PROCEEDINGS OF SCIENCE

Profiling Post-Grid analysis

Akira Shibata[*†]
New York University, 4 Washington Place, NY 10012, USA
E-mail: akira.shibata@nyu.edu

In recent years, a great deal of effort has been put into developing a set of frameworks to realize analysis in the new paradigm of Grid computing. However, much more than a half of physicists' time is typically spent after the Grid processing of the data. Due to the private nature of this level of data processing, there has been little common framework or methodology.
While most physicists agree to use ROOT as the basis of their analysis, a number of approaches are available for the implementation of analyses using ROOT: conventional methods using CINT/ACLiC, fully fledged development using g++, alternative interface through python, and parallel processing methods such as PROOF are some of the choices physicists are confronted with. Furthermore, in the ATLAS collaboration an additional layer of persistency technology, POOL, adds to the complexity.
In this study, various modes of ROOT analysis are profiled for comparison with the main focus on the processing speed and input event size. Input data is or derived from the ATLAS Monte Carlo production.

XII Advanced Computing and Analysis Techniques in Physics Research
November 3-7 2008
Erice, Italy

[*]Speaker.

[†]The author would like to thank a number of people who helped the project one way or another, in alphabetical order : Sebastien Binet, Rene Brun, Jim Cochran, Attila Krasznahorkay, Wim Lavrijsen, Hong Ma, Tadashi Maeno, Stephanie Majeski, Sven Menke, Sergey Panitkin, Fons Rademakers, Scott Snyder, Shuwei Ye

PoS(ACAT08)034

1. Motivation

In an LHC experiment like ATLAS, models of analysis flow have been developed over the past few years in preparation to process data efficiently as we start taking data. Elegant solutions have been proposed to manipulate the data and analyze them on the complex system of computing Grid, and implementation of data management tool and analysis brokering system has been successful. On the other hand, most of the intellectual activities of a physics analysis happens outside such a system, that is, after the information required for the analysis has been defined and delivered to the computing facility physicists have local access to. We call this the "final analysis" stage. Typically, much more than a half of a physicists' time is spent in final analysis, in which data is analyzed repeatedly while measurements and observations are refined. Despite the importance of final analysis, due to its often private nature, there tend to be few common frameworks or methods available or widely known. Meanwhile, implementation of tools in final analysis must be done carefully examining competing technologies available to fulfill the task. This study aims at offering a broad overview of the technologies relevant to the final analysis stage and to compare their performance in a realistic analysis setting.

2. Tiered Analysis Model and data types

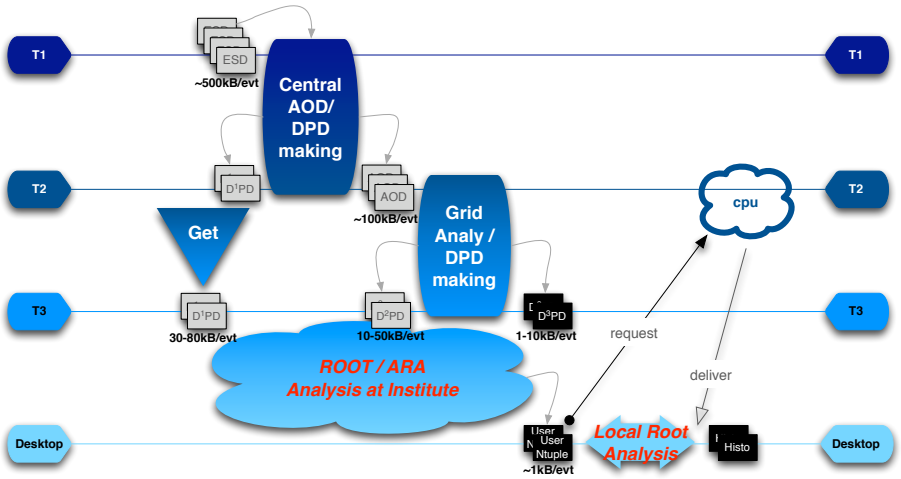

Figure 1: A schematic diagram showing the flow of data in the tiered Grid analysis model. See text for details.

