Optimization of Large Scale HEP Data Analysis in LHCb

Daniela Remenska, Roel Aaij, Gerhard Raven, Marcel Merk, Jeff Templon, Reinder J. Bril

E-mail: danielar@nikhef.nl
On behalf of the LHCb Collaboration

Abstract. Observation has lead to a conclusion that the physics analysis jobs run by LHCb physicists on a local computing farm (i.e. non-grid) require more efficient access to the data which resides on the Grid. Our experiments have shown that the I/O bound nature of the analysis jobs in combination with the latency due to the remote access protocols (e.g. rfio, dcap) cause a low CPU efficiency of these jobs. In addition to causing a low CPU efficiency, the remote access protocols give rise to high overhead (in terms of amount of data transferred). This paper gives an overview of the concept of pre-fetching and caching of input files in the proximity of the processing resources, which is exploited to cope with the I/O bound analysis jobs. The files are copied from Grid storage elements (using GridFTP), while concurrently performing computations, inspired from a similar idea used in the ATLAS experiment. The results illustrate that this file staging approach is relatively insensitive to the original location of the data, and a significant improvement can be achieved in terms of the CPU efficiency of an analysis job. Dealing with scalability of such a solution on the Grid environment is discussed briefly.

1. Introduction
The LHCb experiment is one of the four large experiments conducted on the Large Hadron Collider accelerator, built by CERN. The LHCb computing model defines several stages of event data processing: from online data collection, to filtering, detector simulation, reconstruction, and finally physics analysis. A key fact about the LHC is the production of immense amounts of event data annually (around 15PB). This eclipses the data rate of any other known scientific experiment and is the motivation behind the adoption of the Grid computing paradigm. The Grid storage and computing resources for the LHCb experiment are distributed across several institutes in Europe, NIKHEF being the representative for the Netherlands.

The LHCb physicists working at NIKHEF perform heavy analysis of the collected event data by submitting jobs on the Grid. There is also a motivation for using local in-house computing resources (which do not reside on the Grid) at NIKHEF, as a back-end for this purpose. Checking the correctness of the newly developed algorithms on a small subset of input data, for instance, is much more feasible to do on these local in-house computing resources, in terms of human effort, time and resource usage.

However, the physics analysis performed on such a local computing farm is experienced as inefficient because of several suspected factors, having to do with the bandwidth constrains between the local and Grid resources, the latency of access to the Grid input files, and the underlying protocols for access of the actual data across the network, among other things. The essential difference between using the Grid and the local computing resources for analyzing physics data is in the proximity between the storage and computing resources used for this analysis (latency), as well as their...
capacities, interconnections and performance characteristics. These factors result in the local analysis jobs taking too long to complete, in terms of wallclock time, and thus making the analysis inefficient. To stress the importance, the Workload Management System on the WLCG deals with this problem by sending the job to the same Tier-1 site where the input data is located. The WLCG infrastructure is designed in such a way that the storage and computing elements on a single site are interconnected with high speed dedicated networks, and these elements are inherently physically close, of course.

The LHCb physicists at NIKHEF have attempted to leverage this problem by manually copying the Grid data closer to the offline computing farm and making it available for the local analysis jobs. The closest desirable location for this input data is the local disk space on the computing farm resources. However, the disk space capacities, as well as the policies which prevent direct user control over this local “scratch” space on the computing farm have forced the physicists to instead use a shared general-purpose NFS partition for this purpose. Nevertheless, the manual effort involved in copying, bookkeeping this data and the performance characteristics of this shared storage have made this attempt rather impractical and inefficient.

This paper describes two use cases that were evaluated within the LHCb community at NIKHEF, depicting the main underlying reasons for the experienced inefficiency. Inspired by a similar approach [1] in the ATLAS experiment, a file stager was implemented in the Gaudi framework [2], targeting the wallclock time of the LHCb analysis jobs through exploring a file caching and pre-fetching approach. Finally, the experimental results from using the file stager are presented, accompanied by a discussion on the realistic performance improvements that can be expected from using the proposed solution.

