

AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY

Faculty of Computer Science, Electronics and Telecommunications



BACHELOR THESIS

**INTERACTIVE DATA ANALYSIS OF DATA FROM HIGH
ENERGY PHYSICS EXPERIMENTS USING APACHE SPARK**

INTERAKTYWNA ANALIZA DANYCH Z EKSPERYMENTÓW FIZYKI
WYSOKICH ENERGII Z UŻYCIEM APACHE SPARK

MIŁOSZ BŁASZKIEWICZ, ALEKSANDRA MNICH

FIELD OF STUDY:
Computer Science

SUPERVISOR:
dr hab. inż. Maciej Malawski

Kraków, 2019

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście, samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

.....

PODPIS

Acknowledgements

First and foremost, we want to thank our excellent supervisor and mentor, Maciej Malawski, for his wonderful support and help. His insight, commitment and knowledge, along the willingness to share it with us, was a huge assistance. We sincerely appreciate it.

We want to express our deepest gratitude to the members of the CERN team for a warm welcome, excellent guidance and incredible cooperation. We do believe that our joint work will contribute to the development of computing methods and resources, and thus will prove to be useful for scientists using them every day.

We are very grateful to Leszek Grzanka, Valentina Avati, Jakub Mościcki, Enrico Bocchi, Luca Canali, Javier Cervantes, Jan Kaspar, Prasanth Kothuri, Enric Tejedor, Simone Giani, Danilo Piparo and TOTEM scientists.

This work is supported by the Polish Ministry of Science and Higher Education Grant no. MNiSWDIR/WK/2017/07-01, by PLGrid Infrastructure and HNSciCloud, co-funded by the European Commission under grant 687614.

Contents

1	Project goals and vision	7
1.1	Data analysis challenges in TOTEM experiment	7
1.2	Project motivation	8
1.3	State-of-the-Art	9
1.4	Product proposal	10
1.5	Feasibility studies	11
1.5.1	ROOT files in Spark	12
1.5.2	Jupyter and Zeppelin	13
1.5.3	Results visualization	13
1.6	Risks	13
1.7	Glossary	14
2	Functionality and scope of the project	16
2.1	Users context	16
2.2	External systems	16
2.3	Features of the distributed analysis	18
2.4	The scope of the project	19
2.5	Usage and testing scenarios	19
2.5.1	Typical usage	19
2.5.2	Testing	20
3	Development of the Distributed Analysis	22
3.1	Structure of the analysis application	22
3.2	Technological solutions used in the project	23
3.2.1	Apache Spark	23
3.2.2	Jupyter Notebook	23
3.2.3	ROOT Framework in the project	24
3.2.4	Tools and technologies used for profiling purposes	25
3.3	Major issues encountered during the development and profiling and developed solutions	25
3.3.1	Weighted histograms	25
3.3.2	Data partitioning options	27
3.3.3	Tests with extended resources on Helix Nebula Cloud	31
3.4	Profiling and testing	34
3.5	TOTEM analysis in Scala	35
4	Work organization	40
4.1	Team structure and general work organization overview	40
4.2	Project phases	41
4.2.1	February to March	41
4.2.2	April to June	41
4.2.3	July to September	41

4.2.4	October to December	42
4.3	Project management practices and tools	42
5	Project results	45
5.1	TOTEM analysis with RDataFrame in a Jupyter Notebook	45
5.2	Profiling and testing conclusions	46
5.3	TOTEM Analysis in Scala	47
5.4	Conclusions	48
	Appendices	50
A	Abstract of the poster paper for 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)	50
B	Poster for 2018 IEEE/ACM International Conference on Utility and Cloud Computing	53
C	Selected reports from the test runs in November and December	55
D	Presentation "Distributed data analysis for the TOTEM experiment. Collaboration Meeting Report", 11 September 2018	71
E	Selected presentations from the weekly meetings in TOTEM	82
F	Selected tables	94
F.1	Tests performed on Helix Nebula Cloud in November	94
F.2	Tests performed on Helix Nebula Cloud in December reusing executors	95
F.3	Tests performed on Helix Nebula Cloud in December restarting executors	96
F.4	Tests performed on Helix Nebula Cloud in December with varying size of input	97
F.5	Duration of the DistTree initialization	98
F.6	Tests using local machine using local storage (SSD)	100
F.7	Tests using local machine using remote storage	101
F.8	Tests of Scala version using Prometheus cluster	102

List of Figures

1	CMS detector © 2017 Brice, Maximilien: CERN	7
2	CERN Computing Center © 2012 Brice, Maximilien: CERN	9
3	Helix Nebula architecture	17
4	SWAN web interface (left) and Jupyter notebook (right)	18
5	General pattern of the usage scenario	20
6	Map Reduce pattern in the analysis application	22
7	Discrepancies in histograms	26
8	Differences in the entries numbers for the weighted histograms after distributed execution	27
9	Execution time for different numbers of partitons used with maximum 128 cores on 6 nodes	28
10	Approximate shape of the speed-up curve encountered at the initial stages of the Spark introduction to the project	29
11	CPU usage and kilobytes read from SSD in time of application execution	30
12	DistROOT - a mechanism responsible for assigning entries to tasks in two variants: original (left) and modified (right)	30
13	Speed-up vs partitions chart for the DistROOT dividing by files	31
14	Execution time and speedup acquired using extended resources in December tests	32
15	Comparision of execution time and speedup without "reuse" option in December tests	33
16	Average execution time to input data size	33
17	Execution time acquired by distributing the computations	34
18	Speed-up acquired by distributing the computations	34
19	Comparision of efficiency of parallelization with and without executors restarting	35
20	Comparision of 'Rate CMP' histogram produced by application written in Scala and the same histogram from original analysis in C++	36
21	Summary Metrics of Scala application (from SparkUI)	36
22	Scalability tests of analysis written in Scala performed on Prometheus cluster - one Spark executor per node (24 cores and 120GB RAM per node), all DSs except DS5 as input	37
23	Histogrammar labeling structure, which computes histograms in a single pass	37
24	Execution time for Scala version on Prometheus cluster with DS1 (90GB) as input	38
25	Spark console	38
26	Jupyter notebook which is the basic result of the project.	45
27	Three execution modes of the RDataFrame application	46
28	Comparision of the histograms resulting from the original analysis and the one recreated with RDataFrame	47

1. Project goals and vision

This chapter is an introduction to data analysis challenges that CERN and its experiments will face in the coming years, an overall summary of the project purposes and motivations, and a review of current approaches to the problem. A brief description of the proposed outcome of the project follows, along with an overview of potential tools, technologies, and risks that might concern the project.

1.1. Data analysis challenges in TOTEM experiment

The European Organisation for Nuclear Research is the largest particle physics laboratory in the world. Scientists, working there every day, try to apprehend a better knowledge and understanding of the nature of the world. Such an objective can be achieved only through precise work supported by exceptional machines and devices. The most remarkable example is the Large Hadron Collider (LHC) which is considered to be the largest machine ever constructed. LHC accelerates the beams of particles and collides them in order to observe the particle trails resulting from those collisions in specially constructed detectors. Each detector observes millions of collisions in a single second, and many signals from pixels inside the detector represent each collision. Amount of data processed at CERN has a unique scale, presenting an enormous challenge to the people who are designing, producing and maintaining an infrastructure which can satisfy the needs of such magnitude.

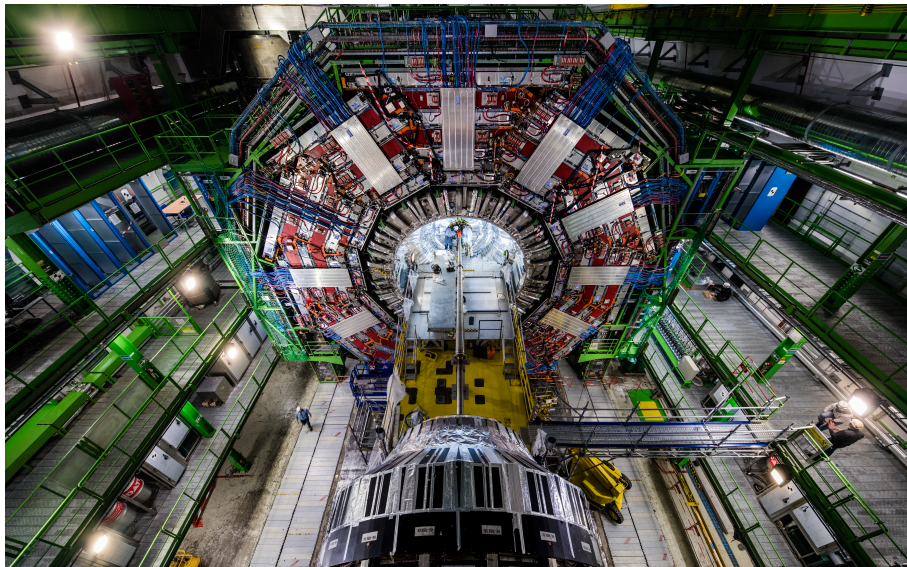


Figure 1: CMS detector © 2017 Brice, Maximilien: CERN

The TOTEM (TOTAL cross section, Elastic scattering and diffraction dissociation Measurement at the LHC) experiment targets forward particle physics, which has unique requirements regarding the beam settings. Contrary to the common LHC experiments, the protons are expected to interact rather than collide. Therefore, location of the experiment's detectors is different from other detectors of the LHC. To be suitable to capture the protons scattered at minimal angles, those devices are positioned approximately 200 meters away from the collision point. This unusual setting is a consequence of the experiment's principal objective.

Raw data from the detectors is a number of signals, which detectors receive shortly after the collision. The first step completed afterward is a reconstruction of the proton tracks inside the net of the detector's pixels. The effects of this operation are stored in a RECO file, which is ready to use in more advanced analysis. In the case of the elastic scattering analysis, which is the case investigated in this thesis, the processing does not end with this step. RECO files can be then converted to the ROOT [33] files, which contain so-called *n-tuples* and this is the form of data which will be used throughout this project.

CERN data is stored in EOS [24] - a disk-based, low-latency storage service developed for scientific purposes. Access to the data is restricted to the members of a given experiment computing group.

Presently, the offline analysis of the data from the detectors is conducted using the CERN Batch Service [19], using sequential applications. Standard operations performed on this data include filtering, applying cuts and generating histograms. To achieve proper results, a person responsible for the analysis have to adjust several parameters for each of the beam runs, and then recalculate the whole model all over again.

In this project, a particular analysis was chosen as an exemplary case to investigate a new model for the TOTEM computations. The deciding factor was the fact that, upon successful completion, the same approach could be easily replicated in other analyses, including those performed right now and in the future, as well. The selected analysis, which fulfills this requirement, is the Elastic Scattering 6500 GeV $\beta^* = 90\text{m}$ 10σ on data from 2015 run. It mainly concerns the proton's elastic scattering and further studies of the theoretical model of the gluons' behavior.

Presently, the analysis is divided into two steps: (1) distillation and (2) distribution.

First, called the distillation step, filters most of the data and keeps only events which have at least 2 out of 3 recorded tracks valid on selected diagonal and have the potential to be the result of the elastic scattering. It is done only once and proved to be a problematic issue from a computational perspective. To perform the distillation of the events, all datasets of more than 200 GB needed to be divided into smaller blocks of 100 GB at most, so that the sequential application and the local cluster could attempt to process this data with predictable success. This step even for DS1, the smallest dataset, took roughly 15 hours.

The second step is repeated many times with different settings and properties. This part involves much more massive calculations than those performed to filter events before. Multiple new columns are defined upon the existing ones, processed as intended by the analyst and summarized as histograms highlighting various aspects of the measured events. In the end, results are delivered and stored as one set of histograms in a ROOT file.

1.2. Project motivation

The primary goal of the project was to evaluate a set of Big Data tools for the analysis of the data from the TOTEM experiment which will enable interactive or semi-interactive work with large amounts of data. Particle physicists working at the TOTEM experiment were working with 4.7 TB of data in 2015. Since that time, in just three years, the amount of data gathered yearly has increased sharply and now may exceed 50 TB. Those numbers will continue to rise, with further upgrades and even more precise devices. CERN is actively seeking new solutions to the



Figure 2: CERN Computing Center © 2012 Brice, Maximilien: CERN

problem with computing power on the organization level. To catch up on these challenges, more advanced IT solutions will be needed.

1.3. State-of-the-Art

Scientists at TOTEM or generally at CERN develop their analysis in the ROOT Data Analysis Framework [33], which is designed especially for the use in High Energy Physics computations. Since its creation, ROOT framework has become a widely accepted solution inside CERN and physics community, yet remained mostly unrecognized outside. The framework offers an extensive variety of the tools intended for the use in physics experiments, statistical analysis, and easy-to-use plotting features. As a solution devised for this particular field, it has several advantages, of which most prominent is a large number of users and thus the active community.

On the other hand, for a long time ROOT had no capabilities to parallelize or distribute the analysis on the cluster easily — these limitations heavily restricted the ability to work with data interactively. RDataFrame, which might resolve partially or entirely the stated issues, was added recently to the framework. This project explores possible application and performance of this new construct in actual physics analysis.

The problem of processing massive volumes of data is quite common in today's data science. One of the most important solutions is a successful, open-source Apache Spark [13] framework. For the last several years, as a part of the Apache Software Foundation and with strong community support, the platform gained a firm position among other big data solutions. Its primary purpose is to enable users to perform a distributed analysis on the cluster, providing scalability and fault tolerance capabilities. This objective is usually achieved quite accurately by the framework itself, yet in the particular application discussed in this work, Spark seems to be not entirely adequate by itself. Prime and most critical factor weakening the use only of the Spark for the analysis of the High Energy Physics data is the fact that it requires all the data to be con-

verted to its native structures beforehand. Additionally, it would be necessary to either abandon the solutions developed in the ROOT framework or port it to the compatible form.

CMS Big Data [21] and Diana HEP [23] teams tackled the problem of the missing method to use Spark with CERN data. Their successful attempt resulted in a connector, which can be used to read TOTEM's n-tuples [36] into the Spark's data frames and process them from this point as any other data. The potential problem is the obligatory conversion of data and complete abandonment of the ROOT framework. The first problem concerns additional time spent on rewriting the same data, and another obstacle is a need for new, physics-oriented tools in a new environment.

A completely different way to distribute the code of the analysis is to use Python's Dask [31]. The idea behind this platform is to add parallelization capabilities to the Python code so that users will be able to take advantage of the scalable performance, without losing the opportunity to write in the well-liked and familiar language at the same time. The framework allows running same analysis at the local machine as well as at the cluster with minimal or no changes to the code. Unfortunately, those benefits come at a certain price. Dask requires the developer to go very deep into technical details of the parallelization. Even though it allows much higher control over the way the parallelization and distribution are working, it also adds another responsibility to the developer, in this case to the physicist. Therefore, this approach was not investigated further, as the number of additional code required for it to run would most probably discourage users from using it.

The proposed solution, discussed at length in the next section, is a combination of the elements from different frameworks and technologies described above. The prospect of combining the advantages of those solutions seems to be very promising, but on the other hand, it might also add unnecessary complexity. Hence, proper validation and performance tests are necessary to prove the correctness and usefulness of the analysis application which is expected to serve as an example of the entire approach.

1.4. Product proposal

The product is a set of the analysis codes and notebooks written in a distributed model, together with their performance profiling, reports, and execution results. Our analysis application has several unalterable requirements to fulfill:

- correctness of the results,
- capacity to work interactively using data-science notebooks,
- scalability of the solution,
- simple and easy way to visualize the results,
- use of existing storage services.

Most typical features of the analysis are the use of the cuts and filters applied to the sets of data. Those operations should be ready, supported outright and possible to apply without any additional work.

Created analyses should be equivalent to the original one written in C++, provided by the TOTEM experiment. The principal requirement for both analyses is that they should give the same results as the original one. This is a primary concern for the users and the team, as substantial differences might lead to the rejection of the whole idea of distributing the analysis.

Data should be obtained directly from EOS (if possible) or can be downloaded to the external storage. The second option was necessary only for the Prometheus cluster and does not pose a considerable problem, as Prometheus is acting only as a testing environment.

The application may or may not preserve the original split of the operations into two steps. The division is reasonable and indeed helpful in this particular case; however, this applies only when the structure of the analysis allows it. Furthermore, it would be good if responsibility for introducing the division was moved to the tools and frameworks and the programmer would not need to implement it manually. Therefore, it would be preferable to not follow this pattern, as combining those steps might allow a higher degree of flexibility for the next analysis developers.

The performance tests and validation of the application are even more significant and demanding part of the project than the initial code of the analysis itself. The underlying reason for the development of this application is to verify the adequacy and advantage of the entire distributed computing model and particular method to do that in the experiment work and to create an example which HEP scientist might use in the future. To achieve those aims, maintaining the highest possible performance at all steps is of utmost importance. This stage of the analysis development took much time and is a large part of the work. The product presented in this project is accompanied by scripts and other auxiliary tools created to validate, measure or visualize the application's performance. Requirements concerning those tools are:

1. Reliability and simplicity.
2. Clear and concise output, tailored to the information sought in a given moment.
3. Should work with popular formats of data so that users are not bound to only a selection of tools.

The goal of the entire project is to investigate a complete and scalable solution, which will enable the scientists to work with the gathered data in a comfortable, interactive environment. If the performance tests result in a positive assessment, the code will serve as an example model of computing that might be used in the experiment. Insights acquired in the course of the project are expected to help with the development of the external tools, as well as help to better determine the right directions in the future.

Alternative version of the analysis, using Spark-ROOT connector and written in Scala, is a part of the product. It is developed in order to get a whole picture of the possibilities for distributing HEP analysis applications.

1.5. Feasibility studies

The overall goal of our project is to evaluate the application of the modern Big Data tools to the data analysis in a particular case. Currently, two key platforms might be listed:

- Apache Spark,

- Hadoop.

Both solutions share a lot of common characteristics, yet substantial differences are present as well. Hadoop consists of several modules, which together constitute the whole framework. Four most important are: Hadoop Common, HDFS, Hadoop YARN, and Hadoop MapReduce. In principle, the comparison should consider only MapReduce module, as Spark does not have the capabilities of the other listed modules.

Spark [13] is a tool created from the Hadoop framework. Some people even consider it to be one of the Hadoop modules, but generally, it works as a standalone platform. It provides the optimization of both, the iterative and the interactive operations, resulting in acceleration even by several orders of magnitude. It is also newer and already more popular.

In this thesis, we will proceed with Spark. This choice was made in the first place because of its established presence of this technology on the available infrastructure. Spark was also found to be more suitable and useful in terms of the needs and expectations out of those two platforms.

Among the advantages of Spark, the most appealing is the speed and the simplicity. The former quality is significant for obvious reasons, and the latter one is crucial for the users. It is very likely that no matter how substantial speedup in terms of the execution time would be, it could still not justify the need for additional time spent on learning and development of the analysis code.

1.5.1. ROOT files in Spark

Data acquired from experiment's detectors and then processed is stored in ROOT files, a format which is part of the ROOT framework. An essential for the Spark working with ROOT files is to provide means which will enable this data to be loaded and usable on the cluster's nodes. At the present moment, two frameworks for this purpose exist: Spark-ROOT connector by Diana HEP team and the DistROOT.

Spark-ROOT allows reading data into native Spark's structures. The project is developed since 2016 and for the moment offers good support of such operations. By using this tool, contents of the ROOT file may be read to the standard DataFrame and then handled like any other data in Spark.

The authors of the DistROOT library took another approach. The primary concept behind this solution is to work with the ROOT files all of the time, instead of converting those files to other formats. Library operates by distributing code and data to the Spark nodes. This idea has some advantages and disadvantages. On the positive side, there is no need to spend time on pre-processing data to convert it to a more suitable format. Furthermore, although the code and the structure of the analysis created in a classic way have to be changed, the change is not so significant, and users can still use the methods and solutions provided by the ROOT framework. The amount of extra knowledge required to run or design analysis for a person familiar with typical ROOT analyses is considerably smaller than in other approaches.

On the other hand, this approach requires to resign from some of the Spark's features. Additionally, this framework is developed for a shorter time than the other one, and thus it may have more limited capabilities.

1.5.2. Jupyter and Zeppelin

A very convenient way to prepare an analysis code is to develop it in an interactive notebook from a web browser rather than in a terminal window. Notebooks offer a comfortable way to split the analysis into meaningful sections, to go over its structure at any time, to explain it with essential paragraphs and, finally, to present charts transparently — no wonder that many data scientists have already moved to webpage-based editors and that in this project we will follow this particular direction.

Nowadays, most popular notebooks solutions are Jupyter Notebooks and Apache Zeppelin. The general opinion seems to suggest that the former one works better with a single machine while the latter operates better with a cluster. Reasons for this belief are probably simple: Zeppelin has built-in integration with Spark (Apache Foundation maintains both products), while Jupyter requires additional plugins making the whole task of distributing computations a little bit tougher.

In this project, we will use the Jupyter Notebooks. Jupyter is a part of the SWAN service and comes already configured for the needs here. Zeppelin does not support the required version of Spark.

1.5.3. Results visualization

An output of TOTEM analysis application is often a set of histograms, which illustrate chosen aspects of the examined data. Not unlike other areas of data science, decent visualization capabilities are here extraordinarily significant.

Many plotting libraries are reliable and widely available. To name a few examples, those are Matplotlib, ggplot or Bokeh. Used in numerous projects and applications, they are already tested by large groups of developers and probably can now be even considered standard-setting solutions in the field. Furthermore, they have a strong base of users and supporters, with multiple manuals and guides, and are quite flexible. Despite these numerous advantages, only one visualization tool will be used in the project, and it is not even similar to the ones listed above.

JsROOT is a plotting engine written in JavaScript, which very closely follows plotting features available in the fully-fledged ROOT framework. Histograms created with this tool not only are similar to the ones to which most of the scientist working with the HEP analyses has become accustomed. It also offers almost the same tools to manipulate displayed histograms as the visualization ROOT framework, including axis scaling, zooming, cutting, recoloring and much more. Combination of those facts makes the JsROOT unbeatable solution in plotting physical data and the reason for which we traded benefits of open-source, well-known plotting frameworks.

1.6. Risks

The project presented in this thesis was exposed to the number of potential risks, which are presented in this chapter. First of all, there are technical issues which may have impeded the work:

1. The project needed to be run and tested in a cluster environment. This requirement made it very dependant on the external systems and their reliability. A threat in this sense was

posed especially by Helix Nebula Science Cloud, which was in the pilot stage during the application development. Clusters continually presented a risk of unexpected behavior, accidental breaks due to failures and stops for maintenance works. A minimizing factor to this risk was the use of more than one distributed platform, and for this, in total, three different and independent from each other platforms were made available to us. Temporary breaks and maintenance works may have caused delays from time to time, but they should not have posed an existential threat to the development.

2. New analysis code relies heavily on the existing frameworks which were in use for a long time by the target users. If those frameworks, e.g., because of their structure, had not allowed us for efficient use of distributed computing methods, the main objective of the project would be compromised. To decrease this risk, we discussed the whole project thoroughly before and prepared possible other ways to take advantage of the prepared analyses and tests.

The critical issue mentioned above discusses the availability of the testing environment. We were prepared to use three testing environments: first is an external commercial platform called Helix Nebula Science Cloud, examined for the suitability for the use at the experiments during the project, second is a regular CERN's cluster, and the last one is Prometheus cluster. While the first two differ in physical machines configuration, they offer the same platform for performing interactive analysis thanks to the SWAN service. Prometheus does not offer the same option, yet there already are devised methods to work with ROOT files, described in the prior year in the thesis [8].

There also may have occurred usual problems of the projects with a similar level of organizational complexity. Possible obstacles on the administrative level included lack of access to data or original code of the analysis, continuously changing objectives, dissolution of the team and problems with the communication within it. To deal with those problems we decided on a transparent way we want to proceed with the project so that we could have completed it independently and regardless of the accompanying obstacles. This way we could pursue our objectives in parallel to the works done with the team. Additionally, we sorted out the problem with the data access at the very early stage to rule this risk out as soon as possible. In the end, none of these risks occurred.

1.7. Glossary

- **CERN** – European Council for Nuclear Research located in Geneva, Switzerland.
- **TOTEM experiment** – experiment for which the analysis was produced.
- **ROOT** – data analysis framework prepared for scientists working in High Energy Physics.
- **High Energy Physics, HEP** – branch of the physics which is CERN's primary area of interest.
- **SWAN** – Service for web-based ANalysis - a platform enabling interactive computations in a cloud which provides Jupyter notebooks configured and connected to Spark cluster

- **EOS** – storage system in which TOTEM and CERN data are saved.
- **RDataFrame** – new construct in the ROOT framework, which usefulness and performance were examined as a part of this effort.
- **Helix Nebula** – commercial distributed computing infrastructure examined for the suitability for the needs of the scientific community.
- **N-tuple** – a generalization of a fixed-length tabular type where each row of data describes a single event.

2. Functionality and scope of the project

This chapter describes the perspective of the scientists working with the HEP analyses, outlines the environment and external systems with which the application will work, describes features that are expected to be supported and introduces the general concept of the project. Plans for the usage and testing schemas enclose this part.

2.1. Users context

The product is addressed to the well-defined group of people. The TOTEM experiment scientists, who are the primary recipients of the product in the long term, are working on similar analyses for quite a long time and already performed the very same analysis in 2015. The difference between what was done three years ago and now is the way of performing those computations. Our analysis does not introduce anything new from the physics perspective in terms of this particular case or research. The goal of this effort, and thus of entire product, is to investigate and ultimately prepare a model approach, which will serve as an example for the analyses that are yet to follow.

Scientists at TOTEM have performed this type of analysis since the beginning of the experiment. Over the years, a clear routine for this kind of activities emerged and was very closely followed.

The product should include the codes of analysis, that use the new approach. All of them, unless explicitly stated otherwise, should meet conditions set out by the TOTEM experiment scientists. First of all, although the number of histograms is not specified and might be lower than in the original, the ones that are created need to be exactly the same. All discrepancies have to be thoroughly checked and, possibly, reconciled. Results validation is a point of utmost importance, as otherwise the analysis code and the execution results might be questioned.

Several objectives were specified in the first stage, which are of utmost importance to users:

- decreasing execution time to minutes instead of hours,
- the code should be easy to read and ready to be modified by the users,
- it should be possible to work with massive datasets in a natural way, without the need to divide datasets manually.
- making the exploratory data stage more interactive.

2.2. External systems

The project has to deal with a number of existing systems, which are essential and not replaceable in any case. Those systems are:

- EOS storage service,
- Apache Spark clusters,
- CERNBox service,

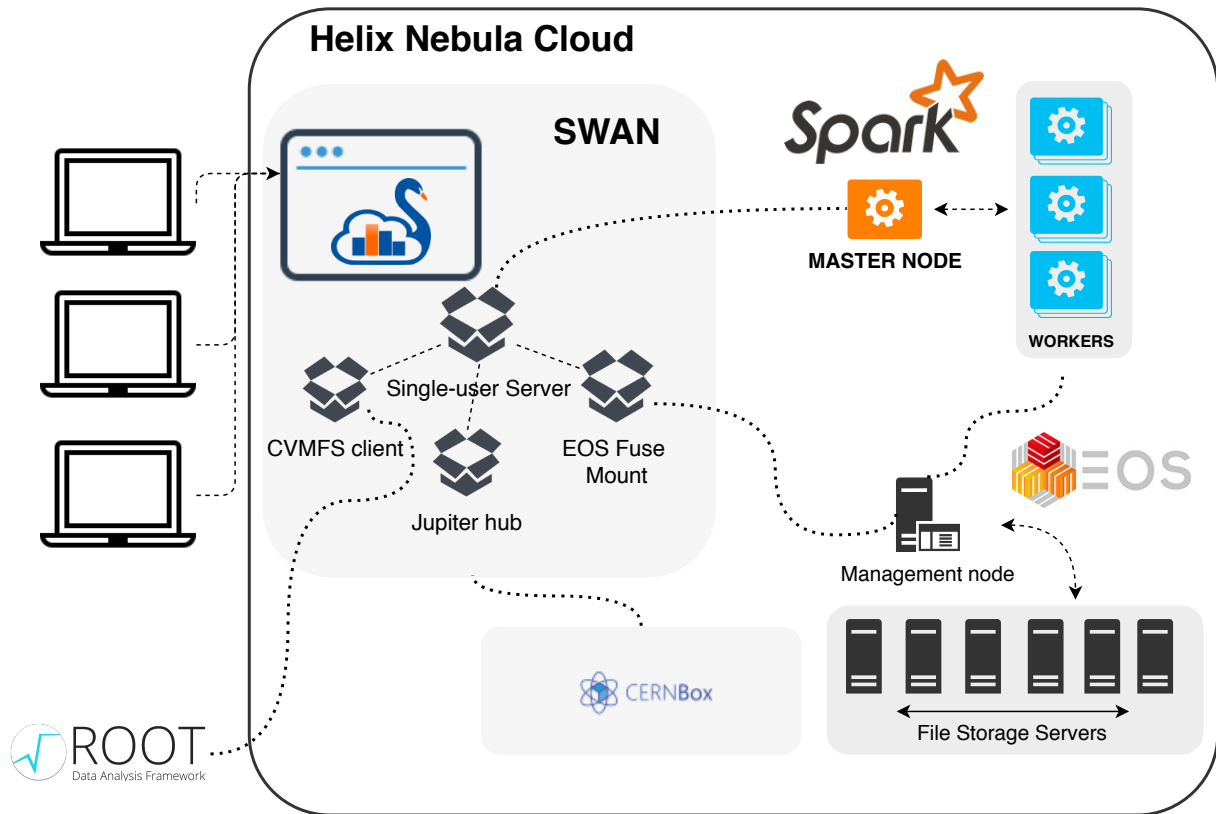


Figure 3: Helix Nebula architecture

- CernVM File System (CernVM-FS or CVMFS).

All of those services had been already tested and used before our project started.

The CERN IT Department provides EOS storage service. Initially designed to persist experiment recorded data, it is now a primary choice for both experiment and users data in the organization. It is a highly scalable solution, which is suitable for storing enormous amounts of data. The structure, management node and virtually unlimited number of storage nodes, allows for a significant horizontal scaling. Presently, it can contain even 250 PB of data. Both, CERN cluster and Helix Nebula Science Cloud, have their own instances with appropriate data for the selected analysis.

CERNBox, a cloud data storage, is a local alternative solution to the Dropbox, customized to the necessities of the scientific community. A vital goal of this solution is to allow a seamless files synchronization between any of the user's connected devices and to enable an easy and quick way to share files between users. At the backend, CERNBox also makes the use of the EOS storage.

CernVM File System is a service responsible for the software distribution. In a way, it is a replacement for the package managers, developed to deploy scientific tools and software on the clusters and other computing infrastructure. It also facilitates rapid updates of the software and simplifies the software distribution process.

