Journal of Physics: Conference Series

PROOF on a Batch System

To cite this article: W Behrenhoff *et al* 2011 *J. Phys.: Conf. Ser.* **331** 072063

PROOF on a Batch System

W Behrenhoff[1], W Ehrenfeld[1], J Samson[1] and H Stadie[2]

[1] Deutsches Elektronen-Synchrotron DESY, Hamburg, Germany
[2] Universität Hamburg, Hamburg, Germany

E-mail: joergen.samson@desy.de

Abstract. The "parallel ROOT facility" (PROOF) from the ROOT framework provides a mechanism to distribute the load of interactive and non-interactive ROOT sessions on a set of worker nodes optimising the overall execution time. While PROOF is designed to work on a dedicated PROOF cluster, the benefits of PROOF can also be used on top of another batch scheduling system with the help of temporary *per user* PROOF clusters. We will present a lightweight tool which starts a temporary PROOF cluster on a SGE based batch cluster or, via a plugin mechanism, e.g. on a set of bare desktops via ssh. Further, we will present the result of benchmarks which compare the data throughput for different data storage back ends available at the German National Analysis Facility (NAF) at DESY.

1. Introduction

The National Analysis Facility (NAF) was extensively used by the participating experiments in 2010 (see reference [1] for details). In section 2, we give some background to the computing environment at the NAF. In section 3, we consider the requirements for parallelisation in high energy physics and introduce PROOF as a framework that meets these requirements. Section 4 presents our scripts that allow the NAF users to start their own PROOF cluster on the NAF batch system. Finally in section 5, we present measurements demonstrating the scalability of PROOF in this setup at the NAF. We consider dCache and Lustre as back ends; dCache as a standard Grid storage technology and Lustre as a massively parallel distributed file system.

2. The National Analysis Facility

The NAF is set up in the framework of the Helmholtz Alliance "Physics at the TeraScale". It is intended as an analysis platform for the LHC experiments ATLAS, CMS and LHCb as well as the future ILC experiments [2]. The access is restricted to members from German high energy institutes. The NAF has been built and is operated DESY and is distributed over the Hamburg and Zeuthen sites. Operation is done in close collaboration with the German groups of the experiments. The NAF provides computing power as well as storage to the NAF users. The computing power consists of a batch system with 1400 cores and additional Grid resources to perform data analysis. Storage is available via Lustre as parallel, high performance file system with InfiniBand connection (230 TB local storage) and via connection to dCache resources at DESY.

The local batch system of the NAF is powered by the Sun Grid Engine (SGE). SGE is an open source batch-queuing system, developed by Sun Microsystems (now Oracle). The choice of SGE as batch system for the NAF relies, among others, on long term experience

with SGE at DESY. The benefits of SGE include advanced scheduling algorithms and policy-based resource allocation. SGE allows fairshare queuing and job scheduling to guarantee equal resource availability between different experiments and users on the NAF. With AFS and Lustre, distributed file systems are available, which are mounted on all cluster nodes at the NAF.

3. Parallelisation in High Energy Physics

Parallelisation in high energy physics (HEP) is straightforward from an algorithmic point of view: A given algorithm is applied to a large number of relatively small sized events. Since the events are independent of each other, the order of processing does not matter. Therefore, it is sufficient to process different events in parallel, using single threaded event processing algorithms. The involved work to parallelise the single event processing is not needed. The main tasks of a framework which does parallelisation on an event level, are to split and distribute the input data, as well as collecting the output of the single jobs.

In an environment with large scale distributed storage – like Lustre or dCache – the input data does not have to be distributed physically, because storage is distributed and the data is already available on every node. Only the bookkeeping of the data distribution has to be done. Merging the output of the single jobs is straightforward, too. In most cases only adding histograms of independent subsets, concatenation of output lists or similar tasks need to be done. The output of data processing in HEP tends to be small compared to the input. This still holds for the sum of temporary outputs, if data processing is split into parallel jobs. Hence, the output can be collected from the worker nodes via network connection or shared/distributed disk space. This parallelisation scheme fits well to what a batch cluster provides without providing specialised parallel hardware (shared memory, etc.).

