



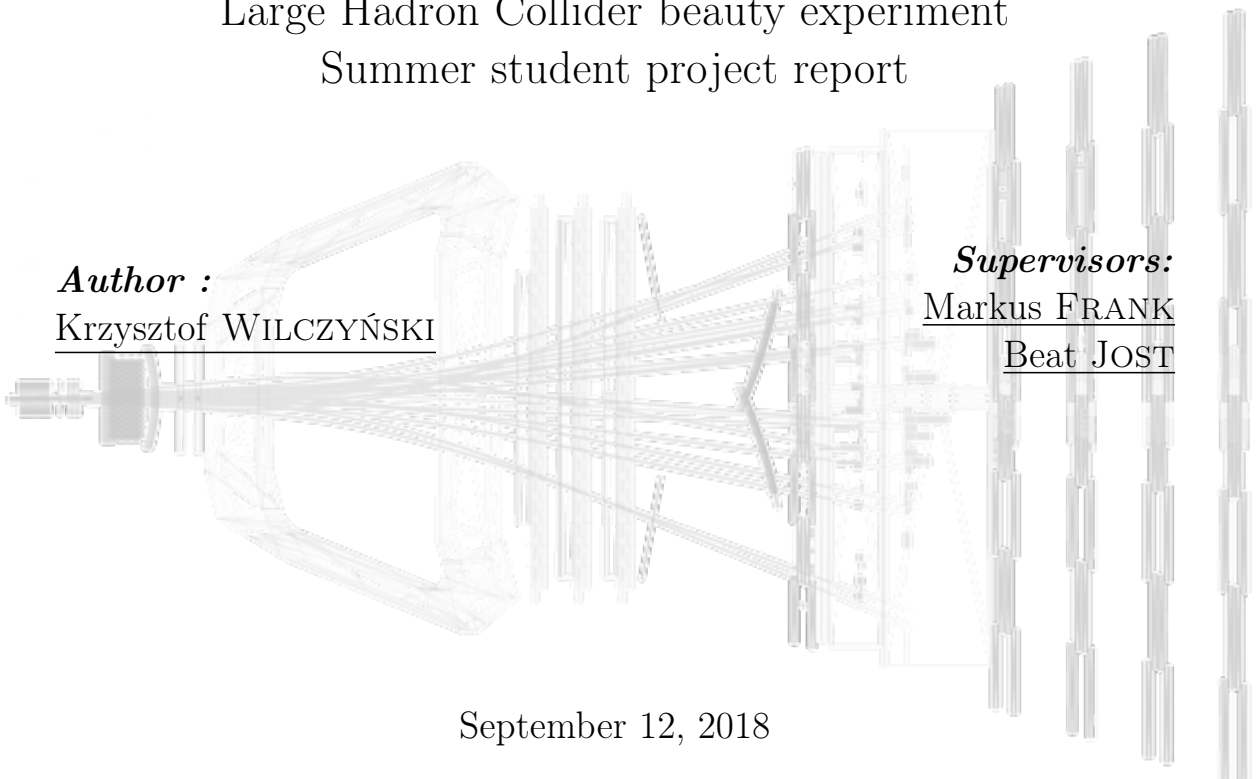
An alternative approach to configure permanent tasks in LHCb Online farm nodes

Large Hadron Collider beauty experiment
Summer student project report

Author :
Krzysztof WILCZYŃSKI

Supervisors:
Markus FRANK
Beat JOST

September 12, 2018



Contents

1	Introduction	2
1.1	Online farm nodes and permanent tasks	2
1.2	Online farm process controller	3
2	Motivation	4
2.1	Farm boot script	4
2.2	Aims of the project	4
3	Development	5
4	Back-end	6
4.1	Database schema architecture	6
4.2	Main API	6
4.3	Front-end connectors	7
5	Front-end	8
5.1	Command line interface (CLI)	8
5.2	Graphical user interface (GUI)	8
6	Derived applications	9
6.1	New boot script	9
6.2	Unit testing script	10
7	Moritz Karbach summer student prize 2018 winner	10
8	Summary	12
8.1	Project, code repositories	12
8.2	Experience	12

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, through API and unit testing, to command line and graphical user interfaces. There was some time scheduled 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 a separate documentation of the system that was born during the project, should the reader be interested in more in-depth information.

