European Organization for Nuclear Research

Technology Department

Machine Protection and Electrical Integrity Group

Magnet Powering Interlocks & Software

Summer Student Program Report

«Using Akka Platform in Unidentified Falling Object Detection on the LHC»

Evangelos Motesnitsalis

Supervisors:

Jean-Christophe Garnier

Kajetan Fuchsberger

August, 2013

Abstract

During my participation in the CERN Summer Student Program 2013, I worked under the Technology Department of CERN and, more specifically, in the Machine Protection and Electrical Integrity (MPE) Group. The MPE Group supports LHC operation and maintains state-of-the art technology for magnet circuit protection and interlock systems for the present and future accelerators, magnet test facilities and CERN hosted experiments.

Within this context, we developed an application that parallelizes the Unidentified Falling Object Detection Algorithm on the LHC Operational Data Analysis Software. For this reason, we used a JVM-based toolkit, named Akka, which parallelizes the execution by creating a number of actors that run simultaneously.

The results of the new approach are presented on the last part of this report. They tend to be quite interesting and promising as we managed to reduce the execution time of the analysis by a factor of 10 on a local machine and the first attempts to execute the program on a cluster have already reduced the execution time by 96%.

Table of Contents

1st Chapter – Theoretical Aspects

1.1 Unidentified Falling Objects

One of the major known limitations for the performance of the Large Hadron Collider is called UFOs (Unidentified Falling Objects). UFOs were observed for the first time in July 2010 and since then they have caused numerous protection beam dumps. Their creation mechanism is still uncertain. However, they are thought to be micrometer sized macroparticles which lead to fast beam losses when they interact with the proton beam with a duration of about 10 turns. [1] The reason for their significant impact on the beam is simple. While the proton beam circulates the LHC ring at around 11.000 times every second and due to the fact that the macroparticle does not have enough speed to go out of the beam orbit, the beam has enough time to interact with the macroparticle several times and increase the number of the protons that are lost with every interaction.

Even though most of the UFO events are much below the Beam Loss Monitor dump thresholds, they can become significantly dangerous when they happen at high energies. At that point the collision between the macroparticle and the proton beam has a much higher impact and can lead to even faster beam losses. Between July 2010 and August 2011, 35 LHC fills in total were terminated by a protection beam dump due to localized fast beam loss events with beam loss events with similar and characteristic beam loss profiles: the temporal loss profile is typically of Gaussian shape with a width of a few LHC turns. Such events were observed in the whole machine and for both beams. Figure 1 shows the temporal loss profile of a typical event.