2. Use cases

The local data processing facilities at NIKHEF are intended exclusively for running analysis jobs by NIKHEF physicists who participate in the LHC experiments. This “exclusiveness” sometimes gives a big advantage, because physicists are not constrained by the fair-share limits of the Grid. One such local facility is Stoomboot, a computing farm consisting of 32 worker nodes, each equipped with dual quad-core Intel Xeon E5335 2.0 GHz processors (64 bit architecture) and 24 GB of memory. Each worker node has around 100GB of local disk space, part of which is reserved as “scratch-space”, for temporary storing the necessary libraries and the output data from a job. All of the Stoomboot worker nodes share a 10Gbps connection to the outside world.

To cover the common usage of Stoomboot for LHCb analysis, two different scenarios were considered, each with four separate file-location setups (Figure 1). The same input files (made available on these four different locations upfront) were used in both scenarios, summing up to a total of 6.4GB, or 287847 events per job. The events in the input files are always considered to be independent, and this holds for any LHCb analysis job. The average event size is 25-30KB.

![Figure 1. Locations of input data for profiling analysis jobs](image)

The emphasis of the tests was to study the effects of the locality of the accessed input data on the analysis jobs, as the main suspected reason behind their perceived inefficiency on Stoomboot. Two metrics were collected for comparison of the approaches: the pure CPU time and the wallclock time of each analysis job. The CPU efficiency was deduced as a ratio between these two values.

2.1. Use case 1: Running DaVinci jobs sequentially over all events

International Conference on Computing in High Energy and Nuclear Physics (CHEP 2010) IOP Publishing
Journal of Physics: Conference Series 331 (2011) 072060
doi:10.1088/1742-6596/331/7/072060
The LHCb physics analysis with the DaVinci [3] toolset most often means processing all the events in input files that are specified for a job. There are many sub-algorithm configurations possible for a DaVinci application, and their sequence is flexible, allowing complex event selection criteria to be defined. One such representative sequence is chosen for the tests. The results of the tests in Table 1 are averaged over four repeated runs.

Table 1. Results from DaVinci sequential runs over all events in the input files

| Files location       | CPU time [min] | Main Event Loop time [min] | Wallclock time [min] | CPU efficiency [%] |
|----------------------|----------------|---------------------------|----------------------|-------------------|
| grid2.fe.infn.it     | 7.36           | 136.0                     | 142.2                | 5.0%              |
| tbn18.nikhef.nl      | 7.26           | 28.0                      | 29.1                 | 25%               |
| NFS partition        | 7.00           | 22.4                      | 24.4                 | 29.1%             |
| local storage        | 6.93           | 8.67                      | 10.0                 | 69.3%             |

As expected, the pure CPU time consumption is almost the same in all cases. However, there is a factor of 14 difference between wallclock times when accessing local input files, and remotely accessing them on a storage element (SE) in Italy, for example. Furthermore, the wallclock time is very sensitive to the physical distance of the SE, even when the same remote file access protocol is used. The high speed dedicated Gigabit network between Stoomboot and the two SEs was never saturated during the tests, so the theoretical bandwidth was not the bottleneck. The small difference between the wallclock time and the main event-loop time is due to the fact that the algorithm initialization and finalization also consumes a portion of the time.

It is evident that the efficiency of the I/O bound DaVinci analysis jobs run on Stoomboot can significantly benefit from having the input files available on a closer storage before the job needs to process them.

2.2. Use case 2: Running DaVinci jobs over a subset of events

LHCb physicists occasionally want to analyze only a small subset of the events that are spread in the input files that they specify. For this purpose, they use Event Tag Collections, which contain metadata (references) about a particular subset of interesting events for analysis present in some data files. This results in sparse skip-sequential event data access, where only a fraction of the events in the files are actually processed by such a DaVinci application. Table 2 contains the testing results from a DaVinci job that reads about 5% of the events in the input files on different locations.

