LHCb-ANA-20XX-YYY
December 9, 2015

Upgrade computing TDR, a roadmap

Many people[1].

[1] *Many places*

Abstract

There is no place like the future.

1 Introduction

This document defines the roadmap towards a Technical Design Review (TDR) document for the computing model of the upgrade LHCb [1] experiment.

1.1 Wider Context

There are many factors which are at play and which will affect the nature of the distributed computing infrastructure which will exist at the time of Run-III

The projections of the combined data volumes of all experiments at the time of Run-III compared to the computing resources that are likely to be fundable, lead us to conclude that by just carrying on as normal, there is a significant risk that the required computing resources will not be sufficient. This assumption is driving all parties concerned, including funding agencies, the WLCG and experiments to devise alternative paradigms.

Relevant factors include:

- It is expected that all funding agencies will face continued tight constraints upon the funding available for computing. It is common to assume that only flat cash will be available (i.e. a constant number of currency units), but it is worth bearing in mind that even this may be optimistic.

- Whereas the LHC programme has been essentially the only data intensive scientific programme in the past, this will not be the case by 2020. Amongst other endeavours the SKA will potentially be set to produce even greater data rates, and other major astronomical instruments such as LSST and EUCLID will produce significant amount of data. Therefore we will not be the only activity seeking large computing resources.

- Recognising the previous reality, all of the major funding agencies within Europe are converging upon a policy of sharing infrastructure across many of the activities for which they are responsible. This means that the major data centres will need to evolve to a more activity-neutral offering, which is more easily used by different disciplines. High on the radar in this respect is deployment of a virtualised infrastructure and a simple single sign on framework.

- It is not axiomatic that provision of resources exclusively in home grown data centres in science laboratories will remain the most cost effective way to provision. Accordingly projects to pilot a hybrid cloud infrastructure are going ahead. These will seek to set up cloud resources provided both by traditional laboratory data centres and commercial providers. We would be prudent to assume that such a model will be prevalent in 2020.

- If EU funds are to be available (and this is a big if) they will only be available for highly shared infrastructures. From the funding agency point of view it is mandatory that particle physics engages in such cost sharing activities. This will probably take

place at a level above the experiments, but the fact that it is happening will have some effects upon HEP provisioning.

- In some quarters there is an expectation to make use of more diverse resources. ATLAS already make use of spare cycles on high end HPC machines for Monte Carlo simulation.

Naturally some of the directions indicated above are speculative, and individuals will have their own guesses as to which are likely to become reality. However it is incumbent upon us to bear all of these points in mind when building our upgrade computing model.

2 Collaborative Tools

The purpose of a collaborative tool is to support a group to accomplish a common goal. Within our collaboration, work us usually organised into groups at different levels (subdetector, analysis, institutions, ...). Our main goal is performing a physics measurement: this is achieved via several steps, from detector maintenance to data taking, processing and analysis, each step generally requiring the interaction of different people and the sharing of information. The goal of the Collaborative Tools session was first to understand the status of collaborative practices within our collaboration, and then start thinking how we can improve how we work together, exploiting tools and techniques currently available. Improvement not only in terms of scientific outcome but also in terms of working conditions.

2.1 Collaborative tools in LHCb: current status

A questionnaire was submitted to LHCb collaboration to understand the the level of collaborative practices and collect feedback. About 100 people answered the questionnaire. The main outcome of the questionnaire was that collaborative tools for archival of documentation and software are widely used within our collaboration. People involved in core software development tend to use standard tools, e.g. svn and gitlab repositories for software archival. Instead people working only on data analysis tend to use many different tools, also commercial ones, which makes sharing documentation and software more difficult. Moving to work organization, the usage of tools such as issue trackers is very common among software developers, contrary to people working only on data analysis. Code review techniques are not popular in both groups. The detailed results of the questionnaire can be found at this link.