Although from the algorithmic point of view parallelisation in HEP is straightforward, a framework to parallelise data processing in HEP still has to overcome some challenges. Some of these are: Data distribution to the worker nodes, including dataset splitting and adaption to varying worker node performance or load; data transportation; collection and merging of output objects.

The key question of parallelisation is the scalability. While the event based approach itself scales almost perfectly, in detail there are various factors that affect scalability. A framework providing parallelisation with the scheme described above still needs time for start up, data assignment and merging of the output objects. This produces an overhead compared to running in a single thread. Additionally, the data splitting and distribution might not be perfect, i.e. the user must wait for the last chunk of data to be processed. Also, the I/O of different worker nodes might interfere. This interference might occur on the level where I/O of independent data (e.g. different files) degrades performance, or on the level where different worker nodes have to access the same piece of data (e.g. two workers access the same file).

3.1. PROOF – The Parallel ROOT Facility

The *Parallel ROOT Facility* (PROOF) [3] extends ROOT [3] to allow transparent analysis of sets of ROOT files in parallel on remote computer clusters or multi-core computers.

One design goal of PROOF is to minimise the difference between local sessions and PROOF sessions. It is designed to work on (C++) objects in the ROOT data store, esp. on *TTree*s. PROOF does automatic dataset splitting (including splitting on a sub-file level, if beneficial), does load-balancing to account varying worker node performance and does the data distribution to the worker nodes transparently to the user. PROOF provides a mechanism for (semi-)automatic merging of common ROOT output objects like histograms, graphs, and n-tuples. For complex projects, esp. those with external dependencies, PROOF provides mechanisms to distribute code to worker nodes in form of source code packages.

A PROOF session follows the client-server principle. The client, which might be an interactive user session or a PROOF enabled stand-alone program connects to a dedicated PROOF master. This PROOF master again connects to a sub master or directly to a set of PROOF worker nodes. The design of PROOF had a dedicated PROOF cluster in mind. Therefore, the master and the clients have to run and listen for incoming connections, before a user can start a PROOF session. There is no support by PROOF to start the PROOF master and worker nodes on demand.

4. Starting a Temporary PROOF Cluster on the NAF Batch System

In the NAF no CPUs are reserved for running a dedicated PROOF cluster. However, the concept of *parallel environments* (*PE*) allows to get support by the SGE batch system to start a PROOF cluster on the batch system. A PROOF specific PE helps to request a given number of simultaneously available CPU slots from the batch system. A single PE job requests a number of slots on the batch system. When the PE job starts, all the requested slots are available and all PROOF worker nodes can be started at the same time. Additionally, this PROOF-PE tries to distribute the slots over different physical machines to minimise interference of different PROOF slaves and to avoid possible I/O bottlenecks.

A script to deploy a temporary PROOF cluster on the NAF batch system needs to be able to do the following tasks. At first, it has to acquire resources from the SGE batch system. Configuration files for each of the cluster nodes and a cluster wide configuration has to be created. The PROOF master and cluster nodes must be started and stopped on demand. The user should be able to change the configuration of the temporary PROOF cluster. Furthermore, some issues need to be addressed. Interference between the temporary PROOF clusters of different users must be avoided (esp. conflicting network ports). The PROOF nodes should start at the same time to avoid waisting CPU time with waiting for the PROOF cluster to provide the requested performance. Finally, the script should check whether job submission was successful and handle errors in case of failures. The needed steps to start such a temporary PROOF cluster are illustrated in figure 1.

