EPJ Web of Conferences **245**, 07035 (2020) https://doi.org/10.1051/epjconf/202024507035
CHEP 2019

Using HEP experiment workflows for the benchmarking and accounting of WLCG computing resources

Andrea Valassi[1,*], *Manfred* Alef[2], *Jean-Michel* Barbet[3], *Olga* Datskova[1], *Riccardo* De Maria[1], *Miguel* Fontes Medeiros[1], *Domenico* Giordano[1], *Costin* Grigoras[1], *Christopher* Hollowell[4], *Martina* Javurkova[5], *Viktor* Khristenko[1,6], *David* Lange[7], *Michele* Michelotto[8], *Lorenzo* Rinaldi[9], *Andrea* Sciabà[1], and *Cas* Van Der Laan[1]

[1]CERN, Geneva, Switzerland
[2]KIT, Karlsruhe, Germany
[3]CNRS-SUBATECH, Nantes, Frances
[4]Brookhaven National Laboratory, USA
[5]University of Massachusetts Amherst, USA
[6]University of Iowa, USA
[7]Princeton University, USA
[8]INFN, Padova, Italy
[9]Università di Bologna, Italy

Abstract. Benchmarking of CPU resources in WLCG has been based on the HEP-SPEC06 (HS06) suite for over a decade. It has recently become clear that HS06, which is based on real applications from non-HEP domains, no longer describes typical HEP workloads. The aim of the HEP-Benchmarks project is to develop a new benchmark suite for WLCG compute resources, based on real applications from the LHC experiments. By construction, these new benchmarks are thus guaranteed to have a score highly correlated to the throughputs of HEP applications, and a CPU usage pattern similar to theirs. Linux containers and the CernVM-FS filesystem are the two main technologies enabling this approach, which had been considered impossible in the past. In this paper, we review the motivation, implementation and outlook of the new benchmark suite.

1 Introduction

The Worldwide LHC Computing Grid (WLCG) is a large distributed computing infrastructure serving scientific research in High Energy Physics (HEP). WLCG was set up to address the scientific computing needs of the four Large Hadron Collider (LHC) experiments, and it integrates storage and compute resources at almost 200 sites in over 40 countries [1]. While the experiment requirements are managed centrally through a well defined process [2], which matches them against the overall amounts pledged by the contributing funding agencies, the procurement and operation of hardware resources are largely delegated to the individual sites, resulting in a very diverse computing landscape.

The compute power provided by WLCG, in particular, comes from a variety of CPUs distributed worldwide, where the specific hardware deployed at one site can be quite different from that at another site, both in terms of cost and of computing performance. A common unit of measurement is therefore needed to quantify the experiment needs and the resources provided by the sites in a given year, and to allow review boards to compare these to the amounts that were actually used [3, 4]. A good evaluation metric of a compute resource, in this context, is one that is highly correlated to its application throughput, i.e. to the amount of useful

[*] e-mail: andrea.valassi@cern.ch

"work" (e.g. the number of events processed by a HEP application) that the compute resource can do per unit time: this is the typical use case of a CPU benchmark [5]. Since 2009, in particular, HEP-SPEC06 (HS06) has been the standard CPU benchmark for all of LHC computing. The total integrated power of WLCG sites in 2017 [6], for instance, was more than 5M HS06: taking into account that the typical worker nodes deployed in WLCG have an HS06 score of around 10 per CPU core [7], this means that LHC computing in 2017 was supported by approximately 500k CPU cores.

While their main motivation is the overall accounting of resources, both on a yearly basis and in the planning of long term projects, HS06 and other CPU benchmarks have many other applications in LHC computing. Individual computing sites use HS06 for their procurement, to buy the CPU resources providing the amount of HS06 pledged to the HEP experiments for the lowest financial cost, also taking into account electrical power efficiency measured in HS06 per Watt. The experiments also use CPU benchmarks for scheduling and managing their jobs on distributed compute resources, to predict the processing time required to complete a given application workload and optimize its placement or smooth termination on batch queues [8] and preemptible cloud resources. CPU benchmarks may also be useful in software optimizations, to compare an application's performance to the theoretical compute power of the machine where it is run, or to the reference performance of another application.

HS06 has served the needs of WLCG for over 10 years, while many things have changed. On modern hardware, users have reported scaling deviations up to 20% [8] from the performance predicted by HS06. It is now clear that HS06 should be replaced by a new CPU benchmark. In the following, the motivations of the choice to develop a new "HEP-Benchmarks" suite and its implementation are described. After reviewing in Sec. 2 the evolution of CPU benchmarks in HEP up until HS06, Sec. 3 describes the limitations of HS06 and the reasons why the new suite is based on the containerization of LHC experiment software workloads. Section 4 summarises the design, implementation choices and status of HEP-Benchmarks, while Section 5 reports on its outlook, a few months after the CHEP2019 conference.

2 HEP CPU benchmarks: from the CERN Unit to HS06

Computing architectures and software paradigms, in HEP and outside of it, have significantly evolved over time and will keep on changing. This implies that CPU benchmarks, and more specifically those used for HEP computing, also need to evolve to keep up with these changes.

CERN Units

The 1992 paper by Eric McIntosh [9] is an essential read to understand the beginnings of CPU benchmarking in HEP and its later evolution. In the 1980's, the default benchmark was the "CERN Unit", whose score was derived from a small set of typical FORTRAN66 programs used in HEP at the time for event simulation and reconstruction. This was used, for instance, to grant CPU quotas to all users of CERN central systems. In the early 1990's, the definition of a CERN Unit was updated, as the task of running the old benchmark on newer machines turned out to be impossible: the set of programs was thus reviewed, to make the benchmark more portable and more representative of the then current HEP workloads and of FORTRAN77. It was already clear, however, that HEP benchmarks should further evolve, for instance to take into account a more widespread use of FORTRAN90, of double-precision arithmetics on 32-bit architectures, and possibly of vectorisation and of parallel processing.