Summer Student Project Technical Report (LINK)

1.1 Online farm nodes and permanent tasks

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

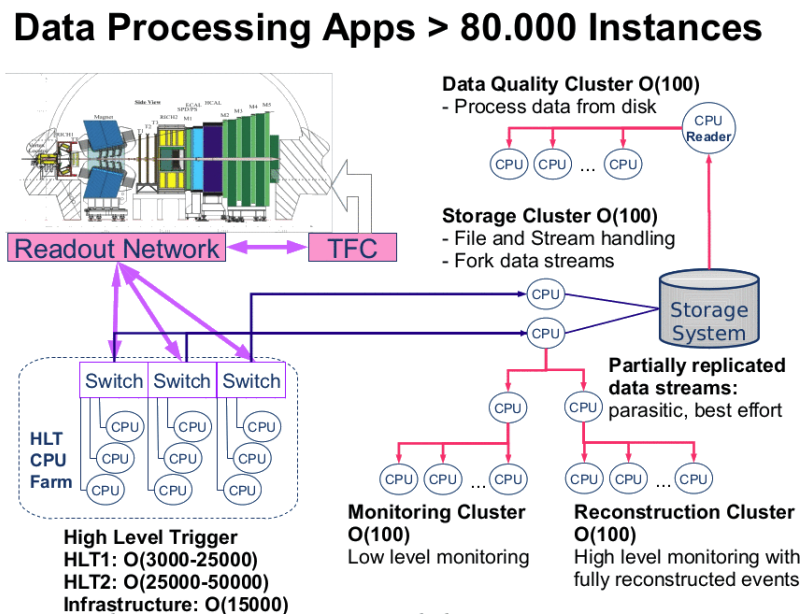


Figure 1: The LHCb Online farm architecture

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 the given group. Those processes are started on the nodes at system boot and continue to run during the entire operation time (unlike special tasks that may be started and stopped 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. This hierarchical structure proved itself to be useful in the current solution and was not changed during the project. The main goal was to make the access to the underlying configuration data easier (more details on the aims of the project can be found in section 2).

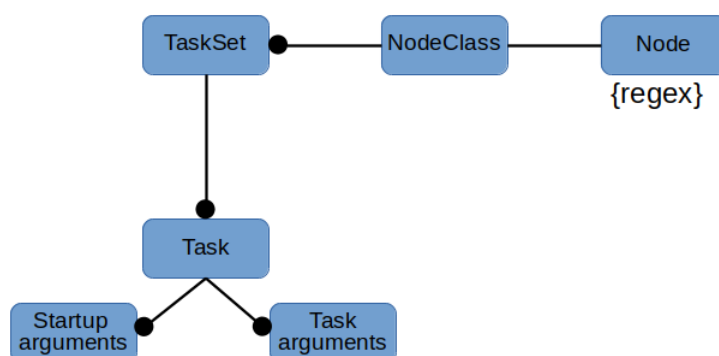


Figure 2: The task hierarchy in the farm

1.2 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 up to 32 nodes).

The above description can be perceived as over-simplified. To find more details about the process controller and underlying DIM (Distributed Information Management system) 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 and their configuration data (parameters) were grouped in a single, large (and increasingly difficult to maintain) 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 are started on the nodes specified by their name (or a regular expression pointing to a group of nodes).

Here is a simplified structure of a pcAdd command:

```
pcAdd(NodeRegex, ScriptToStart.sh, ScriptParameters, pcAddCommandParameters)
```

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

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

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 was installed.

The time has come for an improvement!

2.2 Aims of the project

The main goal of the summer student project was to create a cleverly designed database-driven system 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 (shown on figure 2) are now 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 the integrity of the system
- a single data access point with one-to-one relation between the running tasks and the configuration data has been created - it is easier to find errors and make improvements to the permanent task parameters
- it was possible to create a future-proof and reliable API (application programming interface) for further developments

3 Development

The core concepts and solutions developed during the summer student project can be classified as one of the three following categories:

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

The role of each of the elements listed above is described in further sections of the report (section 4: Back-end, section 5: Front-end and section 6: Derived applications).

4 Back-end

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

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 data structure by the requests.

Many technologies have been considered - from classic SQL (Structured Querying Language) databases to unconventional non-relational 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. It is, however relatively simple to change the database back-end's engine to Oracle or any other mainstream SQL solution should it be needed in the future.

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 simple to use the database to its full potential without knowing the details of its inner 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 | 5. assign |
| 2. delete | 6. unassign |
| 3. modify | |
| 4. get | 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 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 (REpresentational State Transfer)
- JSONRPC (JavaScript Object Notation Remote Procedure Call)
- XMLRPC (eXtensible Markup Language Remote Procedure Call)

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

REST is an unified way of navigating through 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 the URL respecting REST standard.

