HLoader
Summer Student Project Report

Dániel Stein (IT-DB-DBF)

2015 Summer

For 10 weeks —from 15th of June to 21st of August— I was a Summer Student at the CERN Database Group. This report summarizes my project and my progress.
Supervisors: Kacper Surdy, Zbigniew Baranowski. Project ID: 15465.

1 Task Description

Hadoop framework is becoming a top open source player on field of distributed system offering possibility of storing and processing big sets of data in a scalable manner. Nowadays it is also gaining momentum at CERN – there is an increasing interest in solutions based on Hadoop ecosystem in many areas including experiments, accelerator controls archives etc. It appears as a natural replacement of a traditional relational system whenever ad hoc analytical processing is a dominant part of data workloads.

Some of the systems at CERN already have offline replicas of archive data stored on a Hadoop created manually by a system administrator. Such process is very often time consuming and requires applying sequence of actions on the metadata and data itself.

The goal of the project is to automatize the process of data loading between Oracle database and Hadoop cluster by creation of a utility that interfaces with both systems via dedicated tools (Apache Sqoop) and applies all necessary actions in order to delivered ready-to-read data on Hadoop file system for high-end frameworks (like Impala or Spark). The tool should support incremental data loading on a time scope bases and should be configurable for each data set separately.

2 Challenges

Based on the task description and the initial consultations, my task was to build an automated, incremental Oracle–Hadoop data transfer framework and service with the following in mind.

It has to **control and monitor data transfers** using Sqoop, a CLI tool for bulk data transfer. Its aim is to **take the load off the system administrators** by executing the jobs with an automated tool. The system should **improve communication** with the users by notifying them about the status of the transfers.

The service should **handle failures**, retry, notify and prevent in case of an error. By nature **security** is one of the main concerns, only authorized data transfers should be executed. It should be **easy to use** for every user without help. And most importantly the whole solution has to use the **CERN-provided infrastructure**.

3 Solution

This problem was so current and acute, two distinct Summer Student tasks have been created. My task description required a more generic solution, while Anirudha Bose —my project partner— got a task concentrating on CMS Job Monitoring data transfers. The following sections detail our generic solution.

3.1 Overview

The architectural overview in Figure 1 shows both the main parts of the service and also gives an overview of the interfaces these parts communicate through.

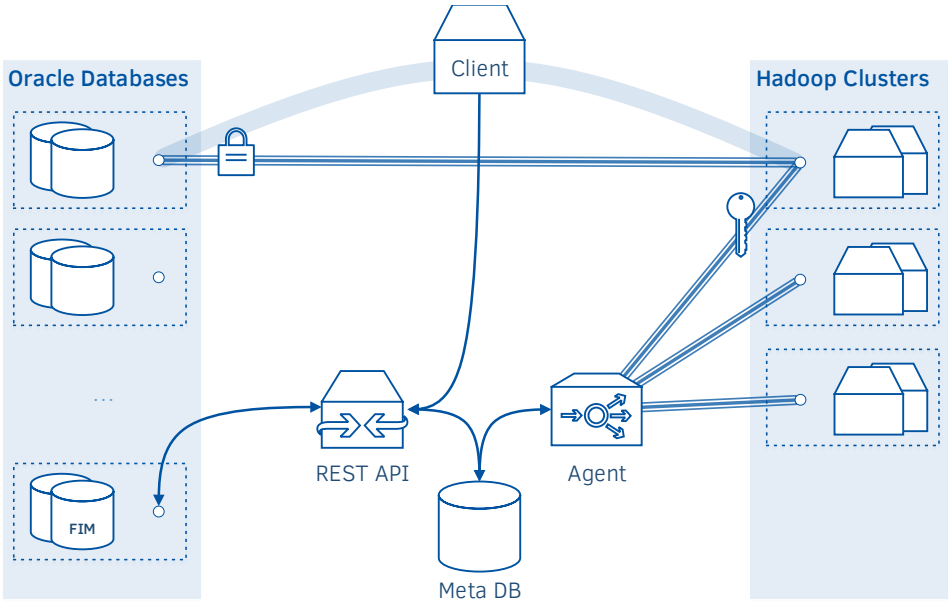

Figure 1: Architectural overview of the solution

Provided Infrastructure The service should bridge the Oracle and Hadoop services at CERN. There is a two node Oracle 11.2.0.4 cluster with shared NAS as a data storage solution, and several Hadoop clusters. Hadoop clusters have various hardware configurations ranging form 17 to 63 nodes in a cluster and having 4 to 16 physical CPU cores per node.