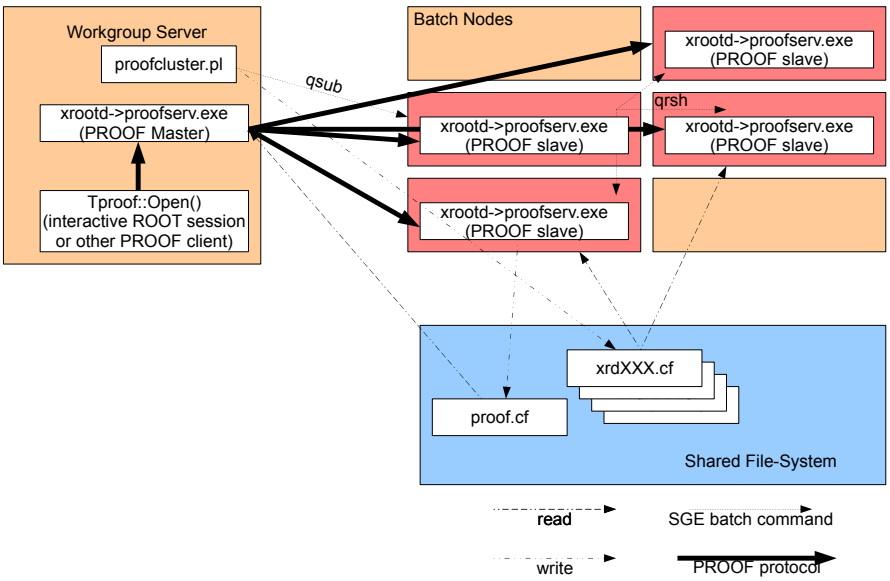

Figure 1. Starting a temporary PROOF cluster

The proofcluster.pl Script To execute the task of configuring, starting and stopping a temporary PROOF cluster in a transparent way, a Perl script was written for CMS [4]. This script is specific to the NAF. Especially, some pathes are hard coded. This script uses the SGE commands to submit a job to a PROOF specific PE. Optionally, the script can set up a CMS specific software environment on the PROOF nodes. This script allows the users to configure their own PROOF clusters. It consists of one single file and thus is easy to deploy.

The Python Reimplementation To increase the maintainability and extensibility of the script a rewrite in Python has been done. Currently, this rewrite is still specific to the NAF, but other environments can be supported via plug-in hooks. A plug-in which deploys a temporary PROOF cluster on a set of bare desktops via ssh is an example for such a plug-in. To further increase the maintainability while keeping deployment easy, a planned feature is to move the development to several files, but merge these files in a build process to create a stand-alone script. For this script itself and further information, see [5].

5. Scalability of PROOF on the NAF

To examine the scalability of PROOF on the NAF, a PROOF enabled analysis has been run several times on temporary PROOF clusters of different sizes.

The analysis used for these tests is implemented in SFrame [6]. SFrame is a PROOF enabled general purpose framework for ROOT based analyses. The benchmark analysis is a real live search for di-tau events with missing E_T. It is a cut based analysis which reads n-tuples (ATLAS *D3PD*s) of the ATLAS SUSY stream. In this analysis $\approx$250 out of $\approx$3300 branches are read. The analysis performs cuts and combinatorial operations and creates $\approx$600 histograms as output.

The D3PDs from the ATLAS SUSY stream are flat ROOT trees partitioned in ROOT files of $\approx$650 MB written with ROOT v5.22. In the tests presented, two different dataset sizes have been used. The smaller dataset consists of 95 files with 2,263,186 events in total, the larger dataset consists of 297 files with 7,124,554 events. The data files were read from the dCache or the Lustre instances for the tests.

To determine the scalability of PROOF within the test analysis, the same analysis is executed several times with different PROOF cluster sizes. For each of the runs, two values are recorded. The wall clock time (T_{tot}) for running the test analysis and the *pure* event rate (r_{full}). The latter is the average peak rate of the events processed when all workers are processing events. Both values of the single threaded case (T^1_{tot}, r^1_{full}) are taken as reference and the speed up (s_T,s_r) relative to these values are calculated for each run with more than one PROOF worker node.

$$s^n_T = \frac{T^1_{tot}}{T^n_{tot}}\,, \quad s^n_r = \frac{r^n_{full}}{r^1_{full}} \tag{1}$$

To rate the results, a simple model is assumed: the total job time is the sum of three terms. First, a term proportional to one over the number of worker nodes, which corresponds to a perfect scaling behaviour of the independent event processing by the worker nodes. Second, a constant term, which models initialisation and clean-up tasks of PROOF and the main program. Finally, a linear term proportional to the number of workers, which models the time needed to collect and merge the output produced by the worker nodes.

