Studying ROOT I/O performance with PROOF-Lite

C. Aguado-Sanchez, J. Blomer, P. Buncic, I. Charalampidis, G. Ganis¹, M. Nabozny, F. Rademakers
CERN, PH-SFT, CH-1211 Geneva 23, Switzerland

Abstract. Parallelism aims to improve computing performance by executing a set of computations concurrently. Since the advent of today’s many-core machines, the full exploitation of the available CPU power has been one of the main challenges. In High Energy Physics (HEP) final data analysis, the bottleneck is not only the available CPU but also the available I/O bandwidth. Most of today’s HEP analysis frameworks depend on ROOT I/O. In this paper, we will discuss the results obtained studying the ROOT I/O performance using PROOF-Lite, a parallel multi-process approach whose results can be directly applied to the generic case of many jobs running concurrently on the same machine. We will also discuss the impact of running the applications in virtual machines.

1. Introduction
With the startup of the LHC accelerator, particle physicists have started to have at their disposal large amounts of data to be analyzed. End-user analysis activities are manyfold, ranging from data reduction to toy simulations to final fits. While fits and simulations are mostly CPU-intensive activities, data mining requires a large amount of I/O which, in the first steps of the whole process, often turns out to be the bottleneck. For example, to go through a data sample of the order of 10 TB with reasonable turn arounds, e.g. 1 hour, would require an effective input rate of the order of GB/s, which is far from being easy to reach with the current I/O hardware.

The recent years have seen an increase in the available CPU in the form of many-cores per machine. The existing applications, mostly based on a sequential programming paradigm, do not automatically exploit the additional computing power. Adaptation of the applications to a parallel programming model requires developer training and full re-writing of the code. This is going to take time and it may be even out of question for a certain number of LHC applications.

In fact, the only realistic way to exploit multi-cores with existing applications is to run many of these applications concurrently. This may not be optimal with respect to some resources - like memory or I/O - but when these are not critical it represents an effective way to run faster.

However, when resources other than CPU are critical, the resulting speed-up in processing rate may be quite different from the one expected from the number of concurrent jobs. In this paper, we investigate the performance of ROOT I/O [1] in concurrent reading access scenarios. To do this, we use PROOF-Lite, a version of PROOF [2] designed for multi-core machines. PROOF-Lite implements parallelism at multi-process level; therefore, it models also the case of many independent ROOT jobs running concurrently on a worker node. Given that most of the LHC applications are based on ROOT [3], we believe that the results of this study are valid for most of the LHC scenarios.

¹ Contact author: gerardo.ganis@cern.ch
2. Understanding I/O performance scaling of ROOT applications

Under simple assumptions, the scaling of processing rate $R_p$ for I/O bound tasks is given by the following formula:

$$R_p(N_p) = N_p \cdot R_1 \cdot \left[1 + \frac{R_1}{R_{IO}} \cdot \left(\frac{N_p}{\min(T_{IO} \cdot R_{IO} \cdot N_p)} - 1\right)\right]^{-1}$$  \hspace{1cm} (1)

Here $N_{IO}$ and $R_{IO}$ are, respectively, the number and the average I/O rate of independent I/O controllers, so that $T_{IO}(\equiv R_{IO} \cdot N_{IO})$ is the total available I/O bandwidth. $R_1$ is the effective processing rate for one process only, which can be written in terms of $R_{IO}$ and of $R_{CPU}$, the rate of purely CPU tasks (event decompression and processing):

$$R_1^{-1} = R_{IO}^{-1} + R_{CPU}^{-1}. \hspace{1cm} (2)$$

The main message coming from this formula is that ideal scaling is perturbed by a saturation term which is small when $R_{IO} \gg R_{CPU}$. This happens when i) $R_{IO}$ is large, because of high performance I/O devices; or ii) when the amount of processing per event is large, so that $R_{CPU}$ is small and dominate $R_1$. The formula also shows that the rate will saturate to $T_{IO}$ for large number of processes$^2$.

