Extreme I/O on HPC for HEP using the Burst Buffer at NERSC

IOP Conf. Series: Journal of Physics: Conf. Series **898** (2017) 082015 doi:10.1088/1742-6596/898/8/082015

Extreme I/O on HPC for HEP using the Burst Buffer at NERSC

Wahid Bhimji, Debbie Bard, Kaylan Burleigh, Chris Daley, Steve Farrell, Markus Fasel, Brian Friesen, Lisa Gerhardt, Jialin Liu, Peter Nugent, Dave Paul, Jeff Porter, Vakho Tsulaia

Lawrence Berkeley National Laboratory, Berkeley, CA 94720 USA

E-mail: wbhimji@lbl.gov

Abstract.

In recent years there has been increasing use of HPC facilities for HEP experiments. This has initially focussed on less I/O intensive workloads such as generator-level or detector simulation. We now demonstrate the efficient running of I/O-heavy analysis workloads on HPC facilities at NERSC, for the ATLAS and ALICE LHC collaborations as well as astronomical image analysis for DESI and BOSS.

To do this we exploit a new 900 TB NVRAM-based storage system recently installed at NERSC, termed a Burst Buffer. This is a novel approach to HPC storage that builds on-demand filesystems on all-SSD hardware that is placed on the high-speed network of the new Cori supercomputer.

We describe the hardware and software involved in this system, and give an overview of its capabilities, before focusing in detail on how the ATLAS, ALICE and astronomical workflows were adapted to work on this system. We describe these modifications and the resulting performance results, including comparisons to other filesystems. We demonstrate that we can meet the challenging I/O requirements of HEP experiments and scale to many thousands of cores accessing a single shared storage system.

1. Introduction

Running HEP experimental workloads on HPC machines presents a significant I/O challenge. One path forward is a fast storage layer, close to the compute, termed a Burst Buffer. Such a layer was deployed with the Cori Cray XC40 System at the National Energy Research Scientific Computing Center (NERSC) at Lawrence Berkeley National Laboratory in the later half of 2015, providing around 900 TB of NVRAM-based storage on 144 nodes used for this study. With the Phase 2 Cori system installed in the latter half of 2016, an additional 144 nodes of storage has been added to the Burst Buffer pool, doubling its capacity and peak bandwidth.

In Section 2 we describe the architecture and software of the NERSC Burst Buffer. We then detail various use-cases for I/O heavy workloads in HEP experiments in Section 3.

2. Burst Buffer Architecture and Software

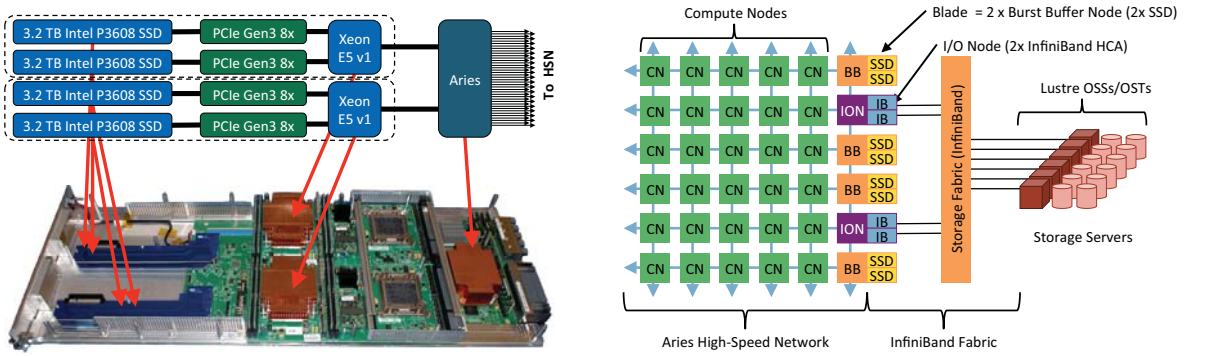
2.1. Cori

Cori is NERSC's newest supercomputer systems and consists of two phases. Phase 2 comprises over 9600 nodes based on the Knights Landing (KNL) architecture and was installed mid-2016.

The Phase 1 system (used for the studies described here) consists of 1632 dual-socket compute nodes with two 2.3 GHz 16-core Haswell processors and 128 GB of DRAM per node. It has a Cray Aries high speed "dragonfly" topology interconnect, and a very high performance Lustre scratch filesystem, that is used as a comparison in many of the results presented here, with 27 PB of storage served by 248 OSTs providing over 700 GB/s peak performance.

