

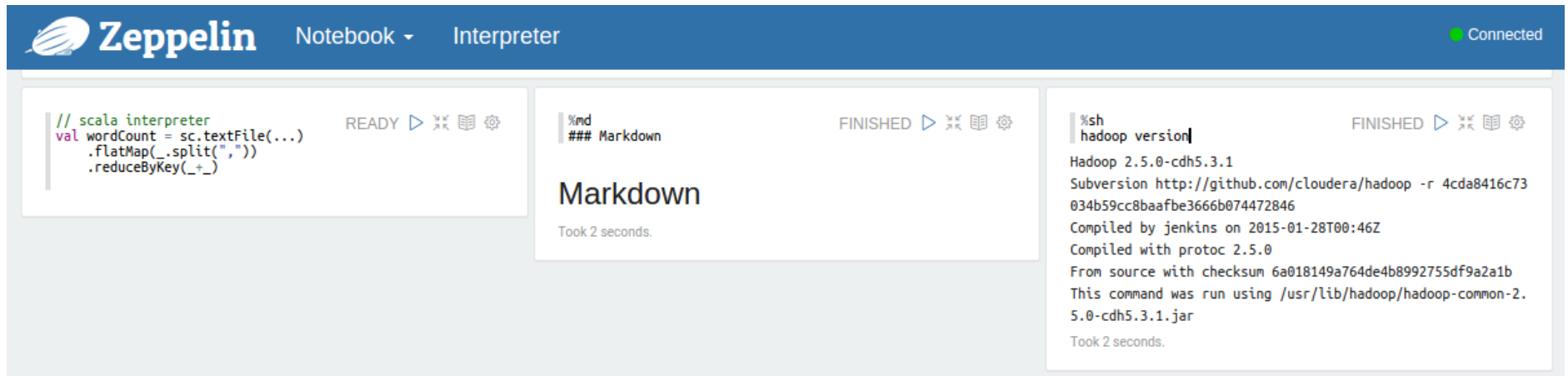
Introducing Apache Zeppelin

AWG – 30/09/2015

Luca Menichetti

Example:
LSF failed jobs inspection
(screenshots)

Multi-interpreter tool



The screenshot shows the Zeppelin Notebook interface with a blue header bar containing the Zeppelin logo, "Notebook", and "Interpreter" tabs, and a "Connected" status indicator. Below the header are three execution boxes:

- Scala Interpreter:** Status "READY". Code:

```
// scala interpreter
val wordCount = sc.textFile(...)
  .flatMap(_.split(" "))
  .reduceByKey(_+_)
```
- Markdown:** Status "FINISHED". Content:

```
### Markdown
```

 Below the code, it says "Markdown" and "Took 2 seconds."
- Bash Shell:** Status "FINISHED". Command:

```
%sh
hadoop version|
```

 Output:

```
Hadoop 2.5.0-cdh5.3.1
Subversion http://github.com/cloudera/hadoop -r 4cda8416c73
034b59cc8baafbe3666b074472846
Compiled by jenkins on 2015-01-28T00:46Z
Compiled with protoc 2.5.0
From source with checksum 6a018149a764de4b8992755df9a2a1b
This command was run using /usr/lib/hadoop/hadoop-common-2.
5.0-cdh5.3.1.jar
```

 Below the output, it says "Took 2 seconds."

Each box can be executed with a different interpreter

- Spark (Scala)
- Pyspark (Python)
- SparkSQL (Spark component)
- Textual interpreters: markdown, html, table, ...
- Dynamic forms
- Bash Shell, Postgres, ... (and [many others](#))



The screenshot shows the Zeppelin Notebook interface with a blue header bar containing the Zeppelin logo, "Notebook", and "Interpreter" tabs. Below the header is an execution box with the following content:

```
print("<html><h1>Hi Zeppelin</h1>")
<img src="https://upload.wikimedia.org/wikipedia/commons/f/f4/USS_Los_Angeles_moorred_to_USS_Patoka,_1931.jpg"
```

Hi Zeppelin



Zeppelin notebook

(interacting dynamically with Hadoop)

The screenshot shows a Zeppelin notebook interface with a blue header bar containing the Zeppelin logo, 'Notebook', 'Interpreter', and 'Connected' status. The main content area is titled 'Batchmon monthly errors study example. Queues view.' and contains several sections:

- Instructions:** A text box with two numbered steps and their sub-points. The first step involves choosing a month and running SQL queries. The second step involves selecting a queue to analyze. The execution status is 'FINISHED'.
- Imports And Function Definitions:** A section for defining variables and functions. The execution status is 'FINISHED'.
- Select The Period To Analyze:** A section with a code block and a dropdown menu. The dropdown is set to '2015 September'. The code defines a path and reads a Parquet file. The execution status is 'FINISHED'.
- Total Number Of Jobs:** A section with a bar chart and a text box showing the total number of jobs: 12,926,974. The execution status is 'FINISHED'.
- Plots:** A section at the bottom for visualizing data. The execution status is 'FINISHED'.