The two later 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 formatting provides the server with 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. As LHCb Online aims to reduce the usage of foreign libraries to guarantee stability after system upgrades, it was decided to use Sencha Ext JS native communication tool: Ajax requests which happen to use JSON data encoding by default.

5 Front-end

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.

5.1 Command line interface (CLI)

The first user interface to be developed was naturally a command line interface. It was 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 much easier than the graphical solution).

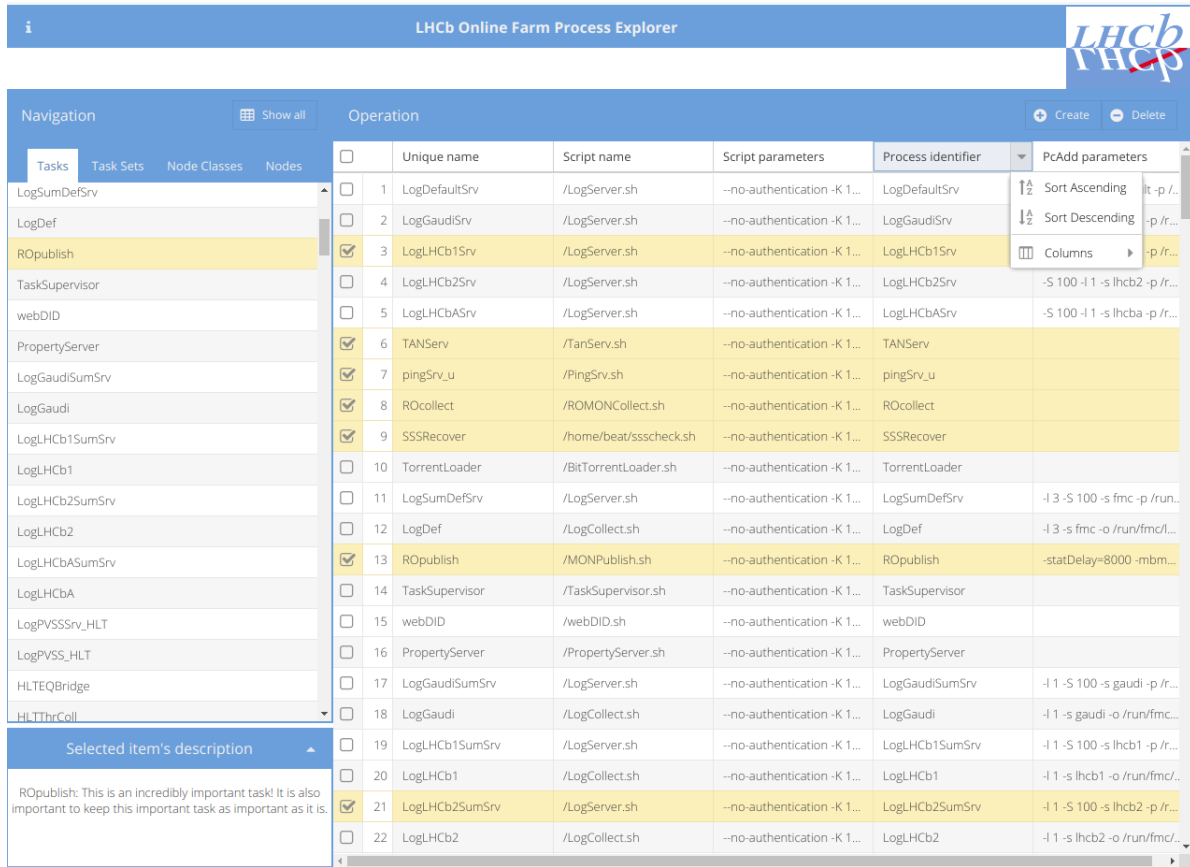
The CLI script is a set of commands that utilize Main API but are easier to operate for humans. Executing each command is assisted by guidance - questions (yes/no) to make sure that the user knows what is happening to the data. After each command, there are also easy to understand status communicates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

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 this part of the project was to make the visualization of task configurations easier and allow the user to reconfigure assignments in task hierarchy using intuitional "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 that allow for easy modifications, creation, deletion and rearranging assignments of the task configuration 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 possible. 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 visual form of the interface (one of numerous views). It will surely be changed and improved further in the future.