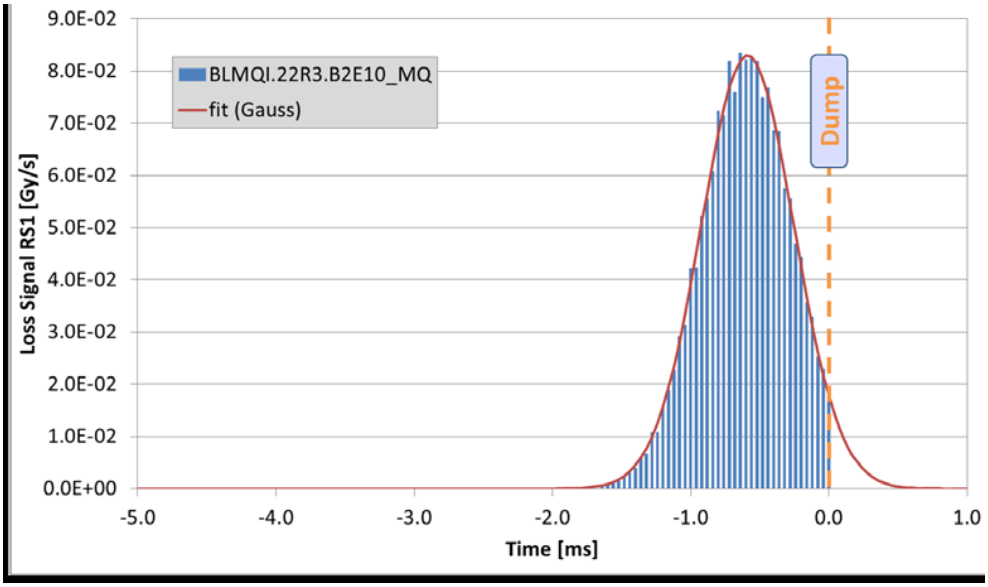

Figure 1 Temporal loss profile of a fast beam loss event on 23.08.2010. The beam loss in the signal exceeded the corresponding threshold and caused the beam dump. The loss occurred on beam 2 in the arc of sector 34. The temporal width of the Gaussian fit is 335 μs (=approximately 3.8 turns).

(extracted from [1] "UFOS in the LHC", Baer et al)

1.2 Beam Loss Monitors and the LHC Logging Database

When a UFO event appears inside the LHC, it is recorded by the Beam Loss Monitors (BLMs). By definition, BLMs are the system that can measure beam losses in the LHC Ring. There are many types of BLMs inside the LHC Ring with the two most common ones named Ionization Chambers and Diamond Beam Loss Monitors. There are more than 4000 Ionization Chambers inside the LHC and there is a significant amount of Diamond BLMs as well.

On a normal state, a BLM produces a zero signal. However, when a UFO event takes place, the protons that are moved out of the beam force the BLM to produce a signal that is proportional to the loss. In case the loss on a BLM is higher than the safety thresholds, the system triggers a beam dump.

All the data that these BLMs produce are constantly stored in the LHC Logging Database. The LHC Logging Database is an Oracle Database that consists of several modules and it stores almost 50 TB of operational (and not physics) data per year. It also deals with more than 40 million queries per day. The data are extracted via a Java API which has limitations on the number of simultaneous extractions per application and, until recently, it only allowed serialized extraction. The Logging Database contains relatively slow time series data that are stored online for at least the lifetime of the LHC and its data are considered useful for long term use even though they could be reconstructed from raw measure data.

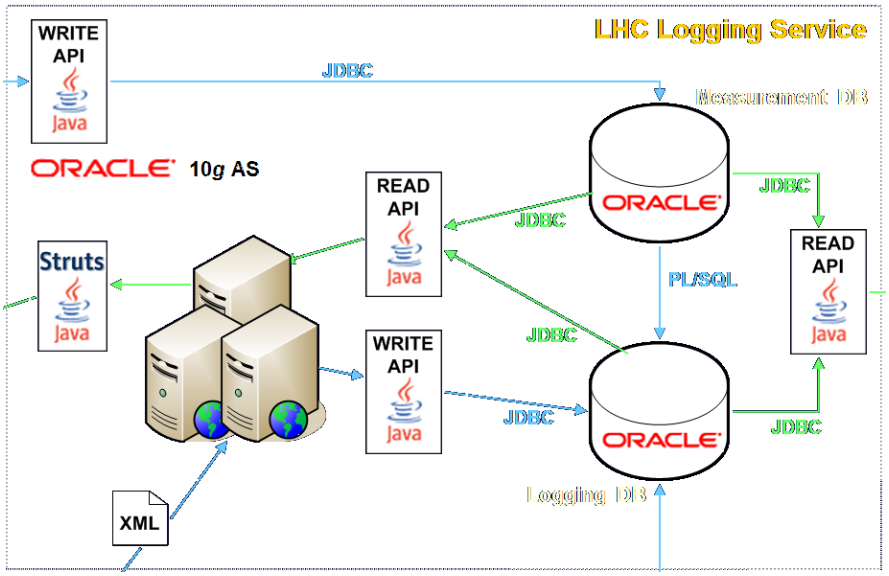

Figure 2 The LHC Logging Database Components

(extracted from [4] "The LHC Logging Service" by Chris Roderick)

1.3 Akka and Actors

Akka is a framework for developing distributed, concurrent, scalable, and fault tolerant applications on the Java Virtual Machine. For fault-tolerance it adopts the "Let it crash" model that self-heal and allows systems to never stop. It consists of

an actor model that raises the abstraction level and provides a better platform to build correct and scalable applications. Akka is adopted by many large organizations in a big range of industries: Investment and Merchant Banking, Retail, Social Media, Simulation, Gaming and Betting, Automobile and Traffic Systems, Health Care, Data Analytics and much more.