Service for Web-based Analysis (abbr. SWAN) provides various useful features like Jupyter notebooks, a CERNBox connector which allows users to comfortably manage and share analysis results and built-in access to the instance of EOS, a storage service. The creators of the

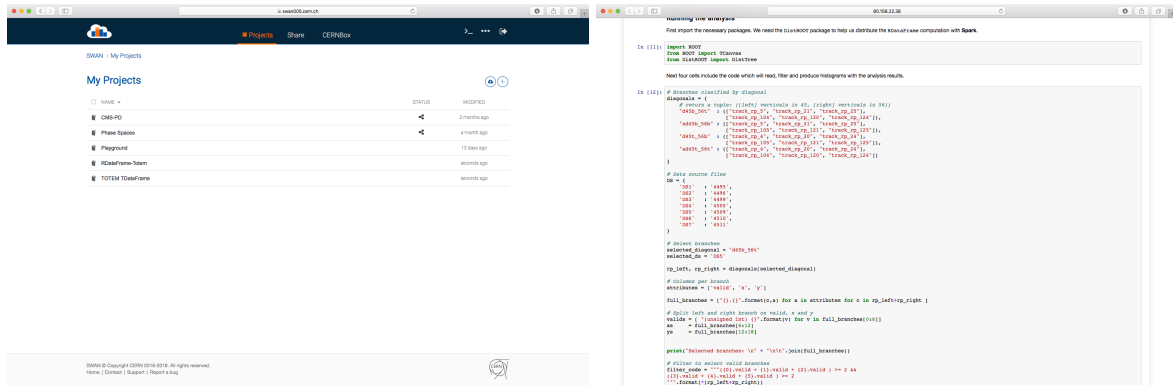


Figure 4: SWAN web interface (left) and Jupyter notebook (right)

platform conceived it as a successor to the old batch system which could address the needs of modern trends in data analysis. The latest versions include a Spark connector, which allows users to easily connect to the computing cluster and manage the parameters from a user-friendly interface.

EOS, CERN Box and SWAN all come in a container-based product called "Science Box" provided by the CERN IT Department. This idea will prepare a simple way to deploy the same, familiar and user-friendly solutions on nearly any cluster, computing grid or computer.

Spark clusters are essential to the project. At CERN Spark cluster is called "Analytix" and has 56 nodes with 780 cores, 1.31TB of memory and 2.22 PB storage in total, however, use is not limited to any particular group and resources are shared across all CERN users. Helix Nebula platform has its own, separate resources dedicated exclusively to this effort, which are: 8 nodes with 32 cores and 64 GB each. At any moment, on a request basis, available is an extension of this resources up to 1824 cores.

2.3. Features of the distributed analysis

Analysis application should be able to work with the basic ROOT format of files. In order limit amount of work done by users, the analysis should be able to handle ROOT files and do any possible additional work by itself. This way, the environment will also allow taking advantage of the EOS service directly, without any intermediary steps.

Notebooks with the analysis code should be prepared in a way which allows quick and comfortable modifications, especially for the parts which are often changed. The expected product here is a new analysis model, which will apply to other cases than the purpose of the original. Users of this product will perform those analyses by making changes directly in the code, without any additional software or graphical user interfaces. Therefore, it is crucial to make sure that their experience with the code and its complexities is as effortless as it could be.

Another important feature is the ability to work on different clusters. It is required, first of all, because of the development and testing environment, which are in a pilot stage and can be discontinued in a short time. In this case, if our application was limited to this particular hardware configuration, the product would be dependent on this particular external cloud. As it was clear from the very beginning, this was not a possibility, and results had to be easily reproducible at least on the CERN cluster.

Analysis output consists exclusively of histograms, charts, and plots. In the case of the translated analysis it is a set of approximately several dozens of graphs, which are mostly histograms, yet there are present also other types of plots. Hence, the notebook has to provide a proper means to visualize results and eventually save them in an appropriate format.

Next requirement is a proper, validated output of the analysis code. The correctness of any version of this particular analysis can be very easily checked, as widely-accepted and confirmed results are already available. This requirement is crucial to the scientists, as any other findings or conclusions drawn from this exercise would be strongly undermined in case of this point failure. The results need to be the same, so histograms of the produced set should describe the same number of data and should have the same shape. The only concession regarding this point could be made because of the computing specificities which can arise from the distributing operations, are in any case unavoidable and finally, do not pose a threat to the final results. For example, insignificant differences in means were expected and were considered likely to be accepted.

2.4. The scope of the project

The scope of the entire project is quite broad. It begins with technical complexities of the cluster environment, as its suitability and usefulness are one of the primary questions under scrutiny. Next, `RDataFrame`, a new concept in the ROOT framework, together with its experimental features are also tested as a part of these efforts. Finally, Spark and distributing computations are put to the test from the users perspective in a particular CERN experiment. Insights gathered in the course of the project are expected to be useful in the further development of the tools.

The work done by the authors of this thesis concerns different parts of this efforts: from cooperation on the development of an actual application which will run on the cluster, through its performance measurements, finally finishing with the initial findings. Those tasks eventually led us to cooperate with all members of the CERN team, to improve the behavior of the framework, the new feature, the cluster and in the end, of the application itself.

The analysis translation to the `RDataFrame` was prepared with another member of the CERN team.

For comparative purposes, an alternative version of the distributed analysis was created using Scala and Spark-ROOT connector. Insights and results gathered in course of the development complete the tests of the application in the `RDataFrame` model.

2.5. Usage and testing scenarios

In this particular project, it was essential to clearly separate usage scenarios from testing. Below, the subsections outline both cases and discuss their main points.

2.5.1. Typical usage

Data for the TOTEM experiment is acquired once a year. This period lasts for a week and is followed by the reconstruction stage. The results of this stage are the n-tuples, which are then stored as files in the EOS service.

An analysis begins only after those stages. The first initial step to start the process is to define the dataset and formulate - even preliminary - ideas which could be checked. The beginning of

the work on the analysis code using RDataFrame can be done in two, fundamentally different regarding user experience, ways. The first is to use the more classical approach, so to write an analysis code in the editor, include there all aspects that are expected to be investigated and proceed further from this point. The second is to do it step-by-step, yet there are some inconveniences. Therefore, the primary analysis code prepared in this product works in the web-based notebooks. This way, the preliminary ideas can be even only vaguely defined even as general directions of the research. The more precise actions can be decided upon later.

In all cases, the results are histograms and graphs. They are either written to the file (only possible for the script execution) or displayed in the notebook. Based on those histograms scientific work can continue. Each time something has to be added or changed, the whole script or most time-consuming part of the notebook needs to be recalculated. In the end, the results are saved in a ROOT file and can be used independently.

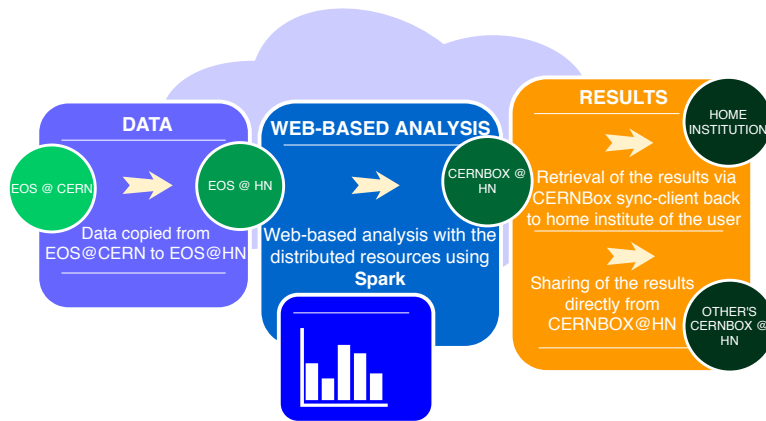


Figure 5: General pattern of the usage scenario

Nowadays with the ongoing cloud's development, it is desirable to have analysis flow including access to the data, computation itself and results fetching, designed in a way which is insensitive to changes in underneath infrastructure. It is essential for the scientist to be able to focus on data analyzing rather than struggle with software and hardware inconsistencies. On Helix Nebula cloud the whole software stack is replicated from CERN environment, and the same environment might be assured through the use of CERN's ScienceBox [17].

2.5.2. Testing

Investigating the performance, execution time and analysis application behavior is a significant area of interest to us and the CERN team at large. The scenarios for testing had to be thoroughly prepared as it has become a significant part of the project.

In all testing scenarios, the results need to be validated at the end. Validation can be performed programmatically, using the ROOT feature to subtract one histogram from another, or manually, checking the histograms one-by-one and comparing the values and the shapes. The

initial confirmation could be obtained upon a general assessment of the shapes and number of events, nonetheless more careful examination is expected for the final results as well.

The second important metric regarding the project's purposes is the speed of the analysis. This step is exposed to be influenced by many more different factors and as many of them as possible have to be eliminated, to avoid damage to the performance results. Two general categories of the performance tests were in use:

- profiling of the analysis application,
- comparative tests of the analysis's newly created versions and the original one.

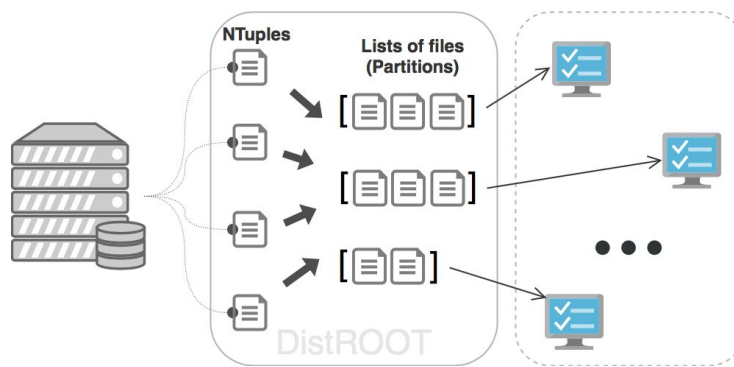
In both cases, due to the shared nature of the environments we have at our disposal, we have to assure the exclusiveness of the resources in use. Otherwise, the analysis could perform better or worse according to the conditions during a single run, imposing incorrect and irrelevant changes to the performance indicators.

For the comparative tests, several additional options and rules were developed. First of all, to get a meaningful comparison, both versions of the analysis have to be run in the same environment and on the same machines. Differences in the environment could result in, e.g., executing the code using different services, and their performance would undoubtedly affect the measurements of the analysis application performance. For instance, the ability of the system to cache certain data is an important factor which should be taken into account. Obviously, even more visible differences would be caused by the differences in the hardware configuration. Those differences could prove to be useful at times but had to be handled carefully to get significant findings.

3. Development of the Distributed Analysis

This chapter details the general concept of the analysis application and technological solutions used in the project. Furthermore, it presents selected issues encountered during development and profiling, with the applied solutions. The chapter finishes with a description of the efforts concerning the alternative version of the analysis, including its development and testing.

3.1. Structure of the analysis application



11

Figure 6: Map Reduce pattern in the analysis application

The analysis was originally divided into two steps: distilling interesting events and computing results, which involved more complex calculations. After the first step, only 2% of data remains to be processed in the second step. However, this split resulted, among other reasons, from the impossibility of handling all data at once. Opportunity to use more extensive datasets would enable entirely new possibilities, e.g., for faster initial data exploration, and will be a necessity in the future. Therefore, the application combines both steps into one to better reflect expectations regarding the future use-cases.

Analysis application follows the Map-Reduce pattern. In order to distribute tasks on the cluster, workload needs to be divided into parts, ideally of equal size, and then to be mapped to the executor. DistTree object, which is part of the DistROOT library, is responsible for this operation. During the profiling stage, a problem with this phase was uncovered and is discussed in section 3.3.2.

Each of the nodes, which has assigned task, performs the computations defined in the function passed to the mapper. Results are returned as a list of histograms.

The final phase is the reduction part. Each of the lists coming from executor nodes needs to be merged. Histograms of the same type are essentially summed, and the final result is another list of histograms, this time containing data computed on all nodes. Merging is performed using TreeReduce operation, reducing the time complexity of the operation from linear to logarithmic.

3.2. Technological solutions used in the project

This section contains basic information on the tools and technologies used in the project.

3.2.1. Apache Spark

Computer clusters, sets of computers working together and connected via a shared network, are powerful tools, which, however, require efficient tools to manage work in order to utilize their potential capabilities fully.

A specific platform for distributed computations selected in this project is Apache Spark [13]. Over the recent year, it has become a top choice for cloud computing and is developed as an open-source project since 2010.

In typical use, the Resilient Distributed Dataset (RDD) is created out of the data, and then parallel operations are performed on this abstraction. In this case, the step is combined with solution integrated into RDataFrame from ROOT framework is chosen instead. Therefore, there is no need for massive preprocessing before and the native TTree structures which are used in the ROOT files are used.

Spark platform is used in this project mainly as a scheduler. It is responsible for the initialization of the executors on the nodes, the tasks allocation on the executors and effective management of the tests execution.

3.2.2. Jupyter Notebook

Advances in the cloud computing and data science domain led to the breakthroughs also in environments for the code creation and the presentation of the results. Standard files containing the source code were replaced by the interactive notebooks, which allow for more flexible management of the code layout, with additional capabilities for code annotating, step-by-step execution and even web widgets.

Project Jupyter proposed one of the leading solutions in this field called Jupyter Notebooks [26]. Formerly known as IPython Notebooks, it is a web-based environment for the creation, development, and presentation of the analysis code combined with results of the code execution, text and multimedia features.

In principle, the structure of the notebook usually consists of a certain number of cells, which are organized in a planned sequence. Each cell can contain a header, paragraphs of text, multimedia content or code. Depending on the environment configuration, a list of the supported languages differ. List of Jupyter kernels, ready-to-use extensions for different languages is already very sizeable and can be extended further. Kernels are responsible for handling the execution requests and delivering results back. By default, Jupyter notebooks are shipped with Python kernel, and most of the popular languages are supported, with particular emphasis on those used in data science.

Jupyter notebooks were a natural choice for the project like this one. Nowadays, this is the leading solution for cloud computing and data analysis, chosen by large industry cloud providers as a default frontend interface. Jupyter Notebooks are also integrated as a part of the SWAN service, which was deployed on the cluster provided for the project. It was essential to

follow the target platform as closely as possible in each of the project phases to gather valuable insights about the code development from an end-user perspective.

3.2.3. ROOT Framework in the project

HEP community developed a very advanced and well-optimized set of tools for advanced data analysis. ROOT was created in 1994 at CERN and since that time is actively developed. Over the years, it has become a standard for the scientists engaged with HEP experiments, though remains largely unknown in the outside world. Important features include:

- tools for fast access to large amounts of data stored in the compressed binary form in ROOT files,
- CLINT, a C++ interpreter,
- graphical user interface,
- mathematical and statistical tools tailored to the needs of data analysis,
- own visualization tools, together with own library to create histograms.

Recent developments are more and more focused on enabling a paralleled and distributed execution of the ROOT code. One of the newest additions is RDataFrame, a new way to handle data in ROOT applications. The general concept is similar to the DataFrames known from R language or Pandas in Python: RDataFrame provides users with high-level API to perform computations on the data coming from different sources, including the TTrees, which most CERN experiments use to organize stored data.

RDataFrame introduces a leading change in the way the calculations are expressed, allowing for a more functional-style approach and lazy execution. It also does low-level optimizations and caching to the performed computations and adds implicit multi-thread parallelization capabilities.

Broadly speaking, operations which can be performed on the RDataFrame are divided into two categories: transformations and actions. Transformations can be used to filter data or define a new column, usually resulting from extensive computations. Purpose of the second type of operations, called actions, is fetching results from the frame. Actions can be lazy, and then the execution is delayed until the resulting data is accessed for the first time, or eager when the operations are performed instantly.

General trends and interesting perspectives for analyses on huge datasets caused the development team to include also features to distribute computations using Spark clusters. The concept implemented in ROOT as RDataFrame, for the reasons mentioned above, is perfect for such purposes. Experimental features, not yet available in the released versions of the framework but ready to use in the form of an external library, will allow to split the workload into the tasks, execute them using a Spark cluster and then merge results. The library is called DistROOT[28] and eventually is meant to be added to the RDataFrame, hiding most of the operation inside.

3.2.4. Tools and technologies used for profiling purposes

One of the challenges encountered at the beginning was the choice of appropriate profiling tools and techniques. The matter was more complicated because of the complexity of the application structure, which was operating on different levels and utilizing many custom-specific modifications.

Spark applications usually provide users with the Spark UI, which conveys a vast wealth of data on the performed computations and presents them in real-time, just as the execution goes. However, use of the Spark UI had significantly been limited in the discussed case for two reasons. First of all, the framework was used here only in a limited capacity to schedule tasks, thus was not able to gather the typical data during the execution. Secondly, the cluster configuration and its network configuration fell short of supplying external access to the monitoring pages of individual Spark executors. However, although its use was limited for the reasons stated above, Spark UI provided data on simple metrics like execution time or the tasks distribution over the executors and hosts.

In the face of those limitations, other ways of tracking the application executions needed to be developed. Several proposals were discussed and examined, and in the end, most of them were independently deployed in various stages to gather different aspects of the application performance.

Cloudera Manager was exceptionally helpful to collect insights about cluster-wide or host-wide statistics during the regular tests involving Spark. Its' clear layout and advanced chart options allowed for quick and efficient work, at times even effortless as the statistics were gathered continuously.

The more classic and incomparably less advanced but also quite comprehensive way to gather the data on device performance was a Unix tool "collectl." Because of its nature, it was utilized mostly in cases of application runs without Spark, usually on one host or a local machine. A considerable advantage was a negligible influence on the system and the device. It proved to be especially useful at the moment in which we were trying to establish how the application effectively handles the data, a potential pre-fetching or cache performance.

All of those tools provided invaluable insights into the application performance and operations. Combination of this data allowed us to understand better the effective activities of the tasks, the RDataFrame framework, and the whole set-up, and to solve many pressing problems.

3.3. Major issues encountered during the development and profiling and developed solutions

During the development and then profiling of the application, many challenges of different nature had to be faced. Below, we discuss several examples of the issues that were solved in the course of the project.

3.3.1. Weighted histograms

Due to the RDataFrame capability to run in three different modes, as a single thread, multi-thread and Spark-distributed application, first translation efforts could concentrate solely on the development of a single-threaded application. Proceeding gradually and including new features

in a step-by-step manner, we separated the outcomes of the subsequent parts and thus obtained valuable insights on the impact that each of the elements creates on the output and the execution time. That way, we could precisely and relatively quickly identify the first issue, which was observed after modifying the code to work in a distributed way with Spark: an inconsistency in the number of events in a few histograms.

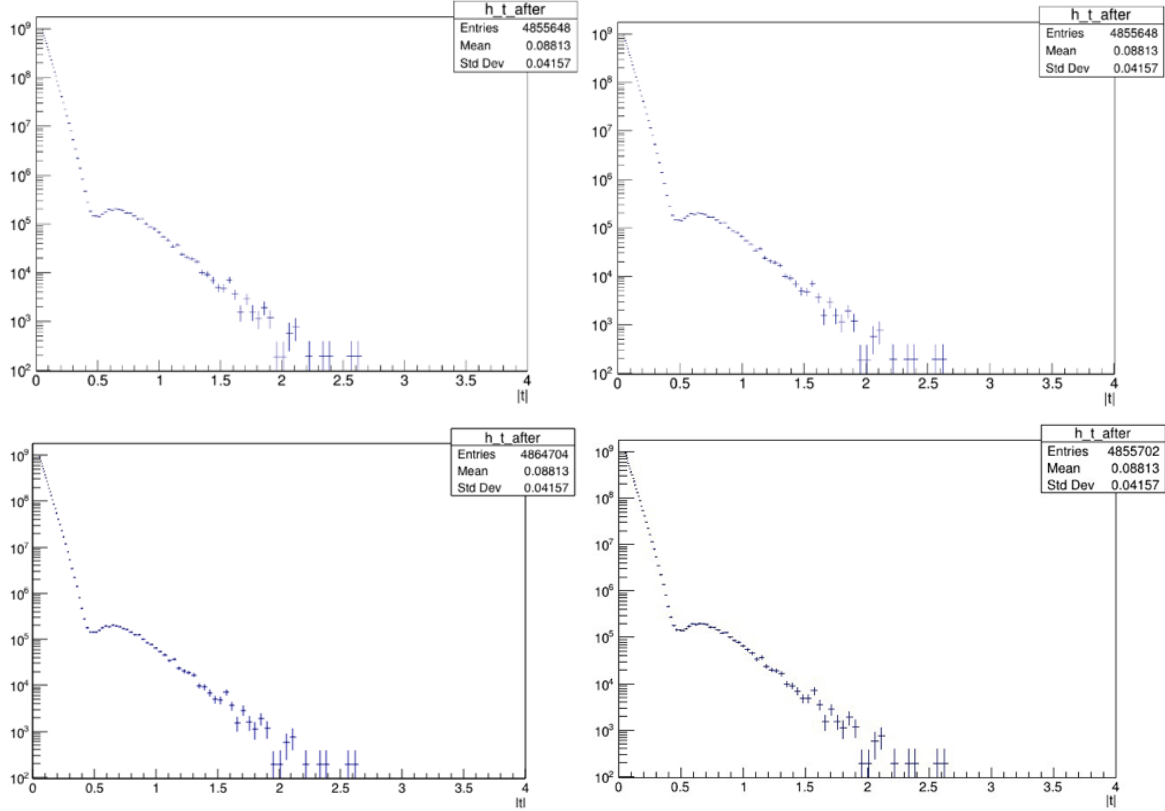


Figure 7: Discrepancies in histograms

sequential original analysis (top left), sequential RDataFrame (top right), distributed RDataFrame with 16 partitions (lower left), original analysis divided into two parts and then merged manually (lower right)

A closer examination revealed that the discrepancy could be observed only in three particular histograms (Fig. 7). In all of them, the number of events differed between original results and the ones retrieved from the distributed execution with Spark. Repeated tests on the same dataset and with the same parameters ended in the same results as in the previous attempt, so there was no random element in the problem.

The difference between both execution modes, single core and distributed, was the introduction of Spark. To be able to run the same analysis, two additions needed to be implemented: the creation of a distributed tree, responsible for dividing data at the beginning, and the mechanism responsible for merging the results from different tasks at the end. A distributed tree can be generated by merely creating a DistTree object from the DistROOT library, a part of the PyROOTSpark project, which aims to simplify PyROOT parallelization with Spark. Merging must have been implemented explicitly in the analysis code at the time of the analysis development. In the case of this analysis, each task returns a list of histograms as a result. To obtain the whole result, all the corresponding histograms in the lists simply had to be added to each other. It is

worth to mention that both steps, dividing and merging, are going to be performed implicitly, as a part of RDataFrame functionality; in the future — there will be no need for the user to specify them in the code. Work on this feature was still in progress in the time of our tests.

	Number of partitions					
Histogram	1	2	17	32	64	90
h_t_after	6690032	6690071	6697284	6706545	6717720	6723669
ob_1_10_0.2	4855647	4855698	4864703	4875740	4888119	4894308
ob_1_30_0.2	4855648	4855699	4864704	4875741	4888120	4894308

Figure 8: Differences in the entries numbers for the weighted histograms after distributed execution

After additional checks, we proceeded to try to identify the sources of the encountered problem. Next attempts revealed that numbers of entries in those histograms depend on the number of partitions into which the data is divided (vide fig. 8). It was another indicator that the fact of distributing the analysis, namely, dividing it into separate pieces, computing and then merging by summing results, is the reason for incorrect results. In the following days, we discussed the issue with other team members focused on the RDataFrame implementation, and then with ROOT experts, who directed us towards possible explanations, including checking further the nature of the histograms we were trying to compute.

The ultimate reason for the discrepancy in the histograms' details was hidden directly in the type of those histograms. The suggestion was that summing weighted histograms might not lead to the same number of entries as in one weighted histogram produced for the whole dataset at once. The fact that the three histograms were weighted was mostly unnoticed; the number of entries was presented as an integer without a fractional part, which could be expected in the case of the weighted histogram. Reasons for this practice were purely historical.

To validate this suggestion, data other than the number of entries needed to be checked. The correctness of the histograms was proved by comparing sums of weights and number of effective entries, which had the same values in all versions of the analysis. As a conclusion, we agreed that the histograms were acceptable in this form and that the issue was present also when the analysis was split and merged manually. The point was raised to consider displaying fractional part of the integer in the histogram visualization.

3.3.2. Data partitioning options

Our main area of interest was the introduction of Spark and its performance in the presented case. We wanted to observe the actual speedup in the execution time and its efficiency in regard to the theoretical speedup which we could achieve with given resources.

In a simplistic view, an expected speedup should be equal to the number of partitions executed in parallel. It would be an obvious overestimation, as we can infer from the Amdahl's law [6]. In the case of this particular analysis, there are excellent perspectives for parallelization. The concept of the code that is behind it is based mainly on executing a same, well-defined set of computations on a large number of entries stored in ROOT files. A part that cannot be parallelized is relatively small, includes only activities required for distributing the analysis: dividing the data into pieces which later will be distributed as separate tasks and then merging the

results of those tasks. Both steps are relatively small: analysis of 4.7 TB returns results in a few dozen megabytes of histograms. Thus, the analysis can be considered a highly parallelizable application.

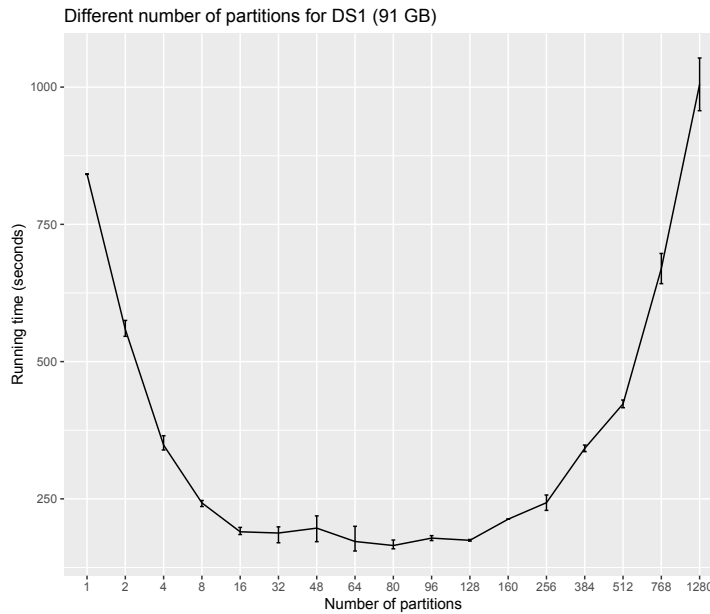


Figure 9: Execution time for different numbers of partitions used with maximum 128 cores on 6 nodes

Problem with significant deviation from these performance expectations revealed itself relatively quickly during the testing when the core development stage was finished. Already in the early analysis test runs emerged a peculiar and strictly undesirable pattern, displayed in figure 9. First tests conducted on only one dataset (ca. 90 GB) with different numbers of partitions and 128 cores showed that the speedup curve is flattening rapidly for more than eight partitions (each assigned to one core). More significant numbers of partitions resulted in even worse results: the execution time started to increase when one core had been assigned to more than one task (it was the case from 128 partitions onwards). According to the general concept, the time should keep getting better or at least stay more or less at the same level, as Spark can manage the smaller tasks more efficiently, substituting failed ones faster and with better capabilities. That was apparently not the case here and to take full advantage of distributing the analysis and utilize more extensive resources, this issue needed to be resolved.

In general, four phases could be observed in the speedup chart in figure 10. Starting with phase one the tasks scale well with the increasing number of cores. Beginning of phase two marks a visible break with the nearly linear speedup; from this point, the problem starts to be visible. Phase 3, resources saturation, is expected behavior: when the number of partitions reaches the number of available cores, the speedup cannot increase further at the high rate. The curve is finished with phase 4, a rapid increase in execution time for vast numbers of partitions.

Multiple different tests and attempts to profile the analysis application were conducted on the Cloud resources, on the local machine and the individual nodes of the cluster.

The aim of the measurements on the local machine and individual cluster nodes was to establish whether there are potential issues with IO, storage or CPU load only inside the standard

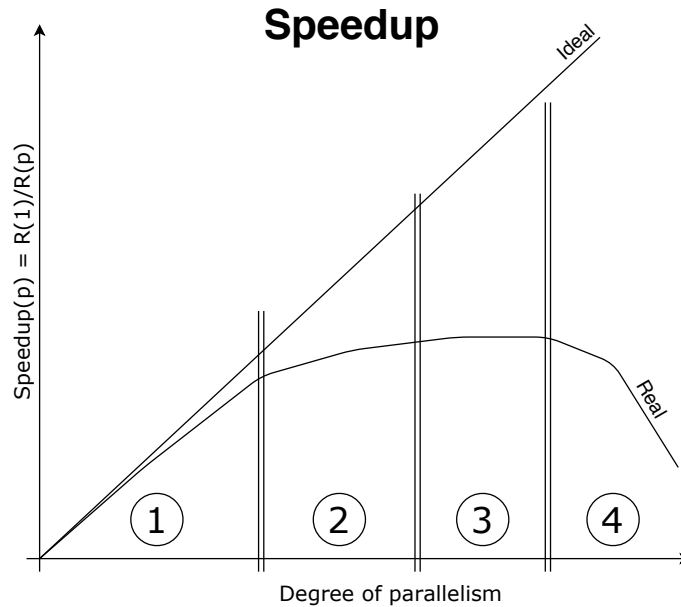


Figure 10: Approximate shape of the speed-up curve encountered at the initial stages of the Spark introduction to the project

application. A number of measurements using basic terminal tool were gathered, together with records of the resources used during analysis execution. An example outcome is presented in fig. 11. The charts were reported to the group and discussed in the presentation (slides attached in Appendix E). The tests did not reveal any extraordinary problems, indicated that CPUs were busy most of the time, remote storage caused little breaks, but in general could not be responsible for the issue. A clear edge of the implicit multithreaded execution over the distributed was discovered (as visible in figure 11).

Meanwhile, another set of tests was performed using the Cloud and Spark.

Finally, the attention turned to the number of CPUs seconds spent in the application, and total bytes read over the network or from the local storage. The outcome was surprising — instead of staying on the same level, both metrics were growing with the increasing number of partitions. It meant that the application is somehow increasing the workload it is meant to process, undermining the potential advantages from distributing the analysis.

The first experiment aimed at explaining performance issues was to change the way entries as assigned to the separate parts. DistROOT had been doing that in a smart way, operating on the number of events and distributing them evenly. Such means are necessary to ensure general equality of the tasks' sizes, which is important to utilize Spark capabilities fully. Uneven distribution of a workload is certainly a highly undesirable situation leading to worse performance on account of the wasted resources assigned to the smaller tasks. Modified DistROOT, prepared for the testing purposes, changed the way it assigned data to the tasks. Now, all events stored in single file were assigned to one RDataFrame, and thus to one executor. It became impossible to parallelize the analysis above the number of files; it also exposed the analysis to the problem of uneven distribution of events. Having all those defects in mind, we proceeded with the test to simplify the tasks creation as much as possible. A new behaviour, illustrated in figure 12, was implemented.