The tiered design for Grid computing facility is already well established, but the exact flow of data in the tiers are becoming clear only recently. The current model that is exercised in the ATLAS experiment is shown in Figure 1. The first level data, Event Summary Data, which contains the full output of the event reconstruction, is produced on T0 and T1 facilities. Typical event size is 500 kB/event at this level. They are subsequently slimmed down to Analysis Object Data [1] and Derived Physics Data and replicated more widely to the T2 facilities. At this point the average event size is around 100 kB/event. Further processing may take place to produce even smaller

data types, which would typically reside in local computing clusters (T3). One may also choose to directly download data from a T2 facility without further processing. Once data is on a local computing facility, the data enters final analysis, producing very small output ntuples (so called "user ntuple") and histograms. These output files are interactively inspected on a desktop or a laptop machine. In addition, it is foreseen that final analysis may require significantly more CPU than locally available. If need arises, T2 facilities may lend CPUs to such ad hoc requests.

Data	Production	Storage + Analysis	Technology	Size (kB)
	Contents			
ESD	Central	T0/T1	POOL	500-1000
	Detailed information from reconstruction			
AOD	Central	T1/T2	POOL	100-200
	Analysis level objects and constituents			
D^1PD	Central	T1/T2/T3	POOL	30-100
	Subset of ESD contents			
D^2PD	User/Group	T2/T3/PC	POOL	5-30
	Smaller subset of ESD/AOD plus user data			
D^3PD	User/Group	T2/T3/PC	ROOT	5-10
	Similar to D^2PD but in native ROOT format			
User NT	User	T3/PC	ROOT	1
	Subset of DPD and Result of local analysis			

Table 1: Summary of data types relevant to physics analysis in the ATLAS experiment. "PC" is a desktop or a laptop machine with direct interactive access.

3. Input Data

Table 1 summarizes the relevant data types as described in the previous paragraph. Some amount of overlap exist in the availability of data between tiers since one type of data is often created in a higher tier and distributed to lower tiers. Derived Physics Data [2] is a relatively new idea and it has significant amount of flexibility leaving us to guess the size and use-case of these data types from current prototypes, many of which exist already. Two persistency technologies are widely used in ATLAS. One is ROOT [3] native persistency, which writes out portable but typically "flat" output[1] files containing only simple data types. Output files of this format are often referred to as "ntuple". This is used heavily in final analysis and are created on T2 facilities and below. The other technology, POOL [4], relies on the ROOT persistency but it adds additional capabilities. This includes metadata based look up of events and more flexibility in writing out complex objects. The POOL technology is used in data types immediately after reconstruction and their close derivatives. Due to the extra dependencies, it requires ATLAS software installation for reading them though recent development in a tool called Athena Root Access (ARA) has enabled

[1] Typically the most complex object stored in ntuples are `vector<vector<double> >`.

near-transparent call back layer[2] for reading these files from ROOT without invoking the ATLAS software framework, Athena [5]. The use of POOL technology is somewhat historical in that the ROOT native persistency was not deemed sufficient when the decision was made to employ it. The problems (e.g. in writing out STL objects) have mostly been resolved by now and the necessity of POOL technology may be challenged in the future.

The data used in this study were produced in one of the latest Monte Carlo based data challenge called Full Dress Rehearsal 2. As DPD size is not known completely and it is interesting to study IO performance dependency on input data contents, several AOD and DPD prototypes were produced on the Grid as shown in Table 2.

Full contents	POOL	Ntuple
Full analysis contents (AOD)	144.22 kB	N/A
DPD contents	31.42 kB	4.87 kB
Small DPD contents	18.74 kB	0.71 kB
Very Small DPD contents	1.06 kB	0.37 kB

Table 2: Summary of data types used in this study and their event size.

In general, the size of ntuple is smaller than the corresponding POOL files. This is partly because it is more efficient to store simple variables than complex objects but also due to the fact that ntuple contents is generally limited because complete information from objects was not copied to ntuples. The full analysis contents consist of analysis level objects such as jets, electrons, muons (multiple versions of them) as well as trigger information and track information. Truth information is not included. DPD contents is subset of AOD but only contain one version for one type of object. Small DPD content consist of track and electron information only while very small DPD only keeps electron information, which is the minimum required for the benchmark analysis used.

