Journal of Physics: Conference Series **396** (2012) 042010 doi:10.1088/1742-6596/396/4/042010

Analysing I/O bottlenecks in LHC data analysis on grid storage resources

W. Bhimji[1], P. Clark[1], M. Doidge[2], M. P. Hellmich[3], S. Skipsey[4] and I. Vukotic[5]

[1] University of Edinburgh, School of Physics & Astronomy, James Clerk Maxwell Building, The Kings Buildings, Mayfield Road, Edinburgh EH9 3JZ, UK
[2] Lancaster University, Physics Department, Lancaster LA1 4YB, UK
[3] CERN, Geneva, Switzerland
[4] University of Glasgow, Department of Physics and Astronomy, Glasgow G12 8QQ, UK
[5] University of Chicago, 5620 S Ellis Ave, Chicago IL 60637, USA

E-mail: wbhimji@staffmail.ed.ac.uk

Abstract. We describe recent I/O testing frameworks that we have developed and applied within the UK *GridPP* Collaboration, the *ATLAS* experiment and the *DPM* team, for a variety of distinct purposes. These include benchmarking vendor supplied storage products, discovering scaling limits of *SRM* solutions, tuning of storage systems for experiment data analysis, evaluating file access protocols, and exploring I/O read patterns of experiment software and their underlying event data models. With multiple grid sites now dealing with petabytes of data, such studies are becoming essential. We describe how the tests build, and improve, on previous work and contrast how the use-cases differ. We also detail the results obtained and the implications for storage hardware, middleware and experiment software.

1. Introduction

The Large Hadron Collider experiments are now storing and accessing petabytes of data at multiple sites with a variety of storage systems. In addition, despite a series of improvements [1] [2], there remains substantial activity on some LHC experiments that is I/O limited. There are also differences between experiments' workflows and I/O patterns and little projection or communication by experiments on expected file access patterns or I/O operations per second (IOPS). This motivates greater understanding of the workloads presented by the experiments and the response of hardware and storage systems to those workloads. Recognising this, in their recent report, the WLCG Storage *Technical Evolution Group* (TEG) recommended that I/O benchmarks be developed; that experiments understand and communicate their I/O requirements and that both applications and storage systems provide mechanisms for understanding I/O [3].

In this context we explore I/O testing. Listed below are some of the reasons for such testing from the perspective of sites, experiments or storage developers. In the rest of this paper we identify certain examples of these using studies in which the authors have been involved. As well as outlining some of the results obtained, we raise some of the features of the method employed and use that to compare what approaches need to be taken more generally for these kind of tests.

- **Sites:**
  - Evaluating vendor supplied storage and informing purchasing decisions.
  - Tuning of hardware or middleware.
- **Storage Middleware Developers:**
  - Tuning their system for use in WLCG environment.
  - Basic functionality testing for new releases.
  - Scale testing of low-level functions.
  - Choice of protocols to use.
- **Experiments:**
  - Application testing.
  - Evaluating data models or changes in file structure.
  - Checking service-level offered by sites.

In section 2 we describe the testing of vendor supplied storage for use at a UK Tier 2 site. In section 3 we detail an analysis of the low-level performance of the internal functions of the DPM middleware. In section 4 we describe a testing framework used for evaluating I/O of the ROOT application [4] and in section 5 we describe a framework for the regular testing of releases of the DPM middleware. Finally in section 6 we contrast these use cases noting where common approaches can be taken.

2. Testing vendor supplied storage

Sites are often presented with a variety of options for storage hardware purchases. Some vendors are keen to engage but, unlike for compute resources where the *HEPSPEC* benchmark exists that can be run without understanding of the experiment software, there is no portable benchmark for HEP I/O. Here we describe stress testing of the DELL "HPC scalable storage building block reference architecture"[5]. The full report is provided in [5], including details for reuse of the tests by other sites.

In this case the importance was to find a simple, portable test, that could provide a realistic level of stress on the supplied servers but that did not require experiment software to be installed or particular expertise. The mandate was specifically to determine the servers suitability as DPM storage pools (DPM is the most popular storage element at UK Tier 2 sites) and therefore the tests made use of the DPM LAN protocol *RFIO* and the WAN protocol *GridFTP* [5].