Figure 11: CPU usage and kilobytes read from SSD in time of application execution

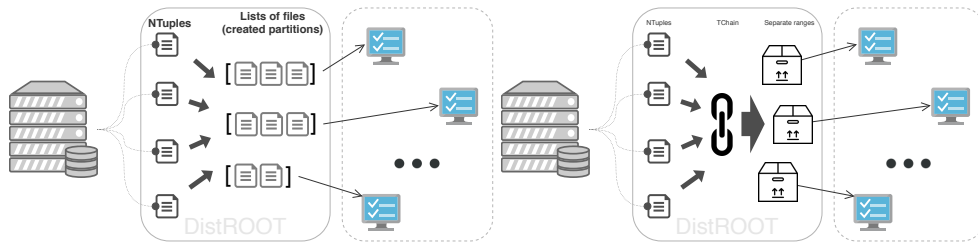


Figure 12: DistROOT - a mechanism responsible for assigning entries to tasks in two variants: original (left) and modified (right)

The outcomes of this test went beyond expectations. Execution time went down from more than 3 minutes to 46 seconds for the first dataset (ca. 90 GB) and from ca. 3 hours to 15 minutes 47 seconds for data from all available datasets (ca. 4.7 TB). New speedup charts, prepared with modified DistROOT, showed that some problems were resolved. The application, although still not ideal, was scaled much more efficiently than before (fig. 13). Maximum speedup for 240 cores reached 120 times, so about 50% efficiency, which is not a perfect outcome, but there were some easily expected losses resulting from much less flexible and limited DistROOT.

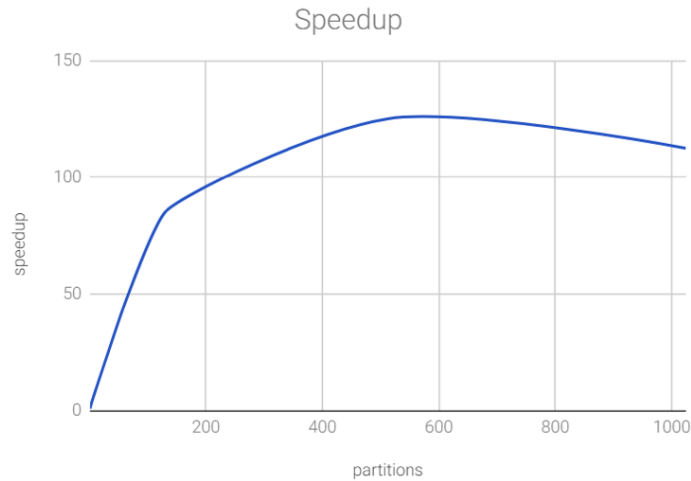


Figure 13: Speed-up vs partitions chart for the DistROOT dividing by files

3.3.3. Tests with extended resources on Helix Nebula Cloud

The final tests performed in December, at the end of the project, included the use of significantly extended resources. The number of ready to use cores rose to 1824, and total memory reached 4.5 TB. Spark cluster with 57 nodes was created, each having 32 cores and 128 GB of physical memory.

Those resources provided us with the means to repeat the scaling test of the analysis application. Previous attempts allowed us to uncover problems with parallelizing tasks and to accelerate the execution time significantly.

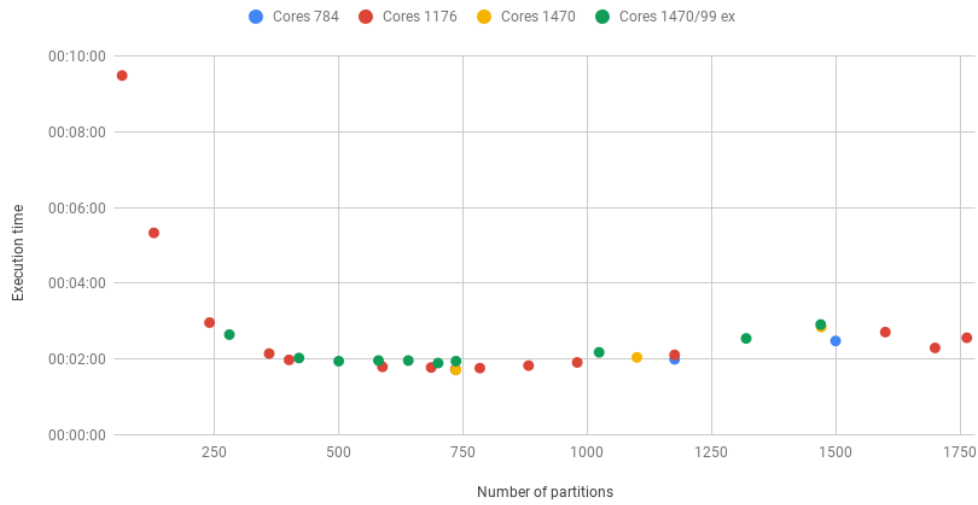
Results attained with those resources were quite good, scalability was performing well but only up to a certain point. We can observe the speedup increase coming from this series of tests in figure 14. Once again visible are four stages of the speedup curve. First, up to approximately 350 cores, a high parallelization efficiency can be observed. Speedup acquired with all of the cores in this stage reaches 250, what is equal to 71% of parallelization efficiency and the trend is generally linear. The second phase lasts to the ca. 750 cores. Speedup increase slows, and the gains from adding next cores are present, but the effects are much less impressive. The fastest execution happens for 750 cores, finishes in 1 minute and 44 seconds, so nearly 300 times faster than one core execution and this is the maximum speedup we acquired with the application. An interesting fact was that Spark failed for any attempt which was made with more than 1750 partitions.

The results seem to suggest that there still might exist some unresolved issues within the application itself. The suspected reason, corroborated by this data and by the findings of other team members, indicate a presence of possible memory leaks, which magnitude cause the application to slow down with a more significant number of partitions and eventually to crash. However, no apparent source of the leak was found. The work on this matter continues.

Additional insight was provided by a series of tests with the slightly modified configuration. Spark option “spark.reuse” alters the default way of assigning awaiting tasks to the executors. In a standard way, the executor is being “reused” possibly many times, e.g., next tasks are being assigned to the same instances of executors which finished previous ones. This modification

Spark Execution Time

Helix Nebula, 7 datasets (4.7 TB)



Spark Speed-up

Helix Nebula, 7 datasets (4.7 TB)

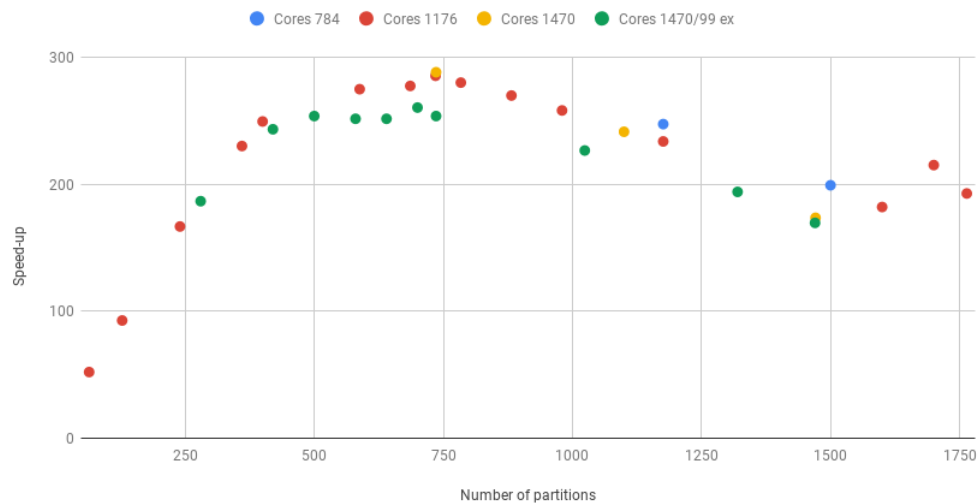


Figure 14: Execution time and speedup acquired using extended resources in December tests

requires executors to be restarted, so their entirely new instances operate tasks. It was essential to check in our case whether it affects the application performance and, in particular, decrease the effects of the suspected memory leakage.

Tests series based on this modification was the last performed on the Helix Nebula. Results are presented in figure 14 with reference values gathered previously. Clearly, the limit of effective acceleration through parallelization moved to the 500 partitions, each computed on a separate core. This result appears to confirm the previous findings and strengthens the case for further examinations in this direction.

The last tests were performed with varying size of the analyzed data. In contrary to previous ones, the numbers of executors (49), cores (735) and partitions (512) used were set to the constant values. The varying factor was the input data size (generated by multiplication of the

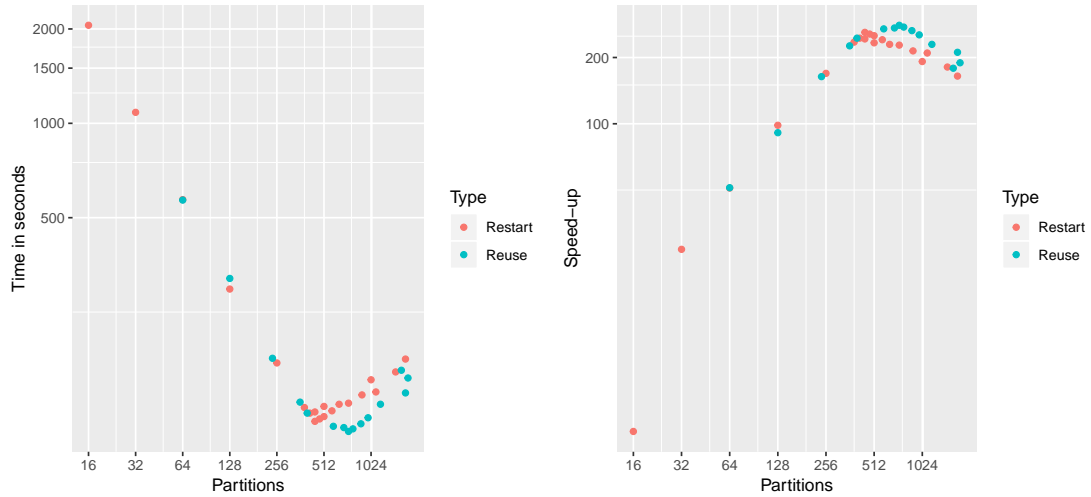


Figure 15: Comparison of execution time and speedup without "reuse" option in December tests

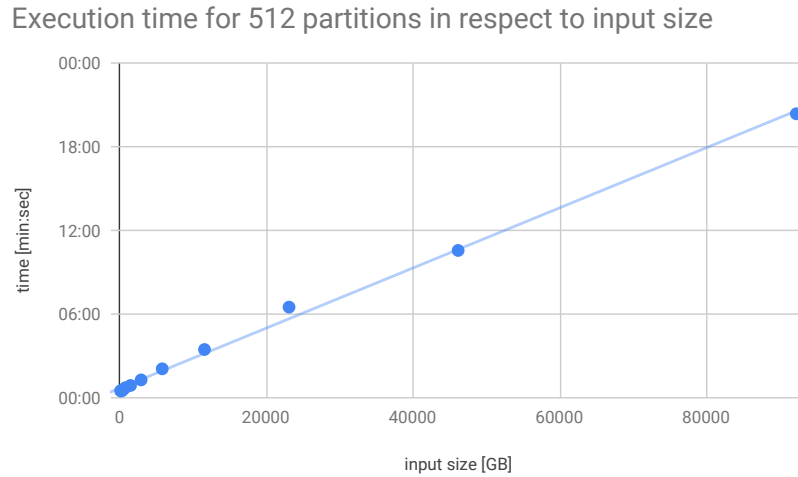


Figure 16: Average execution time to input data size

first subset). We hoped the approach would allow eliminating the influence of possible memory leakage, which was suspected to be dependent on the number of partitions. The most significant dataset tested was 1024 times larger than DS1 (92160 GB in total), and its computation on settings mentioned lasted approximately 20 min 30 s. What is clearly visible in Fig. 16, the relation of the time to the input data size is even less than linear which is quite promising for future applications operating on much bigger datasets. It seems that no CPU or memory bound bottleneck has been reached yet and furthermore application speed-up with data increased appropriately.

3.4. Profiling and testing

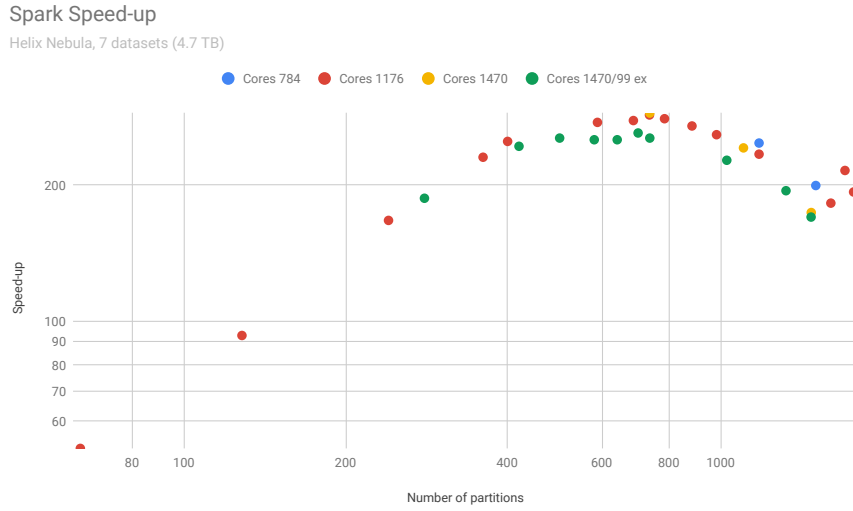


Figure 17: Execution time acquired by distributing the computations

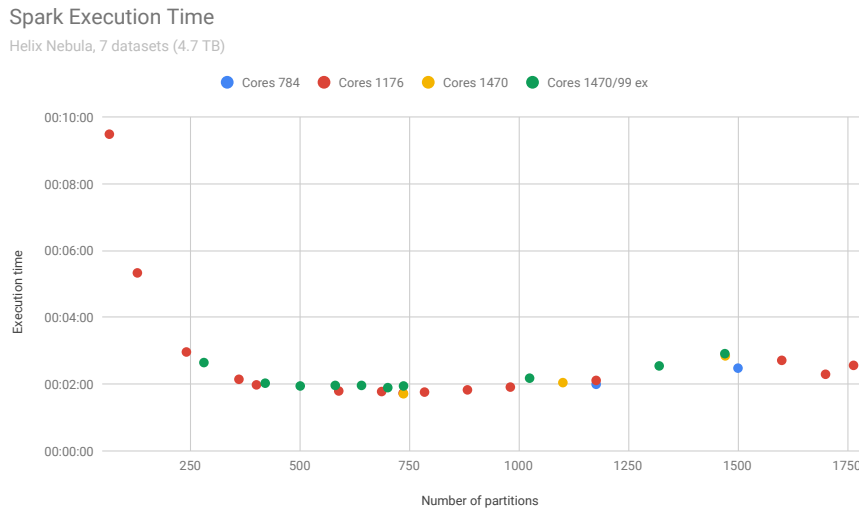


Figure 18: Speed-up acquired by distributing the computations

For the speed-up and efficiency discussed below, only the part executed with Spark is taken into account. Other parts of application include initial initialization of the functions before and saving files afterward, but those times are constant and negligible in this case. However, there is one additional part which duration depends on the size of input data. The initialization of the DistTree object is responsible for computing equally sized workloads basing on fetched metadata. Its length is proportional to the size of the input data. Details of the execution time are presented in Appendix F.5. Possible ways to speed-up this part are to be considered in the future.

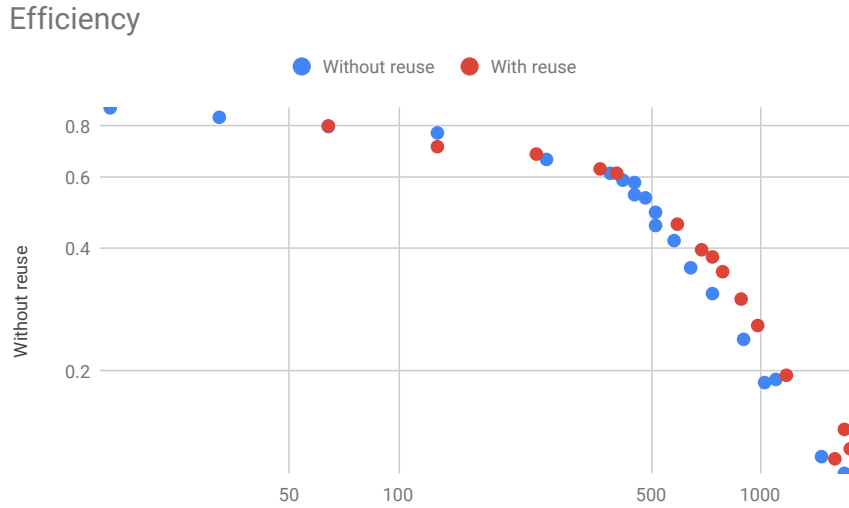


Figure 19: Comparison of efficiency of parallelization with and without executors restarting

Three types of tests could be established:

- based on Spark UI,
- based on Cloudera Manager measurements,
- based on the performance measured on a singular worker node.

Speedup discussed in this chapter can be defined as:

$$S_p = \frac{T_1}{T_p}$$

where T_i is the execution time in an i core environment. The maximum speedup attained with this analysis application is presented in figure 18. Maximum speedup attained only from distributing the analysis is 280 times. Besides, the execution time was also influenced by using RDataFrame, which seems to provide decent optimizations itself. However, those optimizations are not included in these numbers.

The efficiency can be defined as follows:

$$E_p = \frac{S_p}{p}$$

where S_p is a theoretical maximum speedup acquired for p cores, which for simplicity can be considered as equal to the number of available cores. It dismisses obvious parallelizing limitations but provides a simple way to benchmark the effectiveness of utilizing additional resources. Final results of efficiency are displayed in Figure 19 .

3.5. TOTEM analysis in Scala

The main version of the analysis is written in ROOT RDataFrame model, however, what was mentioned in the first chapter, the analysis written in Scala was also considered and imple-

mented. The second solution required more effort and addressing more issues encountered during development, compared to equivalent RDataFrame analysis.

Use of Spark as a critical technology limits the number of languages in which the code might be written. The first approach described earlier in detail, relies on Python both in PyROOT to operate on ROOT structures and PySpark to use Spark scheduling. Knowledge of PySpark overheads in serialization and performing UDFs leads to the conclusion that additional performance speed-up might be gained by writing an analysis in Scala. This is possible thanks to the attempts to rewrite ROOT analysis to Scala that had been already made, and the connector library "scala-spark" used to read ROOT files into Scala RDDs exists, along Histogrammar library to create histograms.

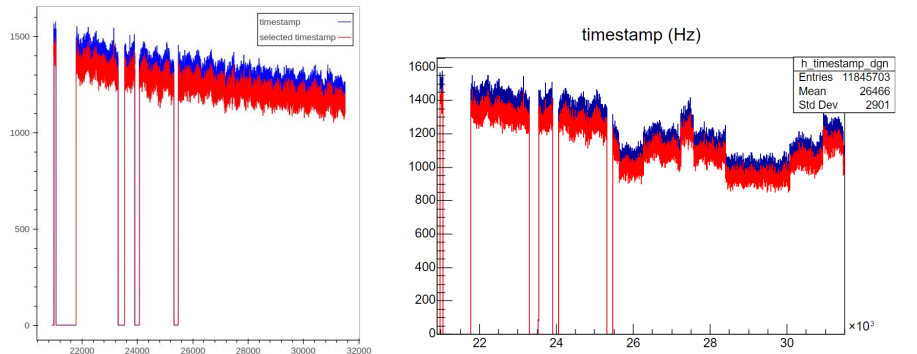


Figure 20: Comparison of 'Rate CMP' histogram produced by application written in Scala and the same histogram from original analysis in C++

The analysis in Scala was successfully implemented in the scope of this project. The code can be found https://github.com/ciastkoMalinowe/Elastic_Scattering_Analysis. What can be observed in Figure 20, in terms of physics validity, the histogram has the same shape and event number as the one produced by original, sequential C++ analysis, so the minimum requirement is fulfilled.

Summary Metrics for 597 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	0 ms	8,7 min	10 min	12 min	24 min
Scheduler Delay	12 ms	15 ms	20 ms	61 ms	18 min
Task Deserialization Time	0 ms	20 ms	26 ms	52 ms	53 s
GC Time	0,4 s	11 s	13 s	16 s	30 s
Result Serialization Time	0 ms	10 ms	10 ms	12 ms	0,3 s
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B

Figure 21: Summary Metrics of Scala application (from SparkUI)

As shown in Fig. 22, the solution is scalable at least up to a certain point. Major performance concern results from Spark-ROOT library approach to read and operate on ROOT data structures file-dependently. That means the file size disproportion influences the load balancing of tasks. As can be seen in Fig. 21, in our particular example of 1148 files of approximately 1.5 GB each, this inbalance limits Spark abilities to efficiently spread computations across nodes. The number of files with input data limits the scalability, but we did not consider it as a significant issue.

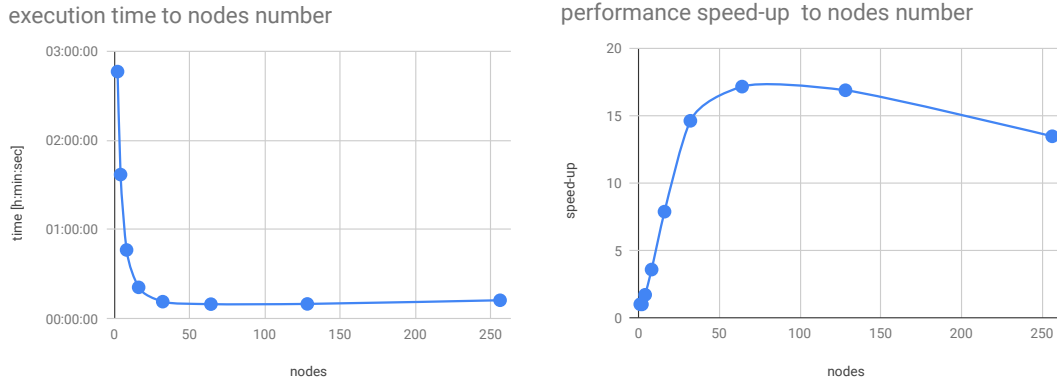


Figure 22: Scalability tests of analysis written in Scala performed on Prometheus cluster - one Spark executor per node (24 cores and 120GB RAM per node), all DSs except DS5 as input

```

val l = Label(
  "timestamp"      -> Histogram(timestamp_bins, timestamp_min - 0.5, timestamp_max + 0.5,
    {x: NtupleSimplified => x.timestamp}),
  "timestamp_sel"  -> Histogram(timestamp_bins, timestamp_min - 0.5, timestamp_max + 0.5,
    {x: NtupleSimplified => x.timestamp, afterSelection}),
  "th_x_diffLR"    -> Histogram(1000, -500E-6, +500E-6,
    {x: NtupleSimplified => x.kinematics.rightArm.th_x - x.kinematics.leftArm.th_x, afterSelection}),
  "th_y_diffLR"    -> Histogram(500, -50E-6, +50E-6,
    {x: NtupleSimplified => x.kinematics.rightArm.th_y - x.kinematics.leftArm.th_y, afterSelection}),
  "th_x_diffLF"    -> Histogram(400, -200E-6, +200E-6,
    {x: NtupleSimplified => x.kinematics.leftArm.th_x - x.kinematics.doubleArm.th_x, afterSelection}),
  "th_x_diffRF"    -> Histogram(400, -200E-6, +200E-6,
    {x: NtupleSimplified => x.kinematics.rightArm.th_x - x.kinematics.doubleArm.th_x, afterSelection}),
  "th_x"           -> Histogram(250, -500E-6, +500E-6,
    {x: NtupleSimplified => x.kinematics.doubleArm.th_x, afterSelection}),
  "th_y"           -> Histogram(250, -500E-6, +500E-6,
    {x: NtupleSimplified => x.kinematics.doubleArm.th_y, afterSelection}))

val timestamp1: Long = System.currentTimeMillis
val r = resultRDD.aggregate(l)(new Increment, new Combine)
val timestamp2: Long = System.currentTimeMillis
val elapsed = (timestamp2 - timestamp1) / 1000

```

Figure 23: Histogrammar labeling structure, which computes histograms in a single pass

There are several inconveniences related to the library used for creation of histograms. In general, Spark lacks proper histogram abstraction with visualization capabilities on a level comparable to JsROOT. Apache Zeppelin provides a bit more advanced solutions but the last version of Spark it is compatible with is Spark 2.1, and it is highly possible it would not be supported anymore. Histogrammar offers back-end for aggregation, but is not optimized for analysis which produces various histograms during single event loop. Although it provides labeling abstraction as a dictionary or map responsible for calculating histograms over a single pass, it is impossible to operate on different histograms types at once.

Scala code was tested on Prometheus cluster where Spark workers were deployed on physical nodes (24 cores and 128 GB RAM each) and used Lustre file system. Unfortunately, despite many efforts, we have not succeeded to run both analyses in the equal environments to perform a fair comparison. Taking into account all the mentioned issues regarding Scala version of analysis, for testing purposes, its scope was reduced to a small subset (19) of 1D histograms originally produced. Obviously, such tests would not enable us to compare both versions but it is an effort to get some insights about the scalability of the solution, possible future use in other analyses with not a significant amount of work needed to add missing parts.

execution time to nodes number (cores number fixed)

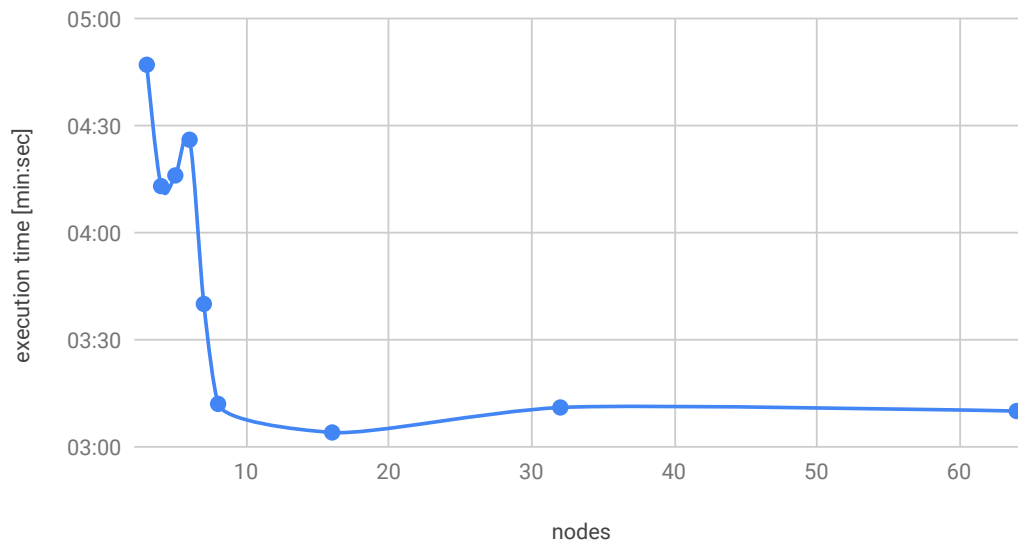


Figure 24: Execution time for Scala version on Prometheus cluster with DS1 (90GB) as input

Additionally, due to unexpected flaws, partially in copied n-tuple files, input data was reduced to 707 files and 2.8 TB in total. Execution with such an input on a single core on Prometheus cluster took 69 hours 51 minutes and 29 seconds, which is surprisingly long. What can be observed on diagrams with test results in Figure 27 is already mentioned: scalability is limited by a number of files (each node is 24 cores; therefore 707 divided by 24 is equal to 30 nodes) but also that not only number of cores used in total but also number of executors influence execution time. Further investigation is presented in Figure 24.

```

2018-12-31 00:10:21 INFO DAGScheduler:54 - Final stage: ResultStage 0 (aggregate at script.scala:143)
2018-12-31 00:10:21 INFO DAGScheduler:54 - Parents of final stage: List()
2018-12-31 00:10:21 INFO DAGScheduler:54 - Missing parents: List()
2018-12-31 00:10:21 INFO DAGScheduler:54 - Submitting ResultStage 0 (MapPartitionsRDD[16] at rdd at script.s
cala:114), which has no missing parents
2018-12-31 00:10:21 INFO MemoryStore:54 - Block broadcast_0 stored as values in memory (estimated size 966.4
KB, free 365.4 MB)
2018-12-31 00:10:21 INFO MemoryStore:54 - Block broadcast_0_piece0 stored as bytes in memory (estimated size
56.5 KB, free 365.3 MB)
2018-12-31 00:10:21 INFO BlockManagerInfo:54 - Added broadcast_0_piece0 in memory on p1125.prometheus:39853
(size: 56.5 KB, free: 366.2 MB)
2018-12-31 00:10:21 INFO SparkContext:54 - Created broadcast 0 from broadcast at DAGScheduler.scala:1039
2018-12-31 00:10:21 INFO DAGScheduler:54 - Submitting 707 missing tasks from ResultStage 0 (MapPartitionsRDD
[16] at rdd at script.scala:114) (first 15 tasks are for partitions Vector(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
11, 12, 13, 14))
2018-12-31 00:10:21 INFO TaskSchedulerImpl:54 - Adding task set 0.0 with 707 tasks
2018-12-31 00:10:36 WARN TaskSchedulerImpl:66 - Initial job has not accepted any resources; check your clust
er UI to ensure that workers are registered and have sufficient resources
2018-12-31 00:10:51 WARN TaskSchedulerImpl:66 - Initial job has not accepted any resources; check your clust
er UI to ensure that workers are registered and have sufficient resources
2018-12-31 00:11:06 WARN TaskSchedulerImpl:66 - Initial job has not accepted any resources; check your clust
er UI to ensure that workers are registered and have sufficient resources

```

Figure 25: Spark console

During each execution, the total number of cores was equal - only the number of nodes (and cores per node) was changed. Execution time for 64 files (90 GB) is lowest for 16 nodes what is not apparent due to networking costs. However, it is relevant to mention that despite homoge-

nous architecture, execution time may vary significantly for the same configuration as Spark may work unpredictably and thus may result in an error as in Figure 25. It is also important for future users, who will need to take this into consideration.

4. Work organization

This chapter provides a comprehensive overview of the information about the work organization in the team, duties and responsible authors, and project phases. It also presents information on the tools and techniques utilized in the project management.