A blue callout box with the text 'Reads directly on HDFS' has an arrow pointing to the `batchmonDataFrame = sqlc.read.parquet(choosenPath)` line in the code block of the 'Select The Period To Analyze' section.

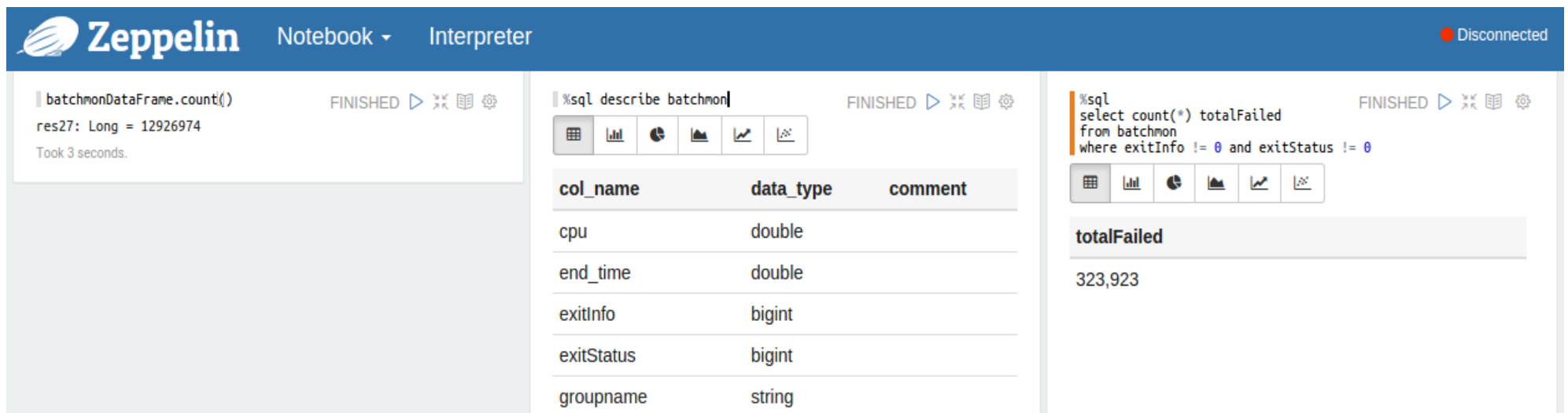
```
val choosenPath = z.select("Select a month", directoriesList).toString()
val batchmonDataFrame = sqlc.read.parquet(choosenPath)
batchmonDataFrame.registerTempTable(tableName)
//z.runAll()
```

chosenPath: String = hdfs://p01001532965510.cern.ch:9000/project/awg/batchmon/jobs/parquet/2015/09
batchmonDataFrame: org.apache.spark.sql.DataFrame = [cpu: double, end_time: double, exitInfo: bigint, exitStatus: bigint, groupname: string, hostfactor: double, job_id: bigint, memMB: bigint, node: string, queue: string, start_time: double, swapMB: bigint, user: string, wall: double, start_date: timestamp, end_date: timestamp, calculated_wall: double]

total
12,926,974

Studying the dataset

- Batchmon data are imported with Flume and daily converted in Parquet
- With SparkSQL reading Parquet files, it is straightforward to load a dataset and to start querying it
- Zeppelin helps further, providing fast/easy visualization, without extracting/moving the data (and without compiling&submitting a jar if you are writing in Scala)



The screenshot shows the Zeppelin Notebook interface with a blue header bar containing the Zeppelin logo, "Notebook", "Interpreter", and a "Disconnected" status indicator. The main workspace is divided into three vertical panels, each representing a code execution block.

Block 1 (Left): Contains the Scala code `batchmonDataFrame.count()`. The output shows `res27: Long = 12926974` and a message "Took 3 seconds.".