2.2. Burst Buffer Architecture

Each Cori Burst Buffer node contains two Intel P3608 3.2 TB NAND flash SSD modules attached over two PCIe gen3 interfaces. These are packaged two to a blade and attached directly to the Cray Aries network interconnect of the Cori system (Figure 1).



(a) A Cori Burst Buffer Blade

(b) Placement of Burst Buffer nodes in the Cori System

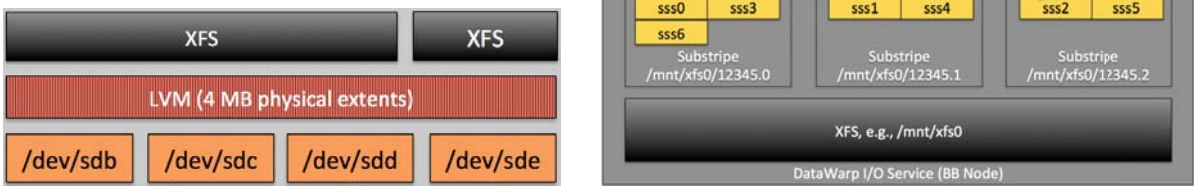
Figure 1: The Cori Burst Buffer Architecture

2.3. Software Environment

In addition to the hardware, NERSC has supported software projects with Cray and SchedMD for the Cray DataWarp software and integration with the SLURM workload manager (WLM). Users allocate Burst Buffer resources via the SLURM WLM that can be striped across different Burst Buffer nodes, or used in a 'private' mode whereby each compute node gets its own namespace which potentially offers improved metadata handling. They can request 'scratch' space that exists for the lifetime of the job, or 'persistent' that can be shared across multiple jobs and users. Details of the entire DataWarp software stack, including the services that create the mount points, can be found in the DataWarp admin guide [1]. A single DataWarp filesystem mount involves several layers:

- Logical Volume Manger (LVM) is used to group the multiple SSD block devices on a single node into one logical block device.
- An XFS file system is created for every Burst Buffer allocation. The Burst Buffer allocation therefore appears to the user as an isolated filesystem.
- The DataWarp File System (DWFS) - a stacked file system based on wrapfs. It handles coordination between the Burst Buffer nodes, staging data in/out of the Burst Buffer allocation, and provides the namespaces (e.g. striped access type) described above.
- Cray Data Virtualization Service (DVS), used for communication between DWFS and the compute nodes.

These layers, and their interaction, are shown in Figure 2.



(a) Schematic of the different layers of the DataWarp software. The Burst Buffer node has 4 SSD block devices aggregated by LVM. When an allocation is requested by the user, an XFS filesystem is created by the DataWarp software.

(b) Schematic of how a file is striped over the DataWarp software on one Burst Buffer node. DVS serves the file to DataWarp in (configurable) 8MB chunks, which are laid out across the three (configurable) substripes on the Burst Buffer node.

Figure 2: The Cori Burst Buffer Software

Example batch script directives are given below to illustrate normal user interaction:

```
#DW jobdw capacity=1000GB access_mode=striped type=scratch
#DW stage_in source=file.dat destination=$DW_JOB_STRIPED/ type=file
#DW stage_out source=$DW_JOB_STRIPED/out destination=out type=directory
srun my.x --infile=$DW_JOB_STRIPED/file.dat  --outdir=$DW_JOB_STRIPED/out
```

This example requests a Burst Buffer allocation for the duration of the compute job ("type=scratch") and that would be visible from all compute nodes ("access_mode=striped"). The allocation is distributed across Burst Buffer nodes in units of 'granularity' (200 GB on the NERSC system at the time of this study). So a request of 1000 GB will normally be placed over 5 separate Burst Buffer nodes (though this is not guaranteed). A file is staged in using the "stage_in" directive, and a directory is staged out at the end of the job using "stage_out". Since the path to the Burst Buffer allocation is unknown when the job is submitted, the user can configure their executable to read in the *$DW_JOB_STRIPED* variable at runtime.

2.4. Benchmark Performance

The performance of the Cori Phase 1 Burst Buffer was measured on installation using the IOR benchmark. Bandwidth tests were performed with 8 GB block size and 1 MB transfer size. IOPS benchmark tests were performed with random 4KB-sized transfers. 1120 compute nodes were used with 4 processes per node. At the time 140 Burst Buffer nodes were in service. Results are given in the table below. All benchmark bandwidth numbers outperform the Lustre filesystem.