4.1. Team structure and general work organization overview

The project in the scope described in this text was realised in the small team of two people. Most of the work, due to its often exploratory nature and small size of the team, was shared between both team members with many and often swaps and replacements, which strongly extended the perspective for each issue. The general division of the work carried out in the course of the project could look like that:

- Both team members were responsible for the following tasks:
 - acquiring necessary knowledge about tools used in the project,
 - recreating the steps to run ROOT applications on the Prometheus cluster,
 - researching possible alternatives for the tools,
 - meeting with the CERN team,
 - an initial analysis of the tests and measurements,
 - cooperating on the analysis code development,
 - conducting performance tests,
 - analysing tests results,
 - creating reports and presentations.
- Aleksandra Mnich was responsible for the following tasks:
 - developing methods and scripts to copy files from CERN's EOS storage to the Helix Nebula Cloud and Prometheus cluster,
 - performing and documenting the results of the tests,
 - contributing to the project documentation,
 - developing the Scala script.
- Miłosz Błaszkiwicz was responsible for the following tasks:
 - creating the project's documentation,
 - executing the performance tests,
 - contributing to the Scala script,
 - coordinating reporting activities.

Our small team joined a larger team at CERN focused on cloud computing research. Responsibilities and tasks often overlapped or required direct work with the members of the CERN team. Thanks to the exceptional arrangement, we have had a fantastic chance to work with the team hand-by-hand at CERN for three months. Apart from that time, we held every month a video conference to share updates on the project with the CERN team.

4.2. Project phases

The project can be divided into four distinct phases. Below, a brief description of the activities in each of them can be found.

4.2.1. February to March

First, the initial stage of the project began in February 2018 and lasted for two months. During that time, we formed our team, completed formalities regarding the commencement of the thesis project and quickly began researching potential tools we were going to use. Our top priority at the moment was to get an early understanding of the project scope and of the defined tools we were going to use later. Both goals were achieved by learning on our own with the help of the project supervisor offered during several meetings we held in that time. This self-study period also included the use of the online courses, of which most substantial was “Big Data Analysis with Scala and Spark” [15] by Dr. Heather Miller available on coursera.org. This course was recommended to us as the first step we should take to get tutorial-like experience with Apache Spark and methods of performing analysis using this tool. Another milestone was receiving access to the Prometheus cluster, which allowed us to recreate steps taken by our colleagues a year before.

4.2.2. April to June

With the beginning of April, the project moved swiftly to the second phase, which in turn lasted three months, until the holidays. The structure of the work in this phase was dictated by two types of regular meetings regarding the project: videoconferences with the CERN team, held once a month, and Project Laboratory classes, also held once a month. With the start of this phase, we received a first High Energy Physics analysis written in C++, which we started to review and access to the CERN’s computing infrastructure. This development meant that actual work could begin, and soon we were exploring CERN’s computing services and preparing a limited, very early, proof-of-concept-type HEP analysis which could be run with Spark. In the meantime, other general experiments with Spark contributed to our knowledge of the framework and platform.

4.2.3. July to September

Definitely, a central phase of the project began in July, when we joined the CERN team in Geneva. During the following three months, our tasks focused on five main areas:

- Analysis translation to the RDataFrame form and development of the Jupyter notebook

- Code profiling
- Infrastructure development
- Analysis translation to Scala

Translation to the `RDataFrame` form was done in cooperation with other members of the CERN team. A most prominent feature developed by our team was a selection mechanism for input files. We also developed some other elements, which were then maintained in the code or abandoned. This part ended with the creation of the notebook, which is a target form for future analyses.

The most interesting part in terms of results, yet quite tedious at times, was devoted to the profiling of the code. Much work was put in order to get a better understanding of the analysis behavior and performance in the distributed environment. Numerous runs and measurements, both from the general perspective of the whole system and of a singular worker, led us to the quite impressive results, which were described at length in chapter 3. This part was also a base to contributing to the infrastructure development, as the results gathered at this point could have indicated potential issues on the technical side and were collected to serve better computing services in the future.

The last area of interest, the translation to Scala, was our own initiative, which goal was to provide the same analysis in another distributed form, which could be compared with the previous approach. Work on this part started in September and continued into the last phase of the project.

The summary of the activities in this period can be found in the presentation of our work at the TOTEM experiment this summer, created for the TOTEM Collaboration Meeting on 11 September, attached to this text in Appendix D.

4.2.4. October to December

The final phase of the project lasted for two months. This period was devoted to finishing the tests, analyzing results and performing two major tests with increased capacity. Unfortunately, cloud services stopped abruptly in October and were resumed at the end of November, which caused quite a severe delay at the last moments of the project. Due to these issues, we could not finish all of the intended goals; however, those works are expected to be continued in the following year with the next projects.

4.3. Project management practices and tools

In different phases, different methods of work were utilized. In the first two phases, and also in the last phase, our routine was dictated by the meetings, video conferences with the CERN team and, in a smaller degree, Project Laboratory (Pracownia Projektowa) classes. During each of the videoconferences, we discussed updates on both sides and decided on broad directions of the work. Nature of the tasks performed in this periods allowed us a little bit of flexibility, so not so often a detailed specifics of the tasks were discussed.

Things were a little different in the middle phase, at the time we were at CERN. Meetings with the team on average took place twice a month, but they were accompanied by much more

frequent talks and meetings with individual members of the team, who were involved in our day-to-day actions in given time.

At CERN we also held weekly meetings with our supervisor at the TOTEM experiment. These meetings were devoted entirely to report updates on the progress we made in the preceding week. Presentations prepared for those meetings are attached in Appendix E. The final presentation, which summarised our entire cooperation at CERN during summer months is attached in Appendix D.

During the intermediate phase, we had an opportunity to work desk-by-desk every day, so we shared our progress with each other much more frequently. It was an excellent advantage whenever one of us was stuck at any point — all the time we could consult the other person and possibly move forward as fast as possible. Additionally, as mentioned before, our standard practice was to swap tasks very often, so that we could check if our approach is the best and most efficient. The daily cooperation allowed us to be flexible in this matter, as within minutes we could brief another person on the state of the problem and try to come up with another solution.

For a short time, there was an attempt to use ClickUp for tasks management. Our approach was at first rather enthusiastic, yet the complexity and possible options outweighed negatively possible positive outcomes of this tool. Our team was simply too small, and our daily, direct cooperation made it completely obsolete and useless. In two weeks it was only adding more work to manage tasks created there, so shortly after we decided to abandon this tool.

Tools utilized in the course of the project:

- Slack

Tool for group communication with file sharing provided as a free cloud service with an available paid version. Has all of the standard ways of organizing a group chat, like channels, message threads, and moderation capacities. Used by our small team of two to communicate with the project supervisor.

- Mattermost

Alternative to Slack and used by the CERN team, it is said to be more private than the competitor and shared as an open-source application. Mattermost had been selected by CERN and is provided as an internal service.

- Electronic mail

A classic way of exchanging messages, in view of the multitude of possibilities it was the most reliable way of messaging other people. We also used a mailing list to distribute the most important messages with the CERN team.

- Sharelatex and Overleaf

Both platforms are listed together because they were merged in the period of writing this thesis. Overleaf proved to be a primary solution for LaTeX documents collaboration in real-time and was used as a text editor for the scientific papers.

- Google Hangouts

Video conference service available alongside Google's Gmail. Easy to use and efficient,

comes preinstalled on Android devices and not too complicated. We took advantage of this solution for quick chats or as an alternative to Vidyo.

- Vidyo

Video conferencing software in use at CERN, alternative to Skype Business. Apart from real-time group video conversations, provides users with other possibilities, like multi-screen sharing, meeting room management or integration with Slack. Available for mobile and desktop platforms alike. In our project, this was the best choice for video conferences from the CERN facilities, possibly with multiple connecting participants, who required decent screen-sharing capabilities.

- Indico

CERN's internal tool; serves as a meeting management service, which allows both planning and managing them. Allows for easy access to the meetings' documents and materials and also provides simple integration with Vidyo.

5. Project results

The primary goal of our work was to perform and test interactive way of analyzing data from High Energy Physics experiments using Apache Spark. The particular scope of the project, described as the project's vision, was completed successfully and thoroughly. The outcomes led to many valuable conclusions and allowed to identify possible advantages, disadvantages, problems, and directions for future development.

Based on the presented work, an conference publication and a poster were created. The abstract can be found in Appendix A and poster in Appendix B.

In general, the matter of distributed computing for CERN experiments still presents many questions that will need to be answered, even within this particular approach. Some of the problems identified here have already led to proper actions and fixes, and some others will need further investigations. The exemplary application of the Big Data software to the analysis done before only in single-core environment presented new possibilities for rapid exploration of massive data sets.

5.1. TOTEM analysis with RDataFrame in a Jupyter Notebook

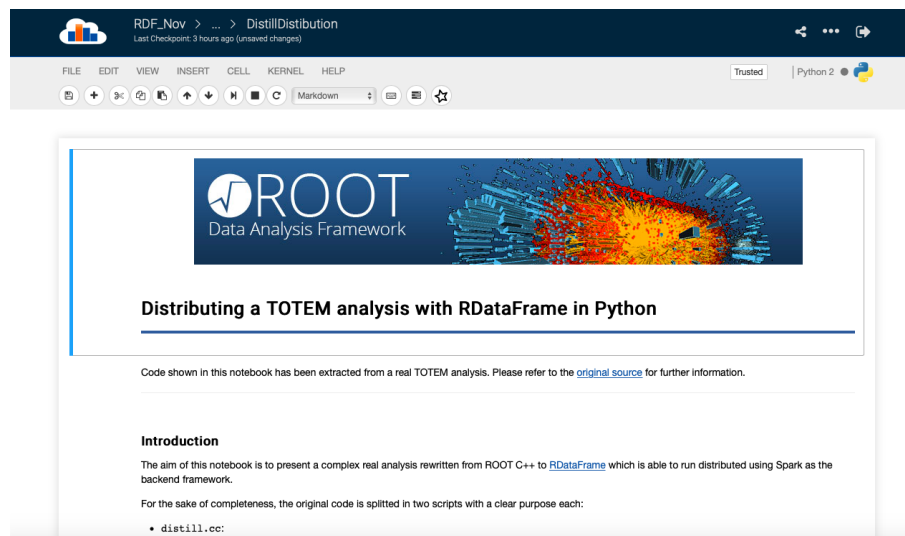


Figure 26: Jupyter notebook which is the basic result of the project.

Foremost step, which was needed in order to continue with the objectives of the effort, was to create an analysis code which would use the RDataFrame model. This objective was completed in cooperation with other CERN team members, who had possessed far more advanced experience with the ROOT framework.

The analysis application fulfills all the requirements set before. Multiple tests confirmed that the application is reliable and can be used on any cluster with Spark and ROOT frameworks. Discussions with potential users suggest that the use is relatively simple for them and similar to the tools they were using earlier. An output may be returned in the same form as previously. Finally, except ROOT files, other sources of data are also supported (e.g., CSV files) and depend on the RDataFrame support.

The significant characteristic of this analysis code is the application of the RDataFrame model. In the time of our work, RDataFrame was an innovation introduced in the newest release of ROOT and was still under heavy development. The concept is in general similar to the data frames known from other languages like Python or R and provides a high-level interface to work with data in ROOT format. Generally, it introduces a bit of a functional approach in the mostly iterative analyses done in C++ ROOT giving users interesting possibilities, like implicit multithreading. Its purpose was defined precisely for what we need it: to perform High Energy Physics computations in a way which could be possible also with the Spark cluster. As a result, three possible execution paths are available (fig. 27).

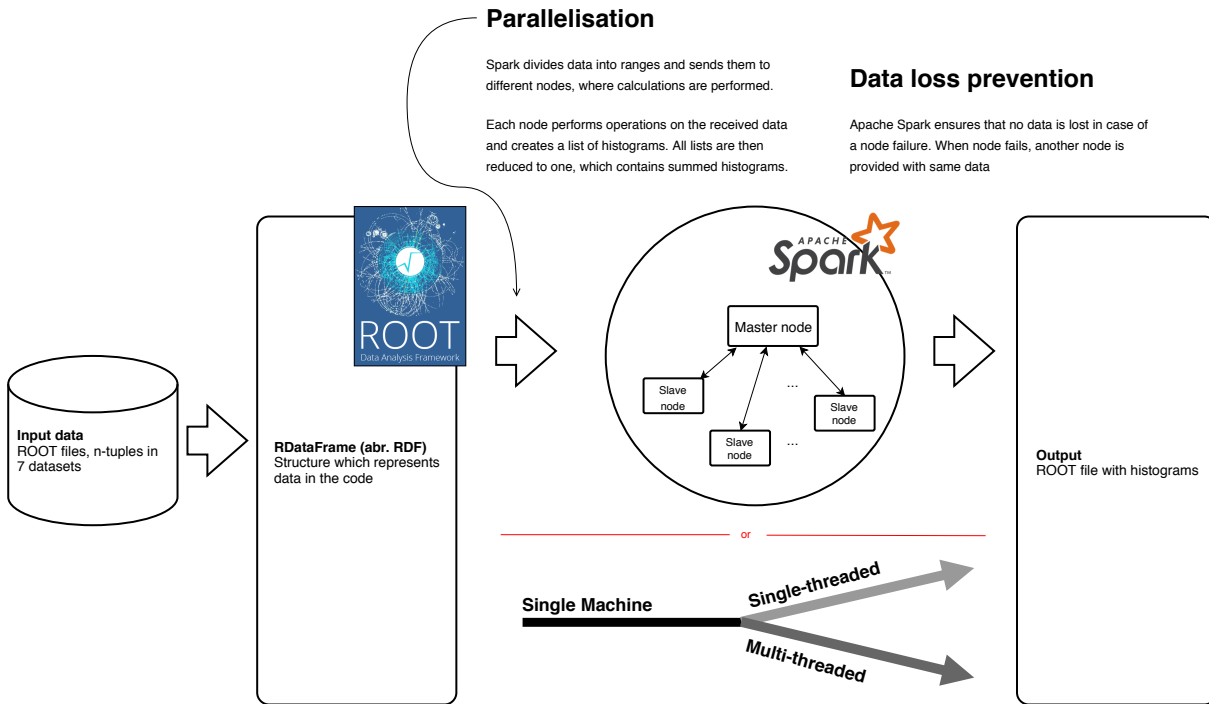


Figure 27: Three execution modes of the RDataFrame application

5.2. Profiling and testing conclusions

Findings on the performance of Elastic Scattering analysis parallelization in RDataFrame are the central theme of our work. The prepared analysis which works with Spark was the first step. Then it was necessary to find out how should we utilize the resources in order to make it as effective as possible. Achieving this goal included numerous tests and accurate reporting so that the progress on this point could take advantage of the cooperation with the CERN experts.

A significant number of reports and presentations were made in the course of our work. Those include extensive spreadsheets with measurements results, presentations for the weekly meetings in TOTEM during summer, reports, and summaries of the tests on the HN in November and December and summary presentation for the TOTEM Collaboration Meeting in September.

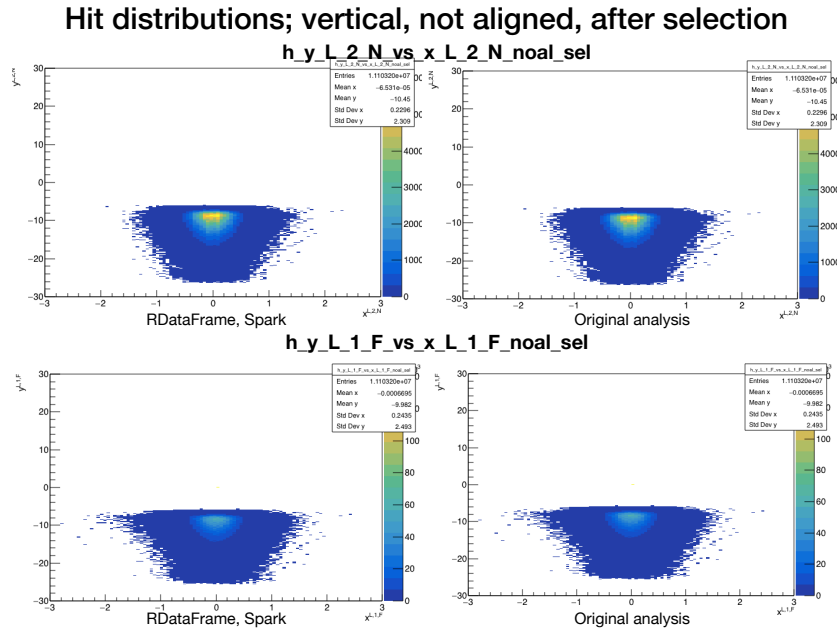


Figure 28: Comparison of the histograms resulting from the original analysis and the one recreated with RDataFrame

Without a doubt the application is scalable only up to a certain point, which exact position depend on additional settings. Further studies of the matter are required to push the scalability limits further, and initial steps were described in section 3.3.

In Appendix E, the first half of one of the presentations is devoted entirely to the results validation (one of the slides, displaying exemplary histograms, is presented in figure 28). This presentation features a comparison of a selection of histograms produced by the new analysis and the ones produced by the original analysis in the C++ ROOT model. One of the slides brings attention to the striking discrepancy in three out histograms, unearthed in the course of this validation. Details of the issue and description of its solution are discussed at length in the 3.3.

5.3. TOTEM Analysis in Scala

Scala version of the analysis was created for comparative purposes. It represents a different approach to distributed computations for HEP analyses. Differences described in the previous chapters were expected to give interesting insights about the potential advantages of the approach and also to provide us with a valuable reference point.

Taking into account all the gathered performance data and amount of time required to write analysis from scratch in Scala (comparing to reuse already written header file in RDataFrame approach) it seems that such approach may not be the most effective in this particular example. The original analysis is highly optimized and tailored for the ROOT framework. In the case of RDataFrame approach, Spark is used mainly as a scheduler which splits the data and distributes it along the nodes. Then each executor runs C++ / ROOT code. This way, Spark minimizes overheads and benefits from physics data tailored ROOT optimizations. On the other hand, pure Scala-based Spark solution has some other advantages: an application written in Scala operating

Scala	DistROOT + RDataFrame
Does not require additional frameworks (only an additional package)	Requires ROOT and specific knowledge to work in this particular framework
Uses standard Spark structures	Allows to use a range of tools from the ROOT framework and to maintain parts of the original code (in case of porting an analysis from any previous sources)
Easy deployment and monitoring using Spark monitoring features (SparkUI)	Spark used as a scheduler which splits the data and distributes it on the nodes. Each executor runs C++ / ROOT code.
Optimizing Spark is subject discussed multiple times and tools are better prepared to handle this task.	Benefits from lesser overheads due to the structure of necessary translations.
Platform enjoys huge interest and popularity, therefore have strong community support. However, the HEP aspect is rather underrepresented.	Better equipped to suit HEP data analyses with proper optimizations and lively community. Conversely, distributing computations is largely in a beginner stage.
Platform is currently an established standard and might be more appealing to new users with experience outside HEP	ROOT framework is developed and used in CERN for years, with its resources and ready analyses many current users find it to be a supreme tool for HEP computations

Table 1: Comparison of advantages and disadvantages of two versions of the distributed analysis

only on Spark internal data structures is fully independent from the infrastructure and does not require any specific framework, which installation might be an additional effort. It can be easily deployed and monitored by Spark monitoring (SparkUI) as event data are stored explicitly in RDDs.

5.4. Conclusions

The project proved that application of the Apache Spark, a widely-approved tool from the big data domain, together with advanced developments in the CERN's frameworks, brings new capabilities and perspectives to the analysis of data from High Energy Physics in CERN's experiments. A combination of tested solutions and open-source platforms allows avoiding the loss of the advantages originating from tools built for years for the HEP community and also to utilize the power of modern approaches to computing. Processing of massive amounts of data, once a nearly exclusive domain of CERN, has become a more common problem. Finding a way to profit from the developments in Big Data and Cloud computing might be a crucial aid in the face of looming challenges for the LHC experiments.

Many insights acquired in the course of the project helped to assess the potential gains and obstacles for performing such analyses in the future. Nevertheless, future efforts will be necessary to introduce the solution presented in this thesis to a broader audience.

We do hope that our contribution will help to make a difference and right choices in this field and will lead to more effective and faster computing, which in effect will open real new possibilities concerning CERN scientists work.

Appendices

A. Abstract of the poster paper for 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)

On the next page abstract for a poster paper can be found.

Big Data Tools and Cloud Services for High Energy Physics Analysis in TOTEM Experiment

Valentina Avati*, Milosz Blaszkiewicz*, Enrico Bocchi[†], Luca Canali[†], Diogo Castro[†], Javier Cervantes[†], Leszek Grzanka*, Enrico Guiraud[†], Jan Kaspar[†], Prasanth Kothuri[†], Massimo Lamanna[†], Maciej Malawski*, Aleksandra Mnich*, Jakub Moscicki[†], Shravan Murali[†], Danilo Piparo[†], Enric Tejedor[†]

[†]AGH University of Science and Technology, Krakow, Poland, Email: {grzanka,malawski}@agh.edu.pl

* CERN, CH-1211 Geneva 23, Switzerland, Email: {first.last}@cern.ch

Abstract—The High Energy Physics community has been developing dedicated solutions for processing experiment data over decades. However, with recent advancements in Big Data and Cloud Services, a question of application of such technologies in the domain of physics data analysis becomes relevant. In this paper, we present our initial experience with a system that combines the use of public cloud infrastructure (Helix Nebula Science Cloud), storage and processing services developed by CERN, and off-the-shelf Big Data frameworks. The system is completely decoupled from CERN main computing facilities and provides an interactive web-based interface based on Jupyter Notebooks as the main entry-point for the users. We run a sample analysis on 4.7 TB of data from the TOTEM experiment, rewriting the analysis code to leverage the PyROOT and RDataFrame model and to take full advantage of the parallel processing capabilities offered by Apache Spark. We report on the experience collected by embracing this new analysis model: preliminary scalability results show the processing time of our dataset can be reduced from 13 hrs on a single core to 7 mins on 248 cores.

I. INTRODUCTION

Researches and scientists in the field of High Energy Physics (HEP) typically perform massive data analysis in two stages: first, they aggressively reduce the amount of information to be processed by filtering the source dataset; second, they run the final steps of the analysis on the reduced dataset using local computing resources, e.g., their laptop or a computing cluster. The goal of this work is to allow scientists to perform analysis on much bigger datasets with interactive or quasi-interactive response times so to let them use different filtering parameters and enable the exploration of new physics.

To achieve this, we explore the use of modern Big Data tools and we deploy them on the elastically-provisioned infrastructure of the Helix Nebula Science Cloud (HNSciCloud) [1], which includes commercial cloud service providers connected to existing HEP computing centers in a hybrid cloud model using the GEANT network. We also explore new parallelisation techniques at the application level by combining the following software ingredients:

- Analysis code from the TOTEM experiment;
- ROOT Data Analysis Framework [2];
- Science Box [3] services – EOS, CERNBox and SWAN;
- Apache Spark task distribution layer.

We compare the performance of this environment with the environment where TOTEM data analysis takes place

currently. In this paper we evaluate data processing and parallelisation aspects of the system. Evaluation of other aspects, such as cloud infrastructure, storage services and integration with end-user work environment, is left for future publications.

II. RELATED WORK

Offline data analysis is currently performed using a highly-optimized ROOT framework [2], which is customized for the needs of HEP physicists. As a comprehensive and tailored solution, it has become the de-facto tool in the CERN physics community. Nonetheless, there are limitations that constrain the way the analysis is performed. Both the lack of parallelization capabilities and the deficiencies imposed by the execution on a single machine lead to the inability of working with large datasets in an interactive way. New additions to the ROOT framework such as the RDataFrame, scrutinized as a part of this ongoing effort, solve part of the former problem.

Apache Spark is a popular, open-source framework for the distributed computing. Its Spark DAG and task schedulers features allow the deployment of map-reduce-type operations in a scalable way on large clusters. In this work, we leverage these features to parallelize and scale ROOT jobs across a cluster. A different approach consists of reading ROOT files into Spark using the Spark-Root connector developed by the CMS Big Data project [4]. This approach, however, requires data processing jobs to be written using native Spark Dataframe APIs, as opposed to using ROOT APIs, as in the rest of this work.

III. ARCHITECTURE

The architecture of our deployment on the HNSciCloud is shown in Fig. 1 and it involves the following components:

- EOS [5] is a distributed storage system used to host all physics data at CERN. A dedicated EOS instance storing a subset of the TOTEM experiment data is deployed on the HNSciCloud, allowing for fast data access from the computing nodes operating in the HNSciCloud.

- SWAN [6] is a web-based platform to perform interactive data analysis. It inherits the Jupyter notebook interface and integrates the ROOT analysis framework with the dedicated ROOT C++ kernel. In addition, it is capable of offloading massive computations to a Spark cluster, it uses CVMFS to access scientific software packages, and provides access to EOS emulating a local filesystem access.

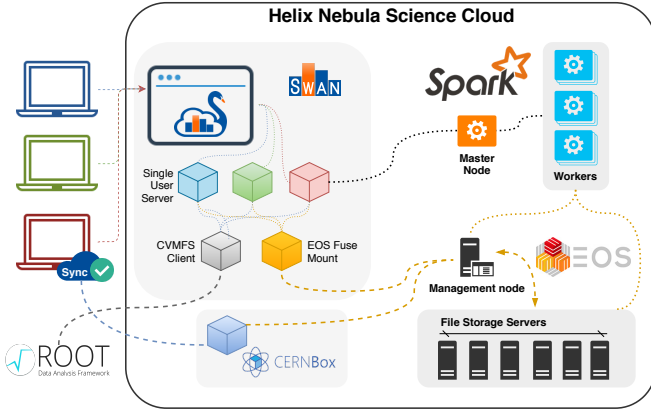


Fig. 1. Architecture and deployment of the distributed analysis components

- A dedicated Spark cluster is deployed on the HNSciCloud and is integrated with SWAN. Specifically, we make use of RDataFrame: a new interface of ROOT that allows the partitioning of the dataset in the form of DataFrame and uses Spark for spreading the computations across worker nodes.

- CERNBox [7] offers a web interface to manage files stored on EOS and allows for easy sharing of SWAN notebooks between users as well as for synchronization of selected folders with personal laptops.

All the Science Box services run in Docker containers orchestrated by Kubernetes, while Spark runs on plain VMs and is configured via Cloudera Manager. The overall deployment consists of 25 VMs, 388 CPUs, 1,450 GB of memory, and 21.5 TB of storage. 288 CPUs and 1,088 GB of memory are reserved for Spark, while 16.4 TB is the physical storage space available for EOS (the actual space available is 8.2 TB due to a replica 2 layout of the stored files).

IV. EVALUATION

The typical analysis scenario consists of three main steps:

- 1) Source datasets are copied from the storage infrastructure at CERN to EOS on the HNSciCloud;
- 2) Physics analysis is performed using the web-based interface of SWAN and the computing resources of Spark;
- 3) The produced results can be viewed interactively with SWAN, synchronized with personal laptops or shared with colleagues via CERNBox.

The physics analysis we used in evaluation is the analysis of the elastic scattering data gathered by TOTEM experiment in 2015 during a special LHC run with optics parameter β^* adjusted to 90 m. The dataset comprises 1153 files summing to 4.7 TB of data in ROOT Ntuple format, and stores 2.8 Billion events representing proton-proton collisions.

The original analysis was written using the ROOT framework and comprises 2 stages: 1. Data reduction, and 2. Filtering based on physics cuts. It followed a traditional approach of implementing a main processing loop which accesses individual events. The output is a set of histograms representing distributions of interesting observables.

The analysis was re-implemented using ROOT's RDataFrame [8], a new high level interface. It preserves the flexibility in the actions that can be performed inside the event-loop, while offering a concise declarative syntax. Implicit multi-threading and other low-level optimisations allow exploiting all the computing resources available transparently, while the data can be partitioned and distributed to multiple nodes by the Spark task distribution layer.

We performed preliminary experiments using a Spark cluster with up to 248 cores allocated to Spark workers. The collected results show that it is possible to substantially reduce the processing time of our 4.7 TB dataset from about 13 hours on a single core to less than 7 minutes on 248 cores.

V. CONCLUSIONS AND FUTURE WORK

In the first phase of pilot deployment we focused on validation of the results at the application level: we assured that the physics results obtained in the new system are correct and correspond to the known and validated results provided by the TOTEM collaboration. In the second phase we focused on optimization of processing: improving the RDataFrame implementation, fine-tuning the Spark cluster and understanding task distribution strategies.

The speedup observed in preliminary tests is promising: if further reduction of time is achieved, it should be possible to use the proposed approach for quasi-interactive analysis of the whole dataset, instead of multi-step batch processing.

Once the preliminary testing is completed, we plan to run more large-scale experiments to evaluate possible performance gains and to better understand the impact of parameters such as data partitioning, distribution or possible caching strategies. The results of this study will be of interest to wider physics community exploring the use of modern Big Data tools.

Acknowledgments: This work was supported in part by the Polish Ministry of Science and Higher Education.

REFERENCES

- [1] M. Gasthuber, H. Meinhard, and R. Jones, "HNSciCloud - Overview and technical challenges," *J. Phys. : Conf. Ser.*, vol. 898, no. 5, p. 052040. 5 p, 2017. hNSciCloud is co-funded by the European Commission under grant 687614. [Online]. Available: <http://cds.cern.ch/record/2297173>
- [2] I. Antcheva *et al.*, "ROOT: A C++ framework for petabyte data storage, statistical analysis and visualization," *Comput. Phys. Commun.*, vol. 180, pp. 2499–2512, 2009.
- [3] CERN. (2018) Science Box. [Online]. Available: <https://sciencebox.web.cern.ch>
- [4] M. Cremonesi *et al.* Using big data technologies for hep analysis. CERN. [Online]. Available: https://indico.cern.ch/event/587955/contributions/2937521/attachments/1684310/2707721/hep_bigdata.pdf
- [5] A. Peters, E. Sindrilaru, and G. Adde, "EOS as the present and future solution for data storage at CERN," *J. Phys.: Conf. Ser.*, vol. 664, no. 4, p. 042042. 7 p, 2015. [Online]. Available: <http://cds.cern.ch/record/2134573>
- [6] D. Piparo, E. Tejedor, P. Mato, L. Mascetti, J. Moscicki, and M. Lamanna, "SWAN: a Service for Interactive Analysis in the Cloud," *Future Gener. Comput. Syst.*, vol. 78, no. CERN-OPEN-2016-005, pp. 1071–1078. 17 p, Jun 2016. [Online]. Available: <http://cds.cern.ch/record/2158559>
- [7] L. Mascetti, H. G. Labrador, M. Lamanna, J. Moscicki, and A. Peters, "CERNBox + EOS: end-user storage for science," *J. Phys.: Conf. Ser.*, vol. 664, no. 6, p. 062037. 6 p, 2015.
- [8] E. Guiraud, A. Naumann, and D. Piparo, "TDataFrame: functional chains for ROOT data analyses," Jan. 2017. [Online]. Available: <https://doi.org/10.5281/zenodo.260230>

B. Poster for 2018 IEEE/ACM International Conference on Utility and Cloud Computing

On the next page poster for the 2018 IEEE/ACM International Conference on Utility and Cloud Computing is presented.