The hardware tested consisted of a pair of Dell *PowerEdge* R710 servers connected to *PowerVault* MD3200 storage enclosure extended by three *PowerVault* MD1200 storage arrays, each of which contained 12 x 2 TB *nearline* Serial-Attached-SCSI (SAS) disks. At 120 TB per server this presents similar, slightly denser, storage servers than in use by many UK Tier 2 sites. The servers were partitioned into file-systems of between 9 and 41TB employing a range of RAID configurations and both *xfs* and *ext4* file-systems. This was connected into the production environment at the Lancaster Tier 2, with three bonded single-gigabit Ethernet links, and accessed by their production cluster which provided 512 cores for use in testing.

Both wide-area and local tests were performed. For the local area tests, scripts were developed that simulated realistic (worst case) production loads and these were submitted to the site batch system, accessing the machine which was setup as a test DPM disk server. The first performed a copy-over-RFIO (*RFCP*) with up to 250 jobs per server, while the second accessed data directly over RFIO at up to 100 jobs per filesystem using the ROOT application [4] with a ≈2GB data file taken from the ATLAS experiment. In both cases the machines became overloaded unless tunings were applied to default filesystem or DPM read aheads. Figure 1 demonstrates for the RFCP test that load on the server increased until the block device read-ahead was increased to 8MB at which point the available network could be utilised. Figure 2 demonstrates the case

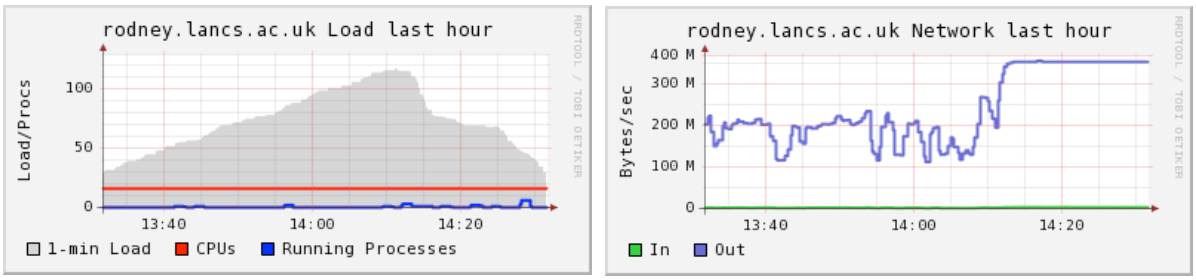

Figure 1. System load and network utilisation for 250 RFCP jobs showing that once the block device read-ahead value is increased the load is reduced and available network is fully utilized.

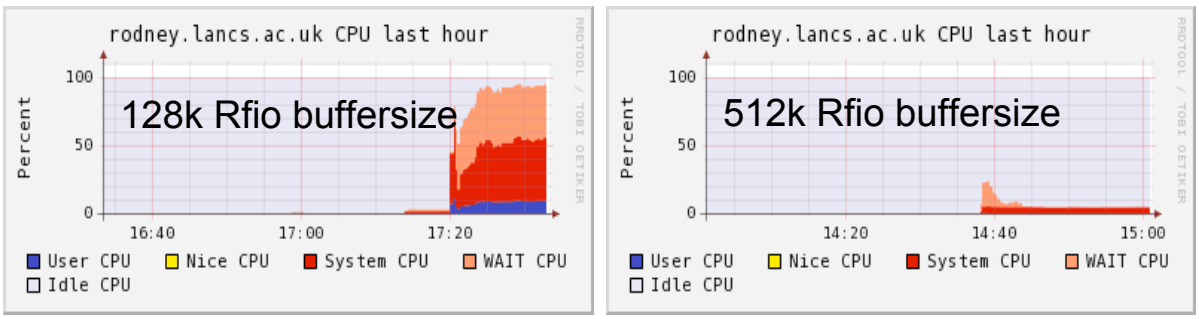

Figure 2. CPU usage for a default RFIO buffer-size of 128k or an increased value of 512k during the direct-RFIO ROOT test.

where 100 direct RFIO jobs resulted in excessive server CPU usage, including WAIT CPU, until the RFIO read-ahead value is increased. Many of the tunings applied in the whitepaper are being applied in production environments by UK sites, and so it was possible with simulated tests to cause similar load as seen in production, which is valuable for testing new hardware.