2.2 Main topics of discussion

During the session we had discussion about the following items:

- **Automated analysis workflows**. Our analysis are usually made of several steps. Creating a detailed workflow of the analysis can have several advantages, e.g. replicate

the analysis on a different sample, share the analysis with other people, keep track (archive) of all the details of the analysis. Among the different tools to develop workflows, *gitlab* is already used with success within our collaboration. This tool makes code sharing and testing easier; as an example, it features the possibility of producing a private branch where algorithm modifcations can be reviewed and tested before being merged with the main branch. It is now also used in a few analyses and has proven to be useful in this environment as well.

- **Reproducible software**. The *everware* project aims at giving users an infrastructure to make easy to run someone else's code. Its main ingredients are the code, archived in a software repository, a dockerfile which describes the environment and an executable README, e.g. a jupiter notebook to describe the analysis flow. Such a tool is also useful to make the usage of code easier. As an example, new machine learning algorithms could be promoted thanks to simple analysis examples hosted in a *everware*-like container.

- **Analysis documentation and preservation**. A fundamental ingredient of an analysis is documentation. People in LHCb use many different media for this (slides, minutes, twiki, jira, ...) and different archival tools. Can we aim at a higher level of uniformity for analysis documentation? The ideal analysis logbook should allow to record different types of information, from text to plots, intermediate results and code. It needs to record any changes to a topic, and allow to share entries with other people for commenting and reviewing. A tool which is intended to help preserving all analysis ingredients, from the very first step to the final publication is is the Cern Analysis Preservation framework LHCb is developing together with Cern-IT and other LHC experiments.

- **Software tests**. Software testing is very important and users should collaborate with core software developers in writing tests. To enforce code testing we should agree on a set of guidelines - e.g. new code or fixes must come with a test - and on a procedure to make sure this guidelines are met - e.g code reviewing.

- **Code review**. Code review is a common practice in industry. It has several advantages, as enforcing a common style of coding, improving overall the documentation of the code, identifying bugs or problematic parts in the code as early as possible, improving coding skills. The goal is to achieve the best possible code and minimise bugs, not judging the author's ability to write code. Code should be commented and merge requests should be as small as possible. Different tools exists but *github* and *gitlab* have easy to use GUIs to comment on specific lines in the code for comments and discussion.

- **Guidelines for contributing**. In order to increase the number of people contributing to LHCb software development we should agree on a set of guidelines. The idea is to create a handbook on how to contribute to LHCb software. This handbook should tell who can contribute, on which subjects, which rules should be followed,

where people can find more information about the software to be developed. This handbook can become one of the Starterkit lessons.

2.3 Conclusions and next steps

From the questionnaire and the discussions during the session we can conclude that it is becoming clear that people want to change towards a more effective collaboration. For almost all technical challenges we already have the tools and solutions. For many there are even working prototypes tested in LHCb at the moment. We have started to identify psychological and organisational obstacles on the way to collaborative working and in the next future we will work to overcome them. We have started to collect documentation about the collaborative tools discussed during the session in the repository at this link.

3 Data processing/access and analysis models

Physics WGs would be OK with having centralised ntuple production, at least for stripping lines going to DST. This would allow a reduced number of DST stream replicas and possibly higher retention. A dedicated WG liaison would take care of organising the "ntupling" that could run at fixed intervals (e.g. two weeks). Such analysis train would allow for a more uniform and efficient data management. It would however require that the WG release their code several days ahead of the train departure. It is also envisaged for some analysis a more direct usage of μDST, without having to pass for ntuple production. This would require a better documentation and standardisation of the muDST format and tools to access it.

In the upgrade the current workflow with a Stripping will break both because of the size of the raw and of the cpu requirements needed to perform periodic strippings. Moreover, with a fully software trigger, the stripping retention would also be very likely to increase to values at which the dataset is not significantly reduced anymore (at least in terms of number of events). Turbo stream should probably become the "default" in the upgrade, many existing stripping lines could in principle make perfectly valid turbo lines with no impact on the analysis feasibility. However some WG would still need information from the entire event, or at least a significant part of it. Therefore, although in a reduced bandwidth fraction with respect to Turbo, the complete event should be kept in some form. Developments in the triggers together with the agreement between online and offline reconstruction would potentially lead to DST production directly as output of the trigger. These DST would contain all the combinatorics performed during the trigger that should not to be re-performed offline (as it is currently done in the stripping).

