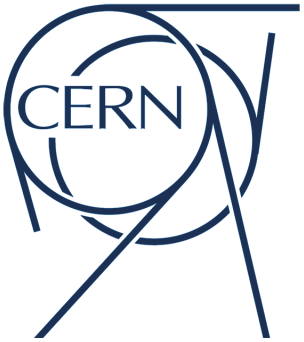
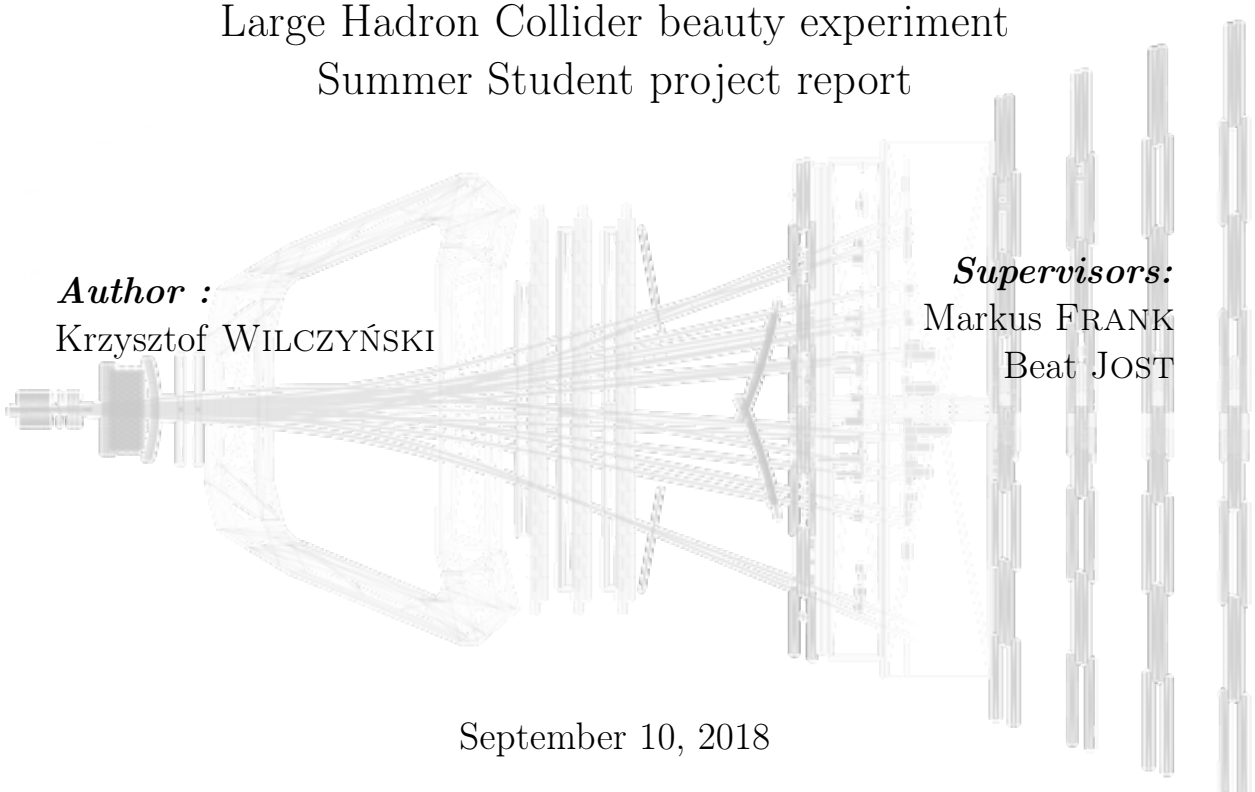


An alternative approach to configure permanent tasks in LHCb Online farm nodes

Large Hadron Collider beauty experiment
Summer Student project report

Author :
Krzysztof Wilczyński

Supervisors:
Markus Frank
Beat Jost



September 10, 2018

Contents

1 Introduction

As a part of LHCb Online group activities, the project had purely applied nature. It could be classified as full-stack IT development, from database back-end, trough API and unit testing, to command line and graphical user interfaces. There was also time for experiments and technology / protocol comparison that was very enriching for the author - a Summer Student.