3. Stress testing of the DPM storage element

The aim of this work was to find limits of the DPM storage element's internal ways of dealing with requests when stressed in a realistic fashion. For full details see [6]. Three tests were developed and added to the DPM package *perfsuite* [7]. These included a file copy and direct file access in ROOT over RFIO, which are similar to those discussed in the last section. However, in addition, a *pretend-RFCP* test was implemented where a file is requested on DPM either with the *dpm_get* or *dpm_put* command, polled with *dpm_getstatus*, opened with *rfio_open*, but closed immediately and a successful file transfer signalled with *dpm_putdone*. This allows testing of the intrinsic limits of the DPM calls rather than hitting limits on server load or network bandwidth. In addition, infrastructure was developed to parse the log files in DPM so that detailed timing info could be obtained.

As shown in figure 3, times were recorded of all the DPM functions involved in the process. This includes the *put* request, which polls for the physical file on a disk server; the *proc* request, which checks if the file exists and returns the transfer URL; and the *putdone* request. It also includes the number and time for *stat* requests, where the client checks if the put operation has completed. As there is a delay between consecutive *stat* calls, multiple such calls reflect a long processing time on the server. The *put* function on DPM contacts the namespace (*DPNS*) daemon for file access rights and runs with multiple *fast* threads while the *proc* function on DPM runs with multiple *slow* threads. There is a delay between the functions in switching between

the threads. Figure 3 shows that an improvement was found by running with 70 instead of 20 slow threads in which case the client only required one *stat* call instead of two and there was a decreased total time for the process in repeated tests.

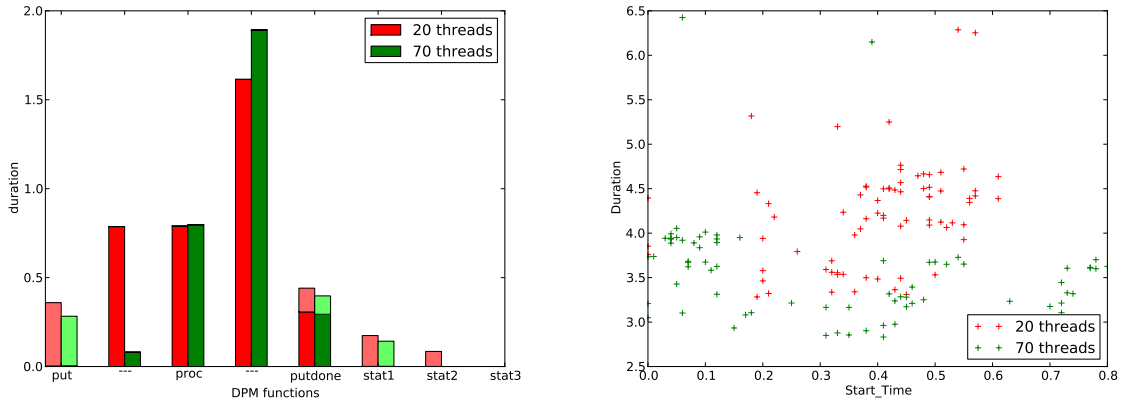

Figure 3. For 20 (left-bar, red) or 70 (right bar, green) slow threads: the average time taken (seconds) for DPM operations (left) and the total time (seconds) for repeated tests (right).

This testing also revealed benefits from increasing the socket queue, which defines how many requests can start to be processed by DPM at a time. Furthermore, it uncovered limitations in file opens within a single directory. This later limitation is due to the fact that when a file entry is created, the *link number* of the parent directory must be increased. To change the parent directory the database will be locked. Therefore file creation in one directory cannot be truly parallel, because all threads have to wait on this database entry for the parent directory. Further specific details on the testing and results are available in [6] and such testing will continue to be used to reveal bottlenecks or limitations within the storage system.

4. ROOT I/O Testing Framework

A testing framework was developed to test the impact on I/O performance of changes within the ROOT application on all different storage system types. This is illustrated in figure 6. It uses the HammerCloud [8] testing system to send continuous jobs to large Tier 2 sites. HammerCloud was modified to take tests directly from an SVN repository allowing a quick development-test cycle for changes. The tests all run on identical data files for comparison. New files (written with new versions of ROOT, for example) can be distributed to all sites within a day. The tests run are highly instrumented and collect a variety of performance information, including ROOT internals and the worker node environment, which are stored in an Oracle database for later analysis. This system enables fast turnaround so that for a new test designed in the morning results can be obtained on production instances of a whole variety of storage systems by the end of the day. It therefore complements stress testing or tests performed in controlled environments. A web interface is provided for display of results [9] allowing for easy monitoring by experiments, developers and sites; together with the ability to store the results as a ROOT *TTree* for more detailed analysis.