Posix File-Per-Process		IOPS	
Read	Write	Read	Write
905 GB/s	873 GB/s	12.6 M	12.5 M

3. HEP Use-Cases

In this paper we focus on new detailed results from several representative experimental HEP projects, for many more use-cases in other science domains see [2].

3.1. ALICE

For the ALICE LHC experiment we construct an I/O intensive analysis that runs on the experiment's event summary file format (ESD) produced during the raw data reconstruction pass. The analysis performs minimal compute by selecting tracks and filling histograms. The data used is a subset of the fully reconstructed data from 2015 Pb-Pb collisions and totals ≈10 TBs in 16k files. The data was staged onto a 26TB Burst buffer allocation, striped across 126 Burst Buffer servers. The analysis was run with 10 input files per single compute core task.

3.1.1. Results: As this is a serial application we can compare the cpu time to the total wall time as a 'CPU efficiency'. On both Lustre and Burst Buffer the CPU efficiency is greater than 90%. If we define a CPU 'idle time' as the difference between wall time and cpu time then we can plot (in figure 3) an effective average 'bandwidth' derived from the data read divided by the idle time per process. As can be seen in figure 3 on this measure, the Burst Buffer out performs Lustre and scales well to large core counts.

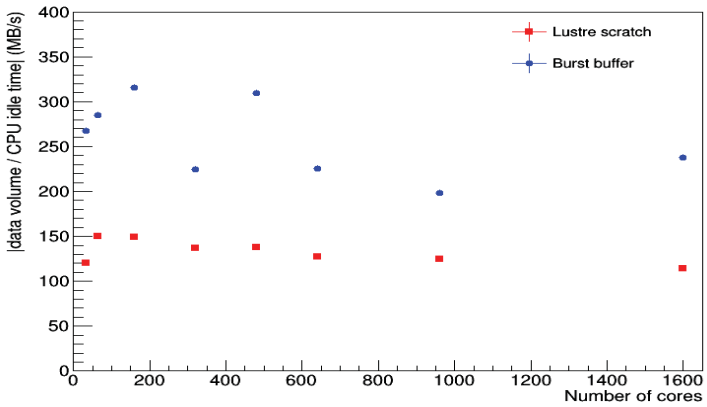


Figure 3: Effective average bandwidth per process for the ALICE ESD analysis application

3.2. ATLAS

ATLAS is now making use of HPC resources for regular production at centers like NERSC. However this is solely to run simulation rather than their more I/O intensive workloads. We explore running more I/O heavy 'derivation' and 'analysis'.

3.2.1. Production Derivation 'Derivation' is the filtering of the 'xAOD' custom ROOT-based format [3] to analysis-specific (xAOD) files. It runs in the ATLAS AthenaMP framework [4] and is therefore a single application per node, but run multi-process with each process reading and writing independent files.

Results: For this study we can obtain I/O stats directly from the 'xAOD::ReadStats' function which makes use of the TVirtualPerfStats interface of ROOT [5]. We then define the 'bandwidth' as the data read divided by the I/O time taken from this ROOT interface. For this study Burst Buffer allocations were varied with 1TB used for 64 and 128 core tests, 2TB for 256 cores, and 10TB for 512 core and higher. This is shown in figure 4 where it can be seen that the Burst Buffer outperforms Lustre and scales well.

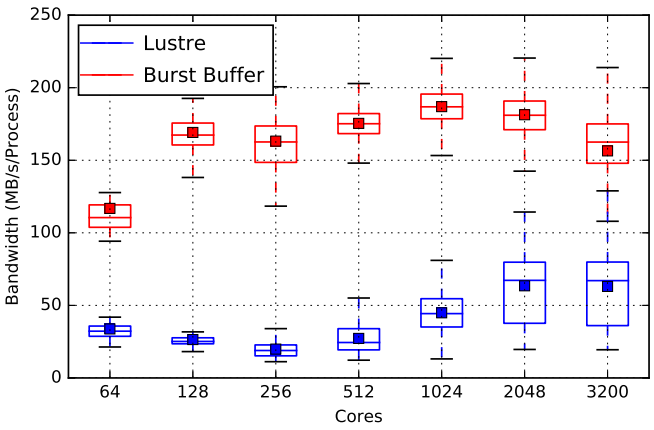
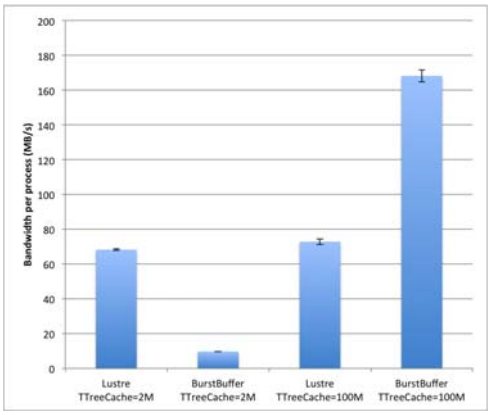