Block 2 (Middle): Contains the SQL code `%sql describe batchmon`. The output is a table with the following structure:

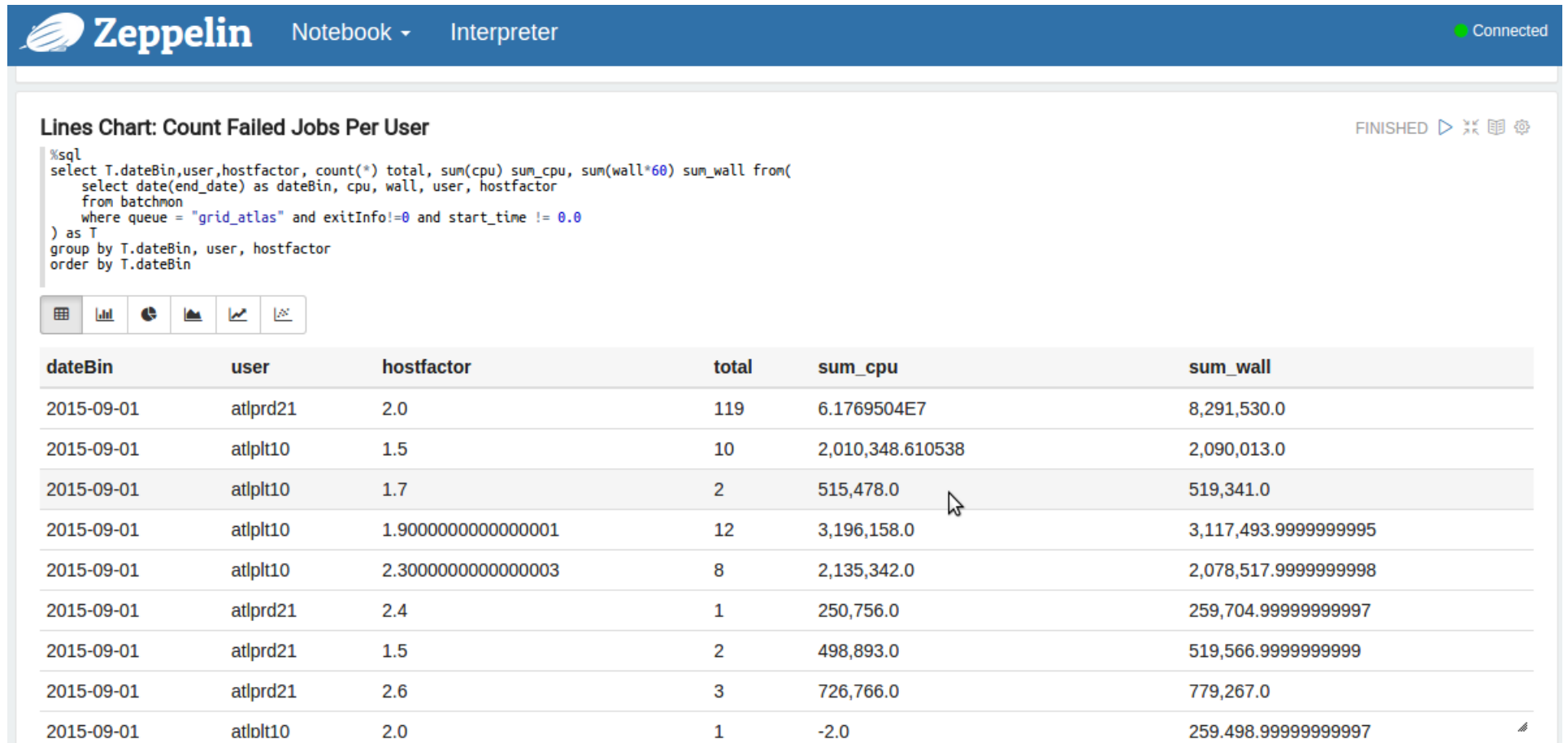
col_name	data_type	comment
cpu	double	
end_time	double	
exitInfo	bigint	
exitStatus	bigint	
groupname	string	