This document aims to give a brief description of the project, without going into technological details. There is also a documentation of the system that was born during the project, should the reader be interested in more in-depth information.

1.1 Online farm nodes

An online farm is a cluster of multipurpose computing machines (also referred to as: CPUs, nodes) that are responsible for high level triggering, online analysis of data, storage, logging, monitoring et cetera. A brief outlook of LHCb Online farm is shown on figure 1.

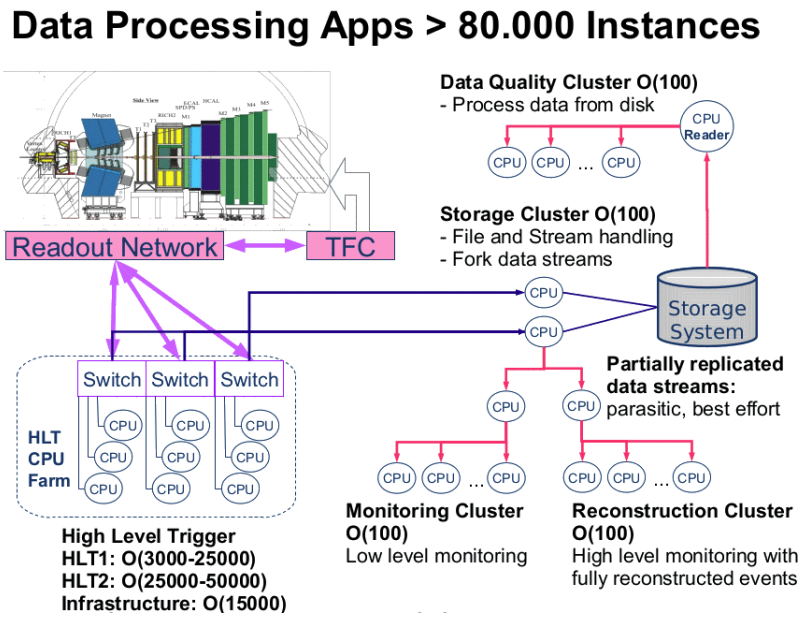


Figure 1: Online farm architecture

1.2 The permanent tasks

Nodes are grouped according to their purpose in the system. The node groups such as "High Level Trigger" or "Reconstruction Cluster" shown on figure 1 are defined by sets of permanent tasks (processes) running on each node in given group. Those processes are started on the nodes at system boot (start) and continue to run during system operation (unlike special tasks that may start and end anytime).

An artificial hierarchy of tasks, task sets, node classes and nodes (top level regular expressions) emerged as a way of imagining and documenting the complex relations between the purpose of node groups and the tasks running on them. A diagram shown on figure 2 describes the parent-child relation between those groups.

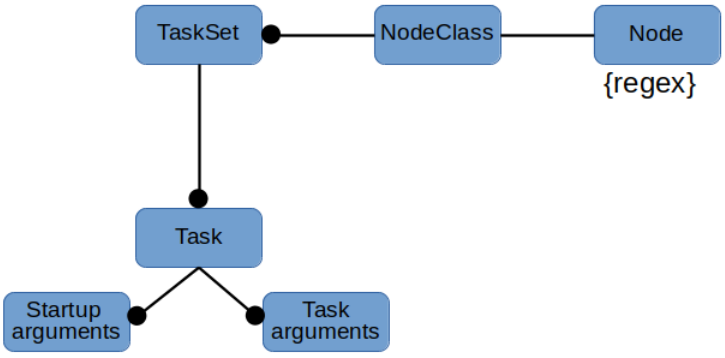


Figure 2: The task hierarchy in the farm

1.3 Online farm process controller

The process controller is a part of LHCb FMC (Farm Monitoring and Control System) that is responsible for starting, stopping and maintaining the processes running on worker nodes in the online farm.