4. Software technology

Table 3 summarizes the modes of analysis widely available to and often used by current ATLAS physicists. Athena is the common framework used within the collaboration and it is a robust system capable of handling a very wide range of tasks including physics analysis. It is globally used for tasks ranging from detector simulation to event reconstruction and there are Grid software frameworks that support distributed operation of the program. While it has full range of tools available that are useful at the final analysis stage, it is not usually considered appropriate for physics analysis. One of the reasons is the development cycle: Athena development tend to involve compilation of packages with complex dependency structure, which tend to take time and expertise.

Analysis using ROOT with no additional framework can provide fast turnaround and fast development cycle. `TTree::Draw` method is a highly interactive tool that requires little code writing. It can be accessed from both CINT and Python interface, though its functionality is limited to

[2]The relevant POOL converter, which is required to convert persistent representation of data into transient representation, is executed automatically when data retrieval is requested.

Mode	Draw	CINT	ACLiC	PyRoot	g++	Athena
Ntuple Compatibility	○○	○○	○○	○○	○○	○
POOL Compatibility	○	○	×	○○	○○	○○○
Compiled/Interpreted	Int.	Int.	Compiled	Int.	Compiled	Both
Language	C++/Python	(C++)	C++	Python	C++	C++/Python
Interactive	○○	○○	×	○○	×	○
Event Loop Mgr	-	TSelector		-	SFrame	Athena

Table 3: Summary of analysis modes examined in this study.

simple operations as it lacks an event loop. Nonetheless, this is often the first method of inspecting data. CINT and ACLiC are provided for more complex analyses. CINT is an interpreted language based on C++ whose interactive capability is realized through dictionaries generated automatically. Due to the inefficiency of an interpreted language, it is not suitable for a large scale analysis. For this compiled mode of operation is provided by ACLiC which can compile CINT scripts on the fly though stricter C++ language requirements need to be satisfied. A simple framework called TSelector is provided to produce a skeleton analysis based on a given input data. This greatly reduces the amount of code writing required by the user and it also provides its own event loop facility that is compatible with PROOF, making it easy to extend the execution to many CPUs. PyRoot is another interactive interface layer for ROOT, which enables using ROOT library from a Python prompt by using Python bindings to the existing interface. Python is a scripting language by design and it often provides simpler syntax than C++ or CINT. For example, type casting is handled automatically in PyRoot. The standard C++ compiler, g++, is highly optimized for its performance. There are a range of development tools available such as debuggers and profilers which helps development. The downside of g++ development is that it is difficult to set up. External packages exist that provide a convenient setup such as Makefile to compile the entire package including object files, executables and dictionaries. One of such packages, SFrame [6], provides an environment much like TSelector but with richer interface to IO using helper functions and configuration using XML. Use of such packages can help development though it could come at a cost of degraded performance depending on the complexity of the framework.

Accessibility to POOL format data exist in most modes listed above except ACLiC, which is currently incapable of compiling code using ARA, though this should not be a fundamental limitation. Support for CINT is somewhat limited while compatibility is higher with PyRoot and g++ compiled code.

5. Methodology

The performance of analysis modes was studied by running a benchmark analysis on the input data types listed in Table 2. The analysis reconstructs Z boson from its dielectron decay and it consists of the following steps:

1. Access electron container (POOL) or electron kinematics branches (Ntuple)

2. Select electrons using particle ID quality ("isEM") and p_T
3. Fill histograms with electron kinematics (p_T and multiplicity)
4. Combine electrons to reconstruct Z boson if two and only two electrons are found
5. Write histograms into a file and finalize

To increase computation, the process above was repeated 10 times in a loop. The analysis is not as complex as a real life analysis though it is not completely trivial. Therefore the performance measurements taken with this benchmark test can be thought of as the upper limit achievable with each mode.

Event execution rate was extracted from a straight line fit on the time measurements obtained with increasing number of input events. The number of events were selected so that timing for all modes are approximately the same[3]. The interception on the y axis indicates the initialization time, which is usually in the range between a fraction of a second to tens of seconds. However, this was not studied here in detail as it becomes negligible in the limit of many input events. Wall time was taken by using the `TStopwatch` tool provided by ROOT. As measurements were taken on a public machine, the timing was affected by other processes running on the machine. To avoid variation of measurements due to this effect, each measurement was taken at least 5 times and averaged. The standard deviation of the measurements is quoted as error in the following results. As an example, Figure 3 shows the measurement summary for AOD input type.