The tests employed include the reading of simple ROOT *TTree* structures by the ROOT application itself, which is common to many HEP experiments. It also includes tests specific to the ATLAS experiment. Tests are done where the whole file is read as well as some with sparse reading of the files (both events or branches). These are carried out with different versions of ROOT, as well as patches or development versions. An example is shown in figure 5 of the CPU

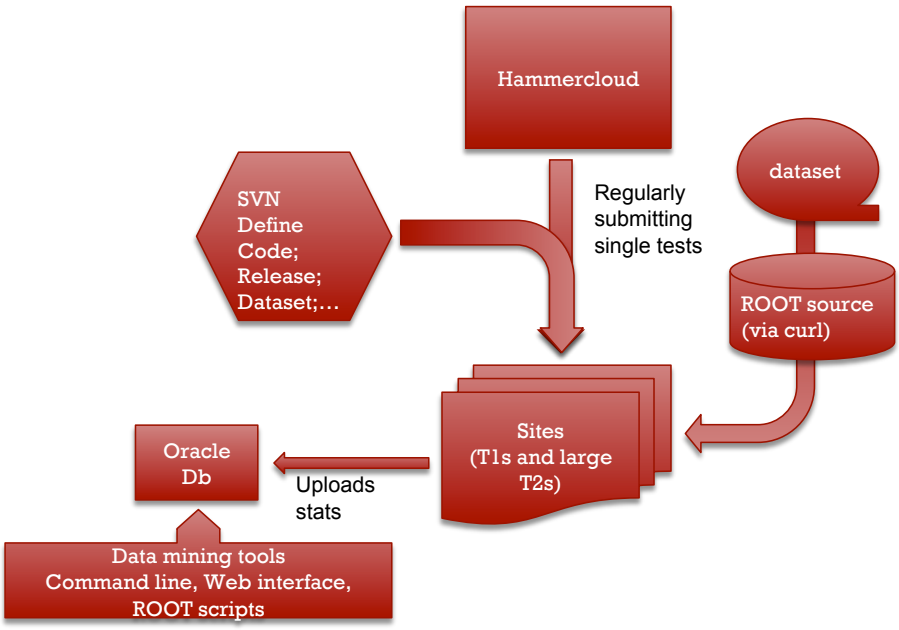

Figure 4. Outline of the ROOT I/O testing framework.

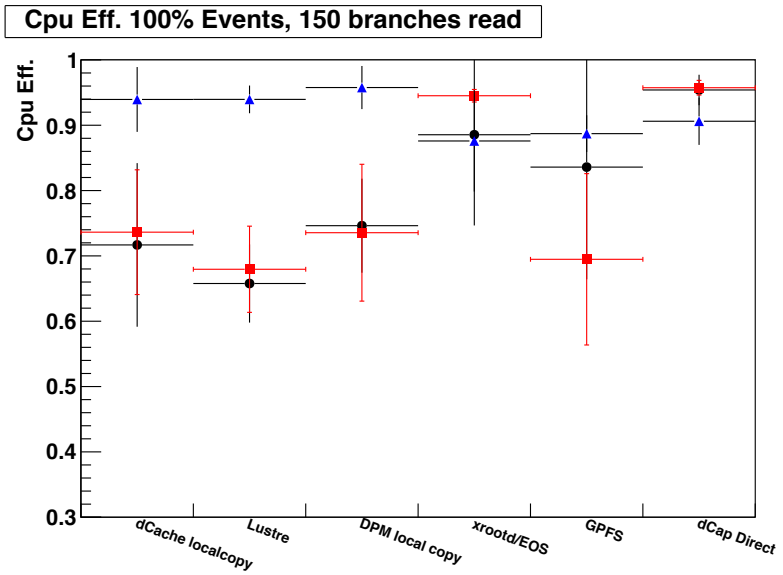

Figure 5. CPU efficiencies for jobs reading 100% of events using ROOT TTreeCache (TTC) and either reading all branches with a TTC of 30MB (blue triangles) or reading around 2% of branches with a TTC of 30MB (black circles) or 300MB (red squares). Errors from RMS spread of results.

