Monitoring & Traceability of Jobs using ElasticSearch for DIRACGrid project

Yash Srivastava
Google Summer of Code participant with CERN-HSF
Indian Institute of Information Technology, Sricity
Andhra Pradesh, India

About DIRACGrid Project

- The DIRAC is an “interware” which means a tool for exploiting (distributed) computing and storage resources.

- The tool performs a variety of roles ranging from the submission of jobs, the management of the data produced, to the orchestration of the distributed resources, while providing active monitoring and key information for the whole LHCb collaboration.

- DIRAC is a generic software, used and extended by several Virtual Organizations (VO). Communities use DIRAC to submit jobs to hundreds of heterogeneous computing resources, with several tens of thousands of jobs running concurrently.

Jobs and it's Monitoring & Traceability

- "Job" is defined as a computational unit, like simulation etc. These jobs contain various parameters like "long", "short", "memory hungry" , "I/O bound", "if job accessed input files" , "on which host job ran" etc.

- The "Job" unit provides a lot of attributes as described in the previous point. Important information can be retrieved from these attributes which can help assess the job's operation.

- Monitoring of jobs is essential as it can help us detect dead processes as well as helping with better resource management.

- Traceability of each of the submitted jobs is key when security checks are needed.

Problem

- The problem with the existing MySQL solution for the DIRACGrid project was that only some of the records are stored and statistical information in not being extracted from the obtained parameters.

- This is due to usage of ***relational database*** (here MySQL), where it becomes difficult to keep and search the data as it is stored in key-value pair format.

- Along with the above problem, the current code is only compatible with Python 2 (precisely 2.7.13) for its development. Python2 support will end in 2 years (approximately in 2020), and more importantly new tools and features are only available within python 3.

Solutions Proposed

- Keeping the mentioned problems in mind, solutions were proposed to work on during the summer so as to support good and efficiency data management as well as working towards newer modules supporting Python 3.

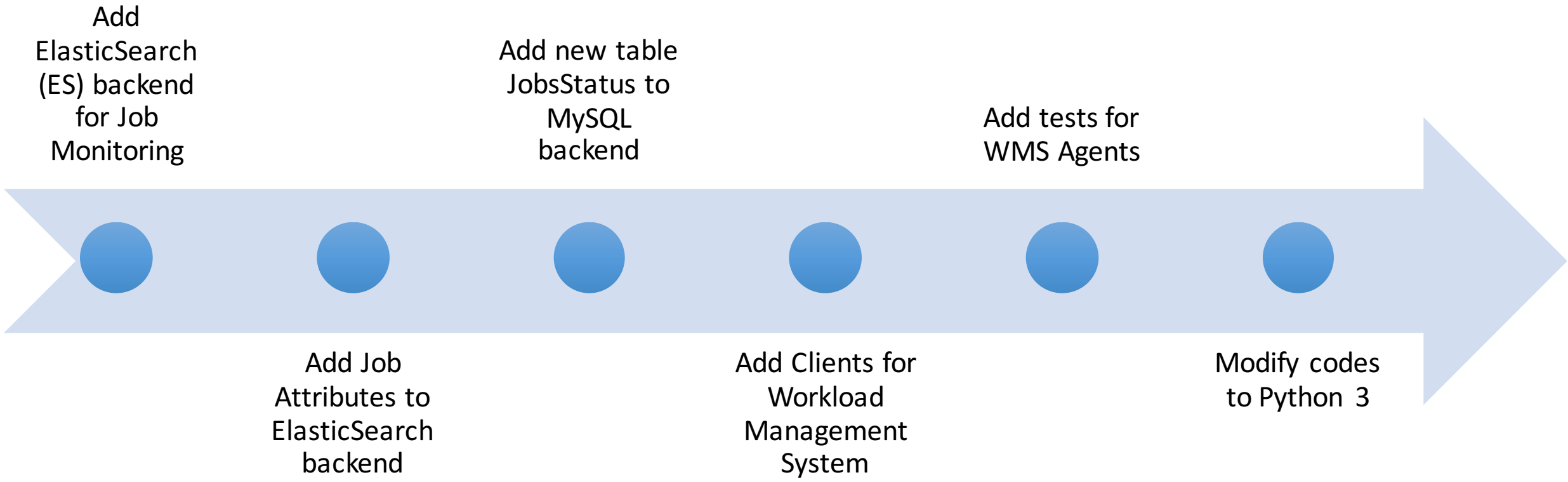
- Shifting to a non-relational database would help the cause of making the queries possible and easier. **ElasticSearch**, which is based on NoSQL (a non-relational DB), a current state-of-the-art would help to achieve this. Hence, ElasticSearch backend is used to store the DIRAC's job parameters, so that querying the data becomes easier.