$$T_{model}(n_{work}) = \frac{T_0}{n_{work}} + T_{const} + T_{lin} \cdot n_{work}. \tag{2}$$

With a given reference time T^1_{tot} the speed-up factor predicted by this model is:

$$s_T(n_{work}) = \frac{n_{work}}{a + b \cdot n_{work} + c \cdot n^2_{work}}, \quad \text{with}\, a = \frac{T_0}{T^1_{tot}}, b = \frac{T_{const}}{T^1_{tot}}, c = \frac{T_{lin}}{T^1_{tot}} \text{ and } a{+}b{+}c = 1. \tag{3}$$

In figures 2 – 4 the speed-up factors for different PROOF cluster sizes are plotted. The triangles give the speed up of the wall clock time for processing a dataset. The right y-axis displays the corresponding time. The line shows the speed up predicted by the simple model, where the solid part of the lines indicates the region which was fitted to the data to determine the model parameters. The circles in the inner graph show the speed up of the event rate as described above. The dashed line in these plots has the slope 1, which corresponds to the expected event rate speed up in case of non-interfering worker nodes.

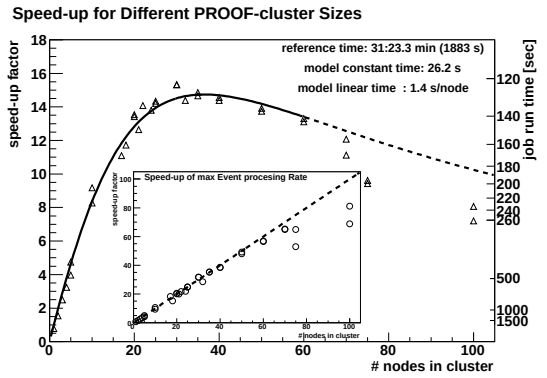

Figure 2. Test *TDL*: speed-up factor of processing ≈7 mil. events read from dCache with different PROOF cluster sizes.

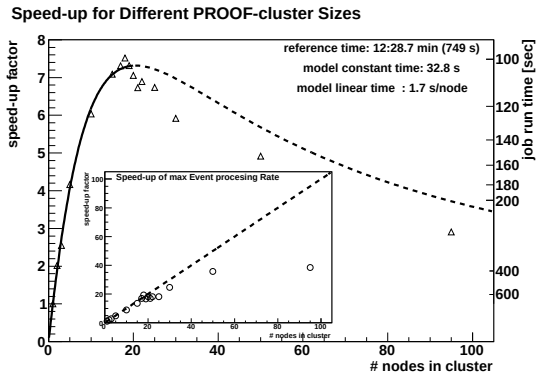

Figure 3. Test *TDS*: speed-up factor of processing ≈2 mil. events read from dCache with different PROOF cluster sizes.

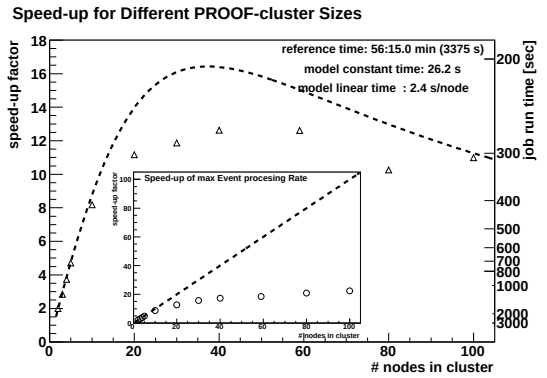

Figure 4. Test *TLL*: speed-up factor of processing ≈7 mil. events read from Lustre with different PROOF cluster sizes.

The three test cases differ in dataset size and storage back end. The first test (*TDL*), shown in figure 2, uses the dataset with 297 files and ≈7 mil. events. The data is read from dCache. The second test (*TDS*, figure 3) uses the smaller dataset with ≈2 mil. event in 95 files. The files again are read from dCache. The last test (*TLL*, figure 4) was done with the large dataset again but Lustre was used as data back end.