Navigation	Operation	Unique name	Script name	Script parameters	Process identifier	PcAdd parameters
LogSumDefSrv	<input type="checkbox"/>	1 LogDefaultSrv	/LogServer.sh	--no-authentication -K 1...	LogDefaultSrv	Sort Ascending
LogDef	<input type="checkbox"/>	2 LogGaudiSrv	/LogServer.sh	--no-authentication -K 1...	LogGaudiSrv	Sort Descending
ROpublish	<input checked="" type="checkbox"/>	3 LogLHCb1Srv	/LogServer.sh	--no-authentication -K 1...	LogLHCb1Srv	Columns
TaskSupervisor	<input type="checkbox"/>	4 LogLHCb2Srv	/LogServer.sh	--no-authentication -K 1...	LogLHCb2Srv	-S 100 -I 1 -s lhcb2 -p /r...
webDID	<input type="checkbox"/>	5 LogLHCbASrv	/LogServer.sh	--no-authentication -K 1...	LogLHCbASrv	-S 100 -I 1 -s lhcb2 -p /r...
PropertyServer	<input checked="" type="checkbox"/>	6 TANServ	/TanServ.sh	--no-authentication -K 1...	TANServ	
LogGaudiSumSrv	<input checked="" type="checkbox"/>	7 pingSrv_u	/PingSrv.sh	--no-authentication -K 1...	pingSrv_u	
LogGaudi	<input checked="" type="checkbox"/>	8 ROcollect	/ROMONCollect.sh	--no-authentication -K 1...	ROcollect	
LogLHCb1SumSrv	<input checked="" type="checkbox"/>	9 SSSRecover	/home/beat/ssscheck.sh	--no-authentication -K 1...	SSSRecover	
LogLHCb1	<input type="checkbox"/>	10 TorrentLoader	/BitTorrentLoader.sh	--no-authentication -K 1...	TorrentLoader	
LogLHCb2SumSrv	<input type="checkbox"/>	11 LogSumDefSrv	/LogServer.sh	--no-authentication -K 1...	LogSumDefSrv	-I 3 -S 100 -s fmc -p /run...
LogLHCb2	<input type="checkbox"/>	12 LogDef	/LogCollect.sh	--no-authentication -K 1...	LogDef	-I 3 -s fmc -o /run/fmc/...
LogLHCbASumSrv	<input checked="" type="checkbox"/>	13 ROpublish	/MONPublish.sh	--no-authentication -K 1...	ROpublish	-statDelay=8000 -mbm...
LogLHCbA	<input type="checkbox"/>	14 TaskSupervisor	/TaskSupervisor.sh	--no-authentication -K 1...	TaskSupervisor	
LogPVSSrv_HLT	<input type="checkbox"/>	15 webDID	/webDID.sh	--no-authentication -K 1...	webDID	
LogPVSS_HLT	<input type="checkbox"/>	16 PropertyServer	/PropertyServer.sh	--no-authentication -K 1...	PropertyServer	
HLTEQBridge	<input type="checkbox"/>	17 LogGaudiSumSrv	/LogServer.sh	--no-authentication -K 1...	LogGaudiSumSrv	-I 1 -S 100 -s gaudi -p /r...
HLTTThrColl	<input type="checkbox"/>	18 LogGaudi	/LogCollect.sh	--no-authentication -K 1...	LogGaudi	-I 1 -s gaudi -o /run/fmc...
	<input type="checkbox"/>	19 LogLHCb1SumSrv	/LogServer.sh	--no-authentication -K 1...	LogLHCb1SumSrv	-I 1 -S 100 -s lhcb1 -p /r...
	<input type="checkbox"/>	20 LogLHCb1	/LogCollect.sh	--no-authentication -K 1...	LogLHCb1	-I 1 -s lhcb1 -o /run/fmc/...
	<input checked="" type="checkbox"/>	21 LogLHCb2SumSrv	/LogServer.sh	--no-authentication -K 1...	LogLHCb2SumSrv	-I 1 -S 100 -s lhcb2 -p /r...
	<input type="checkbox"/>	22 LogLHCb2	/LogCollect.sh	--no-authentication -K 1...	LogLHCb2	-I 1 -s lhcb2 -o /run/fmc/...

Selected item's description

ROpublish: This is an incredibly important task! It is also important to keep this important task as important as it is.

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 Derived applications

The derived applications are all of the programs that do not qualify as back-end (server side) or front-end (client side) solutions but still utilize the data provided by the database, through the Main API interface.

6.1 New boot script

To fully recreate the functionality of the previous solution, a new boot script had been written. As a modular structure of the system has been introduced, the boot script is no longer responsible for 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).

6.2 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.

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.

Those actions apply 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.