To obtain stable measurements of the timing of analysis execution, a number of issues were considered. Misleading results were obtained when the same file was processed twice consecutively. Data on disk is first loaded onto RAM and this remains on RAM as cache until something overwrites it and second access would not initiate another disk read. No method was found to flush disk cache on RAM except to restart the machine to a "cold start". This was impractical considering the number of measurements to be made and time it takes to reboot the system. Therefore files were processed in sequence so that the same file would not be read again until all other files of the same type have been processed. Measurements of ntuple input and POOL input were alternated to further avoid the effect of caching.

The machine used for the timing was a public machine dedicated to ATLAS analysis, located at BNL. It has 3.34 GB memory according to the system specification and has 2 cores of Xeon CPU running at 2 GHz (Model E5335, cache 4096 KB). The kSI2k[4] rating for the machine was 0.9466 and all measurements were normalized by this factor. The input files were stored on a hard drive connected to the machine via NFS.

6. Optimization

While realistic analysis may not be fully optimized, the implementations of the benchmark analysis used in this study were optimized with help from ROOT and Athena experts. This is to

[3] In other words, larger number of events were tried with analyses that are faster. This reduces fluctuations in the results.

[4] SpecINT2000 seconds is a widely used metric of computer performance endorsed by Standard Performance Evaluation Corporation. The faster the system, the larger its conversion factor, or the rating, is. kSI2k stands for kilo-SpecINT2000 and is commonly used because most modern computers have $\sim$1000 SI2k.

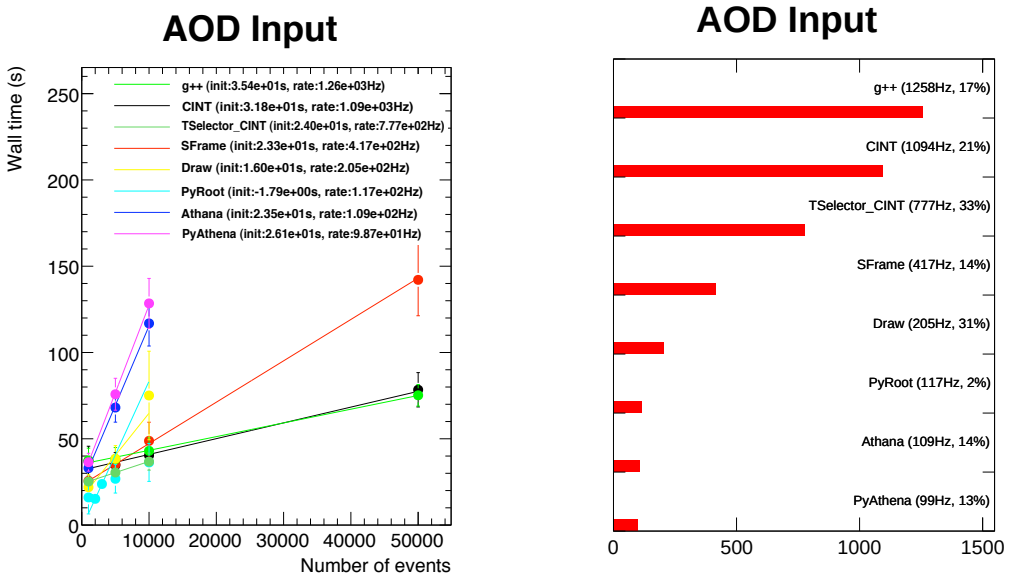

Figure 2: Measurements of AOD analysis speed illustrating the measurement method. The left plot shows the extraction of event execution rate, the right shows the result extracted from the fit.

obtain fair result for every mode with no accidental mistake in any one of them. Several changes were made to the initial code, which sometimes improved the performance significantly. For example, since CINT execution is performed line by line with little runtime optimization, repeated creation and destruction of a variable in an inner loop can be rather inefficient. Another example is the way ROOT trees are read. Trees must be read by branch not as a whole for the best performance. This operation: `chain->GetEntry(entry)` will initiate the reading in of the whole tree, which is wasteful if a small subset of the whole tree is required. In the case of PyRoot, reading in by branch is not supported by default, and a workaround, which is available within ATLAS CVS repository was applied to force read-by-branch.