The test *TDL* has the best speed up with ≈30 – 40 PROOF worker nodes working in parallel. With this number of worker nodes the time needed to process this dataset can be reduced from more than half an hour to below two and a half minutes. Increasing further the number of worker nodes reduces the speed-up factor and increases the time needed to process the dataset, consistent with the model. The explanation is the following: the growth of the linear-time term overcompensates the shrinking of the "one over n" term. In other words, the overhead of collecting and merging the output eats up the additional speed up by the growing number of worker nodes. At around 70 worker nodes, the data and the model start to deviate. The inner plot shows the reason for this: From ≈70 workers on the *pure* event rate does not scale linearly any more, as assumed in the model, i.e. the worker nodes start to interfere. One explanation for

this interference might be the fact that the number of worker nodes approaches the number of files in the dataset. In this case the probability increases that PROOF distributes chunks of the same ROOT file to different worker nodes at the same time. In this case the storage back end might suffer in performance, if two worker nodes access the same file at the same time. However, other interference mechanisms are also possible.

The results of the test *TDS* with the smaller dataset differ from the first test that the best speed up is already achieved with 15 – 20 worker nodes. Also the best speed up is a little above seven in this case instead of around 15 in the first case. This behaviour can be reproduced by the model. However, the total time for processing the smaller dataset is still reduced from about twelve minutes to below two minutes. The deviation between data and our model starts already at 20 worker nodes. Given the smaller number of files, the probability of two worker nodes processing the same data file is higher in this case than in the first test.

The last test *TLL* is similar to *TDL*, except the location of the files in the dataset. In this test, the speed up depending on the number of PROOF worker nodes cannot be described well by the model. The inner plot shows that the *pure* event rate already stops to scale as expected at 10 – 20 worker nodes. The interference between the PROOF jobs in this case is so strong that the *pure* event rate stays almost the same with 20 worker nodes or more. That means, the throughput of the storage back end already saturated at 20 slaves. However, it is worth noting that all input files were stored in the same directory of the Lustre instance in this test. Hence, the throughput is not necessarily limited by the I/O bandwidth of the Lustre instance, but might be limited by the meta data handling.

6. Conclusion

PROOF is a framework to parallelise HEP analysis on an event basis with smart data distribution and output merging. Although PROOF has been designed with a dedicated PROOF cluster in mind, the users of the NAF can use PROOF on the SGE batch system. The scripts presented above allow the users to start their personal temporary PROOF clusters in a transparent and convenient way. The tests and the model show that with help of PROOF the turn around time of real life analyses can be reduced from the order of half an hour to few minutes, if the number of PROOF worker nodes is well chosen. The deviation between the simple model and the data indicates that the I/O back end can be a limiting factor concerning scalability. However, the test with the dCache back end shows that the dCache setup at the NAF is able to provide data to at least 70 worker nodes in this benchmark without suffering of an I/O bottleneck.

Acknowledgments

We thank the NAF members of DESY-IT and DESY-DV for the kind support of this project. The project was partially funded by the Helmholtz Allianz “Physics at the Terascale”

References

[1] Aplin S, Ehrenfeld W, Haupt A, Kemp Y, Langenbruch C, Leffhalm K, Lucaci-Timoce A and Stadie H 2010 The National Analysis Facility at DESY: Status and Use Cases by the Participating Experiments These proceedings
[2] Haupt A and Kemp Y 2010 *J. Phys. Conf. Ser.* **219** 052007
[3] Antcheva I *et al.* 2009 *Comput. Phys. Commun.* **180** 2499–2512
[4] Behrenhoff W 2008 Development of Interactive Analysis Tools for the CMS Experiment. diploma thesis Universität Hamburg URL `http://www.desy.de/~wbehrenh/diplom.pdf`
[5] URL `http://sourceforge.net/projects/proofonbatch/`
[6] Krasznahorkay A *et al.* SFrame - a root analysis framework URL `http://sframe.sourceforge.net/`