Big Data Tools and Cloud Services for High Energy Physics Analysis in TOTEM Experiment

Valentina Avati[†], Milosz Blaszkiewicz[†], Enrico Bocchi*, Luca Canali*, Diogo Castro*, Javier Cervantes*, Leszek Grzanka[†], Enrico Guiraud*, Jan Kaspar*, Prasanth Kothuri*, Massimo Lamanna*, Maciej Malawski[†], Aleksandra Mnich[†], Jakub Moscicki*, Shravan Murali*, Danilo Piparo*, Enric Tejedor*

[†]AGH University of Science and Technology, Krakow, Poland, Email: {grzanka,malawski}@agh.edu.pl

* CERN, CH-1211 Geneva 23, Switzerland, Email: {first.last}@cern.ch

Motivation and Objectives

- **High Energy Physics:** LHC Experiments produce and process large amounts of data: expected growth to over ExaByte after upgrade
- **Current technologies:**
 - ROOT C++ analysis framework
 - batch processing on clusters and grids
- **New opportunities:** big data tools and clouds
 - Apache Spark as distributed data analysis framework
 - Helix Nebula Science Cloud
- **Goal:** evaluate new approaches for interactive data analysis in cloud using real data from TOTEM experiment

Components

- **SWAN:** web-based platform for interactive data analysis, based on Jupyter notebook interface and integrated with the ROOT analysis framework with the dedicated ROOT C++ kernel
- **EOS:** a distributed storage system used to host all physics data at CERN. A dedicated EOS instance storing a subset of the TOTEM experiment data is deployed on the HNSciCloud
- **Spark cluster:** deployed on the HNSciCloud and integrated with SWAN
- **CERNBox:** to manage and share files stored on EOS and SWAN notebooks between users; synchronization with personal laptops.

Approach: Interactive analysis using RDataFrame and Spark

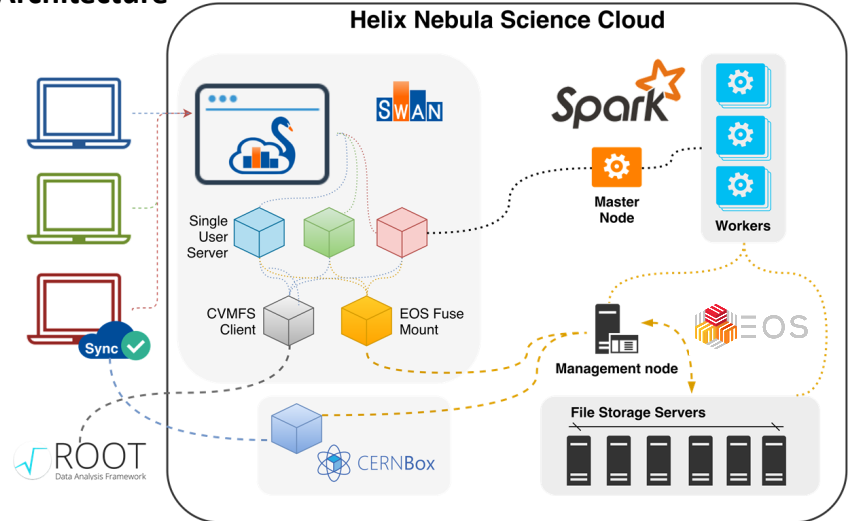
- **Original analysis: batch processing, two phases**
 - Distilling events of interest
 - Computing histograms
- **New analysis:**
 - Reimplemented in RDataFrame
 - Single phase on the full dataset
 - New, high-level API

```
df = RDataFrame(dataset)

df2 = df.Filter('x > 0')
        .Define('r2', 'x*x + y*y')

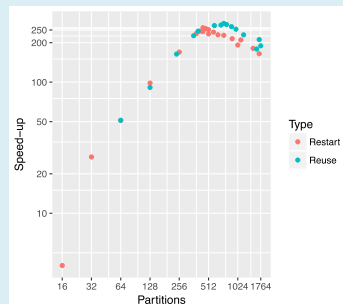
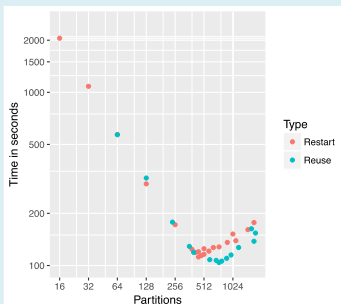
h = df2.Histo1D('r2')
g = df2.Graph('x', 'y')
```

Architecture



Evaluation

- **TOTEM Data:** collected during special run at LHC; sample rich of proton-proton elastic interactions (2015, 6500 GeV $\beta^*=90\text{m } 10\sigma$)
- **TOTAL:** 4.7 TB, 1148 files, 2 814 933 179 events (records)
- **Spark Cluster:** 57 nodes @ 32 cores & 128 GiB, 1824 vCPUs & 7296 GiB total
- **Configurations tested:** Spark executor reuse vs. restart



Conclusions

- **Cloud usage:** Successful migration of CERN software stack (SWAN, ROOT, EOS, Spark) to external cloud service
- **Data Analysis:** new programming model supports real data analysis from TOTEM experiment
- **Performance:** new tools enable nearly interactive data analysis on a full dataset
- **Results:** Computing time reduced from 8h to 1m45s, speedup of 280x

Future Work

- **Tests in multi-user environment:** to evaluate the performance and usability
- **Tests on larger sample (TOTEM and CMS common data):** to push scaling limits further
- **Use of new cloud technologies:**
 - Evaluation of public cloud providers
 - New cloud services (storage, serverless computing)

References

- TOTEM: <http://totem.web.cern.ch>
- ROOT: <https://root.cern.ch/>
- CERNBox: <https://cernbox.web.cern.ch/>
- SWAN: <https://swan.web.cern.ch/>
- EOS: <https://github.com/cern-eos/eos>
- Helix Nebula: <http://www.helix-nebula.eu>

This work is supported in part by the Polish Ministry of Science and Higher Education Grant no. MNiSW DIR/WK/2017/07-01, and by PLGrid Infrastructure.

C. Selected reports from the test runs in November and December

On the next pages reports from November and December tests can be found.

Running Distributed TOTEM Analysis with RDataFrame in Python on the Analytix Cluster (via swan.cern.ch)

In general, attempt to run the analysis code was successful, there are no problems at all, analysis worked as expected. Below, for reference, all of the configuration details and exemplary results from this single run.

Essentially:

- ✓ Same project (re-fetched from <https://github.com/JavierCVilla/RDataFrame-Totem>)
- ✓ Notebook DistillDistribution.ipynb (printout attached at the end)
- ✓ Running on swan006.cern.ch
- ✓ Analytix Cluster

SWAN session parameters:

Option	Values
Software stack	Development Bleeding Edge (might be unstable)
Platform	x86_64-slc6-gcc62-opt
Number of cores	2
Memory	8 GB
Spark cluster	Analytix

Spark Clusters Connection parameters:

Option	Value
spark.executor.cores	8
spark.executorEnv.KRB5CCNAME	./krbcache
spark.driver.memory	4g
spark.executor.instances	8
spark.yarn.dist.files	{KRB5CCNAME}#krbcache,./DistROOT.py,./common_definitions.h,./common_algorithms.h,./parameters.h,./parameters_global.h,./common.h,./common_cuts.h,./common_parameters.h,./generators.root
spark.executor.extraLibraryPath	{LD_LIBRARY_PATH}

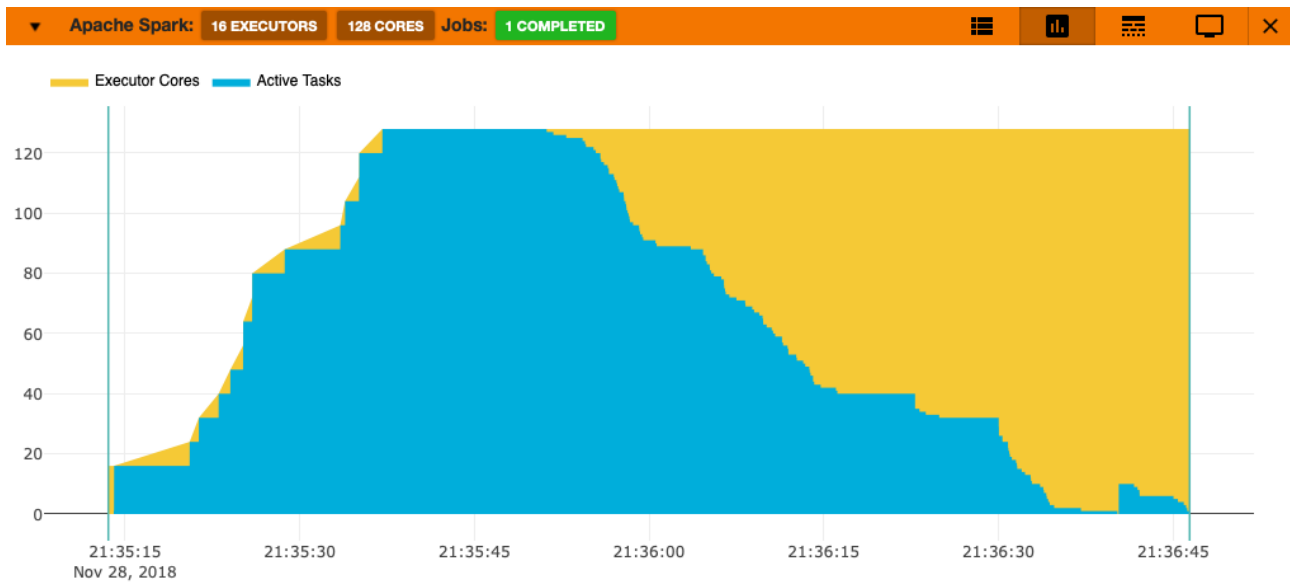
Analysis parameters

Option	Value
Selected dataset	DS1 (91GB) from EOS
Number of partitions	128

Results

Duration: 1m:32s

Executors: 16, cores: 128 (dynamic allocation).





Tasks: 10 ithdp1109.cern				56						13	
				57							
				58							
				59							
				60							
				61							
				62							
				63							
Tasks: 8 ithdp1104.cern				64							
				65							
				66							
				67							
				68							
				69							
				70							
				71							
Tasks: 14 ithdp1104.cern				72						132	
				73							
				74							
				75							
				76							
				77							
				78							
				79							
Tasks: 18 ithdp1106.cern				80							
				81							
				82							
				83							
				84							
				85							
				86							
				87							
Tasks: 9 p05151113126				88							
				89							
				90							
				91							
				92							
				93							
				94							
				95							
Tasks: 16 p05151113126				96						12	
				97							
				98							
				99							
				100							
				101							
				102							
				103							
Tasks: 17 p05151113459				104						130	
				105							
				106							
				107							
				108							
				109							
				110							
				111							
Tasks: 11 p05151113459				112						129	
				113							
				114							
				115							
				116							
				117							
				118							
				119							
Tasks: 12 p05151113550				120							
				121							
				122							
				123							
				124							
				125							
				126							
				127							

Repeated tests at the Helix Nebula Cloud with updated DistROOT

Another series of tests on the Helix Nebula Cloud after the break.

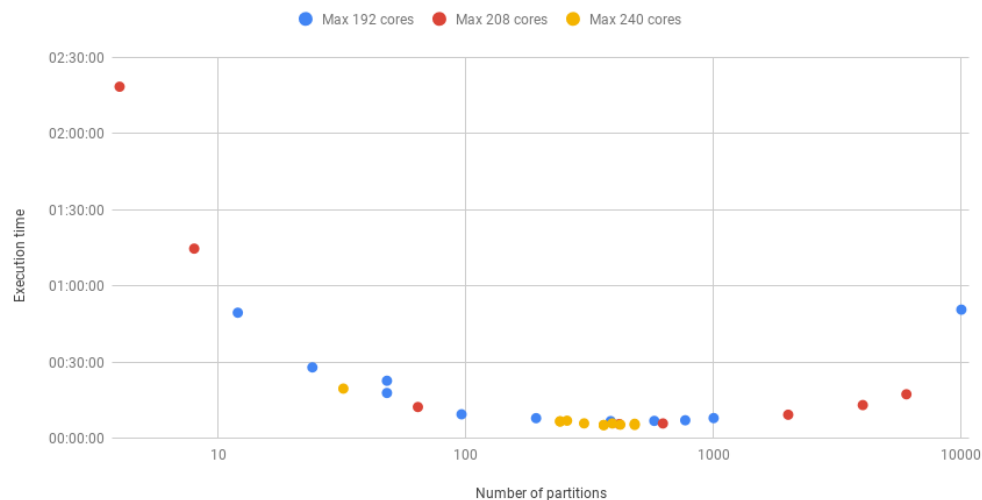
Details of the configuration:

- ✓ 256 cores available (theoretically),
- ✓ project updated from <https://github.com/JavierCVilla/RDataFrame-Totem>,
- ✓ Bleeding Edge software stack
- ✓ Data reading remotely from EOS
- ✓ Using all 7 datasets (4.7 TB)

Key take-away is the standard execution time plot:

Spark execution time

Helix Nebula, 7 datasets, 192 - 240 cores

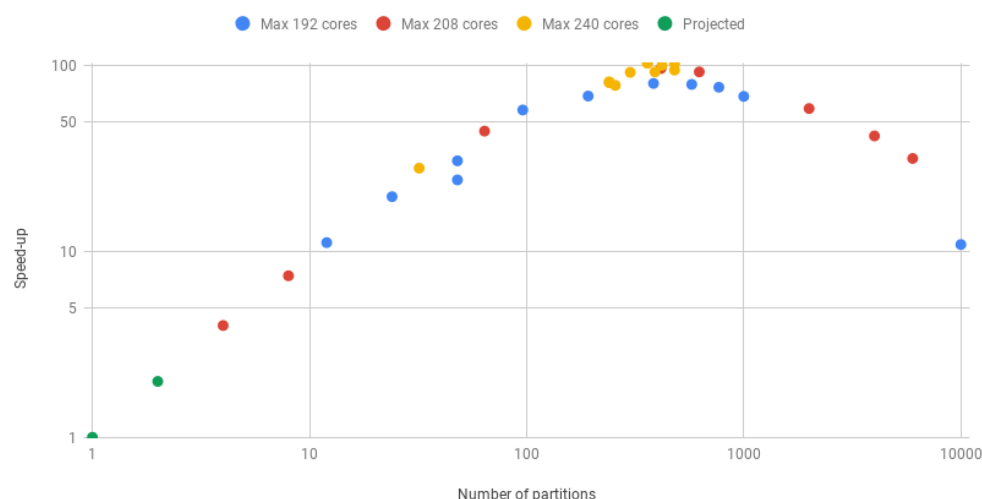


Maximum number of cores allocated to the application differed between the runs, and that is why results are presented in three separate series.

Unfortunately, there was no time to measure a one core execution, and therefore the plot is simply time vs partition. However, execution in the first phase scaled quite correctly previously, so if we assume that the execution time for 2 cores is twice as long as for 4, and similarly for one, then classic speed-up plot will look like that:

Speed-up plot

Helix Nebula, 7 datasets, 192-240 cores



Measured time explanation

The notebook cell which activates Spark computations includes also the preparation of the distributed tree. Time of this part is not included in the numbers above, and was more or less constant: for this datasets it takes each time roughly 3 minutes before Spark begins.

The table indicates also whether the dynamic allocation was enabled for each run. To accelerate testing routine a little, dynamic allocation was initially enabled. In practice it meant that during first seconds of the Spark activity, workers were being allocated and did not start all at once. Duration of this phase depends on the number of needed cores, and in general is a matter of seconds. Workers start their jobs as soon as they are allocated, independently from each other.



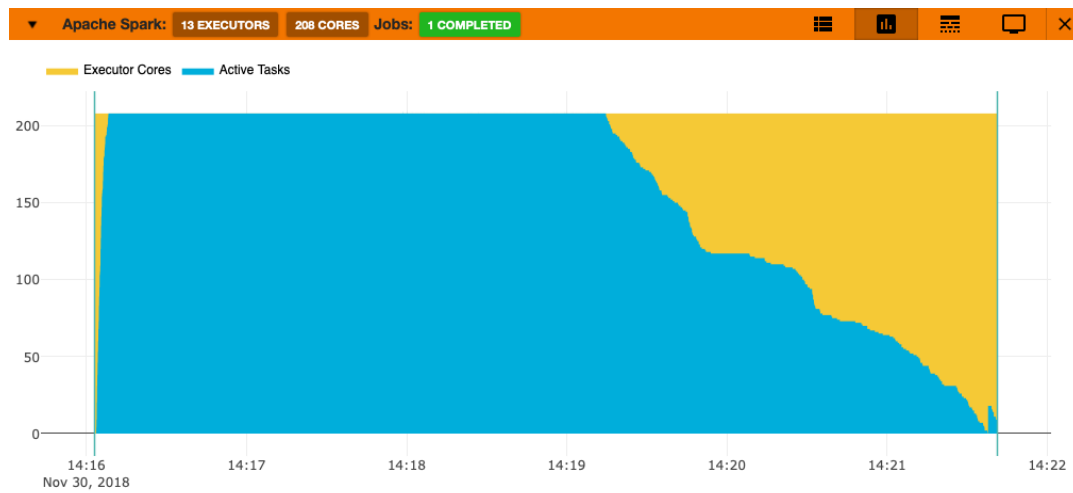
Times presented in the table and on the graphs in this tests are covering all Spark computations (orange field), including dynamic allocation if was enabled.

Helix Nebula TOTEM RDataFrame analysis tests

No	Time	Max Cores	Max Executors	Partitions	Partitions vs cores	Dynamic Allocation
1	00:50:45	115	23	10000	86,96	TRUE
2	00:08:05	192	12	1000	5,21	TRUE
3	00:08:03	192	12	192	1,00	TRUE
4	00:06:53	192	12	384	2,00	TRUE
5	00:07:13	192	12	768	4,00	TRUE
6	00:06:58	192	12	576	3,00	TRUE
7	00:09:34	176	11	96	0,55	TRUE
8	00:22:46	64	4	48	0,75	TRUE
9	00:28:02	64	4	24	375,00	TRUE
10	00:49:33	32	2	12	375,00	TRUE
11	00:17:58	48	3	48	1,00	TRUE
12	00:07:03	240	15	256	1,07	TRUE
13	02:18:29	32	2	4	125,00	TRUE
14	01:14:46	16	1	8	0,50	TRUE
15	00:09:24	208	13	2000	9,62	TRUE
16	00:13:12	208	13	4000	19,23	TRUE
17	00:05:43	208	13	416	2,00	TRUE
18	00:05:58	208	13	624	3,00	TRUE
19	00:17:27	208	13	6000	28,85	TRUE
20	00:12:26	112	7	64	0,57	TRUE
21	00:06:45	240	15	240	1,00	TRUE
22	00:05:50	240	15	480	2,00	TRUE
23	00:05:23	240	15	480	2,00	FALSE
24	00:06:49	240	15	240	1,00	FALSE
25	00:05:21	240	15	360	1,50	FALSE
26	00:05:30	240	15	420	1,75	FALSE
27	00:06:00	240	15	300	1,25	FALSE
28	00:05:58	240	15	390	1625,00	FALSE
29	00:05:20	240	15	360	1,50	FALSE
30	00:19:41	240	15	32	0,13	FALSE

Example of Spark UI screens

Execution without dynamic allocation, 360 partitions, total time: 5m:38s.



Summary of the first tests on Helix Nebula after resources extension

Spark configuration:

- spark.dynamicAllocation **false,**
- spark.executor.memory **60g / 30g,**
- spark.executor.instances **49 / 98,**
- spark.executor.cores **15 / 16 / 24 / 30,**

Environment:

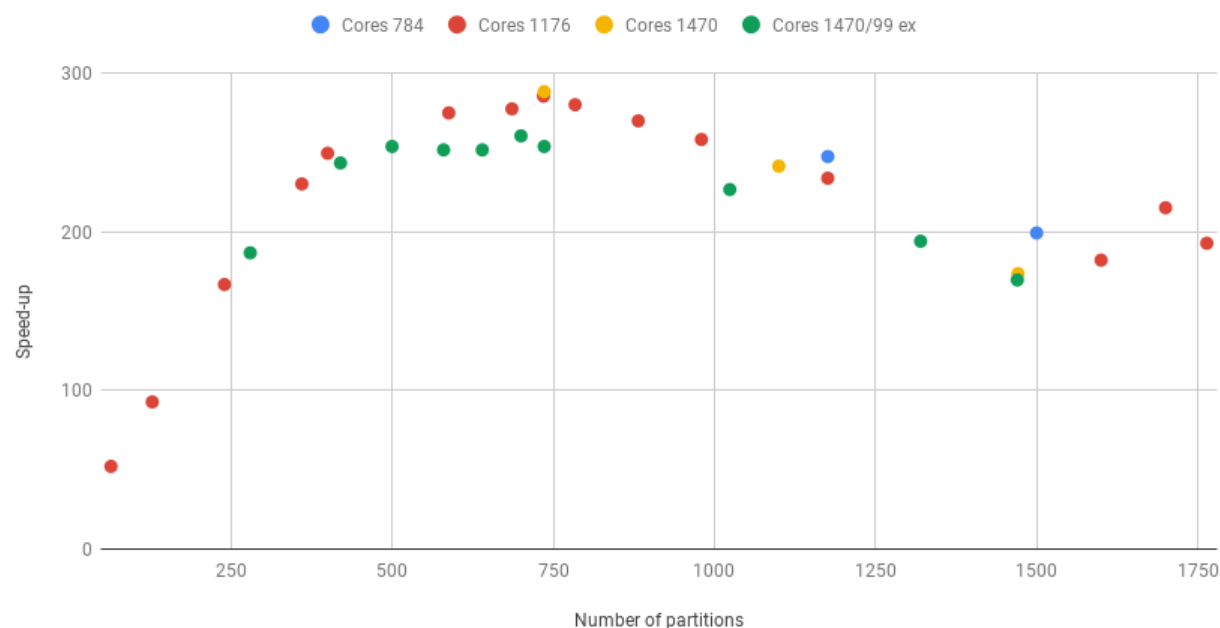
- Software stack **Development Bleeding Edge,**
- Spark cluster **Totem HN**

DistillDistribution notebook from RDataFrame-Totem

General findings

Spark Speed-up

Helix Nebula, 7 datasets (4.7 TB)



Generally, it seems that maximum speed is attained using ca. 750 partitions on more than 750 cores. Using a greater number of partitions only makes the execution time longer.

Execution using 98 executors with 30 GB each seems to take longer time than 49 executors with 60 GB each (but this still needs additional checks).

For the tested configurations, all of the tests with more than 1750 partitions failed. Spark did not return error, but the reduction stage lasted unusually long (more than 20 minutes, while for tests with less than 1750 usually does not exceed 10 seconds at most). Spark UI screenshots are attached for the failed runs 25 and 32 in “Tests details” section below.

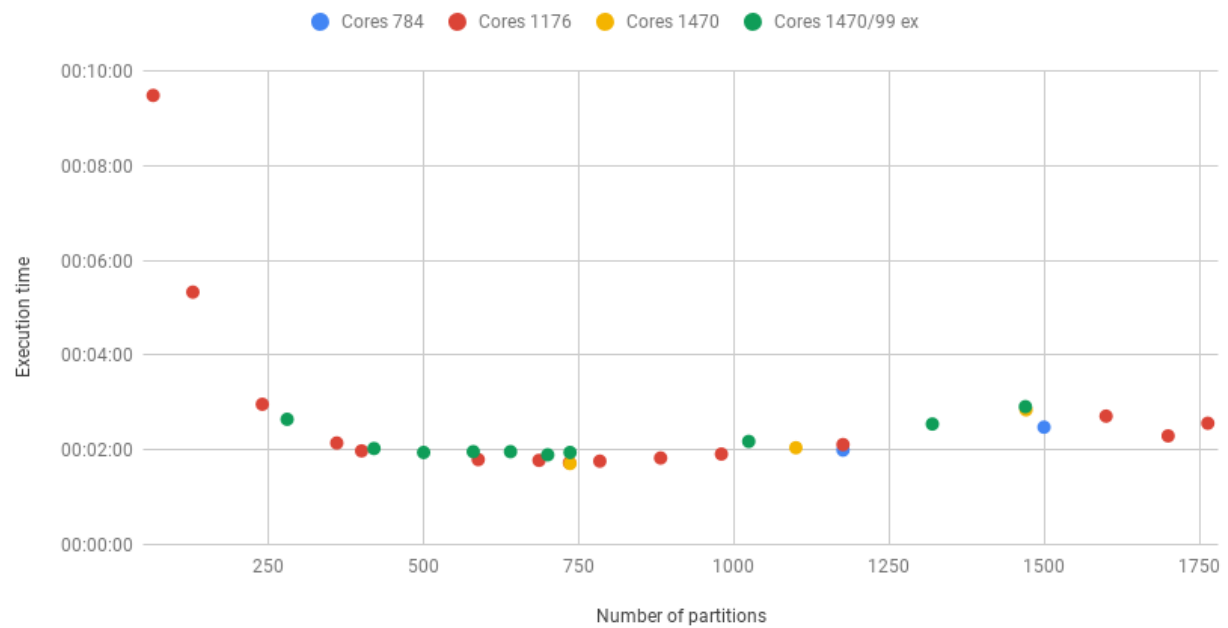
For the speed-up graph, the time of one core execution is 08:15:20, as measured recently (only once, so definitely still needs confirmation).

Overall testing sequence:

1. Tests on 784 cores available (tests 10 and 11)
2. Tests on 1176 cores available (tests 12 to 28)
3. Tests on 1470 cores available, creating 49 executors and 60 GB of memory per executor (tests 29 to 32).
4. Tests on 1470 cores available, creating 98 executors and 30 GB of memory per executor (tests 33 to 40)

Spark Execution Time

Helix Nebula, 7 datasets (4.7 TB)



Tests details

RUN 11.

Changes: -

Spark options: spark.dynamicAllocation **false**, spark.executor.memory **60g**,
spark.executor.instances **49**, spark.executor.cores **16**

Cloudera: 3.2 TiB (memory), 785 (VCores), 50 (Containers), 0 (Pending Containers)

Run	Cores	Executors	Partitions	Time
11	784	49	1176	00:02:00

RUN 12.

Changes: Only number of cores per executor to 24

Spark options: spark.dynamicAllocation **false**, spark.executor.memory **60g**,
spark.executor.instances **49**, spark.executor.cores **24**

Cloudera: 3.2 TiB (memory), 1177 (VCores), 50 (Containers), 0 (Pending Containers)

Run	Cores	Executors	Partitions	Time
12	1176	49	1764	00:02:34

RUNS 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 26, 27

Changes: Only number of partitions

Spark options: spark.dynamicAllocation **false**, spark.executor.memory **60g**,
spark.executor.instances **49**, spark.executor.cores **24**

Cloudera: 3.2 TiB (memory), 1177 (VCores), 50 (Containers), 0 (Pending Containers)

Run	Cores	Executors	Partitions	Time
13	1176	49	1176	00:02:07
14	1176	49	784	00:01:46
15	1176	49	588	00:01:48
16	1176	49	400	00:01:59
17	1176	49	980	00:01:55
18	1176	49	882	00:01:50
19	1176	49	686	00:01:47
20	1176	49	735	00:01:44
21	1176	49	360	00:02:09
22	1176	49	240	00:02:58
23	1176	49	128	00:05:20
24	1176	49	64	00:09:29
26	1176	49	1600	00:02:43
27	1176	49	1700	00:02:18

RUNS 25 & 28

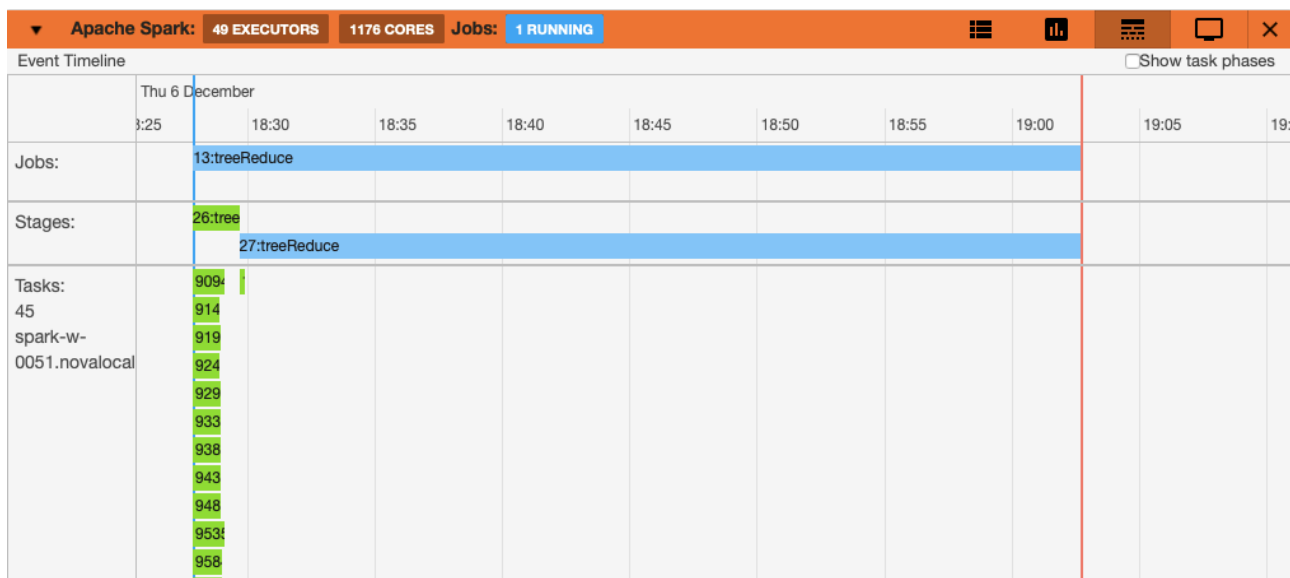
Changes: Numbers of partitions set to 1764 and 1800.

Spark options: spark.dynamicAllocation **false**, spark.executor.memory **60g**,
spark.executor.instances **49**, spark.executor.cores **24**

Cloudera: 3.2 TiB (memory), 1177 (VCores), 50 (Containers), 0 (Pending Containers)

Run	Cores	Executors	Partitions	Time
25	1176	49	1764	N/A
28	1176	49	1800	N/A

▼ Apache Spark: 49 EXECUTORS 1176 CORES Jobs: 1 RUNNING									
Job ID	Job Name	Status	Stages	Tasks	Submission Time		Duration		
▶ 13	treeReduce	RUNNING	1/2 (1 active)	1805 + 1 / 1806		34 minutes ago		-	



Spark UI for 25:

RUN 29.

Changes: number of cores per executor to 30 and number of partitions to 1470.

Spark options: spark.dynamicAllocation **false**, spark.executor.memory **60g**,
spark.executor.instances **49**, spark.executor.cores **30**

Cloudera: 3.2 TiB (memory), 1471 (VCores), 50 (Containers), 0 (Pending Containers)

Run	Cores	Executors	Partitions	Time
29	1470	49	1470	00:02:51

RUNS 30, 31

Changes: numbers of partitions.