Transfer Data When a user contacts the Databases group or the system administrators with a new transfer request, it usually contains the same set of information: what, when, from where and where to transfer the data. Based on these parameters, Sqoop jobs can be automatically created for the average, everyday use cases.

Surely there are many, less important configuration parameters, e.g., whether it should be an incremental update of an existing copy, or just a one time transfer, or the number of map-reduce jobs to spawn.

Execute the Transfer on Behalf of the User After the user provided every necessary information and they were validated, the system can take over. It schedules and executes the job at the requested time and also informs the user of the status of their request.

Update when Needed If the user requested incremental updates and the previous, updating transfer was successful, the service schedules another updating transfer after the given interval.

3.2 Security

While Section 3.1 described the expected user experience and workflow, this section details how we implemented the underlying system and made it secure by design.

CERN SSO Authentication The first thing the user has to do in order to use the service is to log in. To avoid password exchange and the hassle of storing user data and the dangers of authentication, we opted for the CERN central Single Sign-On. Using this solution also allows us to restrict the user base to a preselected group of logins managed centrally, without modifying the source code itself.

Authorization After the user has been authenticated, the service lists all the databases and schemas that the user owns. The FIM database is used for authorization, but only a portion of the owned schemas are available for the user, since —for security reasons— only the needed databases are configured to be used with the service.

Kerberos SSH Tunneling When the assigned time comes and the transfer has to be executed, the scheduler starts an agent that connects to the designated Hadoop cluster. The connection is secured by an SSH tunnel and authenticated using Kerberos.

Secure Password Input In order to hide the database connection password from the other users of the cluster machine, Sqoop provides two options. The agent could either enter the password in interactive mode, meaning the password is not included in the list of running commands, or use a password file. Since the cluster might not be configured to have a shared, secure folder, we opted for the first solution for the time being.

3.3 Modularity

The service is not only built to be secure, but to be easy to modify and extend too. In this section we detail the parts and solutions that make this possible.

DB Connector Agnostic The service uses SQLAlchemy Object-relational Mapping (ORM) technology to communicate with the underlying service. It supports several dialects, meaning we could choose from MySQL, Oracle, PostgreSQL, SQLite and many other dialects. Also using the modularity of the framework, SQLAlchemy could be replaced with another solution for using NoSQL databases as a backend.

Interchangeable Scheduler For each running instance of the service there is a scheduler working in the background. Based on the limitations of the infrastructure and the required complexity of the scheduling algorithm, this module can be replaced.

Flexible Communication with Hadoop While the current solution uses SSH-based communication with the Hadoop clusters, there is at least one more way to start transfers. Apache Oozie, a workflow scheduler for Hadoop could also be used for scheduling and running Sqoop jobs.

Besides these modular components, the frontend client and the Sqoop JDBC drivers can also be changed making the whole solution and framework modular.

3.4 Deployment Infrastructure

This section describes where these modules, the different parts of the solution can and have been deployed using the infrastructure provided by CERN services.

PostgreSQL On-Demand For the backend database we are using the locally managed PostgreSQL on-demand service. With SQLAlchemy, abstracting the database access, we could easily move to another underlying service.

Central Web Services Since the backend and the frontend service requires SSO authentication, we opted for using central web services. From the three available options (AFS (UNIX), DFS (Windows) with 2 different server versions), we chose DFS with IIS 8.5, since it had Python 2.7 installed. To be able to use Flask (a Python web server), FastCGI had to be configured. The full stack looks like the following: DFS | Windows > IIS 8.5 > FastCGI > Python 2.7 > Flask

Separate Agent There are many options for running the agent, since it only requires Python and a few libraries to run properly. As the SSH tunneling is done using the Paramiko library, a pure python SSH interface, it could run virtually anywhere. We will probably use an UNIX OpenStack VM for the agent.

Client Hosted with the REST API Using the REST API for the backend makes the client side decoupled from everything else. Anyone with an SSO cookie can communicate with the interface. The CERN infrastructure requires to have the interface and the client on the same domain, or the SSO will not work properly. Only for this reason the client HTML, CSS and JavaScript files are hosted with the REST API.