7. Results

Table 4 summarizes the result for each mode for each input type. One can first observe the general tendency of performance increase as the input file size becomes smaller. While the performance difference between DPD contents input and Small DPD input is apparent, the performance did not improve with even smaller Very Small DPD input. This is related to the IO bottleneck discussed later in this section.

Even within the tests using the same input, the rate varies widely. Compiled modes are clearly advantageous compared to the interpreted modes. ACLiC and g++ analyses result in the highest event rate of tens of thousands of events processed per second[5]. Addition of `TSelector` does not degrade the performance at all thanks to the simplicity of the `TSelector` framework, while

[5]The small tendency that g++ event rate is lower than ACLiC is not currently understood.

Input Contents	Format	Draw	CINT	CINT TSelector	ACLiC	ACLiC TSelector
Full contents	Ntuple	-	-	-	-	-
	POOL	205 Hz (±31%)	1094 Hz (±21%)	777 Hz (±33%)	-	-
DPD contents	Ntuple	6433 Hz (±14%)	8805 Hz (±6%)	8286 Hz (±7%)	32651 Hz (±18%)	30438 Hz (±24%)
	POOL	208 Hz (±15%)	1999 Hz (±24%)	1478 Hz (±18%)	-	-
Small DPD	Ntuple	7907 Hz (±9%)	11913 Hz (±12%)	11041 Hz (±7%)	60873 Hz (±23%)	55234 Hz (±35%)
	POOL	301 Hz (±7%)	3447 Hz (±11%)	2397 Hz (±20%)	-	-

Input Contents	Format	PyRoot	g++	g++ SFrame	Athena	PyAthena
Full contents	Ntuple	-	-	-	-	-
	POOL	117 Hz (±2%)	1258 Hz (±17%)	417 Hz (±14%)	109 Hz (±14%)	99 Hz (±13%)
DPD contents	Ntuple	839 Hz (±16%)	26617 Hz (±14%)	19750 Hz (±18%)	906 Hz (±25%)	347 Hz (±31%)
	POOL	674 Hz (±21%)	2503 Hz (±22%)	920 Hz (±14%)	346 Hz (±22%)	245 Hz (±17%)
Small DPD	Ntuple	1040 Hz (±14%)	49953 Hz (±17%)	35531 Hz (±16%)	963 Hz (±28%)	362 Hz (±30%)
	POOL	958 Hz (±10%)	4345 Hz (±24%)	2399 Hz (±15%)	780 Hz (±29%)	416 Hz (±26%)

Table 4: Summary of the event processing rate obtained by the benchmark measurements. Normalized by kSI2k conversion factor. To obtain corresponding speed for a specific computer, multiply by kSI2k conversion factor. The results for Very Small DPD was equivalent to those of Small DPD and they are omitted.

SFrame brings down the speed by a noticeable amount. The complexity of Athena framework degrades the event processing rate very significantly. There is clearly a trade off between simple but specific analysis and general but complex analysis.

Interpreted modes are inherently slower due to language limitations. CINT analysis is slower by a factor of 5 or so compared to the same analysis compiled by ACLiC. This was achieved by optimizing the code and avoiding frequent memory allocation/deallocation in an inner loop. Without such caution, the performance can be degraded to a level similar to PyRoot, which turned out to be a much more inefficient option even after optimization. Similar to ACLiC, the addition of TSelector has almost no effect on the performance of CINT. On the other hand the Python version of Athena, PyAthena, is clearly the slowest option suffering from both language and framework

disadvantages.

The Draw analysis is not entirely a fair comparison with the others. As `TTree::Draw` lacks customizable event loop, the analysis only consists of plotting two kinematic variables of electron without Z boson reconstruction. One would expect a similar result to CINT in this case, but one clear disadvantage of Draw is that each histogram making requires one looping over the.