3. Test machine and benchmark suite

**Hardware.** The benchmarks have been run on a HP Proliant server with 4 Intel Xeon X7460 2.66 GHz CPUs (24 cores in total) with VT-x support$^3$, 48 GB of RAM, 1 Gbit/s NIC. The machine has attached a SAS disk, 150 GB, 10000 RPMs, and an SSD disk Intel X25-M, 160 GB. The operating system installed on the machine was Scientific Linux CERN 5 (SLC5) [4].

**Benchmark suite.** The ROOT version used for these benchmarks was the production version 5.26/00 [5] with the latest available patches. The analysis consisted in reading a fraction (about 75%) of events based on the Event class filling some distributions$^4$. The real processing component in the analysis was very small and the global processing rate was dominated by I/O and decompression. For the test, files were created with basket size of 32 kB; the number of files was such that there were at least 2 files per worker in average. At each point the measurement was repeated 4 times, using the average and the RMS to estimate the rate and the error. Between runs the system cache of the relevant machine (so the remote one during network tests) was cleared with a call to the POSIX system function `posix_fadvise` with option `POSIX_FADV_DONTNEED`. Finally, local caching - via TTreeCache - was enabled during the runs.

4. Results of the tests

As discussed in [6], file layout optimizations have an important role in overall reading performance; high fragmentation may introduce important inefficiencies. The results of our tests are a sort of ideal benchmark, because the Event structure has been designed to exploit all possible optimizations.

**Reading from RAM.** The amount of RAM available on worker nodes has increased considerably in recent years, making de facto this component an additional layer of cache in the storage infrastructure. To quantify the specifications of RAM as an I/O device, we have run the benchmark tests reading the data from the `/dev/shm` memory partition. Figure 1a shows

$^2$ Note that the total I/O at saturation depends also on the way files are structured; so the value obtained from these measurements cannot be directly compared with the device nominal specifications.

$^3$ Support for Extended Page Table was not included.

$^4$ The Event class and the analysis selector can be found at http://root.cern.ch/viewvc/trunk/test/Event.h and http://root.cern.ch/viewvc/trunk/tutorials/proof/ProofEventProc.C.h, respectively.
Table 1. Results of the fit to equation 1 of the measurements shown in Figures 1. See text for details.

|      | $R_{CPU}$ | $R_{IO}$ | $T_{IO}$ |
|------|-----------|----------|----------|
| RAM  | 35.61±0.01 MB/s | 326.2±0.1 MB/s | 1797.8±0.2 MB/s |
| SAS  | 37.0±0.3 MB/s   | 53.1±0.3 MB/s   | 70.400±0.002 MB/s |
| SSD  | 45.6±0.1 MB/s   | 61.3±0.1 MB/s   | 139.7±0.2 MB/s   |

the result of the measurement. The scalability in this case is very good. The I/O bandwidth is very large, $O(2 \text{ GB/s})$, and the measurement is in fact a measurement of the CPU-dependent components of the processing rate, i.e. of the decompression rate. We can use equation 1 to extract some effective numbers characterizing the I/O performance. The result of the fit to the equation is also shown super-imposed on the figure. The numerical results of the fit are given in Table 1. We see that the I/O rate is the largest contributor to the overall rate. The job becomes effectively CPU-bound.

Figure 1. I/O scaling curves obtained on the 24-core machine in different scenarios. On plot a (left): reading from RAM. On plot b (right): reading from the SAS HDD (A); reading from the SSD (B). In all cases the super-imposed lines are the result of the fit to equation 1.