Special nodes - "control PCs" are hosts of "pcSrv" process that allows them to supervise processes running on worker nodes assigned to them (each control PC in the current system oversees 200 nodes).

The above description can be perceived as over-simplified, to find more details about the process controller and underlying DIM services, please consult a Researchgate article by Federico Bonifazi, Daniela Bortolotti, Angelo Carbone, Domenico Galli, Daniele Gregori, Umberto Marconi, Gianluca Peco and Vincenzo M. Vagnoni linked below.

The Process Controller for the LHCb Online Farm (LINK)

.

2 Motivation

This section clarifies (hopefully) the reasons behind the development of a new system for permanent tasks configuration in LHCb online farm.

2.1 Farm boot script

Up to this point, all processes started on the farm nodes are grouped in a single, huge python script that prints out ready to execute "pcAdd" commands for a given node name. Those commands are then executed by the control PCs - the tasks ran on the nodes.

Here is a simplified structure of a pcAdd command:

pcAdd(ScriptToStart.sh, <script parameters>, <pcAdd command parameters>)

This method of defining and executing the boot sequence had many clear disadvantages, most importantly:

- modifications of task parameters were difficult (one had t browse the script to find the "hardcoded" configuration)
- it was relatively easy to break dependencies between items in the hierarchy - they were not clearly visible in a huge text file
- only a specialist who knew the boot script very well could use it
- there was no good source of information about tasks that run on a given node as the script contained also duplicated entries - there was no transparent one-to-one relation between running processes and their configuration

The boot script in this form was born as a "quick hack" about ten years ago, when the LHCb Online Farm Monitoring and Control system started. The time has come for an imrovement!

2.2 Aims of the project

The main goal of the summer student project was to create a cleverly designed database-driven solution that would replace the old solution, bringing aid to all of its major problems.

Trough the usage of database back-end it was possible to achieve the following goals:

- the modifications of hierarchical task structure has become easier - database tables are clearer than plain text notation
- thanks to special constraints on the data entries, the system prevents human errors that could lead to breaking dependencies in the hierarchy and system integrity
- a single data access point with one-to-one relation between running tasks and data has been created - it is easier to find errors and make improvements to the permanent task configuration
- it was possible to create a future-proof and reliable API (application programming interface) for further developments

3 Development

To make explaining the solution easier, the project description was split up in the following parts:

- Back-end
  - Database schema architecture
  - Main database API
  - New boot script
  - Unit testing script (internal error prevention)
  - Front-end connectors: JSONRPC, (REST, XMLRPC)
  - Merge of the back-end solutions
- Front-end
  - Command line user interface
  - Graphical user interface (web application)

The back-end elements are the parts of the solution that are not visible for the user. In fact, the operator does not have to be aware of the underlying database and API at all.

The front-end programs are designed with a purpose to make the usage of the system as easy and intuitive as possible. Those are the visible elements that the user interacts with.

4 Back-end

4.1 Database schema architecture

The database schema - organization of data in tables and creation of associations between them is truly the heart of the whole system. This is why it deserved special attention - a lot of work has been put into creating optimal architecture that will support any possible usage of the data and to prevent breaking the integrity of the structure by the requests.

Many technologies have been considered - from classic SQL databases to unconventional non-sql solutions such as MongoDB. As a result of LHCb Online's policy of keeping the foreign dependencies as low as possible, SQLite database engine (an integral part of Python) has been chosen in the end.

4.2 Main API

The Main API is a Python class that offers high-level methods (simple functions) for other applications to access the advanced database operations in a safe way. Thanks to this script, it is very easy to use the database to its full potential without knowing its detailed structure. All low-level database operations (SQL queries) are executed only by the API script when a simple API request containing valid parameters is made by any client application.