The possibility of having streaming at the trigger level would allow different level of information propagation. For instance, one could imagine having streams with no raw banks or only some of them, streams in between a muDST and DST where only part of

the event is saved (e.g. saving only tracks within a cone around the trigger candidate). In addition, all triggered events could be "flagged" into an event index (which could be produced by the successor of the stripping) so that users could quickly read single event via random access, this would remove the need to loop over large files, having to store related info such as flavour tagging and isolation variables and reduce the number of replicas. The result of the event indexing could be stored either in files as it has been proposed in the past (ETC files) or via a centralized service. Such a workflow however requires careful evaluation in terms of performance.

Propositions were made in order to make the Event Indexer service more useful. They include an HTTP/REST API, indexing of the MC and Turbo data, a bandwidth division feature or a functionality to search for events similar to a sample of events given as input. Interest was shown in these features, but it requires more thoughts before deciding whether they could show helpful.

The middleware to support user analysis and productions is and will stay DIRAC. An evaluation of its scalability has been made, taking into account an increase of traffic, data and maintenance. The conclusion is that new technologies must be integrated, and some core part of DIRAC must be replaced, but overall only evolutions are needed, not a revolution.

In order to cope with the increased amount of data, a more dynamic and flexible placement of data could be achieved using the data popularity and taking advantage of the analysis train paradigm.

Efforts from the institutes should be made in order to enhance computing skills of students and PhDs and each working group should keep an up-to-date analysis to be used as a best practice example. In the same spirit as the the starter kit, intermediate tutorials focusing on best practices for the various tools would be appreciated.

4 Event Model Summary

The current event model has been very successful in some aspects and less so in others. It has enabled developers to write code that performs various reconstruction tasks and analysists to create the selections they need by representing reconstructed objects and decay trees in an intuitive way. This ease of use of event model objects should be kept.

At the same time, as a result of the Array of Structures (AoS) design of the event model classes, in its current form exploitation of modern CPU features such as SIMD are more difficult. In addition, current event model objects are not composable, so if information needs to be added to objects without modifying them, they need to be copied. This costs significant memory, which in turn leads to suboptimal usage of resources.

While current event model objects were intended to be read-only after they had been

added to the Transient Event Store (TES), this was not obvious in the way they are retrieved from the TES by algorithms. As a result, paralellisation of algorithms is more difficult and would often require cloning of objects.

Since computing resources will be more and more limited compared the rate of events that needs to be processed, speed and memory efficiency should be core guidelines for the design of the new event model. New event model objects are therefore recommended to:

- be composable,
- have Structure of Arrays (SoA) memory layout,
- become read-only after their initial creation and "filling",
- use `float` precision where possible.

Transformation of event model objects between their on-disk and in-memory representaion will remain needed, for example to be able to efficiently store objects that need double representation during calculations, but fewer bits on disk.

Information such as luminosity and simulation statistics are best stored at the file level. The storage of this type of data should be reviewed to become more extensible and light-weight to allow for example production information and simulation statistics to be added. If this data can be read from the file in a lightweight way, it can be used to configure jobs or to check their configuration.

If the code required to read files is available as a single library, or a small set of libraries, with minimal dependecies, this would facilitate analyses and improve the forward compatibility of files.

To ensure data can be read in the future, data preservation should be an important factor in the way the event data is stored.

5 Hardware and Dataflow

All discussions of hardware must be tightly coupled to discussions of software. While the focus of discussions was what hardware plus software would be sufficiently performant for the online system, several people emphasized the need to process at least as much simulated data as real data. This may require that similar hardware resources be available offline and online. In addition, the life-cycle cost of the hardware and software, including capital costs, operating costs, and development plus maintenance costs for software were identified as critical issues.

We discussed three families of CPUs:

- "Ordinary" Intel and AMD x86 processors;
- ARM64 processors;
- OpenPOWER processors.

It is expected that all three can run our software. *Building the LHCb software stack regularly on an instance of each platform (continuous integration) was identified as an action item.* In addition to planning for Run 3, this will allow us to identify bugs in existing software that are not caught when building on just one architecture.