As a general trend, analysis with ntuple input is faster than its POOL input counterpart. This is partly due to the size of EDM object which are larger than selecting specific variables, and also due to the time taken to convert persistent representation into its transient counter part using POOL converter. The difference between the two is larger for faster modes: a difference by factor of 10 or more is seen for compiled C++ while PyRoot and PyAthena differ only by factor of 1.5 or so. In slower modes, the difference due to input type is washed away by their own overhead. Draw is exhibiting exceptional behavior in this respect and its POOL/ntuple ratio is rather high. This is presumably because Draw loads the entire tree instead of reading in the branches required for the analysis. Its speed therefore is more sensitive to the event size difference between the POOL and ntuple data types.

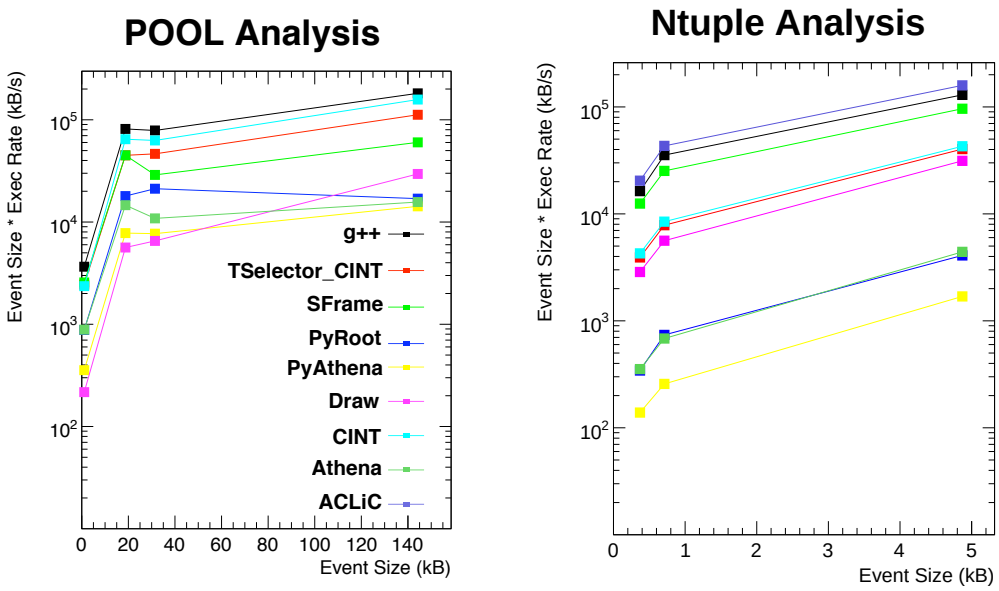

Figure 3: Dependency of analysis speed on input event size. The left plot shows POOL input analysis result and the right shows ntuple input.

Figure 3 shows the mode performance, in terms of data size read per second, against the event size of input data type. It can be seen that the increase in performance for analysis with POOL input data types is limited severely by their event size. This is due to the limitation of disk read speed: there is an effect of IO bottleneck below 100 MB/s, which implies $\sim$30 kB event size. Note that this is for the total event size, the data volume actually required for all of these analyses are the same. In other words, execution performance is constrained by the information that is not required for the analysis. This effect is due to disk caching by the operating system. As soon as a program

request reading of a file, the operating system loads the whole file onto the memory since it has no prior knowledge of what subset of the file is required during execution. This behavior also affects ntuple analyses but less because we only tested event sizes up to $\sim$5 kB/event. The limitation clearly depends on the disk system deployed on the computer, one can expect that the bottleneck will not appear up to higher event size with higher throughput IO devices.

8. Summary and Conclusion

In this study, several modes of ROOT analysis were compared. We obtained measurements of performance for each mode using a simple but non-trivial benchmark analysis implemented in all modes. Realistically, the results can be interpreted as the upper limit of the performance. The results show variation in performance, some of which were expected while others more subtle. An important observation was made that the performance of analysis depends on the event size of the input data, even though the input data consist largely of information that is not needed for the particular analysis. This affected analysis with POOL input data more significantly as the event size of large data types could exceed the IO limitation of the hardware.

For POOL input data, performance of analysis can depend largely on the type of object since object size can be significantly larger than a single variable and time taken to convert persistent representation to transient representation depends heavily on the complexity of the converter. While such details are beyond the scope of this study, the performance of POOL converters are closely monitored on a nightly basis[6] to detect possible degradation in IO performance.