The actors of Akka are very lightweight concurrent object-oriented entities which process messages asynchronously using an event-driven receive loop. They are constrained only by memory of which they consume a few hundred bytes each. Thus, any application can easily create millions of actors. In this way, the programmers focus on workflow and how the messages flow in the system instead of low level primitives like threads, locks and socket I/O.

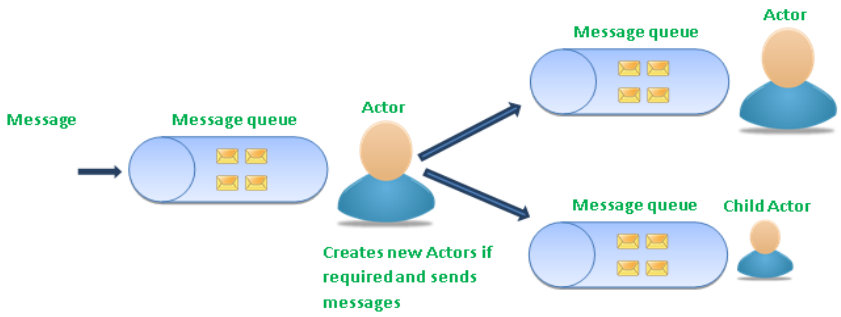

Figure 3 Actors and Messages in the Akka Platform

Actors form a hierarchical tree with actors being parents to the actors they've created. The communication between them is done via messages that can be of arbitrary type such as String, Integer, Boolean, arrays, collection etc. Messages should be defined with rich semantic and domain specific meaning. They should also be serializable and immutable. If they are not, there is a risk of accidentally sharing state between two different Actors which will violate the Actor Model.

The asynchronous communications that is done with the messages is what makes actors reactive and event driven. As a parent, the actor is responsible for handling its children's failures, forming a chain of responsibility, all the way to the top. As a result, when an actor crashes, its parent can either restart or stop it, or escalate the failure up the hierarchy of actors. This model enables a clean set of semantics for managing failures in a concurrent, distributed system and allows writing highly fault-tolerant systems that self-heal.

2nd Chapter - Akka Implementation

The TE-MPE group is constantly seeking new technologies for the Analysis of the LHC Operational Data. In this context, we developed the following use case scenario to test the Akka toolkit and its performance on the UFO Detection Algorithm. Since Akka is a parallelization platform, we expect that migrating to this environment may offer:

- Increased scalability
- Increased reliability
- Increased fault-tolerance
- Increased decentralization
- Reduced execution time

2.1 Previous Approach

Before the Akka parallelized approach, the UFO detection was done in a serial way. More specifically, for every second of the time range under analysis, the program communicated with the LHC Logging Database, extracted the part of the data wanted, and then proceeded to the threshold checking.

In this implementation the execution time was equal to the time range multiplied by a factor of ~5.1. For instance, a 5 minutes analysis in the Logging Database BLM signals needed more than 26 minutes to be completed. However, no previous effort was put on optimizing the algorithm since we only used it as a use case at this point.

A known bottleneck of this approach was the Logging Database. On our test system, a single 1 second extraction needed more than 5 seconds to be completed. Therefore, it was not only the threshold checking that needed parallelization but also the communication with the Logging Database.

2.2. The Akka Approach

The previous approach was implemented in such a way that parallelization was an easy migration. As a first trial, we created as many actors as the seconds under analysis. Therefore, every actor of Akka takes the data of one second, performs the threshold checking on its own and returns the results.