Table 2. Results from DaVinci skip-sequential runs over events in the input files

| Files location       | CPU time [min] | Main Event Loop time [min] | Wallclock time [min] | CPU efficiency [%] |
|----------------------|----------------|---------------------------|----------------------|-------------------|
| grid2.fe.infn.it     | 4.15           | 85.5                      | 119.3                | 3.4%              |
| tbn18.nikhef.nl      | 4.8            | 100.2                     | 119.2                | 4.0%              |
| NFS partition        | 4.68           | 37.4                      | 42.1                 | 11.1%             |
| local storage        | 4.73           | 32.24                     | 35.1                 | 13.5%             |

Similar to sequential reads, the CPU time does not vary with the location of the input data, and the wallclock time is sensitive to their physical distance. Naturally, this affects the CPU efficiency of the analysis jobs in a related way. The sparse skip-sequential access imposes a lot of I/O operations, even in the best case of these test results, where the files are accessed on local storage; the CPU efficiency is still very low. The network latency is overcome in this case, but the disk latency remains an issue, and is amplified compared to the sequential reads scenario, because of the frequent seek operations. The cost of large seeks is substantial [4], leading to latencies $O(10\text{ms})$ which causes a significant penalty when the amount of skipped data is in the order of Megabytes.
In both scenarios, the low CPU efficiency comes from the high network latency when remote file access protocol is used. RFIO’s API [5] provides a remote version of the standard POSIX calls (open, read, write, lseek, close, etc.), enabling byte level access to Grid files. In addition, for every input file accessed remotely, there is more than 60% data transfer overhead measured [6]. On the other hand, copying input files using GridFTP does not impose any extra data volume transfer.

3. Pre-fetching and caching files: a file stager approach
To tackle the inefficiency of the analysis jobs, the first obvious idea would be to make the input files accessible on the local storage on the worker node where the job will be run. Downloading the files using the GridFTP protocol, which enables whole-file transfers between entities, has one limitation: the storage on Stoomboot is not even remotely sufficient to keep all the files necessary for running local analysis jobs, as it is a small-scale computing farm. A workaround to this limitation is to have them available only for the duration of the job execution, and remove them from the worker node’s storage once the job is finished. In practice, this means that the input files would be staged1 “per job,” i.e., at the start of that job, before any event data processing takes place. This approach has another implication: the job will be blocked due to I/O at its start, until all input files become available locally. This situation where four input files are used for a job is depicted in Figure 2 (a).

Download the files on the local storage, and removing them one by one subsequently, “on the fly,” as the job finishes processing them (Figure 2 (b)), does not reduce the introduced blocking due to I/O operations. However, it reduces the possibly high storage demands (growing linearly with the number of input files) per job and can potentially allow more analysis jobs to be executed simultaneously on a single worker node of Stoomboot.

The ever-existing performance gap between CPU and I/O speed has caused a lot of effort to be invested in reducing the application’s runtime by using caching and pre-fetching strategies [3]. The concept of pre-fetching has mostly been based on smart techniques for predicting the future input of an application, and then overlapping the CPU and I/O parts of that application. For the LHCb analysis jobs this “prediction” is trivial: the list of input files is given a-priori in one form or another, in the description of the job. Using the idea of overlapping the CPU and I/O, the application’s runtime can be potentially reduced at the cost of a small increase of local disk storage requirements. This approach is shown in Figure 2 (c). Staging the next desired file to the worker node’s local storage gives the job the time to process the event data of the previously staged file. Ideally, the staging of a file would complete just in time for the job to access it from local storage, thereby reducing the job’s total wallclock time.