Reading from local disks. Figure 1b shows the scaling results obtained when reading from local disk systems on the 24-core machine. The shape is in agreement with the one predicted under simple assumptions discussed in Sec. 2. The results of the fit to equation 1 are super-imposed to measured points. As expected, the rate saturates at the value representing the amount of I/O that the device can provide for this kind of analysis. For example, we see that, in the conditions of the test, the SAS hard drive we can feed 3-4 processes, reaching about 65 MB/s (curve A), and the single SSD can feed 7-8 processes for a global I/O of about 120 MB/s (curve B). For completeness, we report in Table 1 the results of the fits to equation 1. As expected from the crudeness of the model, the fits turn out to be not good in the sense of the standard definition of fit goodness. The small errors, therefore, are not much meaningful and just indicate that the minima were well defined. Since we are dealing with the same analysis performed on the same machine, the model predicts that $R_{CPU}$ is the same in the three case. This parameter is basically measured by the derivative at small $N_p$, so it depends on the effective ramp-up of the I/O device bandwidth, which is the most fragile ingredient behind equation 1. Therefore, we believe that the 25% agreement is rather remarkable.

Reading from remote disks. Another common case is the one where the worker node reads the data to process from remote servers via the network. In such a case an additional
potential bottleneck is the network bandwidth. Figure 2 shows the results of the scalability benchmark reading the data from a single Xrootd server (curve A) and from an Xrootd cluster with five data servers (curve B). In the first case we see that the plateau rate does not reflect the available network bandwidth (about 128 MB/s) but the maximum bandwidth of the HDD behind the Xrootd server. In the second case the local bandwidth of the file server system (about 200 MB/s) is larger of the available network bandwidth, so that the rate is network limited.

![Figure 2. 24-core machine: I/O scaling when reading from a single Xrootd server (curve A) and from a Xrootd system with 5 disks (curve B).](image1)

![Figure 3. Impact of TTreeCache (on the 24-core machine) (curve A: local cache ON; curve B: local cache OFF).](image2)

**Impact of local cache.** Client-side caching is a technique to improve the I/O performance over network. By default, in ROOT, client caching is switched off for local files because no gain is expected at the expense of the (little) overhead required by setting up the caching machinery\(^5\). However, in the case of concurrent access of many processes to the same local device, one can expect a benefit from using the cache, because the read-out can be less fragmented and more efficient. To quantify the effect we have measured the I/O rate scaling for the SSD device on the 24-core machine disabling the usage of the cache and compared the result with the reference values shown in Figure 1a. The two scaling results are shown in Figure 3. Disabling the cache reduces to about 70% the overall I/O rate, because the number of accesses to disk is increased and the efficiency of device sharing reduced.

**Impact of partition information.** Finally we have studied the impact on the I/O performance of the way the disks of a multiple hard-drive system are configured. Performance-wise, RAID systems have the advantage of distributing the load evenly across the disks. For this test we used an 8 core CPU system (2x Intel Xeon 2.8 GHz), running SLC5 [4], 16 GB RAM, 10 Gbit/s NIC, and 24 standard HDD, 750 GB, 7200 RPMs which could be organized in several configurations\(^6\).

On the left of Figure 4 we have sketched the setups. In case 1 all the files were read from the same hard-drive; in case 2 the files were evenly distributed on four hard-drives but the processing order was such to maximize the number of processes reading from the same disk (partition unawareness); in case 3 the files were also evenly distributed but the processing order

---

5 In reading ROOT trees, caching is implemented via the class TTreeCache. After a learning phase of 100 events, the cache is able to know precisely which chunks of the files needed by the application, so that the list of buffers can be read-ahead more efficiently.

6 Courtesy of the ATLAS group at the University of Wisconsin.
Figure 4. Setup of the partition awareness test (left) and results of the test (right) on the 8-core machine. See text for details.

was such that the load on the single disk was kept minimal (partition awareness). The results are shown on the right side of Figure 4. We see that in this extreme case being partition aware almost doubles the I/O rate. This shows how important it is to load-balance disk access to boost performance. For independent processes this is not always possible. However, in PROOF, this can be done provided that the partition information is available.