The API offers the following functions for any of its data elements (tasks, task sets, node classes and nodes):

1. add
2. delete
3. modify
4. get
5. assign
6. unassign
7. inSet

The first four methods allow the client applications to influence the data entry directly, the rest is responsible for creating many-to-many connections in the database by assigning, unassigning and displaying items assigned to a given set (tasks in task set, task sets in node class and node classes in nodes).

4.3 New boot script

To fully recreate the functionality of the previous solution, a new boot script has been written. As a modular structure of the system has been introduced, the boot script is no longer consisting of both storage and execution of the configuration data - it is just an optional client application for the Main API. The inner structure of the script is thus very simple - it just uses Main API's methods to find all tasks that belong to a specific node regular expression (input).

4.4 Unit testing script

In the IT world, unit testing is a name for validation of the program's functionality. In the designed system, the most critical part that needed testing in order to go into production (replace the old solution) was naturally the Main API. This subsystem is used to check if the changes made to the API do not harm its functionality - the script will be ran as a part of daily integrity check.

Unit testing was surely not the most exciting part of the project. In principle, the script had to execute all possible API methods and verify if they work properly. To achieve that, the script first verifies the API's response to the method call and then checks if the requested action had modified the underlying database in the right way.

This naturally applies not only to correctly called methods (all of the parameters provided, no logical errors). The unit testing script also needs to verify if all of the possible errors are handled correctly - right error code is returned and no modifications are made to the database.
After each test method call, the script prompts the result in the command line using unified formatting and font colouring - it is easy to spot an error and find its source. It also returns the number of errors found - it is possible to integrate the script in a larger production unit testing system.

4.5 Front-end connectors

In order to allow web based applications to use the Main API, it was required to create a bridge between server side Python and client side JavaScript. The previous LHCb Online solutions using the same front-end framework (Sencha Ext JS) adopted XMLRPC as the communication protocol.

An important part of the summer student project was to experiment and find the optimal technologies for the system. Therefore, many front-end connector possibilities were considered, most notably:

- REST
- JSONRPC
- XMLRPC (tested in older applications)

All of the listed solutions were implemented as Python web servers and then tested.

REST is an unified way of navigating trough data based on URL routing. For instance, if one would like to delete task called Task_1, a DELETE HTTP request should be sent to http://server/tasks/Task_1 (URL routed to the data point). In short, the Main API methods are mapped onto regular HTTP methods and the data affected is chosen by a URL respecting REST standard.

The later two protocols are based on the same principle - RPC: remote procedure call. The client - graphical user interface (GUI) only needs to send a request containing the called method and its parameters in a specific form (XML or JSON file in the body of the request), using POST HTTP method. The RPC web server then calls the requested Main API method directly and returns its response.

In the end, JSONRPC has been chosen over XMLRPC and REST. It was the optimal solution as:

- The connector does not need any changes when new method is added to the Main API - it will still call it when requested (unlike REST)
- JSON file format is often considered the successor of XML as it does not contain as much text tags. In the end, the data size transferred in XML requests was mostly

the tags. JSON allows sending the same data without unnecessary tags while using way less network bandwidth.

- Sencha Ext JS framework used in the previous solutions needed a special, external library to be able to talk to the XMLRPC back-end connector servers. LHCb Online aims to reduce the amount of foreign libraries used to guarantee stability after system upgrades - it was decided to use Sencha Ext JS native communication tool: Ajax requests which happen to use JSON format.

4.6 Merge of the back-end solutions

When all of the back-end prototype solutions have been finished and tested, the time has come to merge them into a system that would be easy to start and configure (to run on a given port, use given database, run in debug mode et cetera). This integration took a lot of work from Markus Frank's side as he will be the main user and maintainer of the solution.

All of the front-end connectors were merged into a single, elegant web server hosting all of the connector services (as well as file hosting for the main HTML of the graphical user interface). Therefore the system can be fully started and configured with a single command.

5 Front-end

5.1 Command line interface (CLI)

The first user interface to be developed was naturally a command line interface. It is relatively easy and fast to create it, moreover it is still preferred over GUI applications by many users as it can be automated and debugged easier.