Spark options: spark.dynamicAllocation **false**, spark.executor.memory **60g**,
spark.executor.instances **49**, spark.executor.cores **30**

Cloudera: 3.2 TiB (memory), 1471 (VCores), 50 (Containers), 0 (Pending Containers)

Run	Cores	Executors	Partitions	Time
	31	1470	49	1100 00:02:03

RUN 32.

Changes: Only number of partitions to 2000.

Spark options: spark.dynamicAllocation **false**, spark.executor.memory **60g**, spark.executor.instances **49**, spark.executor.cores **30**

Cloudera: 3.2 TiB (memory), 1471 (VCores), 50 (Containers), 0 (Pending Containers)

Run	Cores	Executors	Partitions	Time
	32	1470	49	2000 N/A

Spark UI for run 32:

<div> <div>▼ Apache Spark: 49 EXECUTORS 1470 CORES Jobs: 1 RUNNING</div> <div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> </div> </div>							
Job ID	Job Name	Status	Stages	Tasks	Submission Time	Duration	
▶ 3	treeReduce	RUNNING	1/2 (1 active)	2038 + 6 / 2044	10 minutes ago	-	

RUN 33.

Changes: executor memory, instances and cores.

Spark options: spark.dynamicAllocation **false**, spark.executor.memory **30g**, spark.executor.instances **98**, spark.executor.cores **15**

Cloudera: 3.2 TiB (memory), 1471 (VCores), 50 (Containers), 0 (Pending Containers)

Run	Cores	Executors	Partitions	Time
	33	1470	98	1470 00:02:55
	34	1470	98	736 00:01:57
	35	1470	98	1024 00:02:11
	36	1470	98	1320 00:02:33
	37	1470	98	500 00:01:57
	38	1470	98	640 00:01:58
	39	1470	98	700 00:01:54
	40	1470	98	580 00:01:58
	41	1470	98	420 00:02:02
	42	1470	98	280 00:02:39

Results of the tests

Number	Time	Max Cores	Max Executors	Partitions	Comments
1	00:02:29	784	49	1568	
2	00:02:07	784	49	784	
3	00:01:57	784	49	1176	
4	00:02:23	784	49	1584	
5	00:02:24	912	57	1584	
6	00:01:57	912	57	1368	executor.memory = 58g
7	00:03:32	1140	114	1450	
8	00:04:50	1450	145	1450	executor.memory = 10g
9	00:05:09	1450	145	1450	executor.memory = 20g
10	00:02:29	784	49	1500	
11	00:02:00	784	49	1176	
12	00:02:34	1176	49	1764	
13	00:02:07	1176	49	1176	
14	00:01:46	1176	49	784	
15	00:01:48	1176	49	588	
16	00:01:59	1176	49	400	
17	00:01:55	1176	49	980	
18	00:01:50	1176	49	882	
19	00:01:47	1176	49	686	
20	00:01:44	1176	49	735	
21	00:02:09	1176	49	360	
22	00:02:58	1176	49	240	
23	00:05:20	1176	49	128	
24	00:09:29	1176	49	64	
26	00:02:43	1176	49	1600	
27	00:02:18	1176	49	1700	
29	00:02:51	1470	49	1471	
30	00:01:43	1470	49	736	
31	00:02:03	1470	49	1100	
33	00:02:55	1470	98	1470	executor.memory = 30g
34	00:01:57	1470	98	736	executor.memory = 30g

35	00:02:11	1470	98	1024	executor.memory = 30g
36	00:02:33	1470	98	1320	executor.memory = 30g
37	00:01:57	1470	98	500	executor.memory = 30g
38	00:01:58	1470	98	640	executor.memory = 30g
39	00:01:54	1470	98	700	executor.memory = 30g
40	00:01:58	1470	98	580	executor.memory = 30g
41	00:02:02	1470	98	420	executor.memory = 30g
42	00:02:39	1470	98	280	executor.memory = 30g

D. Presentation "Distributed data analysis for the TOTEM experiment. Collaboration Meeting Report", 11 September 2018

Next pages contain slides from the presentation delivered on 11 September 2018 during the TOTEM Collaboration Meeting. The purpose of the presentation was to provide the experiment members with general overview of the project and advances produced to the date.

Distributed data analysis for the TOTEM experiment

Collaboration Meeting Report

CERN, 11 September 2018

Aleksandra Mnich, Miłosz Błaszkiwicz

Table of contents

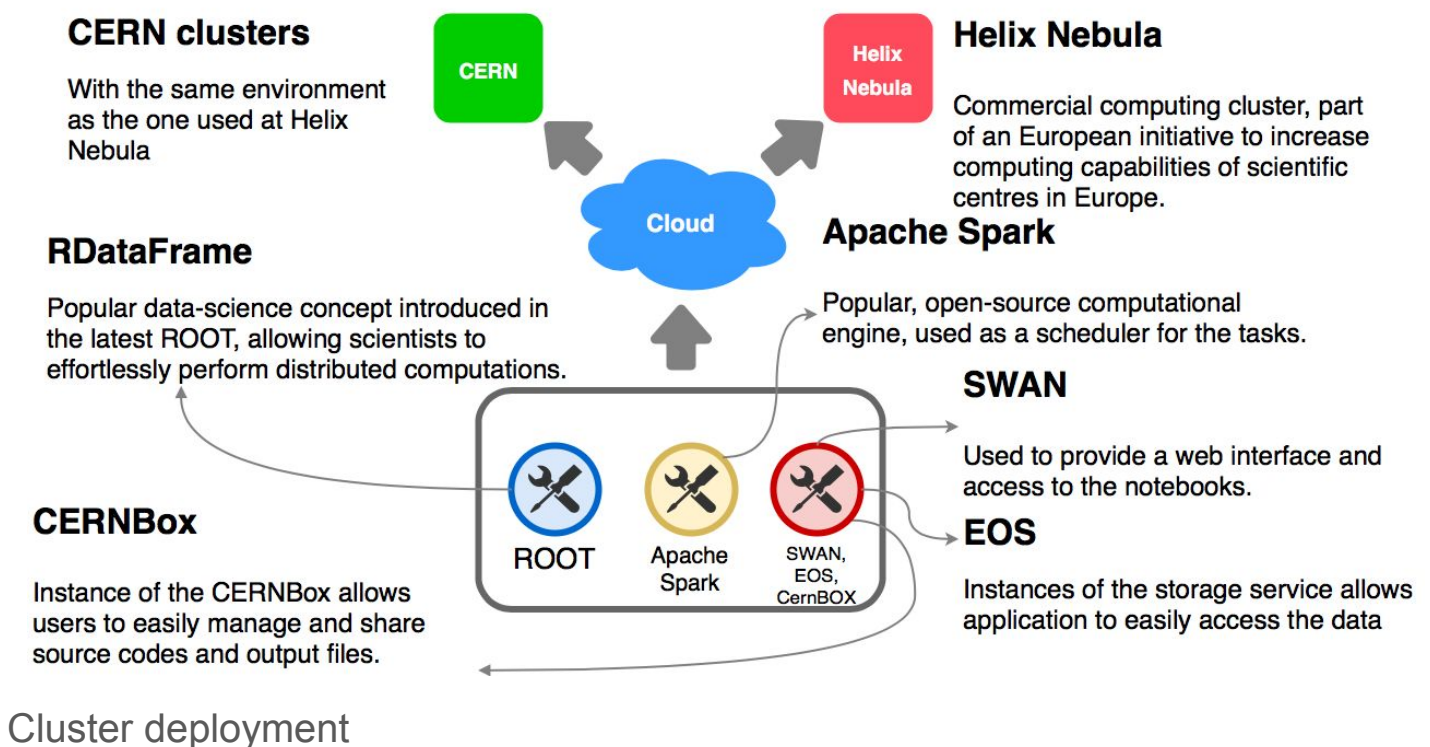
1. Purpose of the project
2. Frameworks, tools, and other concepts used in the project
 - a. Helix Nebula
 - b. ROOT Framework
 - i. RDataFrame
 - c. Apache Spark
 - d. Partitions creation
 - e. Data details
3. Different execution practices
4. Measured speedup of the
5. Our input to the project
6. Conclusions
 - a. Future of the project

Purpose of the project

Why are we doing it?

- To propose a time-efficient way to perform analysis of extensive amounts of data in CERN
- To investigate an impact and usefulness of external solutions for the HEP computing needs
- To prepare a ready model for future analyses performed in TOTEM experiment
- Research towards BSc Thesis due in December

3



4

Helix Nebula



- HN is piloting hybrid Cloud service procurement for scientific application's
- We deployed CERN's software and environment in HN cloud
 - ◆ the usage and the same results are also replicable on both clusters.
- For the moment 8 machines with 32 cores and 64 GB of memory each, however additional resources are available on request basis.
- Resources available to use till the end of November (with a possibility to massively scale out for tests if needed).

5

ROOT framework



- Data analysis framework for Physics.
- RDataFrame, included in the latest version of the ROOT framework (6.15, production release in November):
 - ◆ one of the key new features for analysis developers,
 - ◆ introduces major change from imperative to declarative programming.
- In order to use with Spark, code needs to be rewritten to the RDataFrame model in PyROOT (Python interface for ROOT).

6

RDataFrame in ROOT

- High level API for the n-tuple, introduces declarative instead of iterative approach to the framework
- Column-oriented, conceptually close to the tables
- Allows to use Spark
 - ◆ Data distribution might be managed internally
- More condensed code, with simplified methods performed on the selected columns
- Three execution modes:
 - ◆ Single core
 - ◆ Multi-core (implicitly parallelizing the application)
 - ◆ With Spark

7

Original

```
// explicit event loop
for (int ev_idx = 0; ev_idx <
ch_in->GetEntries();
    ev_idx += evIdxStep)
{
    ch_in->GetEntry(ev_idx);

    // check time
    if (anal.SkipTime(ev.timestamp))
        Continue;

    // diagonal cut
    bool allDiagonalRPs =
        (ev.h.L_2_N.v && ev.h.L_2_F.v
        && ev.h.R_2_N.v && ev.h.R_2_F.v);
    if (!allDiagonalRPs)
        continue;
    h_timestamp_dgn->Fill(ev.timestamp);
}
```

RDataFrame

```
#Read all branches
rdf = RDF(treename, input_file)

# check time
f1 = rdf.Filter("! SkipTime( timestamp )")

# diagonal cut
f2 = f1.Filter("v_L_2_F && v_L_2_N &&
v_R_2_F && v_R_2_N")
h_timestamp_dgn = f2.Histo1D(model,
"timestamp")
```

Code snippets

8

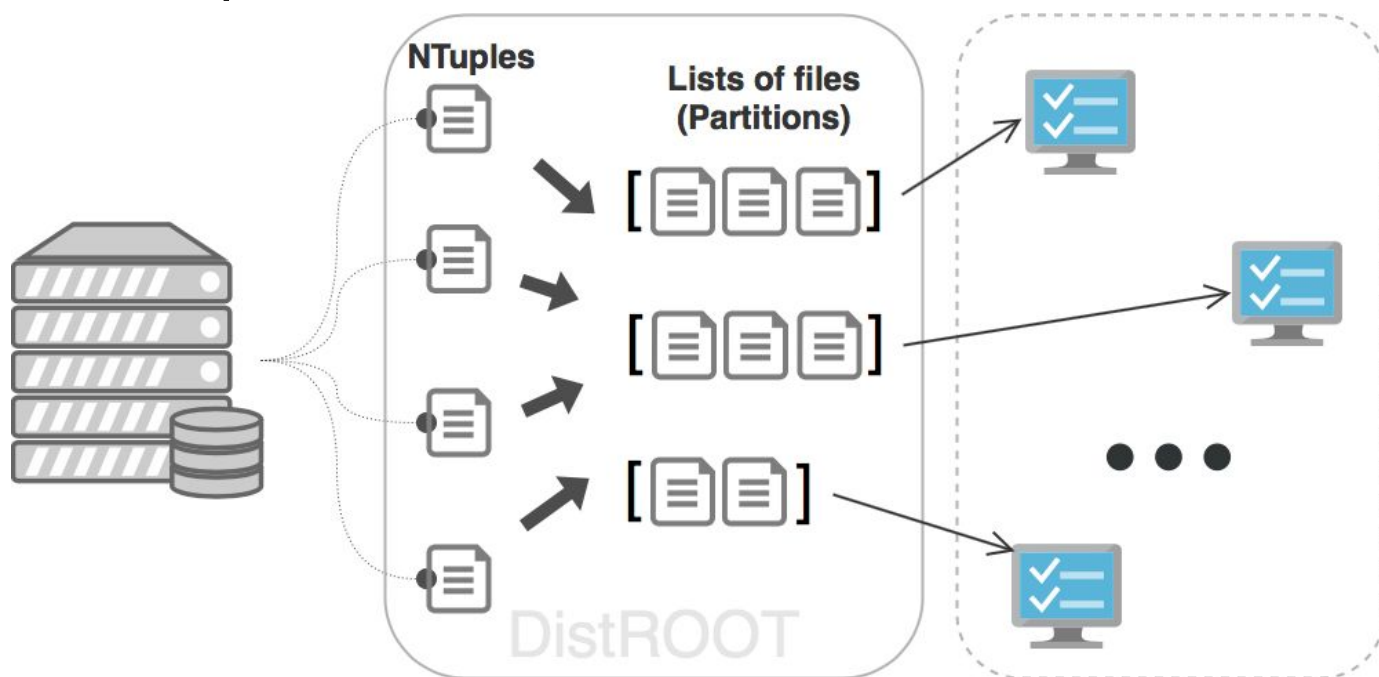
Apache Spark



- Distributed computing engine, which is responsible for managing tasks on a cluster.
- Open-source, widely-popular solution in the Big Data community, which primary aim is to allocate the resources on the cluster and then to distribute tasks appropriately to the nodes' individual workload.
- Can be accessed directly from the notebook.

9

How are partitions created?



10

Original

RDataFrame

Distill

Distilling events of interest

Datasets initially contain hundreds of files on average with TOTEM reconstructed events. To distill only interesting events, simple cut is applied.

Additionally, only 24 out of 1597 branches are saved in the output file.

Both
steps
together

Distillation and histograms computing combined

Whole dataset is distributed to the computing nodes. Each node performs same operations as the original analysis, yet there is no break for creating intermediary ROOT file.

In case of this particular analysis, this is a nonsense. However, we proceeded this way, to simulate working with larger amount of data.

Distributions

Computing histograms

Selection of cuts and filters is applied to the data chosen in the first step.

As an output number of histograms, graphs and profiles are created.

Analysis flow

11

Data for elastic scattering from 2015

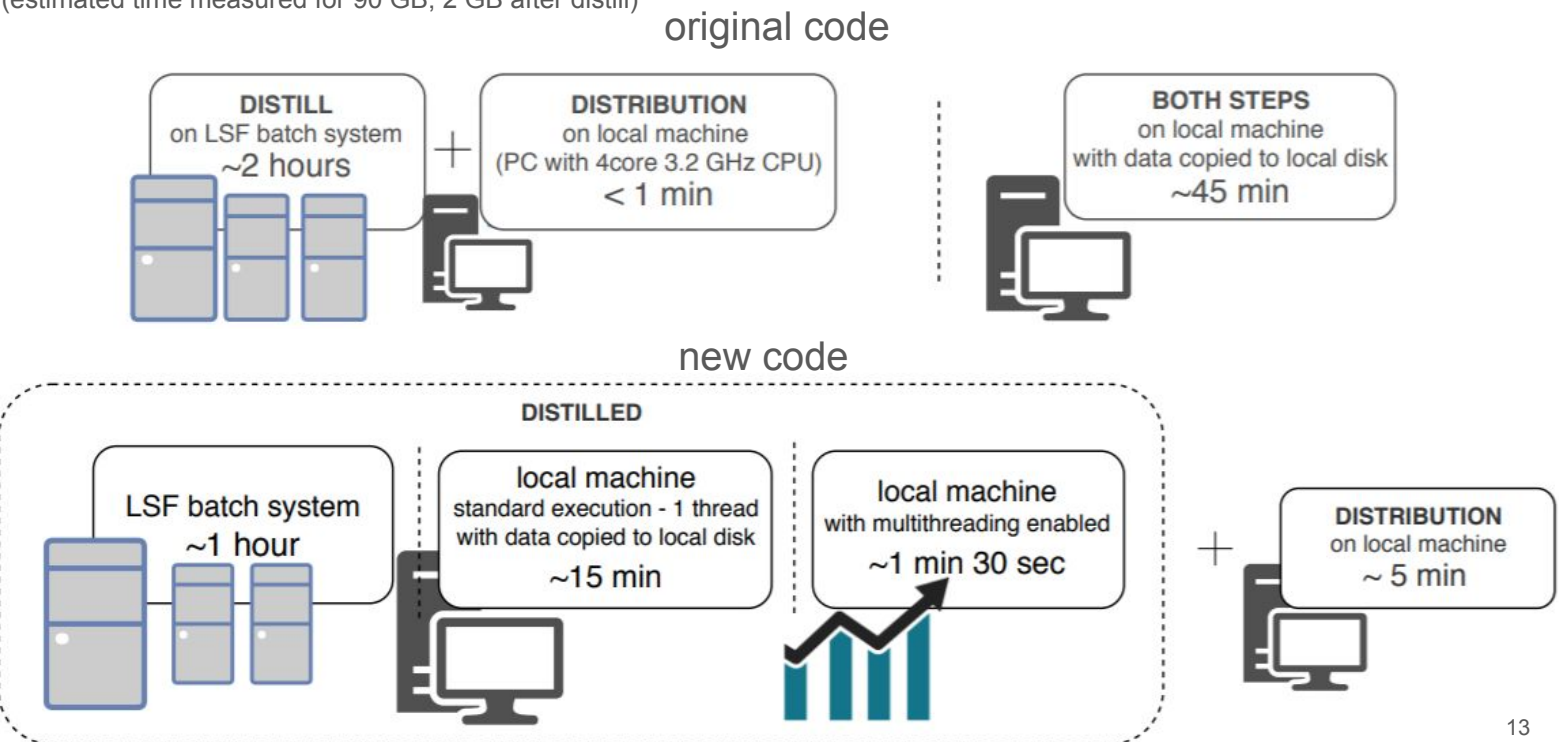
6500 GeV $\beta^*=90\text{m}$ 10σ

Fill number	Dataset	Size	Number of files	Number of entries
4495	1	97.31 GB	59	52 397 909
4496	2	198.96 GB	76	11 244 782
4506	4	405.81 GB	87	232 646 607
4511	6	559.50 GB	127	327 777 691
4505	3	713 GB	177	444 079 803
4510	7	795.67 GB	175	483 203 711
4509	5	2.02 TB	446	1 162 381 676
TOTAL		4.7 TB	1148	2 814 933 179

12

Different execution practices

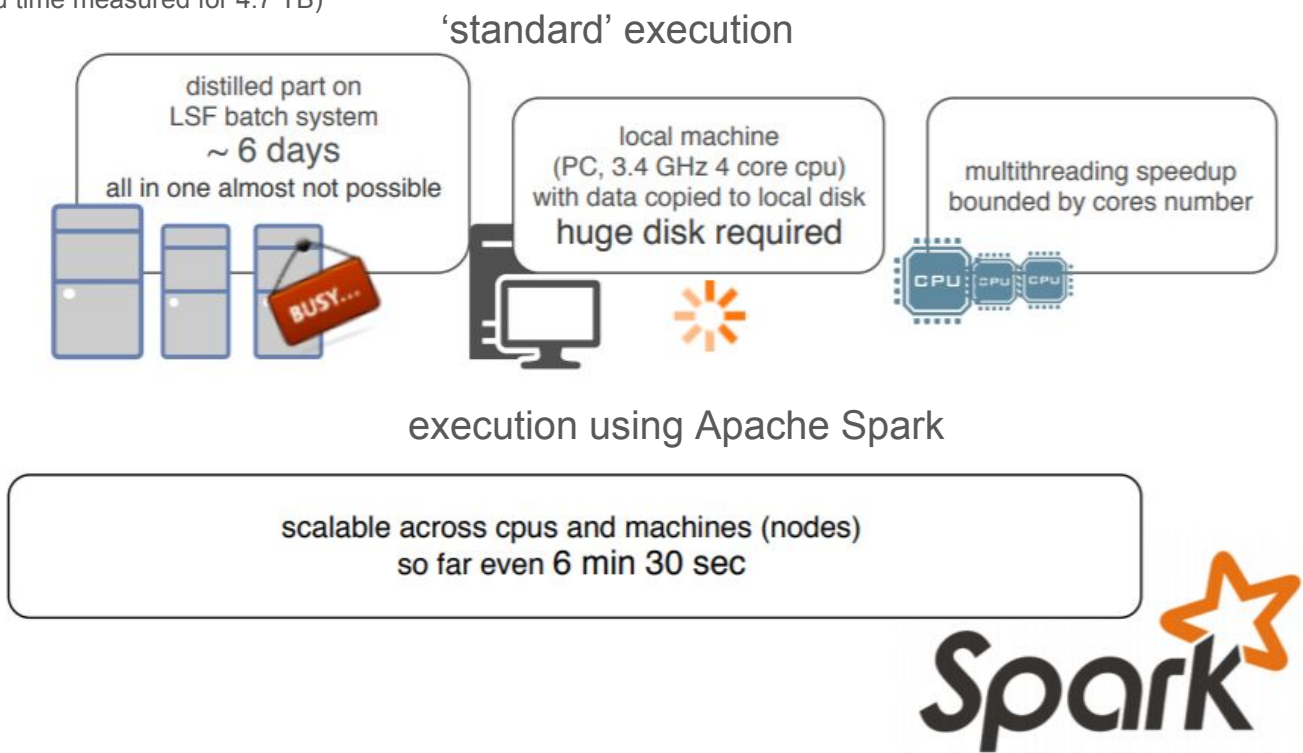
(estimated time measured for 90 GB; 2 GB after distill)



13

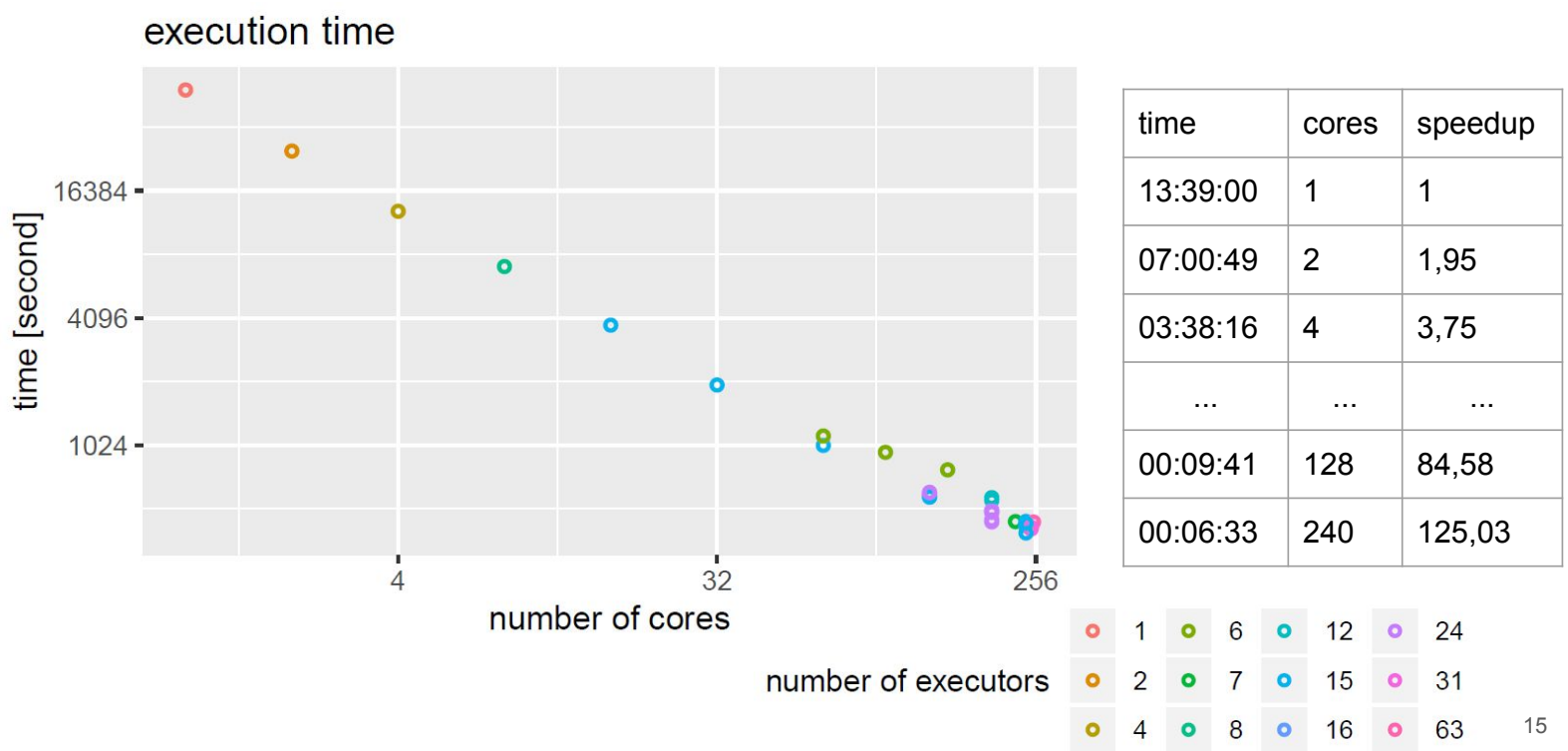
Going distributed

(estimated time measured for 4.7 TB)

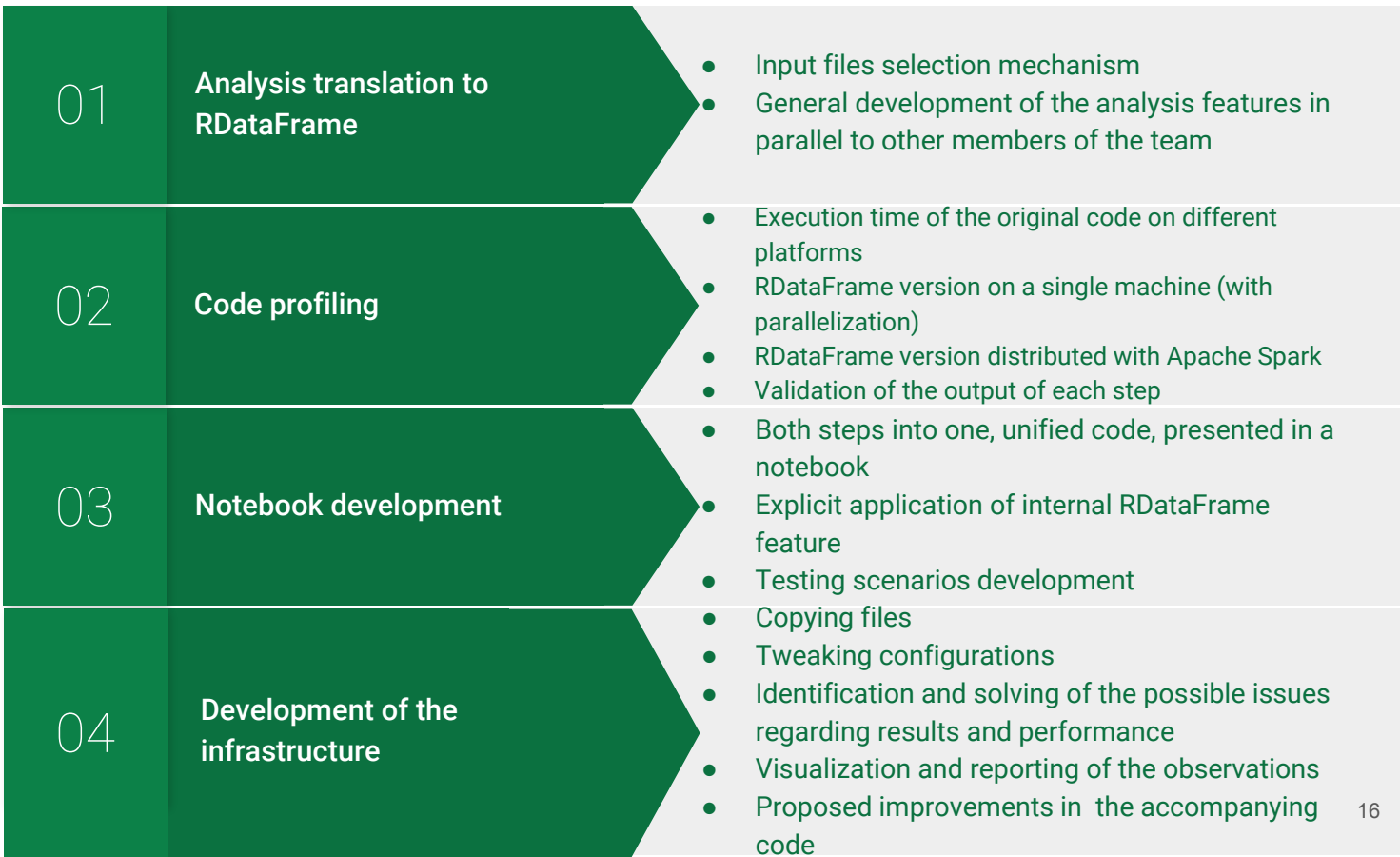


14

Execution time of notebook encapsulating whole (distill + distribution) analysis



Our contribution to the project



Conclusions

- Entirely new possibilities now:
 - Analysis of 5 TB is a matter of minutes.
 - Likely to be accelerated further.
 - No upper limit of the data size - as long as data can be contained in a memory.
 - No need to divide data to blocks, subsets or anything else.
 - Feasible quick data exploration.
- Application scales up with the number of cores - not limited to the single machine.
- Currently maximal measured speedup is about 125 times using:
 - Helix Nebula infrastructure,
 - ca. 240 cores.
- New programming model is an easier, more expressive way of writing a code

Required version of ROOT is still under development. Part responsible for distributing, which used in the project, has not yet been added to the ROOT framework.

17

Future of the project

- We will continue to work on this project for the next three months
- In that time, our goals will be to try to further optimize the application and test it with additional resources.
- The expected effect of those efforts is to prepare a way of performing analysis in a distributed environment, but in particular to supply TOTEM scientists with appropriate scripts, settings, and models to take advantage of the available computing potential.

We would be interested in using this approach in real conditions for not yet completed analysis.

It could be the ultimate proof of the practicality and effectiveness of this approach as well as a mutually beneficial use of Helix Nebula resources.