efficiency (CPU time / wall-time) found at a range of sites. The input file used is 750MB in size and contains a main event *TTree* with around 3000 branches and 12000 events written in ROOT 5.32. The sites employ a variety of storage systems as labeled on the x-axis. In this example, the effect of reading a limited set (around 2%) of branches is shown. It can be observed that sparse reading of branches leads to lower CPU efficiencies on certain sites. Those sites that maintain high efficiencies use protocols capable of vector reads (*dcap* or *xrootd*). It can also be observed

that using a larger than default value of ROOT's memory buffer "TTreeCache" makes little difference to this drop in efficiency though its effect varies on different sites or systems. These tests are now running continuously and this will be used within the cross-experiment ROOT I/O working group towards further developing ROOT features such as "Basket Optimisation" and "Asynchronous Prefetching" [10]. In addition, the test will be further broadened with a view to both including more examples from other HEP experiments as well as working towards a generic benchmark. Finally the framework is being used for site tuning where it will be important to compare to server side monitoring, available for xrootd [11] and http [12] for example.

5. Middleware Testing Framework

While there is an extensive testing process before storage middleware is released to the community, there are often issues that do not appear until it is tested in a real production environment. In addition, there may be new features, such as the implementation of new protocols, that require performance testing that can be most effectively done in production. There are also site benefits from testing middleware versions before they are released. For these reasons a framework for testing pre-production versions of the DPM storage element has been developed. It builds on the ROOT I/O framework described in the last section, but, in addition, the sites install a test DPM headnode and disk server which auto-update (using *yum*) from the testing "epel" repository used by the DPM developers. The framework is illustrated in figure 6. Currently two sites are deployed: Glasgow with an SL5 release and Edinburgh with an SL6 instance. As above, the same tests are run regularly on the sites' production clusters, but reading files now from the test DPM and testing both current functionality and new features such as new protocols. The results of the tests are uploaded to a custom database to allow the storage of middleware specific quantities and the results are displayed on a custom webpage [13]. A Nagios instance is also run to monitor the test machines.

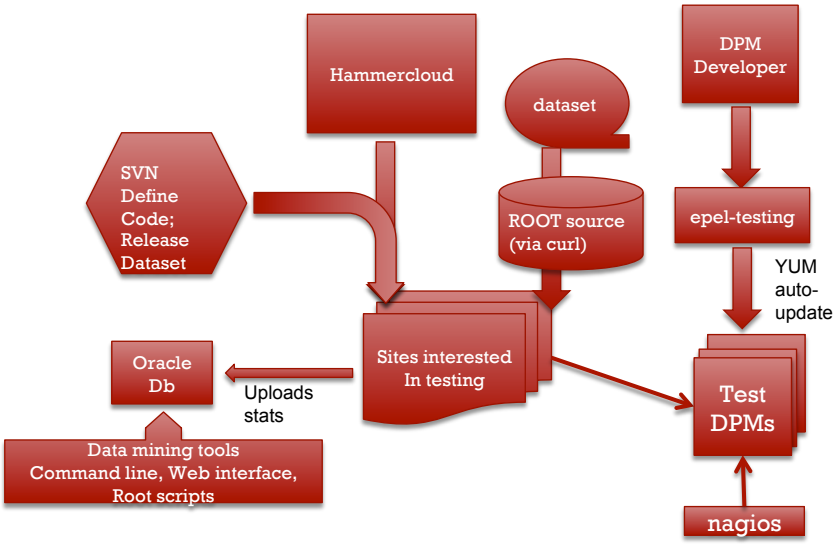

Figure 6. Outline of the middleware testing framework.

Figure 7 shows a snapshot of the web page, illustrating the regular monitoring of reading from the Glasgow test server using RFIO. Such monitoring gives confidence that releases will work in the environment in which they would be used. It would also potentially show if any performance improvements were made in the RFIO protocol. However the development of DPM is not currently focused on RFIO, but on new protocols. Therefore figure 8 illustrates the reading

of files via the new *WebDav*/https interface. This is under heavy development and the test jobs need to patch a version of the ROOT source and recompile in order to be able to read the file. Figure 8 shows the wall-time for reading only 100 out of the total 12000 events in the file and the wall-times are significantly longer than for RFIO. This is most likely because of the SSL encryption used when reading in this way (because of the need for X509 authentication). In order for this to be a better performing option the possibility of redirection to plain http is being investigated with the DPM developers and this framework is now in place to be able to test the resulting implementations.