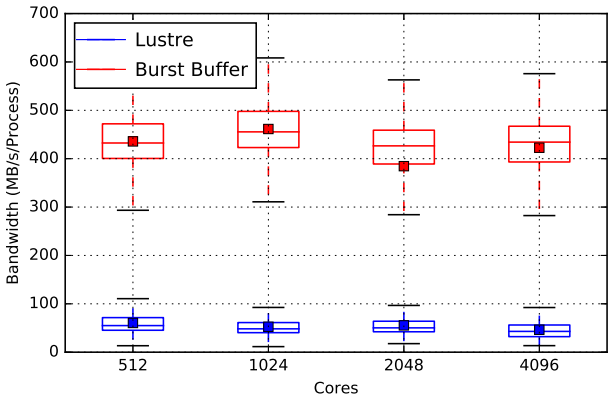
Figure 4: Bandwidth per process for the ATLAS Derivation application

3.3. ATLAS Data Analysis

ATLAS data analysis jobs read filtered xAOD files, perform final analysis-level selections, and fill histograms or write out additional data. As a representative application we use 'QuickAna', a popular high-level ATLAS analysis tool, and the 'EventLoop' framework to execute these tasks [6]. We first analyzed a 475G dataset on a node fully occupied with 32 processes and a 2 TB Burst Buffer allocation. This saw poor performance with default ROOT xAOD settings because the application was issuing over 2 million read calls. This affects running on Burst Buffer more than on Lustre as there is currently no client-side memory caching in DVS, though this feature is in development. We were able to improve the performance by increasing the ROOT built-in memory cache 'TTreeCache' (which pre-fetches and caches in memory only variables of interest [7]) to 100 MB. As shown in figure 5a, this led to substantially less read calls and a 17x read-time performance boost. We also subsequently applied further improvements of using branch-access (more efficient column-like access of variables) and learning pre-fill (that performs only a single read when 'learning' the branches used for the analysis) [7].



(a) Bandwidth per process for the ATLAS analysis application with the default 'TTreeCache' size of 2M and an increased cache of 100M.



(b) Bandwidth per process for the ATLAS Analysis application

3.3.1. Results Having applied the optimizations described above, we then ran on a large 50 TB dataset on a 143 node Burst Buffer allocation. As shown in figure 5b there is around an 8x reading performance advantage with the Burst Buffer that scales well to larger job sizes.

3.4. Tractor and DESI

Tractor is a python-based application which analyzes optical images of the night sky, taken by three telescopes telescopes located at Kitt Peak, AZ and Cerro Tololo, Chile, to produce a 2D map of the distribution of galaxies over about 1/3 the sky. Tractor classifies astronomical sources with a χ^2-fit between models and images. They will combine 2D positions of galaxies with source spectra from the Dark Energy Spectroscopic Instrument (DESI), first light in 2018, to get the distance to each galaxy, thus yielding a 3D Map.

The Tractor configuration involves reading approximately 200 small FITS input files which have a total size of approximately 5 GiB. We collect I/O performance information at the POSIX I/O layer by analyzing the intercepted POSIX calls with a custom tool and the Integrated Performance Monitoring (IPM) tool [8]. This was necessary because the I/O is performed lazily in a Python FITS I/O module which makes it difficult to know when I/O is actually happening. We launch a single instance of Tractor per compute node with 32 child processes and use a Burst Buffer allocation of 3.2 TB. The computation in Tractor is heavily dependent on the light sources identified in the images. Therefore in the scaling test, we replicate the computation in each process by analyzing the same input files.

3.4.1. Results Figure 5 shows the effective bandwidth per process. Figure 6 provides a detailed analysis of I/O: Figure 6a breaks the time spent in I/O into data (e.g. read and write) and metadata (e.g. open and stat) components of I/O, and Figure 6b breaks the time spent in I/O into finer-grained function groups per process.