Actors are very light-weight entities and need no more of 300 bytes to be saved in memory. This proved to be a very good factor of Akka. As a result, we are able to analyse millions of seconds without problems even on a single machine. For example, since every second is dedicated to one actor and we can have more than 2.5 million of actors per GB of heap, we need only one GB of memory to analyse 2.5 million seconds (almost one month of data).

The first executions of the parallelized version were unexpectedly disappointing. We noticed that the execution was reduced only by 9% for a 5 minutes data analysis. Thus, while the Serial approach consumed ~1630 seconds for a 5 minutes data analysis, the first Akka parallelized version consumed ~1490 seconds for the same data.

Further analysis of the code proved that the Logging Database extraction was still done in a serial way. The reason for this was the existence of multiple blocking calls on the extraction code which did not allow simultaneous communications with the database. Nevertheless, replacing the jar files of the logging extraction by a newer version allowed the Akka actors to accomplish simultaneous extractions. The results of the final Akka approach are discussed in the next chapter of this report.

3rd Chapter - Results

Apart from the decentralization and the fault-tolerance that would be offered by Akka in either case, the reduction of the execution time proved to be quite momentous. Experiments showed that the parallelized version was executed more than 10 times faster than the serial approach. In Figure 4 we present the execution times of the Akka approach for 4 sets of BLM data. The size for the first chunk is 30 seconds while for the second, the third and the fourth we have one, two and three hours of data, respectively.

It is obvious that our test implementation is able to complete with its execution time equal to less than the half of the time range that is analysed each time. However, we noticed that for small parts of BLM data (below one minute) we experience a small bottleneck due to the actor creation loop that increases the duration to almost 70% of the time range under investigation. In terms of comparison to the serial execution, analysing parts of data that contain more than 10 minutes of BLM data in a serial way tends to be inhibitive. The chart indicates that 5 minutes of data analysed in a serial way need the same amount of time to be executed with a parallelized execution of 60 minutes data. This fact means that for the same amount of data, Akka needs only 8% of the Serial execution time to complete its task.

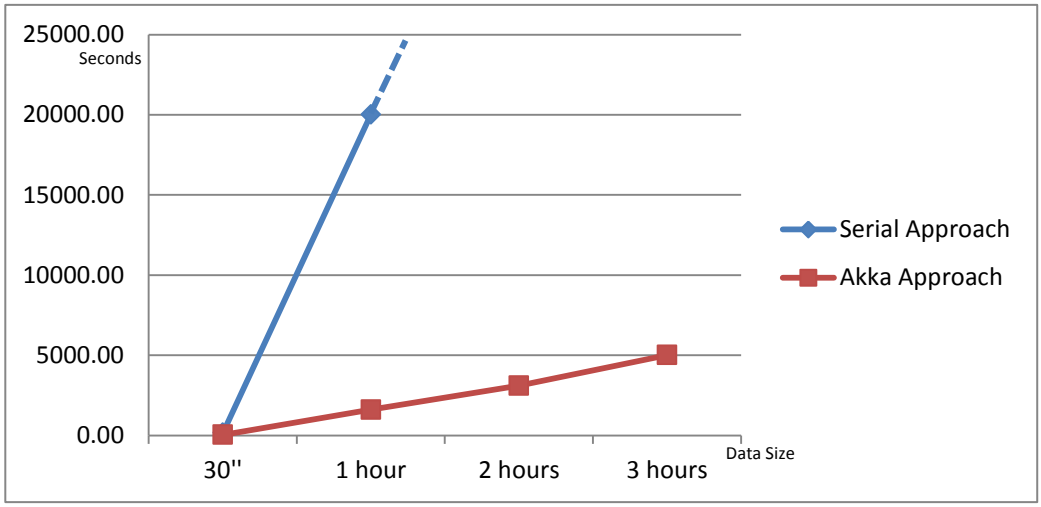

Figure 4 Time Comparison for the Serial and the Parallel Approach

4th Chapter – Discussion/Conclusion

The hypothesis that the parallel version of the UFO Detection Algorithm would significantly reduce the execution time is supported by the experimental data. The reduction factor is equal to 92% of the serial execution time. We expect that by running the new approach on a cluster with several nodes, this reduction factor will increase even more with the Logging Database Service to be the final limiting factor.