- It is also important to move the project to python3 or **make it 2to3 tool complaint**: 2to3 tool is used to facilitate the conversion from python2 to python3. Hence, writing code in compliance with this tool, so that codes can be easily ported to python3 when needed.

Tasks Completed

- Extending the current job monitoring system, currently based upon relational databases, by using ElasticSearch (NoSQL DB), a state-of-the-art solution.
- Code Submission complaint with 2to3 tool, to make python3 porting trouble free.
- Unit and integration tests for the working implementation.
- Documentation for the working implementation.

Milestones


Add ElasticSearch (ES) backend for Job Monitoring
Add Job Attributes to ElasticSearch backend
Add new table JobsStatus to MySQL backend
Add Clients for Workload Management System
Add tests for WMS Agents
Modify codes to Python 3

Add ElasticSearch backend for Job Monitoring

- The Job Monitoring module in Workload Management System used MySQL backend to store and access Job Parameters. But the only caveat of using MySQL is that it is a relational database and also limits the queries that can be processed due to the relationship between keys.

- Keeping the above in mind, using ElasticSearch (ES) which is NoSQL DB (a non-relational database) seemed a good choice and hence set up ES indices and wrote functions ***setJobParameters and getJobParameters.***

- The functions accessed the indices and wrote and retreived the values available, as clear from terms *set and get.*

Add Job Attributes to ElasticSearch backend

- The Jobs table contains a set of values which are most commonly accessed by the queries processed as per the requirement. Hence, it becomes important that these values are moved to ES backend, as it would make query processing efficient as well as open up newer queries that can be performed.

- The attributes that were moved to ES are: *Job Group, Owner, Proxy, Submission Time, Running Time.*

- In order to set and retrieve these attributes, function ***setJobParameters*** is modified to accept these attributes as *kwargs* (keyword arguments) and then set these values as and when specified by the user.

- A new function is introduced in ElasticJobDB.py named ***getJobParametersAndAttributes*** which returns both parameters and attributes mentioned above when given a JobID.

Add new table JobsStatus to MySQL backend

- As discussed in previous slide, not all the values linked to particular JobID are accessed as often than some of the columns of the Jobs table.

- Hence, separating ***status*** values from the Jobs table is an efficient way of accessing these values as these are most often queried and would make the processing more efficient in terms of traversing rows and columns of the table when compared to much-loaded Jobs table.

- The columns of the new tables are as follows: *JobID (primary key), Status, MinorStatus, and ApplicationStatus.*

- Along with this, two functions were written/modified to access the new table:
    1. *getJobStatus* (to retrieve values from the table)
    2. *setJobStatus* (to set values in the table)

Add Clients for Workload Management System

- It is desirable that WMS agents don't access the DB's (both ElasticSearch and MySQL) directly and instead access them via running service. These services are initiated using *RPClient()* and hence need to be initiated and used to access or write to DB's.

- But at the same time, it is also needed that the modules themselves don't call these RPCClients and use the ***Client*** class which initiates the service.

- Keeping the above points in mind, I added the following clients in WMS and replaced their RPCClient invokes to these classes:
    1. JobStateUpdateClient.py
    2. JobManagerClient.py

Add tests for WMS Agents

- It is of common knowledge that with development, testing becomes an important part of the whole process. It is essential as we keep changing the codebase, we need to ensure that new changes don't affect the existing functionality and doesn't disturb the whole process.

- Since the tests for WMS agents were quite limited, I added the tests written in accordance with 'pytest' module for the following agents:
    1. JobAgent.py
    2. JobCleaningAgent.py
    3. PilotStatusAgent.py
    4. StalledJobAgent.py

Future Work

- As the current codebase is in transition from Python2 to Python3, I would be contributing towards incorporating code changes required to support both the versions.

- Some the existing and newer tests don't have enough coverage of the modules available in the project. Hence, expanding tests would be major contribution ahead checking functionality in both the python versions.