We discussed two general classes of accelerators in detail:

- GPUs, including both NVIDIA and AMD boards;
- Intel Xeon Phi boards with circa 60 CPUs each.

The GPUs may be more performant in terms of bang per buck and bang per watt. Daniel Cámpora compared a vectorized CPU velopix algorithm with an OpenCL velopix algorithm implemented on a traditional Intel Xeon node, an NVIDIA GTX 980, an AMD HD 7970, and an Intel Xeon Phi. The cost for a standard unit of performance was judged to be roughly a factor of three lower for the AMD GPU than for a traditional CPU node, and that for the NVIDIA GPU only somewhat worse than that of the AMD GPU. Alexey Badalov discussed a co-processor manager in our session, and Illya Shapoval presented a description of GaudiHive in a parallel session. The common conclusion is that low latency interfaces for Gaudi-like algorithms to use accelerators seamlessly should work, be it on the same system or in an offloaded execution. The question of *how many events in parallel are required to benefit from an offloaded execution* is dependent on the specifics of the algorithms in consideration, host-device transmissions and efficiency of the scheduling, and should be addressed in demonstrators to come.

Marco Corvo described integrating GPU hardware and an interface to Gaudi into the EFF. To decide whether to adopt GPUs, Xeon Phi's, or other accelerators as part of the vision for Run 3, *we should rapidly develop an inventory of up to six key algorithms used in the HLT, and build demonstrators to see whether these technologies can provide enough additional performance to justify the additional development effort required to build and maintain the requisite software.* It was noted that much of the design work to make algorithms parallel and vectorize them will also improve performance on "ordinary" Intel CPUs. Rainer Schwemmer reported that the existing HLT software is currenlty using about half the CPU cycles available on the farm due to cache misses, and one might reasonably hope for 90%. Manuel Schiller discussed how to design/re-design software to be cache-friendly and pointed out that the advocated approaches also promote parallelization and vectorization. As it is possible that writing high-quality, properly vectorized code for Intel CPUs will increase performance significantly, *we need to develop and evaluate optimized CPU versions of the demonstrator algorithms in parallel with those for the accelerators to make useful comparisons* (excuse the pun).

Utilizing modern vector and parallel architectures effectively requires a high level of software design and coding expertise. Therefore, *we should provide extensive training and mentoring in this area*. This should include

- hands-on workshops to teach modern, high-quality C++ and parallel programming languages such as CUDA and OpenCL;
- formal code reviews, both to provide feedback to the original coders and to provide opportunities for the reviewers to learn from others' examples;
- clear documentation of design choices so that later developers understand what was done and why; this might allow good designs to serve as templates for later code development.

Collaborating with other experiments has the potential to advance our understanding of what is required for Run 3, and should be pursued as an integral part of the software R&D plan. For example, grid resources are often "common". If we want to use particular architectures offline (CPU or accelerator), they are more likely to be supported if other experiments also adopt them. As lawyers like to say, *time is of the essence;* we need to implement the action items ASAP. Repeating what was said before:

- We should build the LHCb software stack regularly on ARM64 and OpenPOWER platforms in addition to x86 platforms.
- We should rapidly develop an inventory of up to six key algorithms used in the HLT, and build demonstrators using accelerators to see how much additional performance these technologies might provide. In parallel, we need to develop and evaluate optimized CPU versions of the same algorithms to make useful comparisons.
- We should provide extensive training and mentoring in developing state-of-the art software.

As usual, all is easier said than done. Identifying and deploying resources for these activities is necessary to move forward with the road-map and TDR.

6 Framework and Scheduling

Discussions of software must be coupled to discussions of hardware – it is the evolution of the available computing hardware over the last decade that is forcing us to rethink the basics of our framework.

Even though Moore's law has continued to evolve over the last decade, and will for the next few years, the trend of ever faster single-threaded CPU performance has been broken about ten years ago. Instead, the additional transistors have been invested into multiple CPU cores on a die, and, within these cores, into wider executation units. Furthermore, the gap between the processing units (PUs) and memory has gotten wider, and feeding the processing units with data is becoming more than ever a bottleneck.

