DEVELOPMENT OF A NEW SYSTEM FOR DETAILED LHC FILLING DIAGNOSTICS AND STATISTICS

A. Calia, K. Fuchsberger, G.H. Hemelsoet, D. Jacquet, CERN, Geneva, Switzerland

Abstract

In the CERN accelerator complex the Super Proton Synchrotron (SPS) is used as injector of the Large Hadron Collider (LHC). Statistics on the injection and beam availability in 2015 showed that too much time is spent at injection. Reducing this time could improve LHC availability and luminosity over the year. Currently, useful data to diagnose the problems is sparse and shown in different applications. Operators time is wasted in debugging and checking for the source of the problem before trying another injection. A new Software application for diagnostics of the LHC Filling is under development which collects data from multiple inputs of the CERN Control System and concentrates them in one central view. The inputs are processed and matched with a set of rules (or assertions) that need to be fulfilled for an injection to be successful. Whenever a problem occurs, the operator can check the Filling Diagnostic for hints on what is the source of the problem during the injection. Filling Diagnostic also produces statistics of the LHC injections and the causes of failed injections, this will allow significantly better analysis of the LHC performance for next year.

MOTIVATION

In order to optimize integrated luminosity for the LHC, the time for the turnaround (time between end of stable beams and the next start of stable beams) has to be minimized. A big part of the turnaround time is spent at injection which is still the least reproducible and least automated part. One reason for the variation in time is the necessity for manual measurements with pilot beam before nominal beam can be injected. A second period where time is lost, is when taking nominal beam from the injectors. In the ideal case, one injection would simply follow each other with a time difference of about 40 s between them, as given by the length of the cycle of the Super Proton Synchrotron (SPS), the direct injector of the LHC. However, there are various reasons why injections can fail, for example:

- An interlock can appear which prevents injection (e.g. coming from a power converter after steering the transfer line).
- No beam is produced in the injectors (e.g. because of a failure of some equipment or because the timing system, which is responsible for transporting the injection request to the injectors, does not accept the request).
- The beam is considered as 'not good enough' to be injected and is blocked by a system called the SPS Beam Quality Monitor (BQM).
- An operational mistake can also trigger an interlock. Classical examples for this category are forgotten operational switches or overlooked latching Injection Quality Checks (IQC).

Some of these problems are easy to find (e.g. forgotten switches), others are much harder to diagnose, especially for people on shift with less experience. In any case, each time something like this happens at least one SPS cycle is lost because the problem only materializes (and becomes diagnosable) when an attempt is made.

Several attempts were made to quantify the time lost and the reasons for it. For example, Fig. 1 shows a comparison of the required number of injections, the received injection events and the number of injections which could have been done in the time spent. It was shown that on average about 50 % more injections could have taken place than actually took place within the timespan of filling [1].

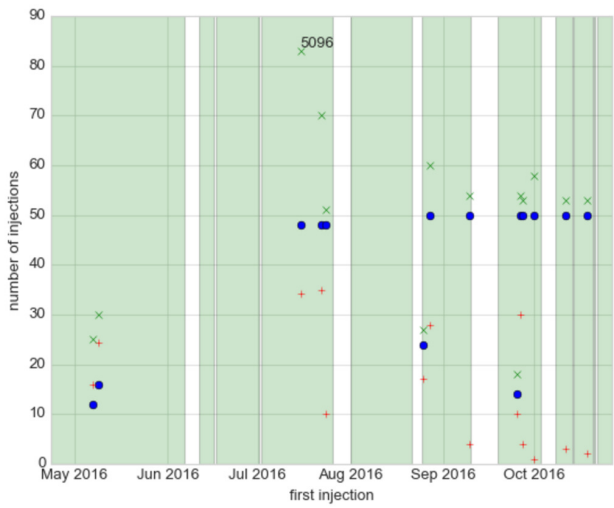

Figure 1: Number of injections for clean turnarounds (No fault tracked during the turnaround). Blue dots indicate the required number of injections, green crosses the number of injection events that happened and red crosses the estimated number of injections that could have been done more than the required ones in the time period spent for filling.