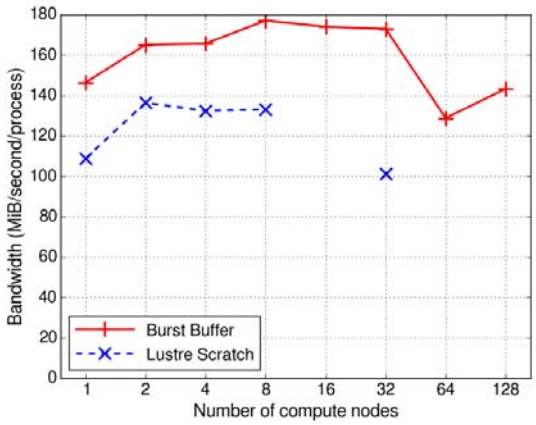
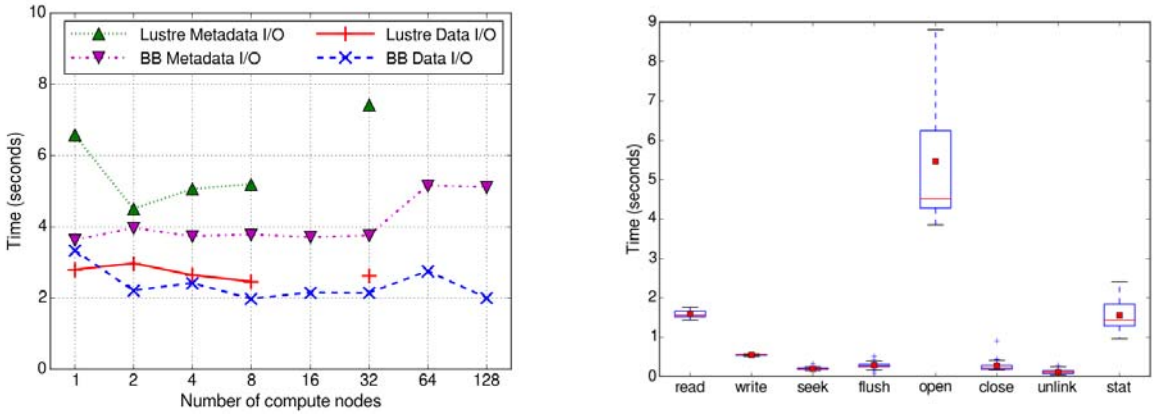


Figure 5: Bandwidth per process for the Tractor application

Figure 5 shows that the effective bandwidth per process is consistently higher when using the Burst Buffer. Lustre results at 16, 64 and 128 compute nodes were not performed. Figure 6a shows that metadata operations are responsible for the majority of I/O time in every single experiment and can take up to 3 times longer than the corresponding data operations. The Burst Buffer reduces data and metadata time for all experiments, except the 1 node configuration, and generally reduces metadata time more significantly than data time. Figure 6b shows the 3 operations with the highest average time to be read, open and stat. There is significant run time spread per process for the open and stat metadata operations.



(a) Scaling of data and metadata components of I/O (b) Time per I/O function in the 32-node Lustre run

Figure 6: Detailed breakdown of I/O costs in the Tractor application

The Burst Buffer can improve metadata time because each Burst Buffer allocation has its own metadata server which can be hosted on any one of the Burst Buffer nodes in the global pool. This is in contrast to the Lustre configuration in which there is a single shared metadata server for all jobs. This can lead to saturation of metadata performance as well as significant variability depending on the other jobs running at the same time.

The metadata operations in Tractor come from accessing hundreds of small FITS files per parent process and from extensive python module loads (present in every python application). The FITS files are placed on the Burst Buffer or Lustre according to the experiment, however, the python modules are always loaded from a GPFS file system. The analysis shows that Tractor needs more than raw I/O bandwidth in order to address its dominant I/O bottleneck.

3.5. H5Boss

We also studied data from the Baryon Oscillation Spectroscopic Survey (BOSS) using data from SDSS. We performed 1000 randomly generated queries to extract small amount of stars and galaxies from millions of such objects, which simulates an exemplar analytics pattern in this field. These query operations involve thousands of file open/close operations and random and small read/write I/O operations. Among them, the 'file open' step includes opening 2444 hdf5 files, reading the file group information, and searching the selected fiber datasets. The file read/write operation (as named 'file copy' in the figure 7), reads the selected fiber objects, each is about 130 KBytes, and then writes to the single shared hdf5 file. The IO pattern in such workflow is random (because the user typically unpredictably specifies the fiber IDs) and small (each fiber object and catalog table is less than 1 MBs). We ran on the final release of SDSS-III complete BOSS dataset which comprised 2444 HDF5 files, a total of ≈3.2 TB. A 4.4 TB Burst Buffer allocation was used, striped across 22 nodes.