18

References

Github repository containing the code of the analysis:

<https://github.com/JavierCVilla/RDataFrame-Totem/>

More information about the available tools:

<http://www.helix-nebula.eu>

<http://eos.web.cern.ch>

<https://root.cern.ch>

<https://swan.web.cern.ch>

[https://root.cern/doc/master/classROOT_1_1RDataFrame.ht
ml](https://root.cern/doc/master/classROOT_1_1RDataFrame.html)

<https://cernbox.web.cern.ch>

<https://spark.apache.org>

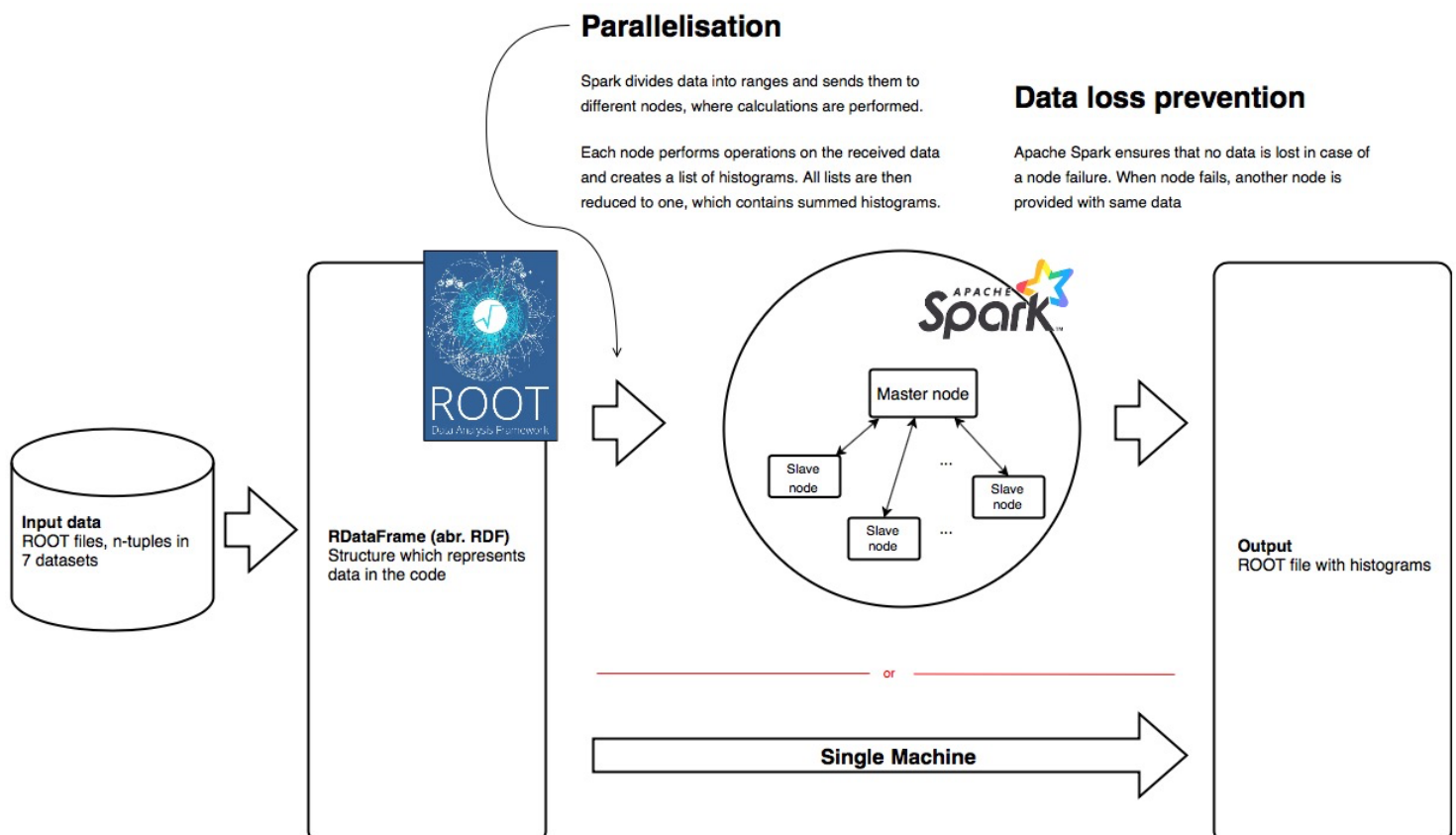
E. Selected presentations from the weekly meetings in TOTEM

Two presentations delivered at the weekly meetings in TOTEM, presenting the state of the project in the middle and the end of our stay in CERN.

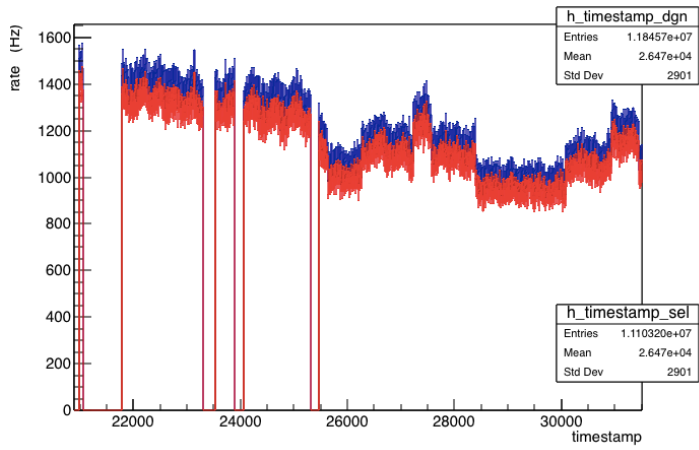
State of the project

Interactive data analysis of data from high energy physics experiments using Apache Spark

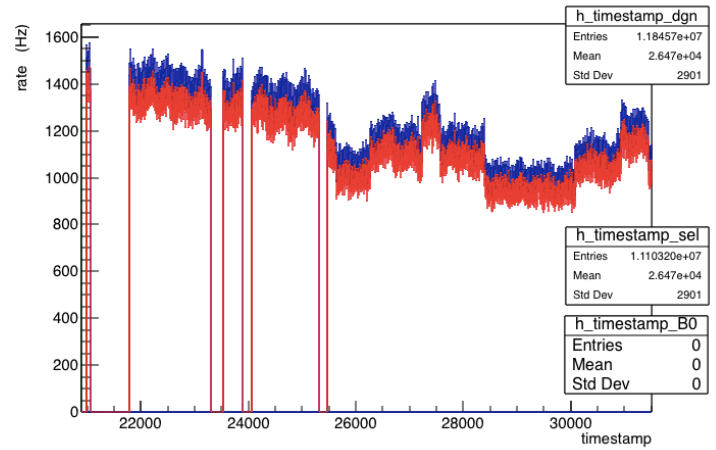
9 August 2018



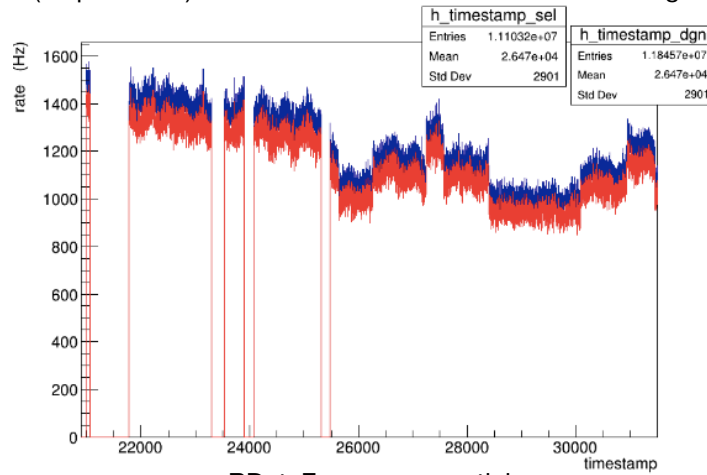
Rate CMP



RDataFrame with Spark (16 partitions)

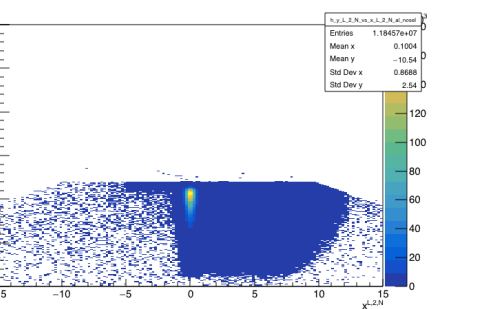
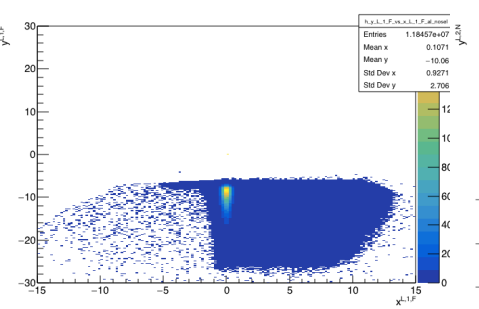
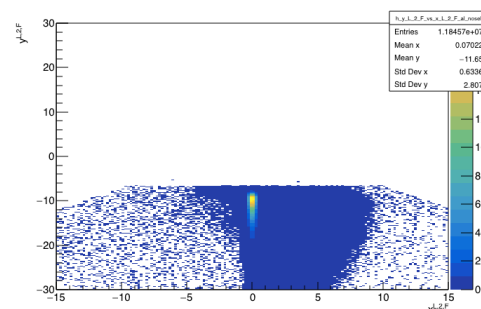
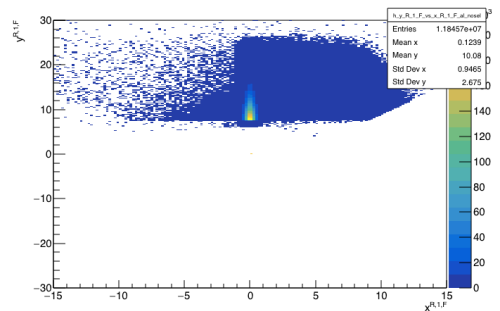
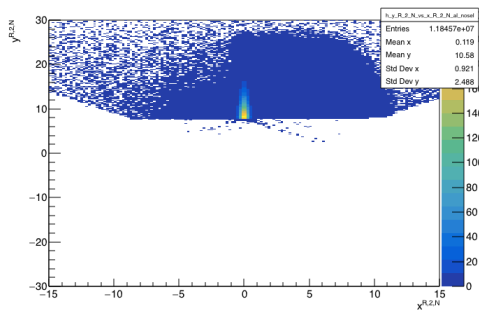
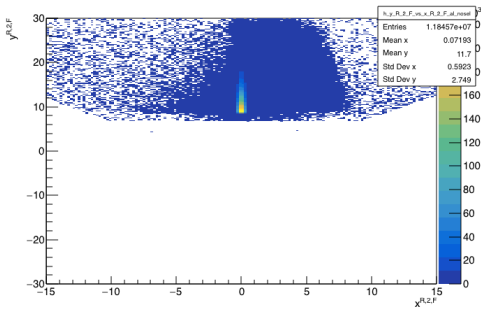


Original analysis (sequential)



RDataFrame, sequential

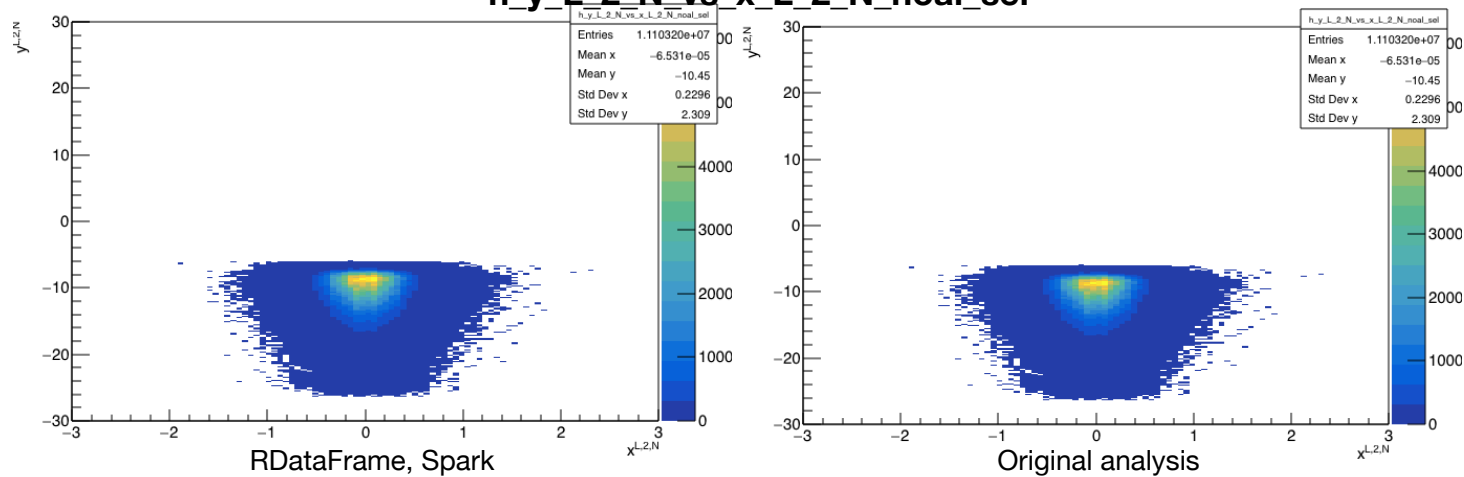
Hit distributions; vertical, aligned, before selection



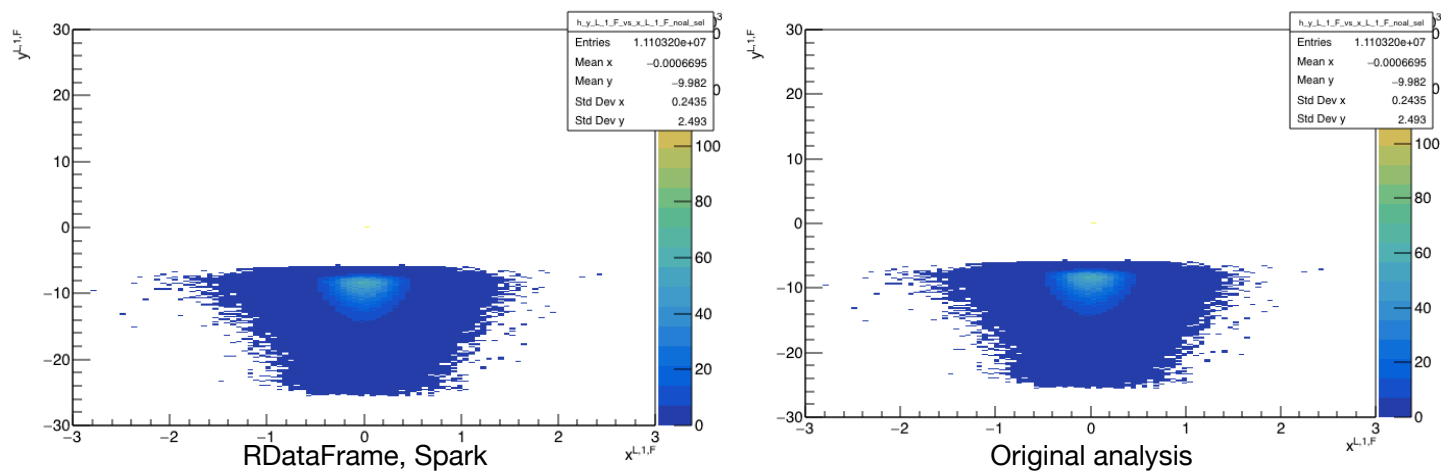
Results of the RDataFrame with Spark

Hit distributions; vertical, not aligned, after selection

h_y_L_2_N vs x_L_2_N_noal_sel

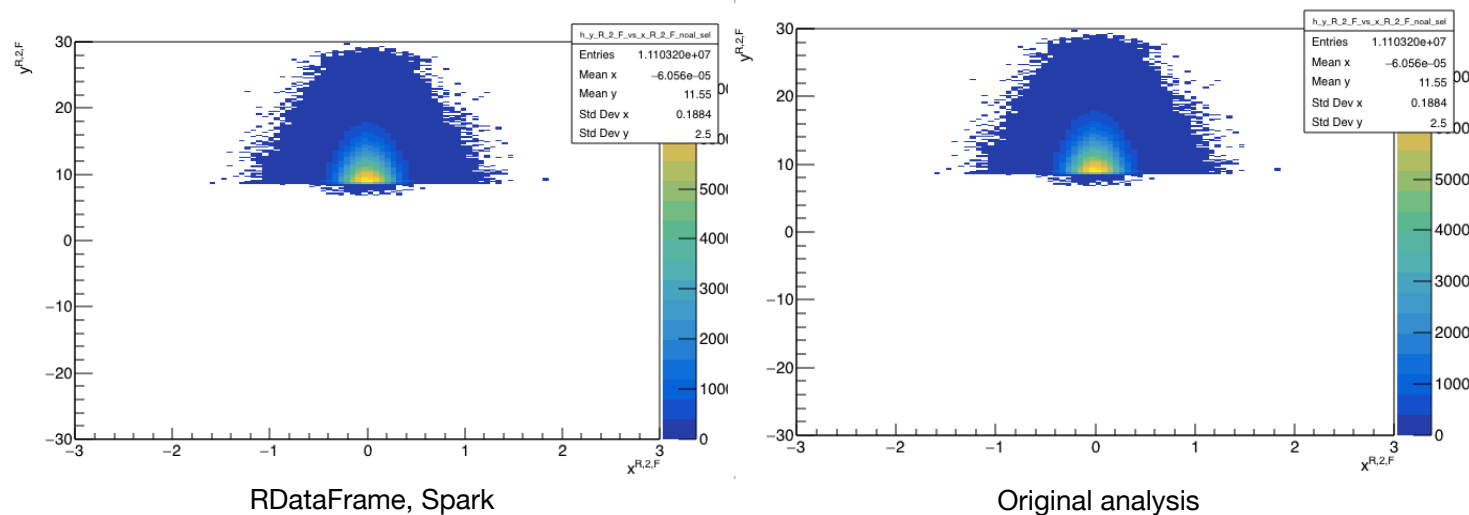


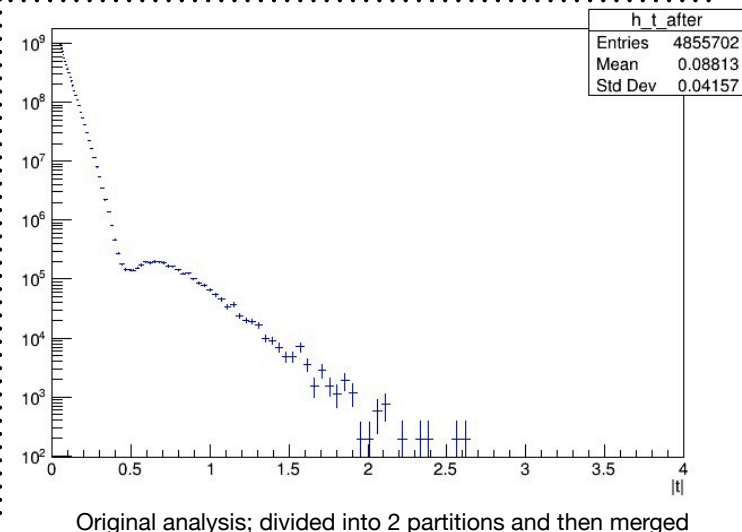
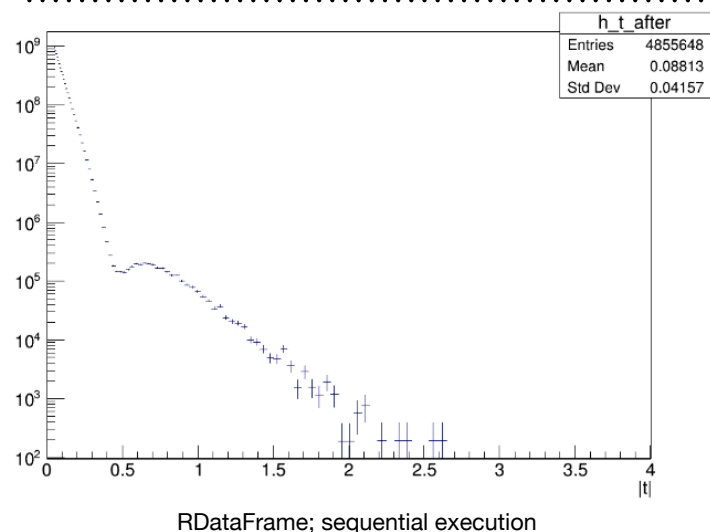
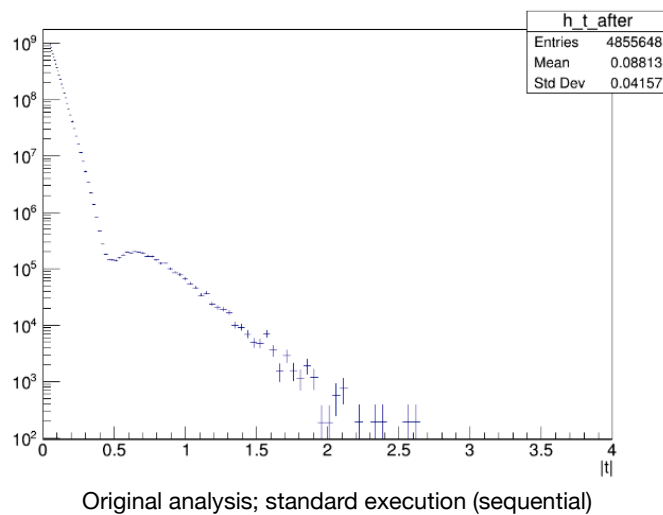
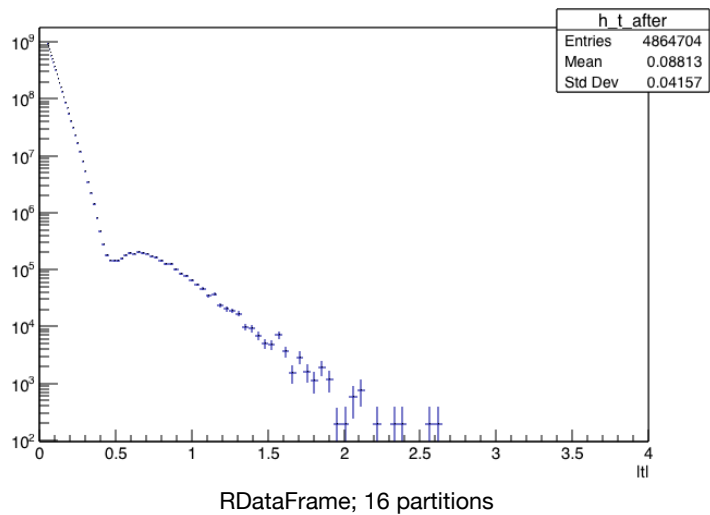
h_y_L_1_F vs x_L_1_F_noal_sel



Hit distributions; vertical, not aligned, after selection

h_y_R_2_F vs x_R_2_F_noal_sel



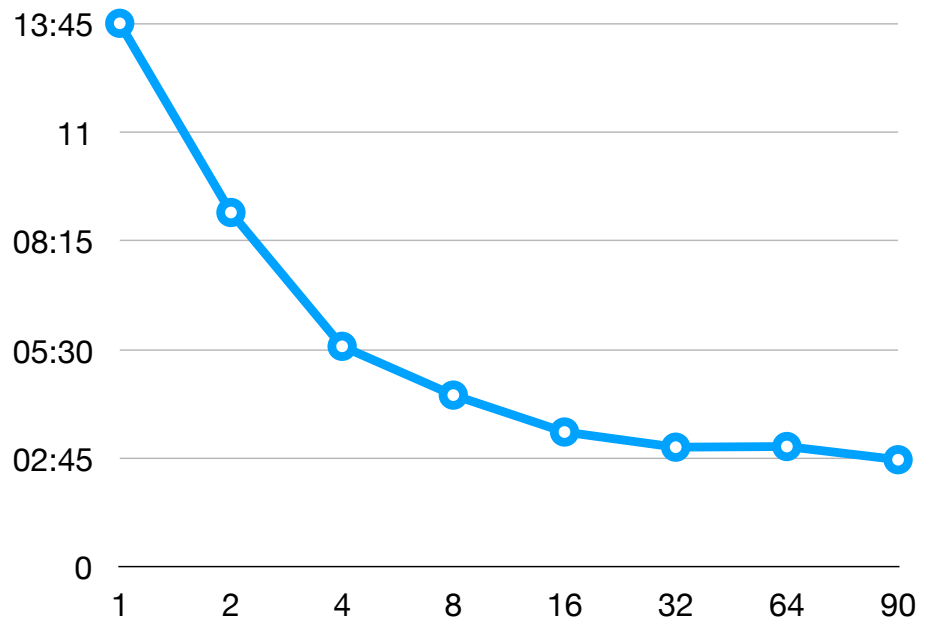


Differences in the number of entries in h_t_after

	Original	2 partitons	16 partitons	32 partitons	64 partitons	90 partitons	Orginal 2 partitons
ub	6 690 032	6 690 071	6 697 284	6 706 545	6 717 720	6 723 669	-
ob_1_10_0.2	4 855 647	4 855 698	4 864 703	4 875 740	4 888 119	4 894 308	-
ob_1_30_0.2	4 855 648	4 855 699	4 864 704	4 875 741	4 888 120	4 894 308	4 855 703

Spark running time

Number of executors	Running time
1	13m 45s
2	8m 58s
4	5m 35s
8	4m 21s
16	3m 25s
32	3m 2s
64	3m 3s
90	2m 43s



Questions?

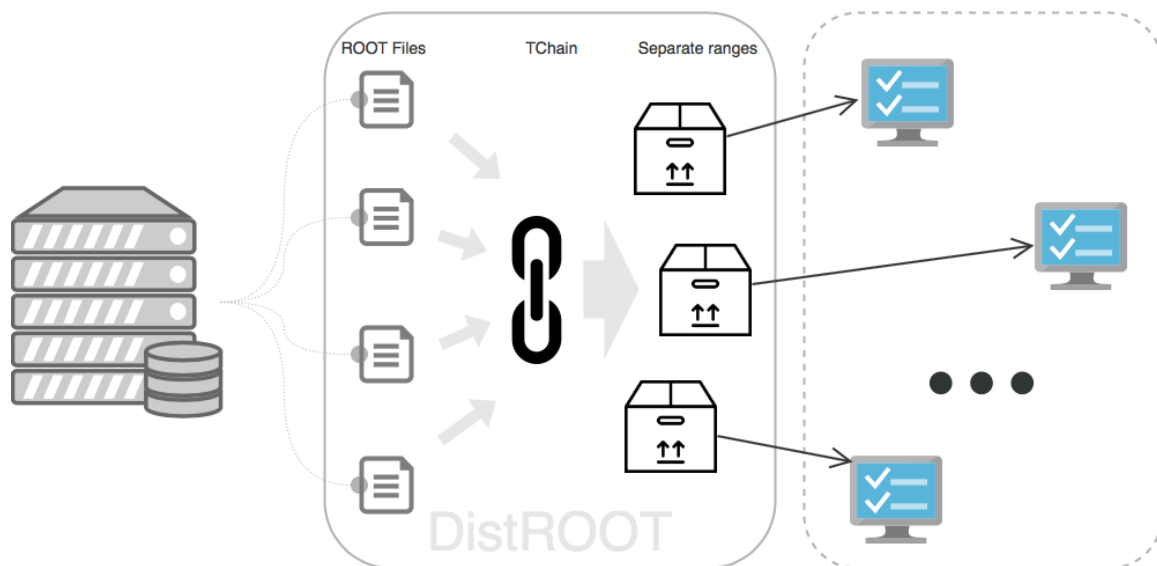
Summary of the week ^(one and half)

Interactive data analysis of data from high energy
physics experiments using Apache Spark

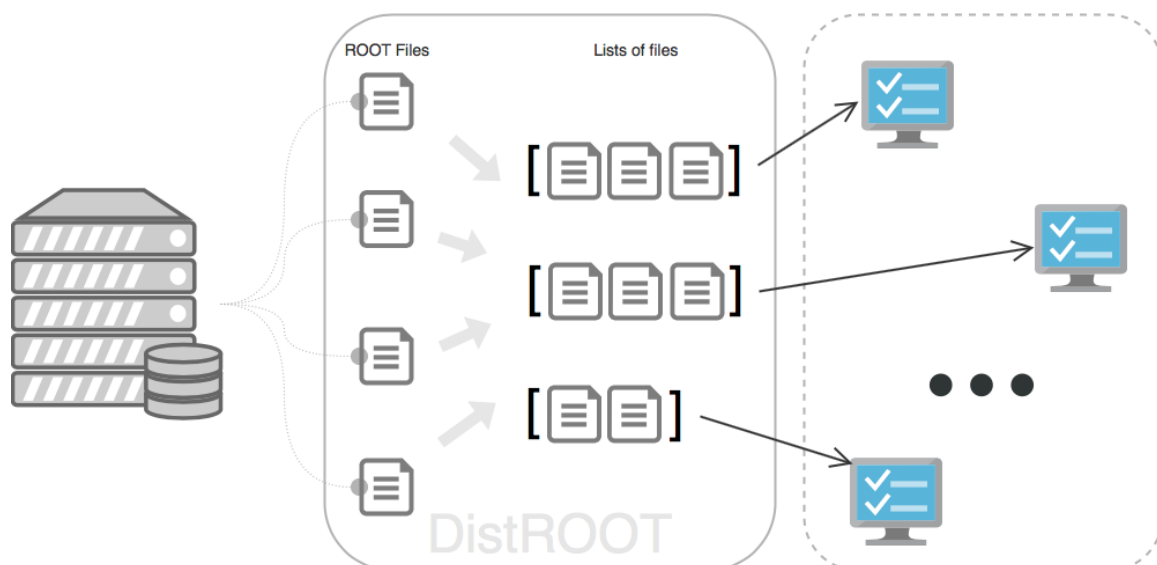
4 September 2018

Change in DistROOT

DistROOT had been using ranges of entries to distribute data evenly across the nodes