Besides these rough estimations, a more detailed analysis remains difficult because the required data is partly not logged and partly not diagnosable automatically at the moment.

To improve this situation, the development of a new system was launched which finally should solve both described shortcomings: improving on-line diagnostics (and thus shorten reaction time) and storing data for long-term statistical analysis.

ISBN 978-3-95450-182-3

DATA SOURCES

To gather all the information and to extract meaningful information out of it, the filling diagnostics system has to collect information from many different systems and combine them with certain rules. Currently, the system uses data from the following sources:

- The **Injection Sequencer** is the central place that orchestrates the injection process. It has the knowledge about the filling scheme which should finally be in the machine and sends out the injection requests to the timing system.
- A subsystem of the **timing system** (CBCM) accepts (or does not accept) the request, depending e.g. if the beam can currently be produced in the injectors or not. If the request is accepted then this system forwards the request so that the beam is produced in the injectors.
- The injection **Beam Interlock Controllers** (BICs) can potentially prevent the injection, based on different inputs.
- The **LHC Software Interlock System** has a big internal logic tree which can prevent injection.
- The **SPS Software Interlock System** decides through several logic trees if the beam can be safely extracted to the LHC.
- The **SPS Beam Quality Monitor** (BQM) decides if the beam was "good enough" to be sent to the LHC.
- The **LHC Injection Quality Check** (IQC) can detect and bad injections and prevent future injections (if not unlatched by the operators).

ANALYSIS STRATEGY

The analysis of the aforementioned inputs is declaratively described in two analysis modules, one for each ring of the LHC. Each analysis module consists of several so-called assertions. Each of these assertions represents one condition to be evaluated.

At each evaluation, every assertion will be resolved into one of four possible states with the following meaning:

- **SUCCESSFUL:** The underlaying condition evaluated to `true`.
- **FAILED:** The underlaying condition evaluated to `false`.
- **ERROR:** An error occurred during the evaluation, so that the state of this assertion could not be determined.
- **NONAPPLICABLE:** The state of assertion is irrelevant in the given context. For example, conditions referring to ion beams would potentially indicate FAILED during the proton run, they are irrelevant in this case and can be masked out by this mechanism.

Only if all of the assertions are SUCCESSFUL or NONAPPLICABLE, then the whole analysis is considered as SUCCESSFUL.

It has to be noted, that as long as the conditions do not cover all potential failure scenarios, a successful analysis does still not mean that beam is injected. While some scenarios might be practically impossible to cover, it is of course the goal in the longer term to get close to a perfect prediction.

A particular challenging part of the whole project turned out to be the alignment of the different inputs and choosing the right moment of the analysis trigger. For example, the BIC interlocks are published immediately, while SPS SIS is published a few seconds later, at the end of the SPS cycle.

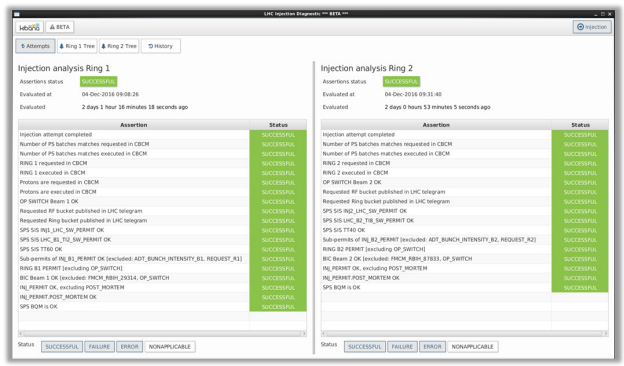

Figure 2: screenshot of the new filling diagnostics application.

The result of the analysis is displayed to the user through a dedicated Graphical User Interface (GUI). Figure 2 shows a screenshot of this GUI, displaying the actually evaluated conditions. It displays the result of the analysis modules for both beams next to each other. The displayed results can be filtered by the user and are kept until the next injection. Dedicated views can visualize the underlaying expression tree and also keep a history of previous injections.

ARCHITECTURE AND FRAMEWORKS

To cope with the challenges described in the previous section, it was essential to choose the right technologies for the task at hand.