3.5.1. Results Figure 7 shows the time in seconds for three steps in the workflow. The first two steps are to read the fiber object, and the third step is a similar operation but to read the catalog tables. It can be seen that there are significantly lower I/O times on Burst Buffer than on Lustre. The results suggest the H5Boss workflow benefits from improved random and small I/O performance on the Burst Buffer particularly in the 'File object copy' stage. Overall there was a 5.5x speedup for this entire workflow.

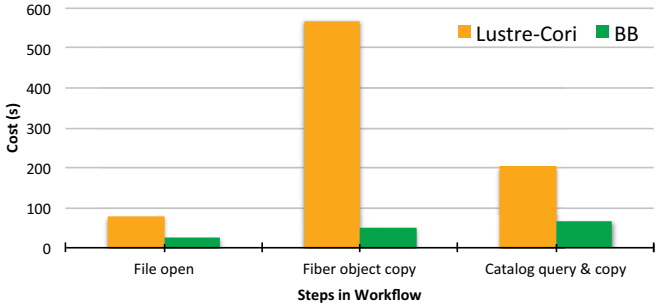


Figure 7: Time for each step in the H5Boss workflow

4. Conclusions

NERSC has successfully brought a Burst Buffer into production with its new Cori system. This offers a novel approach to creating flexibly-sized, on-demand filesystems backed by high-performance NVRAM hardware. The Phase 1 system used at the time of these studies is capable of around 900 GB/s bandwidth and 12.5M IOPS. The Phase 2 system now available at the the time of writing is capable of around 1.7 / 1.6 TB/s Read/Write bandwidth and 28 M read IOPS.

We have demonstrated use of this system here for experimental HEP Workflows and shown it substantially improves I/O over a comparable Lustre filesystem. No application saturated the Burst Buffer performance as indicated by the scaling plots presented here: it was able to sustain the I/O request rate at scale. But, in addition we have seen that HEP applications need more than raw I/O bandwidth. For example, Tractor spends more time in metadata operations than data operations and other applications can be metadata-intensive too. The metadata server per Burst Buffer allocation can deliver higher performance and isolation compared to Lustre where these are shared between all running jobs.

These benefits have been demonstrated across a variety of use-cases from Nuclear Physics, Particle Physics and Cosmology and at the scale of 1000s of cores and 10s of TBs of data. In each case the experiments were asked to provide their most I/O intensive workloads and, in each case, I/O is not now a significant barrier to those applications. Therefore, there is no I/O-related barrier to experimental HEP/NP running these workloads on the NERSC Cori system.

References

[1] Cray 2016 DataWarp Administration Guide URL `http://docs.cray.com/books/S-2557-5204/S-2557-5204.pdf`
[2] Bhimji W, Bard D, Romanus M, Paul D, Ovsyannikov A, Friesen B, Bryson M, Correa J, Lockwood G K, Tsulaia V *et al. Proceedings of Cray Users Group* URL `https://cug.org/proceedings/cug2016_proceedings/includes/files/pap162.pdf`
[3] Buckley A, Eifert T, Elsing M, Gillberg D, Koeneke K, Krasznahorkay A, Moyse E, Nowak M, Snyder S and van Gemmeren P 2015 *J. Phys. Conf. Ser.* **664** 072045
[4] Calafiura P, Leggett C, Seuster R, Tsulaia V and Van Gemmeren P 2015 *J. Phys. Conf. Ser.* **664** 072050
[5] ROOT 2016 TVirtualPerfStats URL `https://root.cern.ch/root/html604/TVirtualPerfStats.htm`
[6] Adams D, Calafiura P, Delsart P A, Elsing M, Farrell S, Koeneke K, Krasznahorkay A, Krumnack N, Lancon E, Lavrijsen W *et al.* 2015 *J. Phys. Conf. Ser.* **664** 032007
[7] Maier T, Benjamin D, Bhimji W, Elmsheuser J, van Gemmeren P, Malon D and Krumnack N 2015 *J. Phys. Conf. Ser.* **664** 042033
[8] Integrated performance monitoring for hpc URL `https://github.com/nerscadmin/IPM`