Results show that the Akka Toolkit is an excellent environment to build elastic and decentralized applications. During the execution, we noticed that even when an actor fails to complete its task, the overall system performance is insignificantly affected. The fault-tolerance that Akka provides is an essential need for every software that runs for the LHC accelerator.

In conclusion, the Akka high-level abstraction and hierarchy proved to grant a stable and efficient way to analyse crucial data on the LHC. Since the first results were quite prosperous, we are convinced that a more systematic use for the Analysis of the LHC Operational Data will boost its performance drastically.

Time Execution Comparisons					
Time Range	From 2012-12-13 15:34:30 to 2012-12-13 15:35:00	From 2012-12-13 15:00:00 to 2012-12-13 16:00:00	From 2012-12-13 16:00:00 to 2012-12-13 18:00:00	From 2012-12-13 12:00:00 to 2012-12-13 15:00:00	From 2012-12-13 12:00:00 To 2012-12-13 18:00:00
Duration	30 seconds	1 hour	2 hours	3 hours	6 hours
Serial Approach	2m37s	5h33m20s	---	---	---
Akka Approach	22s	26m48s	51m45s	1h23m33s	2h42m9s

Bibliography

Printed References

[1]. T. Baer (CERN, Switzerland and University of Hamburg, Germany), M. Barnes, B. Goddard, E. B. Holzer, J. M. Jimenez, A. Lechner, V. Mertens, E. Nebot Del Busto, A. Nordt, J. Uythoven, B. Velghe, J. Wenninger, F. Zimmermann (CERN, Switzerland). UFOs in the LHC. Proceedings of IPAC2011, San Sebastián, Spain, pages 1347-1349.

[2]. T. Baer (CERN, Switzerland and University of Hamburg, Germany), M.J. Barnes, E. Carlier, F. Cerutti, B. Dehning, L. Ducimetiere, A. Ferrari, N. Garrel, A. Gerardin, B. Goddard, E.B. Holzer, S. Jackson, J.M. Jimenez, V. Kain, A. Lechner, V. Mertens, M. Misiowiec, R. Moron Ballester, E. Nebot del Busto, L. Norderhaug Drosdal, A. Nordt, J. Uythoven, B. Velghe, V. Vlachoudis, J. Wenninger, C. Zamantzas, F. Zimmermann (CERN, Switzerland), N. Fuster Martinez (University of Valencia, Spain). UFOs in the LHC after LS1. Proceedings of Chamonix 2012 workshop on LHC Performance, pages 294-298.

[3]. Maria Hempel. Application of Diamond Based Beam Loss Monitors at LHC (Doctoral dissertation), April 4, 2013.

[4]. Chrid Roderick. The LHC Logging Service. UKOUG Conference 2006, Birmingham.

Online References

[1]. Wiki on previous UFO Detection Approach

http://wikis/display/MPESC/MapReduce

[2]. Wiki on LHC Analysis Software

http://wikis/display/ACCTEST/AnalysisFramework

[3]. Akka Documentation

http://doc.akka.io/docs/akka/2.2.0/java.html

Appendix

Akka UFO Search Pseudo Code

1. Fetch Data from the Logging Database;
2. Import the positions of the BLMs;
3. Import the Running Sums and their Thresholds;
4. Define the Time Range for the Search;
5. Search for UFOS:
   For every second
   Create an actor who will:
   Actor: 5.1. Fetch the data for its second;
   Actor: 5.2. If all the Thresholds in the Running Sums are passed;
   Actor: 5.3. Add the UFO to a List;
6. Print the List to the Logging Output;