Reactive Streams [2] were selected as the way of retrieving data from the devices of the control system. The key advantages of Reactive Streams is the fully asynchronous data processing and the ability to handle backpressure, which is an vital feature when combining slow and fast devices. In order to manage the lifecycle of the required Reactive Streams a new library has been created, Streaming Pool [3, 4]. This library abstracts the management and the creation of Reactive Streams and, with proper extensions, the access to LHC devices.

Once the stream of data are established, they are analyzed and compared with predefined conditions, in order to determine the result of the analysis. All the conditions are specified using the Tensorics Expressions library [5, 6]. A

Tensoric Expression is a node in a tree of expressions that can either be resolved by a Tensorics Resolver (custom logic) or can contain a single, resolved, value. Resolving the tree means walking from the leaf nodes to the root and resolving all the, unresolved, nodes in the way. In the use case of the LHC Filling Diagnostic, the root of the tree represents the final condition of the analysis and the leaf the data coming from the LHC control system. In this context, resolving a node means checking that a specific signal from the controls system matches the desired value.

The combination of Streaming Pool and Tensorics Expressions produces a powerful on-line analysis framework that can be adapted to any use case.

The analysis modules are expressed using a Java internal domain specific language (DSL). Listing 1 shows an example of the description of an assertion within an analysis module.

Listing 1: Assertion example within and analysis module.

```
assertThat(INJ1_LHC_SW_PERMIT).
   isEqualTo(true).
   withName("SPS SIS INJ1_LHC_SW_PERMIT OK");
```

In this particular example, it is checked that a particular permit (`INJ1_LHC_SW_PERMIT`) is `true` at the time of the evaluation of the module. The last part (`withName`) configures the text under which this assertion will appear in the GUI. The usage of a DSL makes it easy to to understand and to extend the behavior of the modules, even at some time in the future.

FIRST EXPERIENCE AND OUTLOOK

A beta version of the system became operational only a few weeks before the end of the run in 2016. Already then it showed promising analysis results. The focus of this first version was to get a first set of assertions in place and to build a framework which makes it easy to add more conditions and input signals during the run.

Development of this system was paused during the long shutdown because of other priorities relevant for the LHC restart. After the startup, the LHC software team will be able to refocus on this system. The first goal is to make logging functionality operational so that the data can be analyzed afterwards and new strategies can be derived from it.

Later, more conditions are planned to be put in place, based on the actual experience. A strong focus will also be to disentangle more and more conditions which depend on each other and mask out more and more redundant information, so that it becomes clear at one glance what is the root cause of a failed injection, so that it becomes easier for the operations team to take the right decisions quickly.

Some effort also will have to go into stabilizing the underlaying frameworks and making them easily reusable in other contexts. Even some more advanced concepts (like machine learning, for example) could be considered in the long term.

CONCLUSION

A new diagnostics system for the LHC filling phase was put in place in 2016 and first version of it was available towards the end of the run. Despite being still of limited functionality, this first version proved that the applied concepts were capable of producing the desired functionality. The underlaying frameworks were improved along the development and will be reusable for many other purposes. Some more work is planned for 2017 to make this system a key tool of LHC operations.

ACKNOWLEDGMENT

The authors would like to thank the whole LHC operations team for their common voice which triggered this project and for their continuous feedback on the implementation. Further a lot of thanks goes to the BE-CO-APS and the TE-MPE-MS section for all the past and future collaborations for software developments.

REFERENCES

[1] K. Fuchsberger, “Turnaround and precycle: analysis and improvements”, proceedings of 7th Evian workshop, CERN, Switzerland (2016).

[2] http://www.reactive-streams.org

[3] A.Calia et al, “Streaming Pool - Managing Long-Living Reactive Streams for Java”, ICALEPCS’17, Barcelona, Spain (2017).

[4] https://streamingpool.github.io

[5] K. Fuchsberger et al, “A framework for online analysis based on Tensorics Expressions & Streaming Pool”, ICALEPCS’17, Barcelona, Spain (2017).

[6] https://tensorics.github.io

ISBN 978-3-95450-182-3