The CLI script is a set of commands that utilize Main API but are easier to operate for humans. Executing each command is assisted by guidance - questions (yes/no) to make sure that the user knows what is happening to the data. After each command, there are also easy to understand status communicates that inform about success or give an explanation of an error the operator has commited.

5.2 Graphical user interface (GUI)

When all of the required parts of the project were working properly, the time has come to develop the graphical user interface. The main aim of it was to make the visualization of task configurations easier and allow the user to reconfigure assignments in task hierarchy using drag and drop technique. All the tasks, task sets, node classes and nodes

were grouped in reconfigurable and sortable grids and all of the API methods have their graphical representation - popup windows allow modifications, creation, deletion and re-arranging assignments of the task data that is then processed by front-end connector and by the Main database API.

The interface has been developed using Sencha Ext JS ver. 6.2.0, a JavaScript framework which makes the creation of unified, similarly looking interfaces across the whole Online system relatively easy. It is basically a library of widgets that is used widely in the industry and thus it has a big community of programmers that exchange experiences in online forums.
Figure 3 shows the current visuals of the interface. It will surely be changed and improved further in the future.

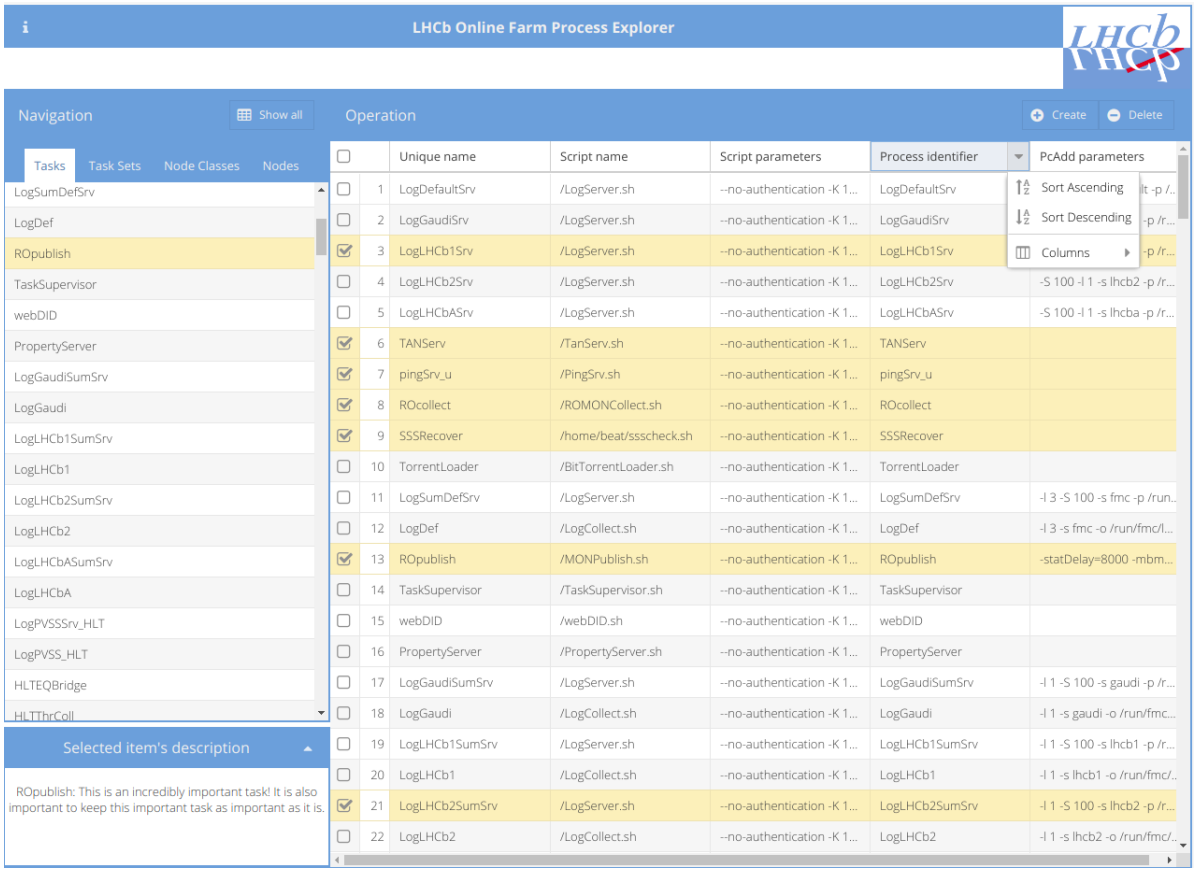


Figure 3: An outlook of the graphical user interface

By no means was the development easy - it was a great opportunity for a summer student to improve programming skills and get to know a new framework.

6 Moritz Karbach summer student prize 2018 winner