5. Virtual Machine tests

Virtual machines are becoming more and more popular in HEP because they represent an effective way to facilitate the deployment of the complex experiment software systems on a heterogeneous environment. The cost to pay is usually a performance penalty for I/O. The tests we present here aimed to quantify the performance penalties on PROOF-Lite due to virtualization in a real use-case.

The tests were run on the 24-core machine using KVM and XEN as hypervisors [8, 9], and CernVM [10] as virtual software appliance based on SLC5. The analysis was a PyROOT-based, I/O bound, ATLAS analysis run within PROOF-Lite with ROOT 5.26/00b and Python 2.5.4p2. The I/O devices tested were RAM, physical disk and network via Xrootd.

The results are shown in Figure 5. These have been obtained using vanilla kernels for both the host and the guest. The measurements have been done up to a maximum of 16 workers. Overall we observed a penalty between 10÷30 % for all three cases (RAM, physical, network), somewhat increasing with the number of processes. In-depth analysis [11] showed that most of the penalty comes from sub-optimal performance of some system calls in a virtualized environment; in particular, UNIX sockets, used for inter-process communications in PROOF-Lite, and some time-related system calls (e.g. gettimeofday() and wait4(), used mostly by the ROOT event loop), were found to be under-performing with default settings. For example, for KVM, the penalty can be essentially removed by using the TSC clock source on the guest and by optimizing the linux kernel setting the CONFIG_KVM_GUEST and CONFIG_HIGH_RES_TIMERS parameters.

7 Feature available starting with ROOT 5.27/04.
8 The analysis was kindly provided by the authors of reference [12].
9 At the time of the tests the SLC5 default kernel was 2.6.18-194.8.1.
6. Conclusions

In addition to be an effective way to exploit multi-cores using multi-processing, PROOF-Lite is a powerful tool to study ROOT I/O in concurrent access scenarios. We showed that the ultimate aggregate I/O approaches the device I/O bandwidth, provided that the file structure is optimized. Network I/O requires adequate servers behind to saturate the available bandwidth. Finally, we have presented the results of measurements done using a virtual machine which indicate that the virtualization I/O overhead can be mostly reduced with proper kernel optimizations and careful programming.

Acknowledgments

We would like to thank the ATLAS group of the University of Wisconsin, in particular Neng Xu and Wen Guan, for providing access to the 8-core machine used for some of the tests presented.

Related work

The ATLAS group at Brookhaven National Laboratories has presented at this conference a study which covers some of the aspects discussed in our study [12]. The results are generally in agreement. One of the problems with virtual machines spotted in their study was traced back to a bug in PROOF-Lite which was fixed in the version used in our tests.

References

[1] I. Antcheva et al., Comput.Phys.Commun.180:2499-2512,2009. See also http://root.cern.ch/.
[2] See http://root.cern.ch/drupal/content/proof. See also the previous reference.
[3] C. Eck, et al., LHC computing Grid : T.D.R., CERN-LHCC-2005-024, and references there in.
[4] See http://linux.web.cern.ch/linux/scientific5/.
[5] See http://root.cern.ch/drupal/content/root-version-v5-26-00-patch-release-notes.
[6] See P. Canal et al., ROOT I/O: The Fast and Furious, contribution #329 to this conference.
[7] See http://xrootd.org.
[8] See http://www.linux-kvm.org/ and references there in.
[9] P. Barham et al., Xen and the Art of Virtualization, Oper.Syst.Rev., vol.:37,no.:5,164-177, 2003
[10] P. Buncic et al., Eur.Phys.J. Plus 126(1) 18 10.1140/epjp/i2011-11013-1.
[11] Tools used: Strace, http://sourceforge.net/projects/strace/; SystemTAP, http://sourceware.org/systemtap/.
[12] S. Panitkin et al., Experience with PROOF-Lite in ATLAS data analysis, contribution #98 to this conference.