1 Staging is a term used interchangeably with downloading. The terminology is adopted from the process of staging files from MSS (tape-based mass storage system).
In theory, a job may finish processing a certain file before the next necessary file has finished staging, in which case, the job execution will be blocked until that file is fully available on local storage. A typical setup of DaVinci sequence of algorithms imposes at least a few milliseconds of processing time per event, translating to a few minutes of processing time per file. Staging files from Tier-1 SEs using GridFTP has shown to take place rather fast, with an average of 1-2 minutes per file, making it unlikely for a sequential event-processing job to be blocked due to staging an input file. Unlike sequential event processing, blocking of a job execution due to I/O (waiting for a file to finish staging) can indeed occur, depending on the percentage of preselected events in these files that are actually processed by the job.

The Gaudi framework [2] plays a central role in the development of the physics analysis software within the LHCb collaboration, and as such, its existing Services and Tools components serve as a base for the design and implementation of the file stager. Concurrency in staging and processing files is achieved by spawning child process instances which take care of staging the individual input files. A two-stage fallback mechanism is also provided in case the local disk space requirements for staging the input file(s) on Stoomboot cannot be met: first, the stager will attempt to stage a file on the local NFS partition accessible from Stoomboot (the rationale for this choice comes from the results in section 2). Failing that, it will attempt to create a replica (if not present) on the closest SE, preferably at Nikhef. The job will then read the data from this replica location using the available remote access protocol implementation on that SE.

The end-user is presented with a simple interface for interaction with the file stager through the job description file. Using this interface, the default values of the properties which steer the execution of
4. Performance evaluation

To be able to evaluate the performance achievements of the file stager, the same use cases from section 2 were used to devise tests, with the attempt to depict a realistic workload of LHCb analysis jobs on Stoomboot. A total of 500 analysis jobs were selected for testing. A reference for every job run with the file stager was necessary for comparing and determining the performance improvements. Therefore, for every analysis job using the file stager, a matching (equivalent) analysis job without the stager was devised, resulting in 1000 jobs overall. The jobs were submitted to Stoomboot in batches of 50 over a few days of time, in the course of which both idle and heavy-load periods on the farm were observed. The tests resulted in 2.7TB of data (belonging to 2500 files) being staged over the entire testing period.

The input datasets for the jobs were selected from a random production run (using the LFC) with no particular preference. To avoid being limited by the reading speed of the physical storage on any single Tier-1 SE accessed by these jobs, each job-pair was configured with a unique input dataset, by supplying it with the appropriate LFN handles.

The parameters of interest for evaluation were the CPU efficiency and the failure rate of the analysis jobs (caused exclusively by the new file stager software). The failure rate of the analysis jobs, defined as the percentage of failed jobs, was indicated as an important parameter for evaluation because of the overhead of re-submitting the failed jobs, in case of occurrence of failures which prevent the jobs from finishing successfully.

From the analysis jobs that were not using the file stager, 7 (1.4%) failed to create a connection to one or more remote input files, and as a result could not process the events in those files. Connection timeouts occurred for several of the analysis jobs that were using the file stager, whenever the download speed would drop to less than 1Mb/sec, which was typically the case when many jobs landed on a single worker node and started staging files at the same moment, occasionally saturating the network link of that node. The jobs were able to recover from such timeouts by retrying the staging procedure using a different SE as a source, but their CPU efficiencies were nevertheless affected by this. However, 11 (2.2%) jobs did not finish successfully. After several connection timeouts caused by the unsuccessful attempts to stage a file over the saturated network, the connection was closed.

The CPU efficiency measurements from the tests are depicted in Figure 3 and Figure 4 (μ represents the average CPU efficiency). Most of the jobs that were not using the file stager had a poor CPU efficiency. Only around 30 (6%) of the jobs had CPU efficiency larger than 35%. When comparing these numbers with the analysis jobs that were using the file stager, the benefits become apparent. Even though the standard deviation of the measurements is larger in this case (σ =21.9), it is evident that around 90% of the data falls in the interval between 35% and 100% CPU efficiency.