3.5 Stored Information

Figure 2 shows all the data that we store. We maintain a list of the available Oracle servers and Hadoop clusters. We keep every parameter the user has provided about the jobs, and for every execution, every transfer we store the actual status and metainformation. Also, the logs coming from various sources are also stored.

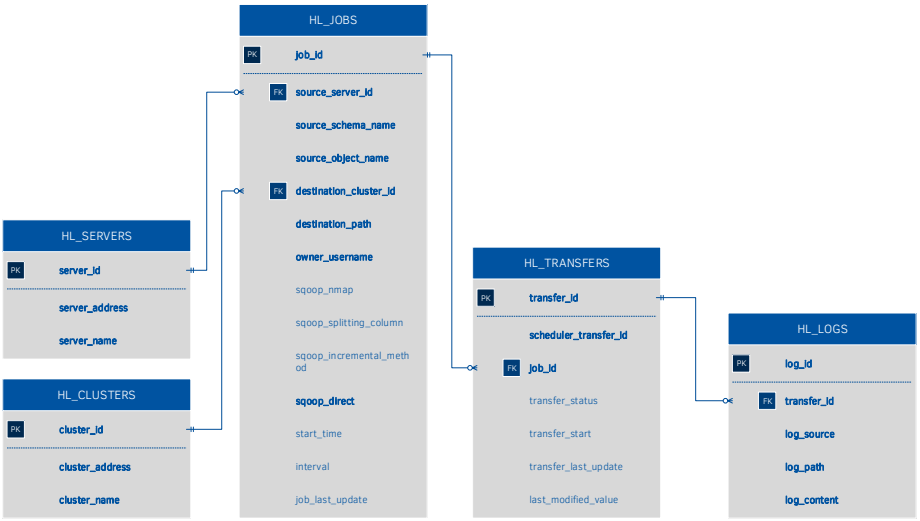

Figure 2: Database schema for the backend

3.6 Restrictions, Constraints

For security reasons and the length of the Summer Student program we had to introduce restrictions. For example we only allow tables and views to be imported, making the Oracle database responsible for evaluating and checking the queries, whether the user can do what they want. The source databases are also limited, making gradual introduction for new users and servers possible.

The destination folder structure on the Hadoop clusters is preset, the user could only change a relative path from the preset base. This serves security with limited access rights and avoids collision and unauthorized access. While the Sqoop commands have diverse parameter and switch configurations, our service only supports a small set of them.

3.7 Current State

This section presents the current state of the parts of the solution. **Client** developed by Katarzyna is in progress, in the meanwhile the REST interface can be used for interaction with the backend. The **REST API** developed by me is almost ready, the new job processing interface needs a few improvements, and the job modification interface is missing. Other missing interfaces (for transfers, logs) are easy to make. The **Scheduling** part of the agent is developed by Anirudha. It is basically ready, can schedule jobs and update itself after job description modifications. The **Runner** part of the agent developed by me is working for initial imports, soon will be able to execute incremental updates. With some improvements, it will work perfectly with SSH and REST monitorig.

4 Summary

During the summer we have managed to design and develop an easily expandable framework and service for transferring data from Oracle to Hadoop. We designed it with automation in mind, minimal administrator intervention is needed.

4.1 Future Work

While we achieved to finish the core part of the project, there are several extensions we would like to see in the future. The system could gain from supporting alternative runners like Oozie, or use the now under development version of Sqoop. Integration with Hive and resolving the restrictions not responsible for security is also subject of future work. Releasing the source code on GitHub with an Open Source license could also mean additional contributors to the framework.

4.2 Contributors

Besides Anirudha Bose and me, many supervisors (Antonio Romero Marin, Domenico Giordano, Kacper Surdy, Katarzyna Maria Dziedziniewicz-Wójcik, Manuel Martín Márquez, Zbigniew Baranowski) were working on this project helping us to achieve what we have, which we are very thankful for.

4.3 Workflow Tools

To coordinate a project of this size, we needed some development and workflow tools. We used the local GitLab service and GitHub to collaborate, Jira for following the progress of the project, Slack for everyday communication and Jenkins for testing and CI.