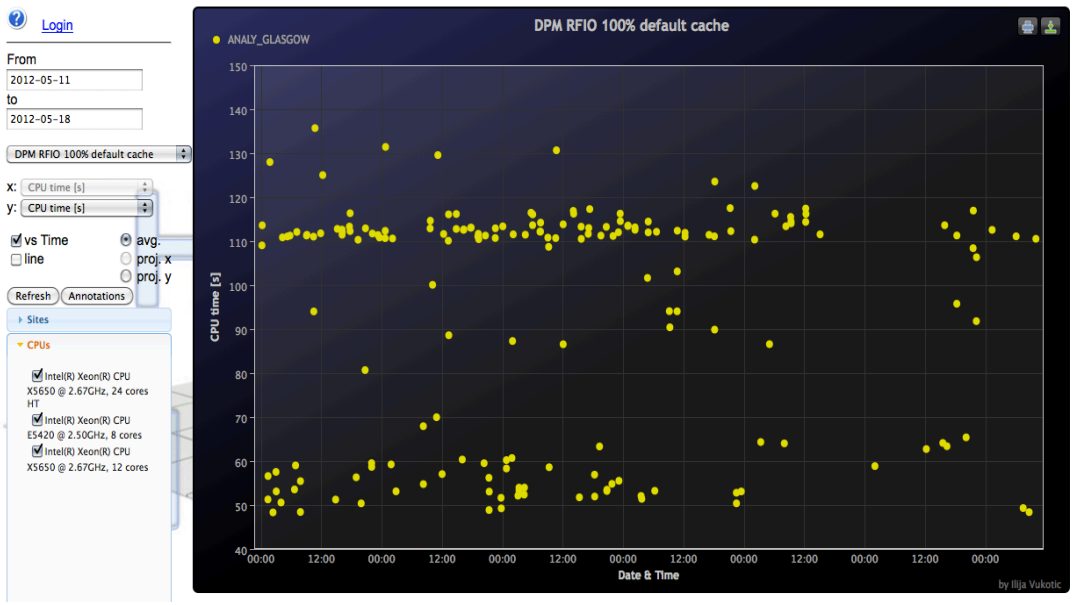

Figure 7. Snapshot of the web visualisation tool for the middleware testing framework showing the tracking of CPU time for reading a file via RFIO on the test DPM server.

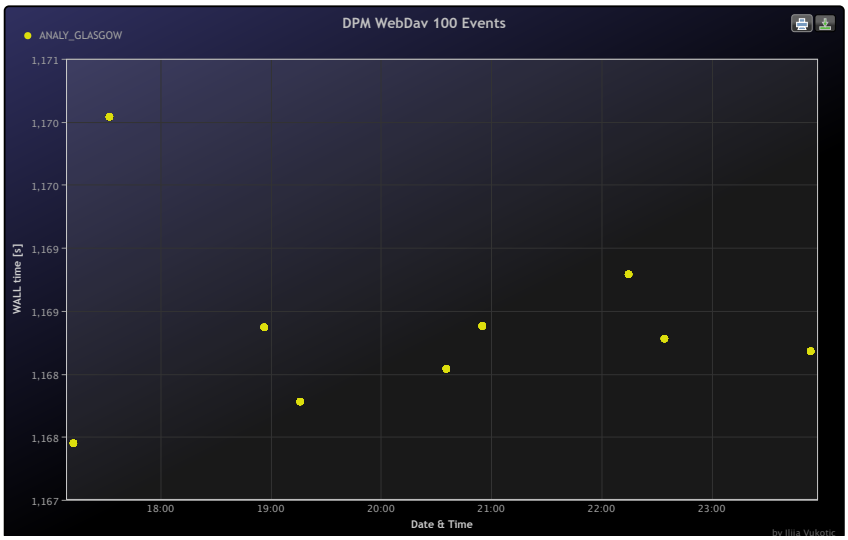

Figure 8. Wall time for reading 100 events within ROOT from DPM's WebDav / https interface.