As a result of these trends, decisions made (more than) ten years ago are no longer appropraite. For example, levaraging the 'embarrasingly parallel' nature of distinct collisions, by runnning multiple independent processes is actively contributing to increasing cache-miss, which subsequently stall these processes. Hence it is seems neccessary to bite the bullit, and deal with multi-threaded processing of events instead. This comes with its own complications and overheads, but will help in a more efficient utilization of the cache, as multiple cores will work on a coherent dataset.

However, one has instead to face the problems of enabling multiple cores to work on closely related data without trampling on each others. The solution to this problem which has been accepted in industry (eg. Microsoft, Apple, Google, Intel, ...) is what is known as a 'task-based' system. A demonstrator of such a task-based system has been implemented on top (or to be more correct, beneath) the current Gaudi frameweork.

From this demonstrator it has become clear that, even though it is possible to 'transform' Gaudi into a task-based system, the domain-specific code which is written on top of Gaudi has to be changed for the system to be able to scale properly – both in performance, as well in understanding its configuration and operation.

The main changes pertain to making algorithms and tools 'stateless' so that their main methods of execution can be wrapped inside independent tasks, and (thus) be run in parallel. Also, their input and output data must be explcitly declared in a manner which is accessible to the framework, so that the framework is able to properly schedule these tasks dynamically, insuring that all cores remain busy.

By performing these changes, the domain specific code is not cluttered with the details of the concurrency framework. This is not only needed to allow us to follow the evolution of the C++ standard in this area but also to avoid people making assumptions on the specific implementation which will still need to evolve. As a side-effect, these changes typically make the individual algorithms more focussed on a single task, making them easier to comprehend, and also more 'regular', i.e. some of the current 'boiler plate' is abstracted away, and dealt with in dedicated, shared framework code.

To be able to write algorithms this way, several requirements have to be made on the event model. First and foremost, once data is globally made available, it must remain immutable. This is due to the fact that the order of execution becomes more dynamic, only bound by control and data flow dependencies. Changes 'in situ' complicate this tremendously, not only for the required scheduling, but also for people to comprehend the behaviour of the system.

7 Conclusion

We gonna do what they said can't be done.

Acknowledgements

We express our gratitude to our colleagues in the CERN accelerator departments for the excellent performance of the LHC. We thank the technical and administrative staff at the LHCb institutes. We acknowledge support from CERN and from the national agencies: CAPES, CNPq, FAPERJ and FINEP (Brazil); NSFC (China); CNRS/IN2P3 (France); BMBF, DFG and MPG (Germany); INFN (Italy); FOM and NWO (The Netherlands); MNiSW and NCN (Poland); MEN/IFA (Romania); MinES and FANO (Russia); MinECo (Spain); SNSF and SER (Switzerland); NASU (Ukraine); STFC (United Kingdom); NSF (USA). We acknowledge the computing resources that are provided by CERN, IN2P3 (France), KIT and DESY (Germany), INFN (Italy), SURF (The Netherlands), PIC (Spain), GridPP (United Kingdom), RRCKI (Russia), CSCS (Switzerland), IFIN-HH (Romania), CBPF (Brazil), PL-GRID (Poland) and OSC (USA). We are indebted to the communities behind the multiple open source software packages on which we depend. We are also thankful for the computing resources and the access to software R&D tools provided by Yandex LLC (Russia). Individual groups or members have received support from AvH Foundation (Germany), EPLANET, Marie Skłodowska-Curie Actions and ERC (European Union), Conseil Général de Haute-Savoie, Labex ENIGMASS and OCEVU, Région Auvergne (France), RFBR (Russia), GVA, XuntaGal and GENCAT (Spain), The Royal Society and Royal Commission for the Exhibition of 1851 (United Kingdom).

References

[1] LHCb collaboration, R. Aaij *et al.*, *LHCb detector performance*, Int. J. Mod. Phys. **A30** (2015) 1530022, arXiv:1412.6352.