Block 3 (Right): Contains the SQL code `%sql select count(*) totalFailed from batchmon where exitInfo != 0 and exitStatus != 0`. The output shows a table with one row:

totalFailed
323,923

SparkSQL queries (1)

tabular result



SparkSQL queries (2)

result visualization



Notebook ▾ Interpreter

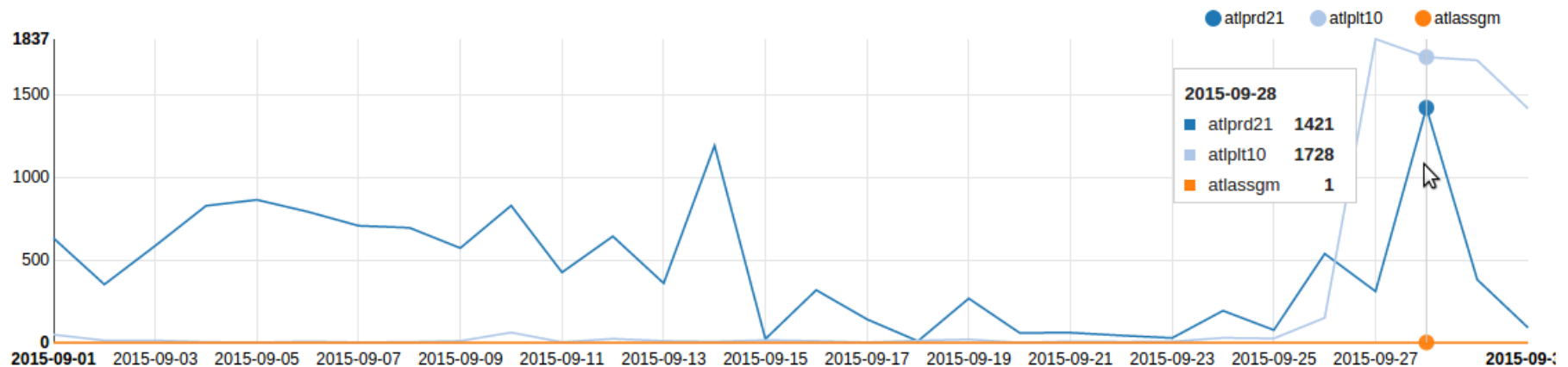
Connected

Lines Chart: Count Failed Jobs Per User

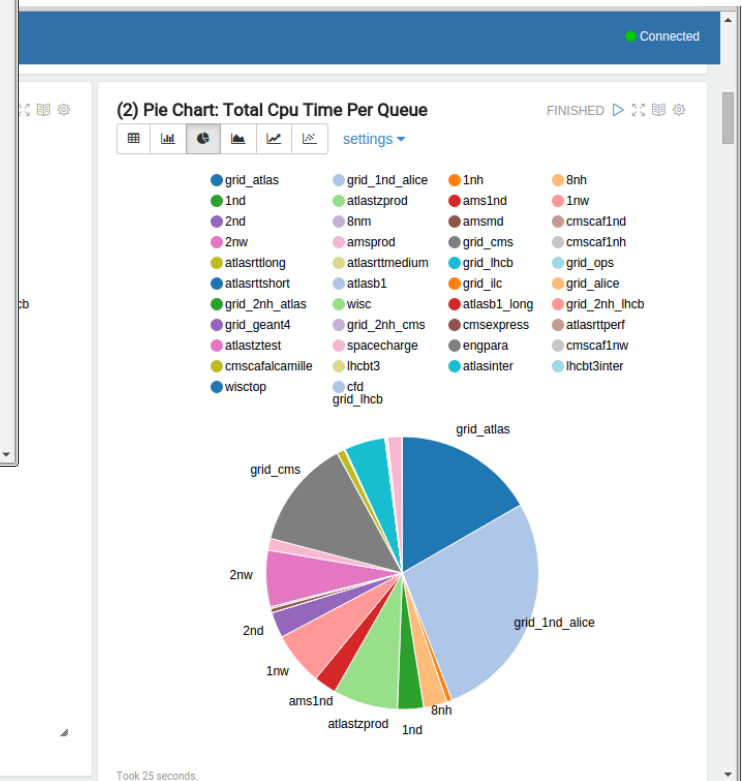