7 Moritz Karbach summer student prize 2018 winner

This summer student project has been awarded with 2018 Moritz Karbach prize which is a collaboration award 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 sum up their work in CERN (after roughly two out of three months of their stay). 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 \(LINK\)](#)

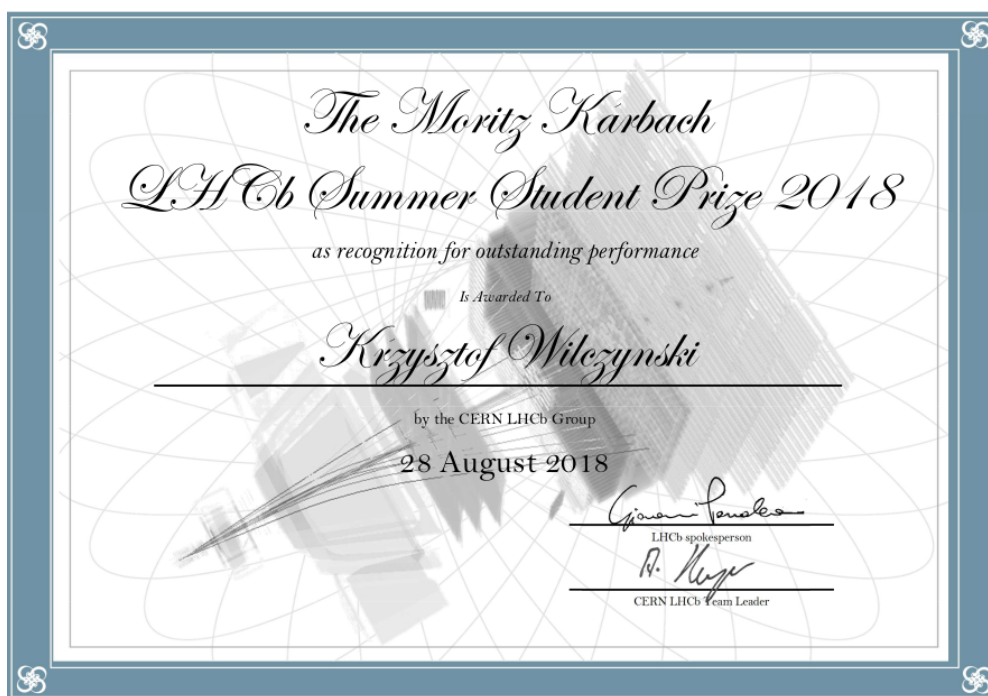


Figure 4: The Moritz Karbach prize diploma

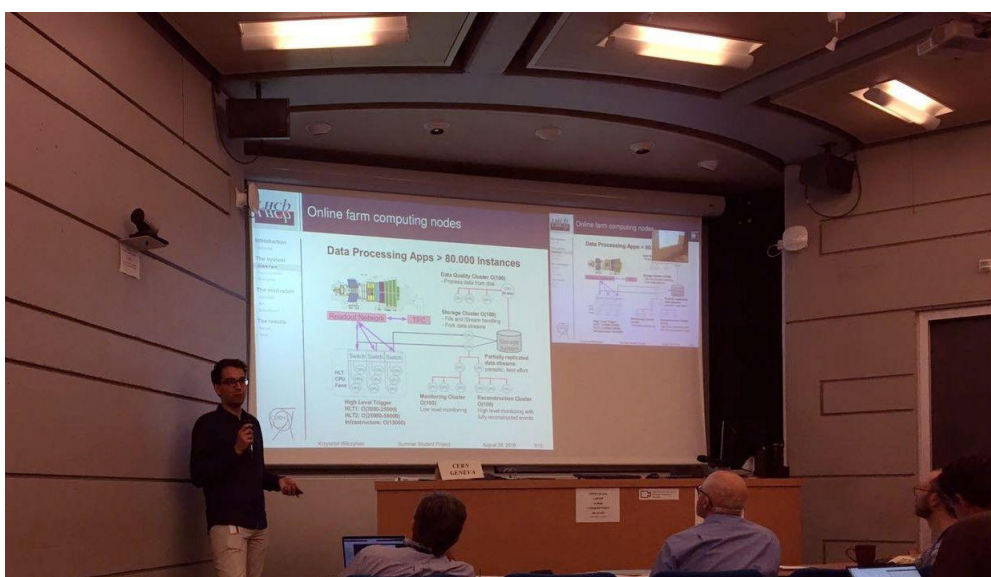


Figure 5: A photo from the presentations session

8 Summary

8.1 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. The official, production repository can be found here:

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

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

Thank You, CERN!