While interpreted languages have obvious performance disadvantages, their benefit can be attributed to types of operations other than event level processing of large number of events. They are indispensable tool for interactive analysis, and a language like Python can seamlessly combine external modules to the analysis environment.

A number of refinements can be considered to improve this study. The scaling pattern observed for large event size may appear differently if the input files were placed on a more efficient file system or a local hard drive. An analysis with higher complexity may reveal more subtle dependencies of performance. In particular, an analysis that writes out large output file may behave significantly differently from the current benchmark analysis.

In this paper, the performance of analysis modes were compared on a single computer only. Lately, however, the method for utilizing multiple cores in an analysis is in rapid development as new CPUs offer a number of cores per chip and new computers house multiple chips. Analysis modes that can adapt for multi-core processing, such as TSelector, can easily provide large performance increase in such systems. Such criterion must be considered in addition to the single machine performance obtained in this study when one makes a choice of method to write a new analysis. Performance profiling of many-core driven analysis must be in the scope of future studies.

9. Appendix - DPD Making Performance

Since DPD making usually takes place on the Grid, its performance was not the focus of this study. Nonetheless, DPD making performance can be measured with the same tools that were

[6] see http://athena-infoioperformance.web.cern.ch/athena-infoIOperformance/

Output Content	Output format	ESD Input (649.18 kb/evt)	AOD Input (144.22 kB/evt)	D[1]PD Input (18.74 kB/evt)
DPD contents	Ntuple (4.87 kB/evt)	14 Hz (10%)	15 Hz (8%)	-
	POOL (31.42 kB/evt)	-	3 Hz (15%)	3 Hz (7%)
Small DPD	Ntuple (0.71 kB/evt)	32 Hz (14%)	45 Hz (15%)	66 Hz (4%)
	POOL (18.74 kB/evt)	-	8 Hz (9%)	11 Hz (3%)
Very Small DPD	Ntuple (0.37 kB/evt)	86 Hz (24%)	88 Hz (12%)	208 Hz (10%)
	POOL (1.06 kB/evt)	-	11 Hz (11%)	11 Hz (3%)

Table 5: Summary of the event processing rate for DPD making. Normalized by kSI2k conversion factor. To obtain corresponding speed for a specific computer, multiply by kSI2k conversion factor.

used in this analysis. Performance of DPD making is an important parameter for estimating the throughput of jobs that would be running on Tier 2 computers and therefore affects the hardware requirements of such clusters whose main purpose is to produce Derived Physics Data. For DPD making, Athena is the primary tool since full access to POOL EDM is often required in such jobs. Table 5 shows the event processing rate of DPD making processes. For each output type, different input data types were examined. One can see that the processing rate for ntuple making is a function of both input and output event size. It implies that it is the most efficient to produce smallest ntuples from smallest possible input. Similar tendency can be observed for POOL format DPD making though the sensitivity is much smaller. It generally takes much longer time to produce POOL DPDs though this is subject to further optimization in the future, as POOL DPD making software is relatively new.

References

[1] P. Califiura et al. AOD format task force report. *ATLAS Notes*, SOFT-PUB(011), 2006. Available from World Wide Web: `https://twiki.cern.ch/twiki/bin/view/Atlas/AODFormatTaskForce`.

[2] D. Costanzo, I. Hinchliffe, and S. Menke. Analysis model report. *ATLAS Notes*, com-gen(001), 2008.

[3] R. Burn and F. Rademakers. Root - an object oriented data analysis framework,. In *AIHENP'96 Workshop*, volume A. 389, pages 81–86, Lausanne, Sept 1997. Nucl. Inst. Meth. in Phys. Res.

[4] I. Papadopoulos et al. POOL - the LCG persistency framework [online]. Available from World Wide Web: `http://pool.cern.ch/talksandpubl.html`.

[5] ATLAS Collaboration. *Computing Technical Design Report*. Number CERN-LHCC-2005-022. CERN, 2005.

[6] A. Krasznahorkay et al. Sframe [online]. 2008. Available from World Wide Web: `https://twiki.cern.ch/twiki/bin/view/Main/SFramePage`.