Another approach to the DistROOT task is send to nodes ranges of entire files



Ranges of entries

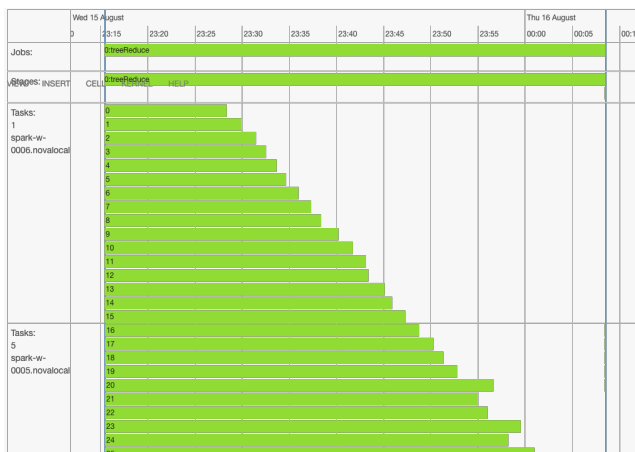
Ranges of files

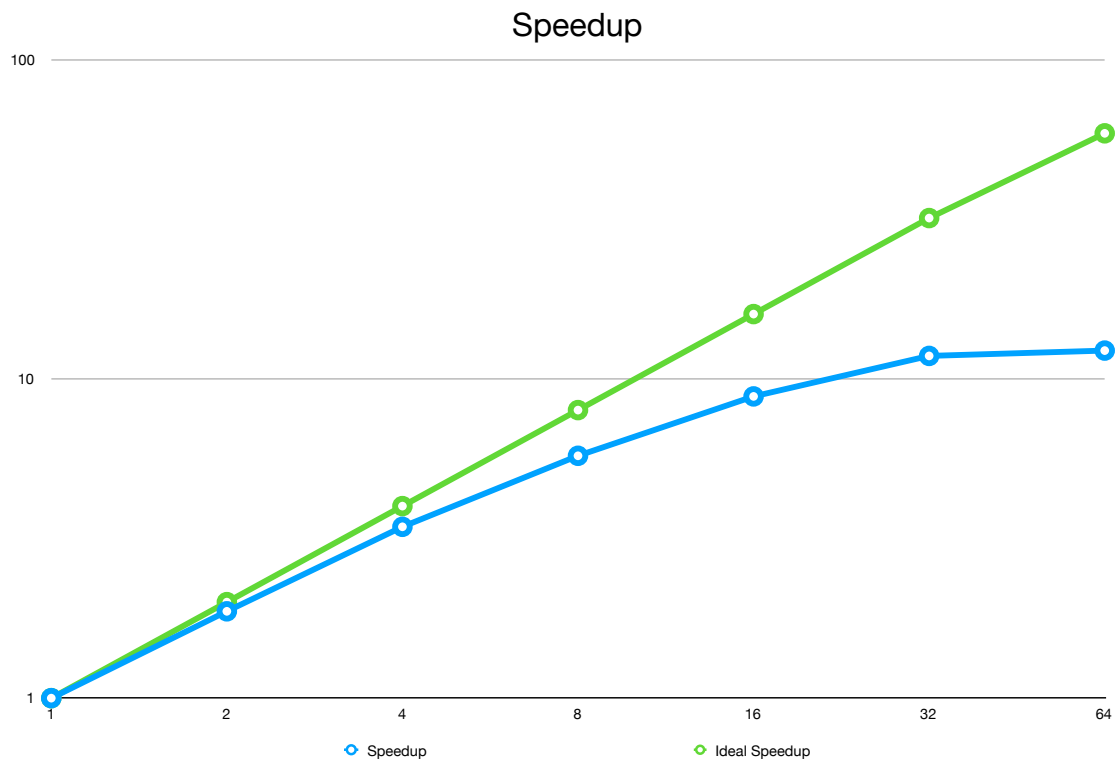
~ 3h

15m 47s

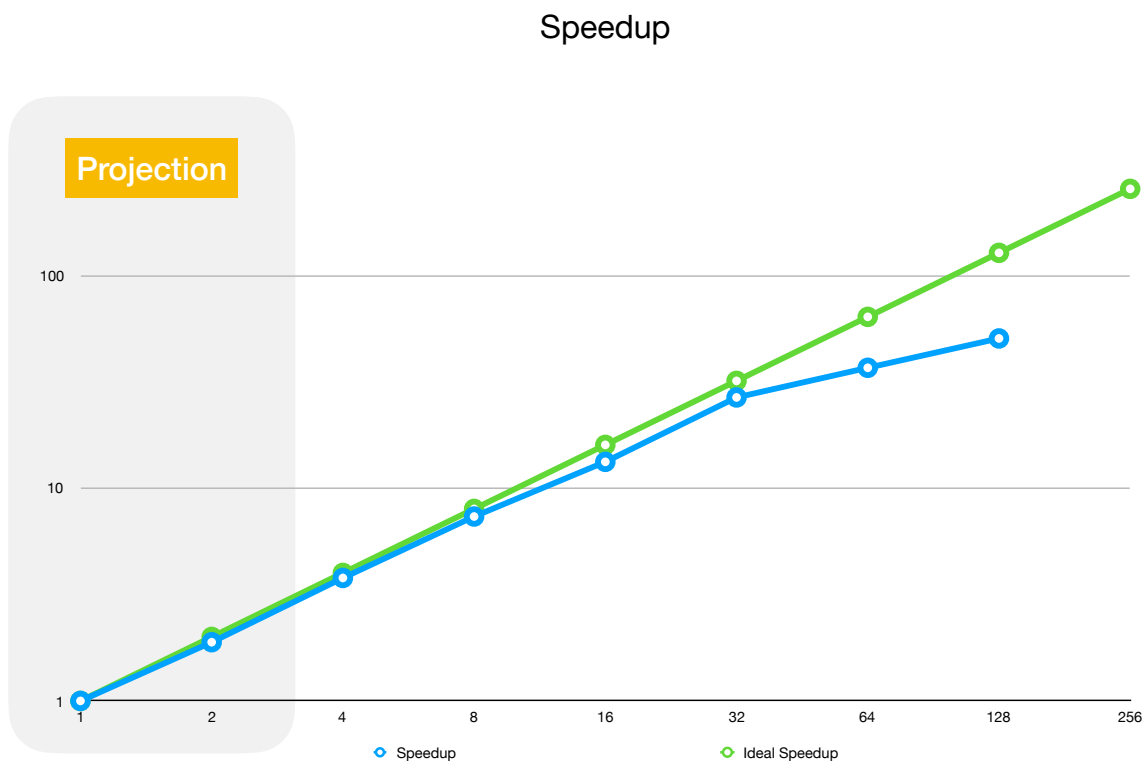
Ranges of entries

Ranges of files

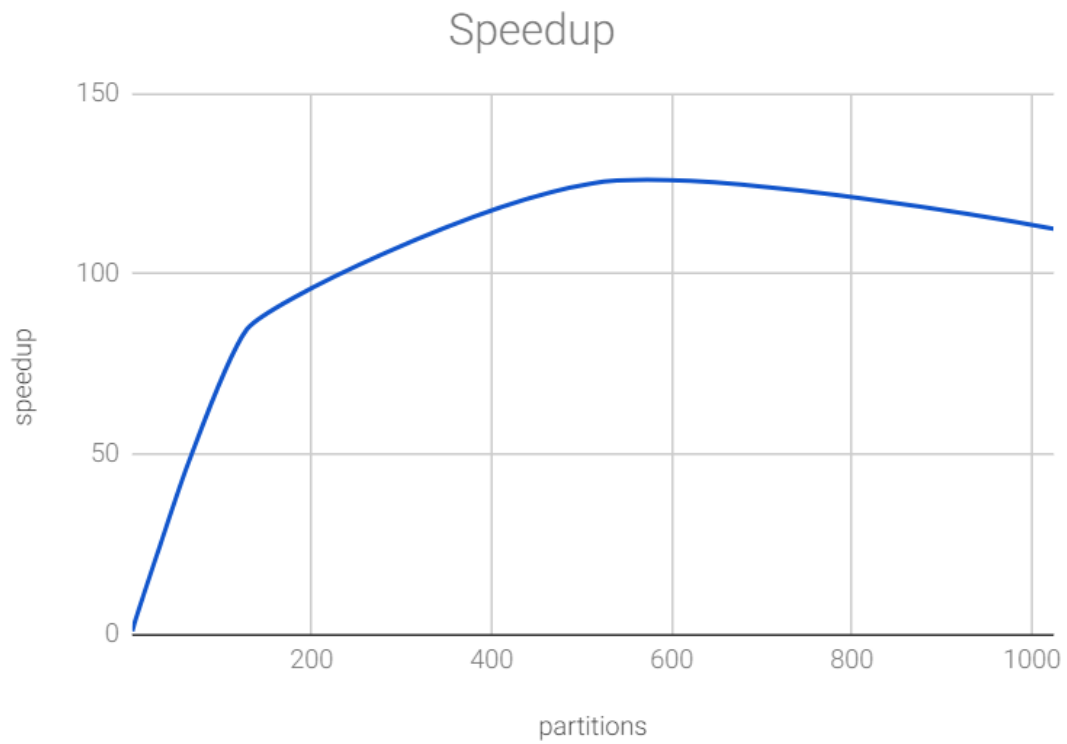




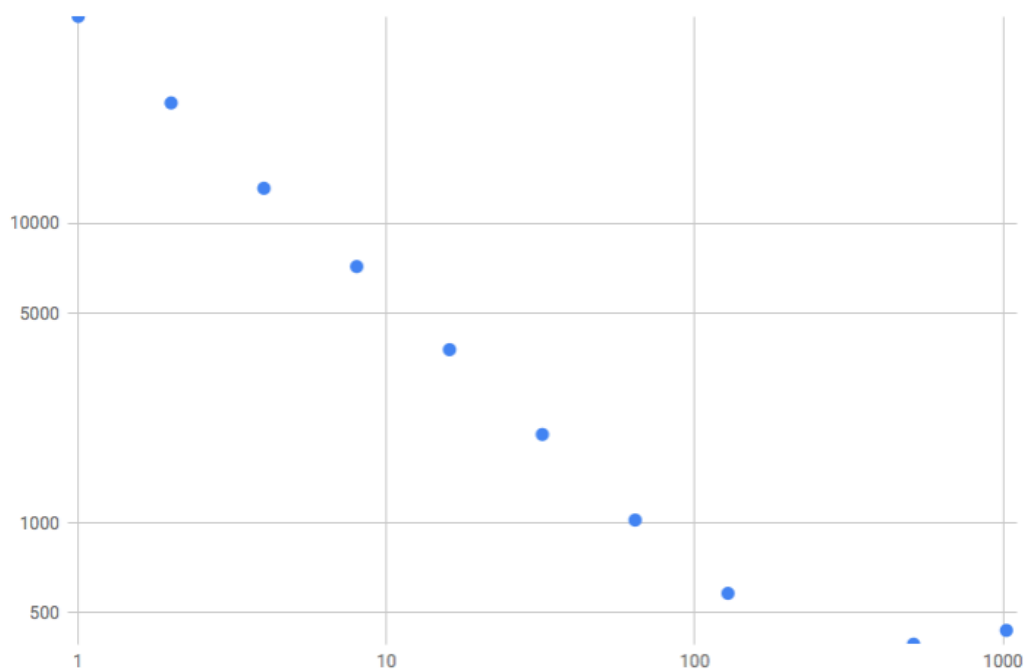
Helix Nebula, DS1 (90 GB),
distributing ranges of files,
4 executors, 64 cores, 20 GB per each node



Helix Nebula, DS5 (1.7 TB),
distributing ranges of files,
4 executors, 64 cores, 20 GB per each node



Helix Nebula, all 7 datasets (4.7 TB),
distributing ranges of files,
15 executors, 240 cores, 32 GB per each node



Helix Nebula, all 7 datasets (4.7 TB),
distributing ranges of files,
15 executors, 240 cores, 32 GB per each node

Execution times for different modes

Spark 1 executor (64G mem and 4 cores)	3:56
Spark 2 executors (32G mem and 2 cores each)	4:00
Spark 4 executors (16G mem and 1 core each)	4:02
Script (each part run one at the time)	2:50 - 3:01
Script (each part run at the same time)	2:55 - 3:03
Script with multithreading (4 threads)	4:20
Script without multithreading (1 thread)	9:55

F. Selected tables

F.1. Tests performed on Helix Nebula Cloud in November

Number	Time	Partitions	Cores	Executors	Partitions	Dynamic Allocation
1	00:50:45	115	115	23	10000	TRUE
2	00:08:05	192	192	12	1000	TRUE
3	00:08:03	192	192	12	192	TRUE
4	00:06:53	192	192	12	384	TRUE
5	00:07:13	192	192	12	768	TRUE
6	00:06:58	192	192	12	576	TRUE
7	00:09:34	96	176	11	96	TRUE
8	00:22:46	48	64	4	48	TRUE
9	00:28:02	24	64	4	24	TRUE
10	00:49:33	12	32	2	12	TRUE
11	00:17:58	48	48	3	48	TRUE
12	00:07:03	240	240	15	256	TRUE
13	02:18:29	4	32	2	4	TRUE
14	01:14:46	8	16	1	8	TRUE
15	00:09:24	208	208	13	2000	TRUE
16	00:13:12	208	208	13	4000	TRUE
17	00:05:43	208	208	13	416	TRUE
18	00:05:58	208	208	13	624	TRUE
19	00:17:27	208	208	13	6000	TRUE
20	00:12:26	64	112	7	64	TRUE
21	00:06:45	240	240	15	240	TRUE
22	00:05:50	240	240	15	480	TRUE
23	00:05:23	240	240	15	480	FALSE
24	00:06:49	240	240	15	240	FALSE
25	00:05:21	240	240	15	360	FALSE
26	00:05:30	240	240	15	420	FALSE
27	00:06:00	240	240	15	300	FALSE
28	00:05:58	240	240	15	390	FALSE
29	00:05:20	240	240	15	360	FALSE
30	00:19:41	32	240	15	32	FALSE
31	08:20:15	1	1	1	1	FALSE
32	04:34:20	2	2	1	2	FALSE

F.2. Tests performed on Helix Nebula Cloud in December reusing executors

Number	Time	Max Cores	Max Executors	Partitions
1	00:02:29	784	49	1568
2	00:02:07	784	49	784
3	00:01:57	784	49	1176
4	00:02:23	784	49	1584
5	00:02:24	912	57	1584
6	00:01:57	912	57	1368
7	00:03:32	1140	114	1450
8	00:04:50	1450	145	1450
9	00:05:09	1450	145	1450
10	00:02:29	784	49	1500
11	00:02:00	784	49	1176
12	00:02:34	1176	49	1764
13	00:02:07	1176	49	1176
14	00:01:46	1176	49	784
15	00:01:48	1176	49	588
16	00:01:59	1176	49	400
17	00:01:55	1176	49	980
18	00:01:50	1176	49	882
19	00:01:47	1176	49	686
20	00:01:44	1176	49	735
21	00:02:09	1176	49	360
22	00:02:58	1176	49	240
23	00:05:20	1176	49	128
24	00:09:29	1176	49	64
26	00:02:43	1176	49	1600
27	00:02:18	1176	49	1700
29	00:02:51	1470	49	1471
30	00:01:43	1470	49	736
31	00:02:03	1470	49	1100
33	00:02:55	1470	99	1470
34	00:01:57	1470	98	736
35	00:02:11	1470	98	1024
36	00:02:33	1470	98	1320
37	00:01:57	1470	98	500
38	00:01:58	1470	98	640
39	00:01:54	1470	98	700
40	00:01:58	1470	98	580
41	00:02:02	1470	98	420
42	00:02:39	1470	98	280
43	00:02:56	1200	40	1201
44	00:02:06	1200	40	786
45	00:02:16	1200	40	840

F.3. Tests performed on Helix Nebula Cloud in December restarting executors

Number	Time	Max Cores	Max Executors	Partitions
1	00:02:41	1470	49	1470
2	00:02:08	1470	49	735
3	00:02:19	1470	49	1100
4	00:02:07	1470	49	640
5	00:02:52	1470	49	256
6	00:04:56	1470	49	128
7	00:01:56	1470	49	512
8	00:09:31	1470	49	64
9	00:18:04	1470	49	32
10	00:02:32	1470	49	1024
11	00:02:57	1470	49	1700
12	00:34:15	1470	49	16
13	00:02:16	1470	49	896
14	00:02:01	1470	49	576
15	00:01:52	1470	49	448
16	00:02:05	1470	49	512
17	00:01:54	1470	49	480
18	00:02:04	1470	49	384
19	00:01:59	1470	49	416
20	00:02:00	1470	49	448

F.4. Tests performed on Helix Nebula Cloud in December with varying size of input

Cores	Executors	Partitions	Input	Entries in histo_75	Time
735	49	1470	1 DS1	4987706	00:58
735	49	1470	2 DS1	9906971	00:51
735	49	1470	4 DS1	19721210	00:52
735	49	1470	8 DS1	39295197	00:54
735	49	1470	16 DS1	78326678	01:08
735	49	1470	32 DS1	156199156	01:25
735	49	1470	64 DS1	311712724	02:05
735	49	512	1 DS1	4941104	00:32
735	49	512	1 DS1	4941104	00:31
735	49	512	1 DS1	4941104	00:27
735	49	512	2 DS1	9842444	0:28
735	49	512	2 DS1	9842444	0:29
735	49	512	2 DS1	9842444	0:30
735	49	512	4 DS1	19613434	00:33
735	49	512	4 DS1	19613434	00:25
735	49	512	4 DS1	19613434	00:28
735	49	512	8 DS1	39101463	00:40
735	49	512	8 DS1	39101463	00:44
735	49	512	8 DS1	39101463	00:45
735	49	512	16 DS1	78003392	00:51
735	49	512	16 DS1	78003392	00:56
735	49	512	16 DS1	78003392	00:51
735	49	512	32 DS1	155728463	01:19
735	49	512	32 DS1	155728463	01:17
735	49	512	32 DS1	155728463	01:14
735	49	512	64 DS1	311059245	02:05
735	49	512	64 DS1	311059245	02:04
735	49	512	64 DS1	311059245	02:04
735	49	512	128 DS1	621673527	03:18
735	49	512	128 DS1	621673527	03:34
735	49	512	128 DS1	621673527	03:30
735	49	512	256 DS1	1243091194	06:22
735	49	512	256 DS1	1243091194	06:30
735	49	512	256 DS1	1243091194	06:37
735	49	512	512 DS1	2486091745	10:41
735	49	512	512 DS1	2486091745	10:19
735	49	512	512 DS1	2486091745	10:40
735	49	512	1024 DS1	4972183490	20:00
735	49	512	1024 DS1	4972183490	20:42

F.5. Duration of the DistTree initialization

Time [s]	Entries	Dataset
3.226341009	52397909	DS1
2.997829914	52397909	DS1
2.999531031	52397909	DS1
3.001531839	52397909	DS1
3.008586884	52397909	DS1
3.079858065	52397909	DS1
6.05289197	112445782	DS2
5.733387947	112445782	DS2
5.561378956	112445782	DS2
5.717045069	112445782	DS2
5.716299057	112445782	DS2
5.674022913	112445782	DS2
19.31308103	418285672	reduced DS3
18.70503902	418285672	reduced DS3
18.80461907	418285672	reduced DS3
19.41429806	418285672	reduced DS3
18.65827012	418285672	reduced DS3
18.72057199	418285672	reduced DS3
10.88036394	232646607	DS4
10.47255707	232646607	DS4
10.41627097	232646607	DS4
10.43485498	232646607	DS4
10.28416109	232646607	DS4
10.42988801	232646607	DS4
51.11124492	1162381676	DS5
50.90236402	1162381676	DS5
50.72539806	1162381676	DS5
50.49534607	1162381676	DS5
50.7720561	1162381676	DS5
50.31584096	1162381676	DS5
14.94506311	327777691	DS6
14.54974389	327777691	DS6
14.41645598	327777691	DS6
14.4067452	327777691	DS6
14.13497996	327777691	DS6
14.22196198	327777691	DS6
22.07105494	483203711	DS7
21.30257893	483203711	DS7
36.9278841	810981402	DS6+DS7

Time [s]	Entries	Dataset
36.6484611	810981402	DS6+DS7
36.75380111	810981402	DS6+DS7
28.80123997	650932279	DS3+DS4
33.80995297	650932279	DS3+DS4
28.9175241	650932279	DS3+DS4
26.79912901	595649493	DS7+DS2
26.9591279	595649493	DS7+DS2
26.51588988	595649493	DS7+DS2
40.32295299	923427184	DS6+DS7+DS2
39.15513301	923427184	DS6+DS7+DS2
39.14362001	923427184	DS6+DS7+DS2
52.95346284	1229267074	DS6+DS7+DS3
52.26283598	1229267074	DS6+DS7+DS3
51.72136617	1229267074	DS6+DS7+DS3
43.96997094	1013935165	DS3+DS7+DS2
43.74999189	1013935165	DS3+DS7+DS2
43.24927711	1013935165	DS3+DS7+DS2

F.6. Tests using local machine using local storage (SSD)

DS	real	user	sys	Test description
DS1	46m13.953s	44m57.542s	0m7.810s	c++ without CMSSW
DS1	42m58.030s	41m41.579s	0m8.281s	c++ without CMSSW
DS1	45m17.651s	44m1.030s	0m8.324s	c++ without CMSSW
DS1	45m25.795s	44m9.095s	0m8.323s	c++ without CMSSW
DS1	44m57.818s	43m40.915s	0m8.326s	c++ without CMSSW
DS1	45m32.535s	44m15.701s	0m8.321s	c++ without CMSSW
DS1	45m25.822s	44m8.979s	0m8.334s	c++ without CMSSW
DS1	5m15.017s	4m50.939s	0m3.427s	python 1 thread (without implicitMT)
DS1	5m6.978s	4m44.784s	0m3.304s	python 1 thread (without implicitMT)
DS1	5m15.324s	4m52.663s	0m3.298s	python 1 thread (without implicitMT)
DS1	5m18.884s	4m55.764s	0m3.224s	python 1 thread (without implicitMT)
DS1	5m24.909s	4m56.724s	0m3.467s	python 1 thread (without implicitMT)
DS1	5m22.753s	4m54.284s	0m3.370s	python 1 thread (without implicitMT)
DS1	8m9.297s	6m10.378s	0m7.946s	python 1 thread (with implicitMT)
DS1	8m7.421s	6m11.285s	0m7.718s	python 1 thread (with implicitMT)
DS1	8m10.853s	6m14.479s	0m7.954s	python 1 thread (with implicitMT)
DS1	8m8.309s	6m9.964s	0m7.793s	python 1 thread (with implicitMT)
DS1	8m12.450s	6m14.467s	0m8.104s	python 1 thread (with implicitMT)
DS1	8m9.308s	6m11.988s	0m8.042s	python 1 thread (with implicitMT)
DS1	3m43.467s	5m32.908s	0m6.462s	python 2 threads (with implicitMT)
DS1	3m41.677s	5m33.171s	0m6.555s	python 2 threads (with implicitMT)
DS1	3m42.088s	5m36.375s	0m6.569s	python 2 threads (with implicitMT)
DS1	3m44.947s	5m38.663s	0m6.632s	python 2 threads (with implicitMT)
DS1	3m43.167s	5m34.170s	0m6.499s	python 2 threads (with implicitMT)
DS1	3m44.906s	5m35.766s	0m6.756s	python 2 threads (with implicitMT)
DS1	1m52.120s	5m28.603s	0m5.970s	python 4 threads (with implicitMT)
DS1	1m55.857s	5m32.108s	0m6.298s	python 4 threads (with implicitMT)
DS1	1m53.997s	5m33.771s	0m6.328s	python 4 threads (with implicitMT)
DS1	1m55.938s	5m31.940s	0m5.874s	python 4 threads (with implicitMT)
DS1	1m49.572s	5m25.951s	0m5.706s	python 4 threads (with implicitMT)
DS1	2m1.541s	5m30.825s	0m6.461s	python 4 threads (with implicitMT)
DS1	1m31.853s	5m23.482s	0m6.832s	python 8 threads (with implicitMT)
DS1	1m38.014s	5m27.094s	0m6.951s	python 8 threads (with implicitMT)
DS1	1m32.444s	5m21.313s	0m7.078s	python 8 threads (with implicitMT)
DS1	1m34.564s	5m24.672s	0m7.495s	python 8 threads (with implicitMT)
DS1	1m31.725s	5m25.017s	0m7.247s	python 8 threads (with implicitMT)
DS1	1m35.151s	5m21.871s	0m7.280s	python 8 threads (with implicitMT)

F.7. Tests using local machine using remote storage

DS	real	user	sys	Test description
DS1	32m21.088s	7m1.294s	0m10.910s	python 1 thread (without implicitMT)
DS1	32m4.592s	7m1.684s	0m11.287s	python 1 thread (without implicitMT)
DS1	21m41.116s	6m39.499s	0m11.377s	python 1 thread (with implicitMT)
DS1	21m4.080s	6m31.367s	0m10.892s	python 1 thread (with implicitMT)
DS1	10m9.281s	6m13.252s	0m10.254s	python 2 thread (with implicitMT)
DS1	10m36.886s	6m12.886s	0m10.755s	python 2 thread (with implicitMT)
DS1	7m26.479s	6m12.909s	0m11.848s	python 4 thread (with implicitMT)
DS1	5m36.042s	6m1.122s	0m11.042s	python 4 thread (with implicitMT)
DS1	4m18.298s	5m53.037s	0m15.463s	python 8 thread (with implicitMT)
DS1	3m41.407s	5m56.551s	0m16.866s	python 8 thread (with implicitMT)
DS1	52m40.332s	38m31.375s	0m13.552s	c++ without CMSSW

F.8. Tests of Scala version using Prometheus cluster

Nodes	Cores (total)	Time [s]
1	24	9891.875161
2	48	9978.380832
4	96	5815.453725
8	192	2766.703505
16	384	1256.310684
32	768	676.70679
64	1536	576.54776
128	3072	585.810265
256	6144	734.424722

References

- [1] I. Antcheva et al. ROOT: A C++ framework for petabyte data storage, statistical analysis and visualization. *Comput. Phys. Commun.*, 180:2499–2512, 2009.
- [2] J. Blomer, P. Buncic, and T. Fuhrmann. CernVM-FS: Delivering Scientific Software to Globally Distributed Computing Resources. In *Proceedings of the First International Workshop on Network-aware Data Management*, NDM '11, pages 49–56, New York, NY, USA, 2011. ACM.
- [3] J. Cervantes Villanueva, J. T. Palma Méndez, and E. Tejedor Saavedra. Parallelization and optimization of a High Energy Physics analysis with ROOT's RDataFrame and Spark, Sep 2018. Presented 13 Sep 2018.
- [4] M. Cremonesi et al. Using Big Data Technologies for HEP Analysis.
- [5] M. Cremonesi, O. Gutsche, B. Jayatilaka, J. Kowalkowski, S. Sehrish, L. Canali, V. Dimakopoulos, M. Girone, V. Khristenko, E. Motesnitsalis, S.-Y. Hoh, J. Pazzini, M. Zanetti, P. Elmer, J. Pivarski, A. Svyatkovskiy, I. Fisk, and A. Melo. Using Big Data Technologies for HEP Analysis.
- [6] I. Foster. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.
- [7] M. Gasthuber, H. Meinhard, and R. Jones. HNSciCloud - Overview and technical challenges. *J. Phys. : Conf. Ser.*, 898(5):052040. 5 p, 2017. HNSciCloud is co-funded by the European Commission under grant 687614.
- [8] P. Grzegorz Bogdał, Paweł Nowak. Application of Advanced Big Data Techniques in Parallel Processing of Datasets in High Energy Physics Experiments, 2018.
- [9] E. Guiraud, A. Naumann, and D. Piparo. TDataFrame: functional chains for ROOT data analyses, Jan. 2017.
- [10] O. Gutsche, L. Canali, I. Cremer, M. Cremonesi, P. Elmer, I. Fisk, M. Girone, B. Jayatilaka, J. Kowalkowski, V. Khristenko, E. Motesnitsalis, J. Pivarski, S. Sehrish, K. Surdy, and A. Svyatkovskiy. CMS Analysis and Data Reduction with Apache Spark. *CoRR*, abs/1711.00375, 2017.
- [11] L. Mascetti, H. G. Labrador, M. Lamanna, J. Moscicki, and A. Peters. CERNBox + EOS: end-user storage for science. *J. Phys.: Conf. Ser.*, 664(6):062037. 6 p, 2015.
- [12] J. T. Moscicki. *Understanding and Mastering Dynamics in Computing Grids: Processing Moldable Tasks with User-Level Overlay*. Universiteit van Amsterdam, 2011.
- [13] Apache Spark. <https://spark.apache.org>.

- [14] **Batch Jobs On Lxplus.** <https://twiki.cern.ch/twiki/bin/view/Main/BatchJobs>.
- [15] **Big Data Analysis with Scala and Spark.** <https://www.coursera.org/learn/scala-spark-big-data>.
- [16] **CERN.** <https://home.cern>.
- [17] **CERN Science Box.** <https://sciencebox.web.cern.ch/sciencebox/>.
- [18] **CERN Service Portal: Batch Service.** <http://information-technology.web.cern.ch/services/batch>.
- [19] **CERN Service Portal: LXBATCH.** <https://cern.service-now.com/service-portal/function.do?name=LXBATCH>.
- [20] **CERNBox Service.** <http://information-technology.web.cern.ch/services/CERNBox-Service>.
- [21] **CMS Big Data.** <https://cms-big-data.github.io>.
- [22] **CMS Big Data.** <https://cms-big-data.github.io/>.
- [23] **Diana HEP.** <http://diana-hep.org>.
- [24] **EOS Service.** <http://information-technology.web.cern.ch/services/eos-service>.
- [25] **Helix Nebula Science Cloud.** <http://www.helix-nebula.eu>.
- [26] **Jupyter Notebook.** <https://jupyter.org>.
- [27] **Prometheus.** <http://www.cyfronet.krakow.pl/komputery/15207,artykul,prometheus.html>.
- [28] **PyROOT Parallelization with Spark.** <https://github.com/etejedor/root-spark>.
- [29] **PySpark Package.** http://p01001532067275.cern.ch:8088/proxy/application_1516620698330_100021/executors/.
- [30] **Python.** <https://www.python.org>.
- [31] **Python Dask.** <http://docs.dask.org/en/latest/why.html>.
- [32] **RDataFrame Class Reference.** https://root.cern/doc/master/classROOT_1_1RDataFrame.html.
- [33] **ROOT Data Analysis Framework.** <https://root.cern.ch>.
- [34] **The SWAN Service.** <https://swan.web.cern.ch/>.

-
- [35] TOTEM Experiment. <https://totem-experiment.web.cern.ch/totem-experiment/physics/>.
 - [36] TOTEM Ntuple. <https://twiki.cern.ch/twiki/bin/view/TOTEM/CompNtuple>.
 - [37] A. Peters, E. Sindrilaru, and G. Adde. EOS as the present and future solution for data storage at CERN. *J. Phys.: Conf. Ser.*, 664(4):042042. 7 p, 2015.
 - [38] D. Piparo, E. Tejedor, P. Mato, L. Mascetti, J. Moscicki, and M. Lamanna. SWAN: a Service for Interactive Analysis in the Cloud. *Future Gener. Comput. Syst.*, 78(CERN-OPEN-2016-005):1071–1078. 17 p, Jun 2016.
 - [39] D. Piparo, E. Tejedor, P. Mato, L. Mascetti, J. T. Moscicki, and M. Lamanna. SWAN: A service for interactive analysis in the cloud. *Future Generation Comp. Syst.*, 78:1071–1078, 2018.
 - [40] H. Riahi, A. Aimar, A. Álvarez Ayllón, J. Balcas, D. Ciangottini, J. M. Hernández, O. Keeble, N. Magini, A. Manzi, L. Mascetti, M. Mascheroni, A. J. Tanasijczuk, and E. W. Vaandering. Integration of end-user cloud storage for cms analysis. *Future Generation Computer Systems*, 78:1079 – 1082, 2018.