6. Comparison of different use cases

In table 9 we match the use cases discussed above against possible features of the testing framework. Commonalities have been seen and similar tests used in the examples described here but there are also important differences. For example, experiments have the expertise to run their code and have the need to test it, while sites and vendors may not and only require that a test be realistic. Also different purposes for the tests motivate instrumentation at different levels such as hardware, storage-system internals or the application. Testing which provides a monitoring purpose such as middleware-release testing, application monitoring or site service testing, requires an automated system. These monitoring systems can also, in practise, only run single tests at a time, while, to find points of contention, middleware or site tuning testing needs to occur at large scale. It can also be noted that for running on different storage types, it is useful to use an experiment delivery system (such as via HammerCloud) which already makes all the required customisations. This however, is not possible outside the production environment where instead customisations to the tests are required.

Example	**Vendor Storage**	**Low-level Middleware**	**Middleware framework**		**ROOT I/O Framework**	
Use	Vendor Kit/ Site Tuning	Middleware Scale tests	M/ware Function	M/ware Features Protocols	Site quality level	VO soft / data
Automation	✖	✖	✔	✖	✔	✔
Scale	Stress	Stress	Single	Both	Both	Both
Environment	Test	Test	Production			
Instrumentation	Hardware	Middleware	Application			
Realism	✖	✖	✔	✔	✔	✔✔

Figure 9. A comparison of the example testing frameworks discussed in this paper.

7. Conclusions

The need for I/O testing and analysis is growing and new focus will need to be put on this in the coming years to deal with increasing volumes of data and potentially decreasing hardware performance. The requirements for such testing come from all parties involved but the purposes and goals differ. Here we have given an overview of several testing projects that have been used to test vendor solutions, tune sites, and develop middleware and applications in the LHC community. We have found that valuable improvements can be made from site tuning of read-aheads, from middleware internals such as process threads and from application buffers such as ROOTs TTreeCache. We have also built regular testing to ensure the continuing functionality and performance of experiment and middleware software. In constructing these tests, we have exploited commonalities and provide much that can be built on for future evaluations, however we have also seen that there is a need for customisation in each use case. In the future, this work will be developed on further towards the WLCG TEG goals of generic benchmarking tools and increased I/O information for sites, developers, and experiments.

Acknowledgments

The authors wish to thank the help of Johannes Elmsheuser and Daniel van der Ster with HammerCloud tests for the ROOT framework; Jean-Philippe Baud, Alexandre Beche and

Ricardo Rocha for help with DPM low level tests; Alejandro Ayllon for help with the middleware testing framework; and DELL for their support for the server testing.

References

[1] Vukotic, I *et al.* 2011 Optimization and performance measurements of ROOT-based data formats in the ATLAS experiment. *J. Phys.: Conf. Series* **331** 032032
[2] Canal, P, Bockelman, B, Brun, R 2011 ROOT I/O: The Fast and Furious J. Phys.: Conf. Ser. **331** 042005
[3] Report of the WLCG Technical Evolution Groups in Storage and Data Management URL https://espace.cern.ch/WLCG-document-repository/Technical_Documents/
[4] Brun R and Rademakers F 1997 *Nucl. Instrum. Meth.* A **389** 81
[5] Bhimji, W *et al.* 2012 Dell HPC Scalable Storage Building Block Disk Pool Manager (DPM), URL http://www.dellhpcsolutions.com/
[6] Hellmich, M, 2011 MSc University of Edinburgh, URL http://www.ph.ed.ac.uk/∼wbhimji/GridStorage/StressTestingAndDevelopingDistributedDataStorage-MH.pdf
[7] Perfsuite URL https://svn.cern.ch/reps/lcgdm/perfsuite
[8] Vanderster, D C *et al.* 2010 Functional and large-scale testing of the ATLAS distributed analysis facilities with Ganga *J. Phys.: Conf. Series* **219** 072021
[9] URL http://ivukotic.web.cern.ch/ivukotic/HC/index.asp
[10] Rademakers, F 2011 Root Framework for a Federated World URL http://indico.in2p3.fr/contributionDisplay.py?contribId=9&confId=5527
[11] Tadel, M *et al* 2012 Xrootd Monitoring for the CMS experiment. To be published in *J. Phys.: Conf. Series*
[12] URL https://svnweb.cern.ch/trac/lcgdm/wiki/Dpm/WebDAV/Monitoring
[13] URL http://ivukotic.web.cern.ch/ivukotic/DPM/index.asp