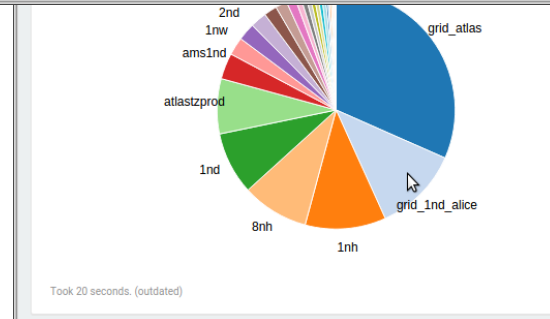
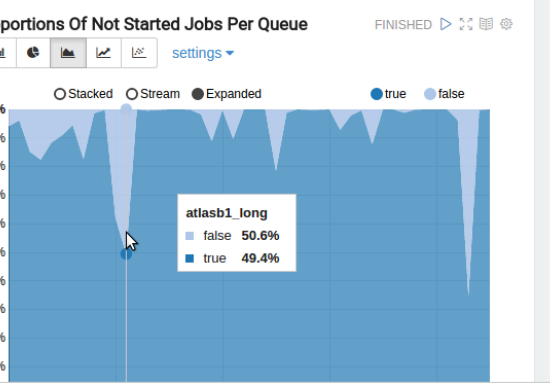
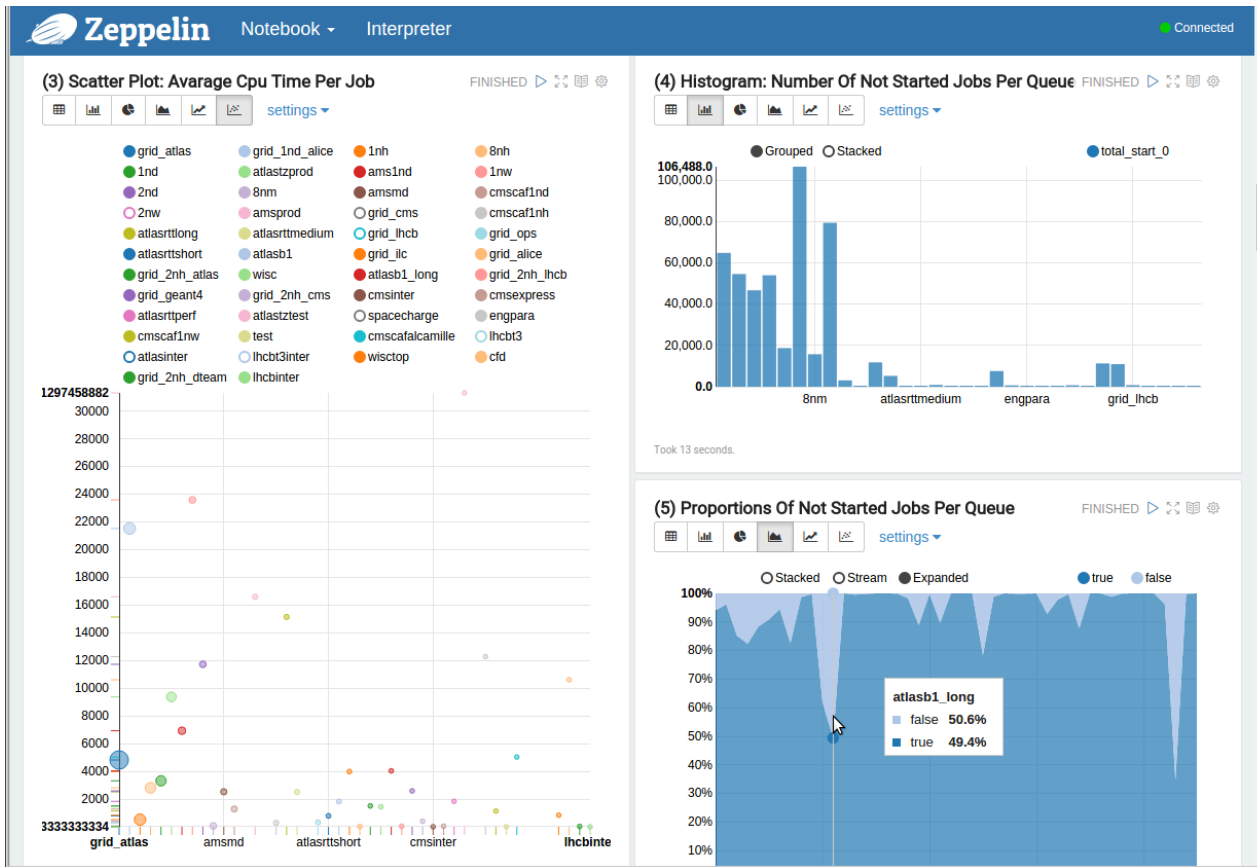
FINISHED ▶ ✕ 📖 ⚙️

```
%sql
select T.dateBin,user,hostfactor, count(*) total, sum(cpu) sum_cpu, sum(wall*60) sum_wall from(
  select date(end_date) as dateBin, cpu, wall, user, hostfactor
  from batchmon
  where queue = "grid_atlas" and exitInfo!=0 and start_time != 0.0
) as T
group by T.dateBin, user, hostfactor
order by T.dateBin
```