This summer student project has been awarded with 2018 Moritz Karbach prize. It is a collaboration award (LHCb) given to summer students "as recognition for outstanding performance" annually, in memory of Moritz Karbach, a young LHCb physicist who lost

his life in a climbing accident in April 2015.

To be qualified for the prize, the students had to give a brief (10 minutes + 5 minutes for questions) presentation of their projects and work in CERN. More details about the criteria of the award and a list of the winners from previous years (along with the presentations of the projects) can be found under the following link:

The Moritz Karbach summer student prize (LINK)

Figure 4: The Moritz Karbach prize diploma

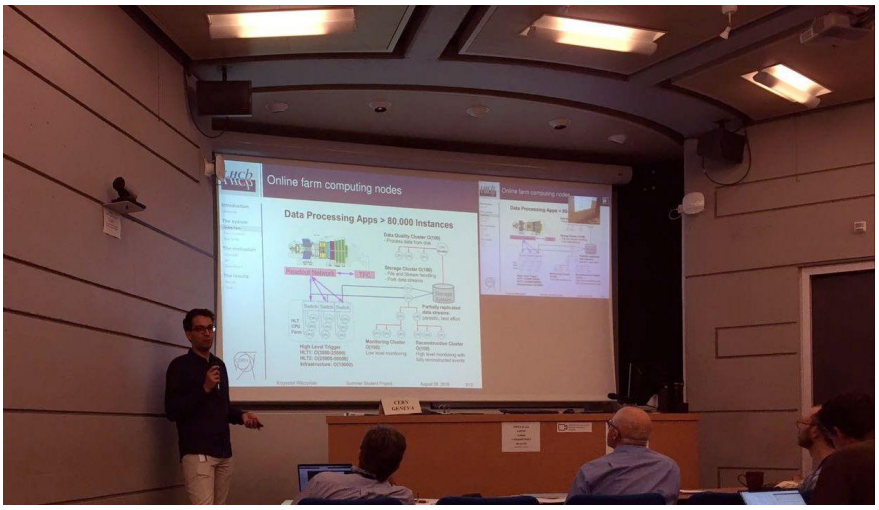


Figure 5: A photo from the presentations session

7 Summary

7.1 The project, code repositories

All of the project's goals have been successfully achieved and the system will soon be deployed in the production environment - the Online farm. All of the software developed during the thirteen week long internship (and some technical documentation) can be found in a Bitbucket repository linked below.

The author's open-source code repository (LINK)

After the student's departure, the project will be maintained and developed further by the main supervisor - Markus Frank who keeps it here:

The project maintained by Markus Frank in LHCb Gitlab repository (LINK)

7.2 The experience

The summer student project in LHCb Online group was a great opportunity to develop programming skills while attending amazing lectures covering various topics ranging from particle physics, through computing to accelerator engineering. The lectures, together with numerous visits in research facilities, fascinating workshops and the applied project made up the best, horizon-broadening summer a curious student could dream of.

The CERN Summer Student Programme was indeed an experience like nowhere else on Earth.

Thank You!