4G
2G
0
09:50
Jan 19
09:55
10:00
10:05

# Roadmap

- Expand the range of available Sinks and Sources
- Smart recommendations on infrastructure needs derived from patterns of resource utilization of jobs
- Work with the opensource ecosystem to improve it and target more use cases


# Thank You.

Questions?

email: yiannisg@ie.ibm.com