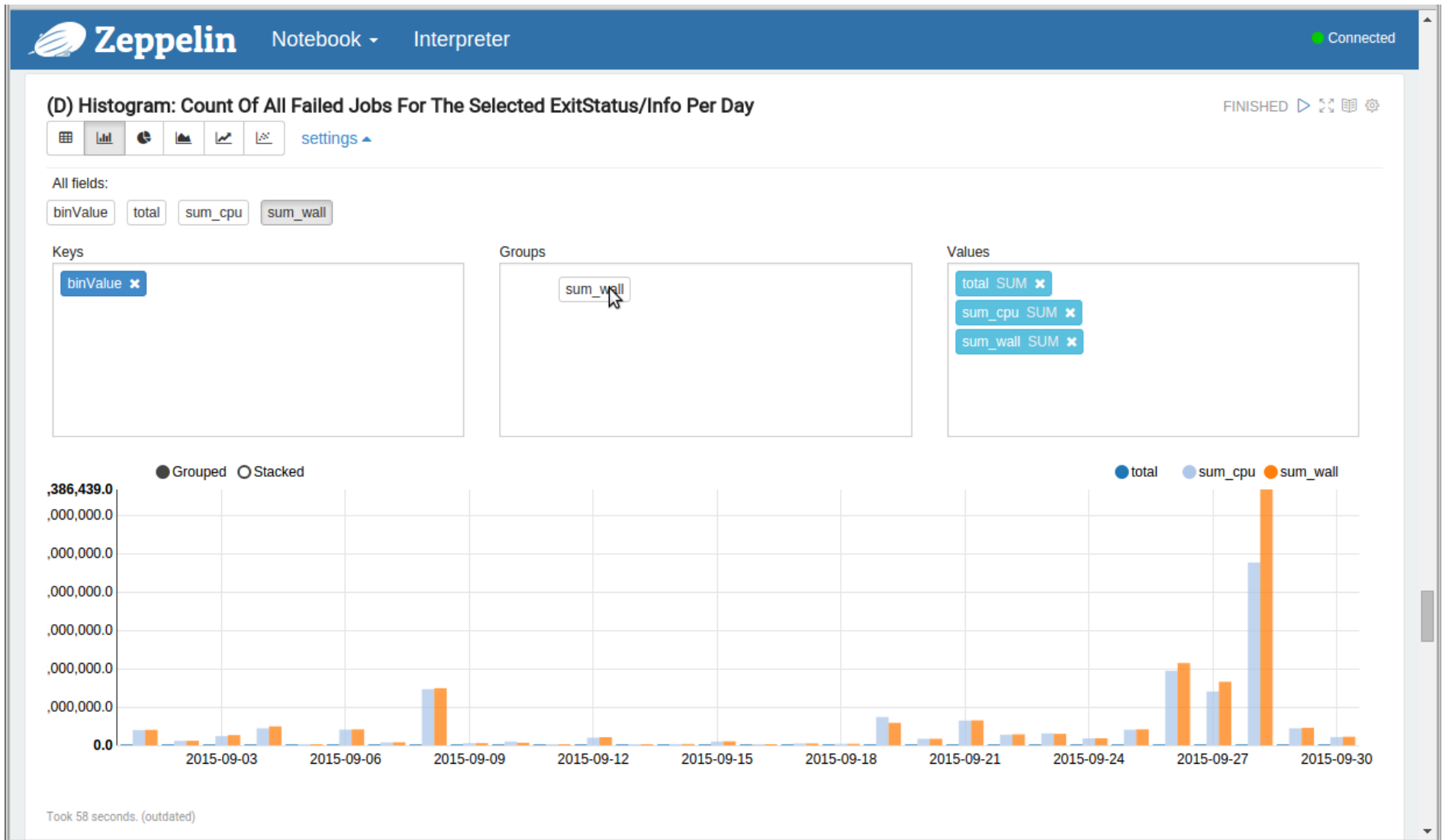
📅 📊 📈 📉 📊 settings ▾



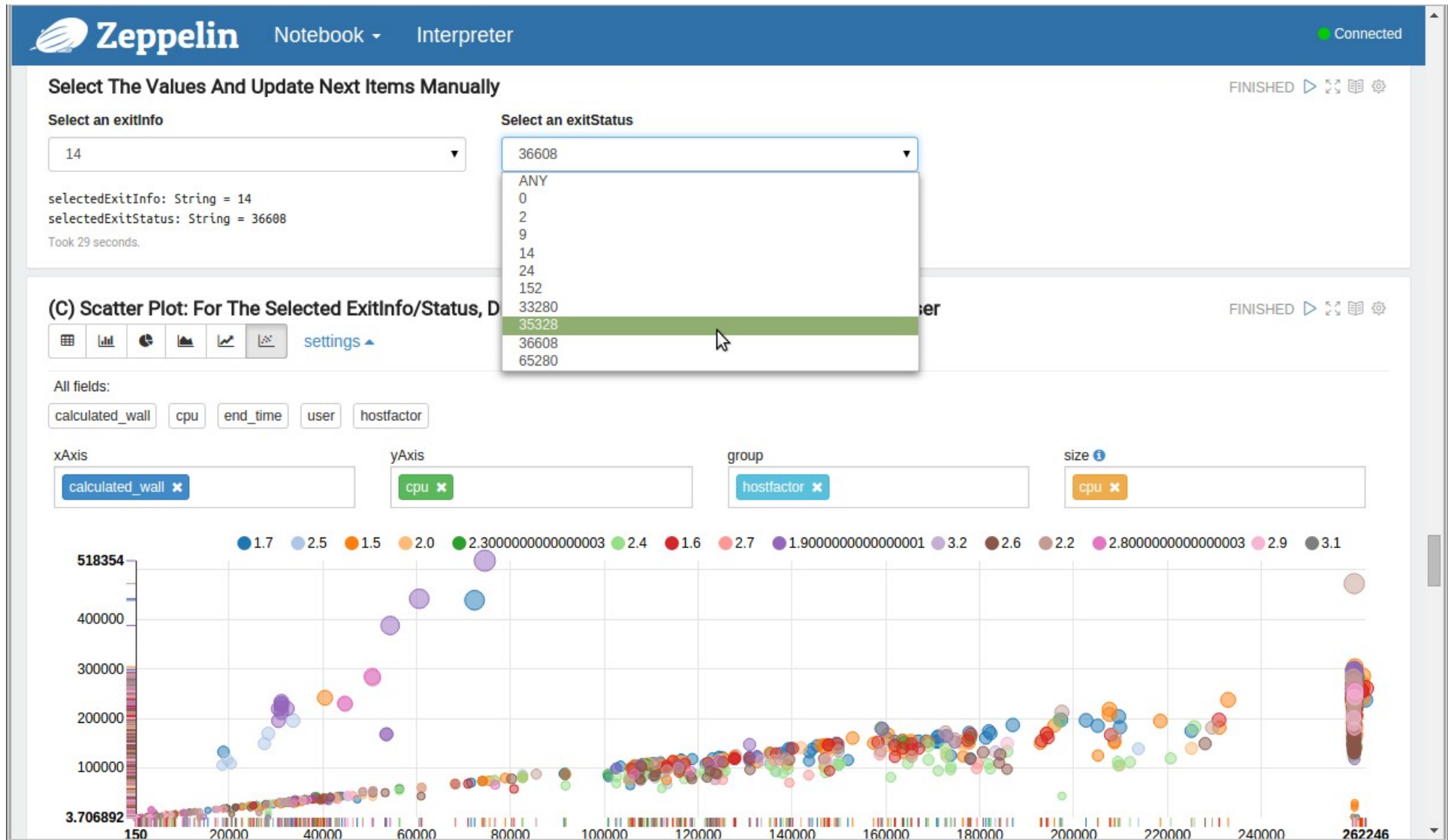
Different kinds of plots can be used to visualize the tabular result



Drag&Drop/interactive plot control



User-friendly, interactive plot customization exploiting dynamic forms



Combining Spark Scala with RDD/DataFrames SparkSQL visualization, improves the complexity of plots