![Figure 3. Histogram of CPU efficiencies (%) for 500 sequential analysis jobs which did not use the stager](image1)

![Figure 4. Histogram of CPU efficiencies (%) for 500 sequential analysis jobs which used the stager](image2)
worker nodes where the jobs were executed. In addition, every child process spawned for the purpose of staging a file also imposes a certain CPU time. However, the Stoomboot scheduler has no notion of multi-threaded or multi-process jobs, so a smaller portion of the CPU will effectively be dedicated per analysis job than if the stager is not used. For an idle worker node with 8 cores, for instance, 8 submitted jobs belonging to the same batch (therefore arriving in Stoomboot’s queue at the same moment) will be sent to a single worker node and will spawn child processes in approximately the same time period. As a consequence, the actual available CPU portion per running job on that worker node will be somewhat smaller.

Comparable results were obtained when testing the skip-sequential event processing use case. Therefore, similar conclusions can be drawn with respect to the efficiency improvements and robustness achievements of the stager in this case. The average CPU efficiency achieved with the file stager was in this case 10% smaller than in the case of sequential event processing. The selection criteria were chosen such that around 50% of the events in the input files were actually processed by the jobs.

5. Conclusions

In this paper, we presented a file staging solution to the problem of inefficient LHCb analysis jobs run on a local computing farm. The file stager is implemented in the Gaudi framework, and provides a more efficient access to the data which (by definition) resides on the Grid. The mechanism deployed is based on pre-fetching input files from the Grid storage, before an analysis job needs the data in each corresponding file. The pre-fetching occurs while computation takes place, which gives the file stager the time to transfer the desired data to the worker node. Ideally, the pre-fetching (staging) would complete just in time (or preferably earlier) for the CPU to access the necessary event data for processing, thereby increasing the efficiency of the analysis job.

One central trait of the solution is that it makes temporary use of the local disk space of the computing farm worker node where the job is run as well as the Grid FTP services for staging the files, in return for a more efficient analysis job execution. Unlike the original remote file access which the analysis jobs use for acquiring the input data for processing, the proximity of the Grid storage becomes irrelevant if the file stager software is used.

The efficiency improvements and robustness achievements were quantified. It was shown that the CPU efficiency of the sequential event-processing analysis jobs run on Stoomboot can be increased by a factor of 2 to 3 (depending on the conditions present during the job execution) by employing the file stager. However, the robustness of the current implementation needs improvement.

The applicability of the file staging solution in the Grid environment as a back-end for the actual analysis job execution requires no conceptual changes in the current implementation. The software needs to be deployed and globally accessible on the same repository (at CERN) with the official LHCb software stack. In principle, the limiting factor in the local user analysis on Stoomboot is the proximity of the Grid storage where the data resides (accumulating latency of reading remote data) as well as the network connectivity between the worker nodes and the Grid storage, which should not be a limiting factor when the analysis jobs are run on the Grid. Their efficiencies can be further investigated. Nevertheless, for analysis jobs that can be classified as I/O bound, the CPU wallclock time improvements from using the file stager software in the Grid environment are expected to be comparable with the results from section 4. The current LHCb computing model defines a static and pre-defined distribution of file replicas across Tier-1 sites. As bandwidth between the WLCG resources becomes less of a scarce resource, similar approach to the file stager can be adopted, where the input files are “cached” on-demand in the proximity of the analysis jobs execution environment. However, if such a caching mechanism is deployed globally on the WLCG, this can have an impact on the local disk capacity preferences of the purchased Tier-1 worker nodes.

References

[1] ATLAS File Stager URL https://twiki.cern.ch/twiki/bin/view/Main/FileStager
[2] Cattaneo, M. 1998 GAUDI Framework Architecture Design Document
[3] DaVinci physics analysis software for the LHCb experiment URL
VanDeBogart, Frost, and Kohler. June 2009 *Reducing Seek Overhead with Application* -Directed Prefetching. Proceedings of the 2009 USENIX Annual Technical Conference, San Diego, CA.

CERN Advanced Storage Manager URL [http://castor.web.cern.ch/castor/](http://castor.web.cern.ch/castor/).

Remenska, D. 2010 *A file staging approach to optimizing large scale HEP data analysis* (TU/e Eindhoven)