Largely speaking, CPU benchmarks can be grouped into three categories [10]: kernel, synthetic and application. Kernel benchmarks are based on libraries and small code fragments that often account for the majority of CPU time in user applications: an example is the LINPACK benchmark [11], based on the LINPACK matrix algebra package. Synthetic benchmarks are custom-built to include a mix of low-level instructions, e.g. floating-point or integer operations, resembling that found in user applications. Two examples are the Whetstone [12] and Dhrystone [13] benchmarks. Application benchmarks are based on actual user applications. From a user's perspective, benchmarking a machine based on the user's own application is clearly the best option, although in practice this is often impractical.

The CERN Unit had been designed to be based, as much as possible, on real HEP applications, rather than on kernel or synthetic benchmarks. McIntosh made this clear in his 1992 paper, where, however, he also commented that new approaches may be needed for the future, as at the time he considered it virtually impossible to capture a modern event processing program involving over 100k lines of code, several external libraries and one or more databases or data sets. For reference, the CERN Unit was shipped as a tarball that required less than 50 MB of disk space in total, for unzipping, building and executing all included programs, using the compiler and operating system found on the machine to be benchmarked.

SPEC CPU benchmarks and SI2K (SPEC CINT2000)

In his review of existing benchmarks outside HEP, one option that McIntosh mentioned as being perhaps the most useful for HEP was the SPEC benchmark suite. After the CERN Unit, indeed, all of the default CPU benchmarks used in HEP have been based on the SPEC benchmark suite, up until today. SPEC (Standard Performance Evaluation Corporation) [14], founded in 1988, is a nonprofit corporation formed to establish, maintain and endorse standardized benchmarks of computing systems. SPEC distributes two different categories of CPU benchmark suites, focusing on integer and floating-point operations. In the HEP world, several versions of the SPEC CPU integer benchmark suite have been used since the early 1990's, starting with SPEC CPU92 [15]. In particular, CINT2000 (the integer component of SPEC CPU2000), known informally as SI2K, was the CPU benchmark used in 2005 by the four LHC collaborations for their Computing Technical Design Reports [16–19].

Since about 2005, however, many presentations at HEPiX conferences pointed out a growing discrepancy between the performances of HEP applications and those predicted from the SI2K scores of the systems where these applications were run. In 2006, a HEPiX Benchmarking Working Group (BWG) was set up specifically with the task of identifying the appropriate successor of SI2K. In 2009, the BWG suggested [15] to adopt a new HEP-specific benchmark, HEP-SPEC06 (HS06), based on the then latest SPEC suite, CPU2006 [20].

HEP-SPEC2006: a HEP-specific version of the SPEC CPU2006 C++ benchmark suite

HS06 is based on a subset of SPEC CPU2006 including the seven benchmarks written in C++ [21], three from the integer suite and four from the floating-point suite. In line with the general approach [10] followed in SPEC CPU suites, these seven programs are not kernel or synthetic benchmarks, but represent instead real applications, mostly from scientific domains, although not from the HEP domain. HS06 differs from the SPEC CPU2006 C++ suite in that it includes a few HEP-specific tunings: for instance, the programs must be built using gcc in 32-bit mode also on 64-bit architectures, and with other well defined compiler options [7], and they must also be executed in a specific configuration on the machine to be benchmarked, as if the available processor cores were all configured as independent single-core batch slots to run several single-process applications in parallel.

HS06 was identified as valid successor of SI2K by the HEPiX BWG for essentially two reasons [15]. First, the HS06 score was found to be highly correlated to throughput on a large number of diverse machines in a test "lxbench" cluster, for each of many typical HEP applications. The test machines were typical WLCG worker nodes, all based on x86 architectures, but including single-core and multi-core CPUs with different speeds and from different vendors, and with a diverse range of cache and RAM sizes. The test applications covered four main HEP use cases [22], generation (GEN), simulation (SIM), digitization (DIGI) and reconstruction (RECO), including programs contributed by all four LHC experiments. The second reason for choosing HS06 was that its CPU usage pattern, as measured from CPU hardware counters using perfmon [23–25], was found to be quite similar to that observed on the CERN batch system used by the LHC experiments (in particular, the fraction of floating point operations was around 10% in both cases). The memory footprint of the SPEC CPU2006 tests in HS06, around 1 GB, was also comparable to that of typical HEP applications, requiring up to 2 GB (while the memory footprint of the older SI2K benchmark was only 200 MB).

3 The issues with HS06 and the choice of a new benchmark

In summary, in 2009 HS06 was chosen because, while it is based on a set of C++ applications from domains other than HEP, HS06 had been found to be sufficiently representative of HEP's own typical applications, both in terms of throughput and of CPU usage patterns. The problem today is that, since a few years, it has become clear [26–28] that this is no longer the case.

To start with, the throughputs of HEP applications, mainly of ALICE and LHCb [8], have been reported to deviate up to 20% on some systems from those predicted by HS06. In addition, important differences are now observed between the CPU usage patterns of HS06 and HEP applications, as measured from performance counters using Trident [28] (a tool based on libpfm [23] from perfmon): in particular, with respect to the HS06 benchmarks, HEP workloads have a lower instructions-per-cycle (IPC) ratio and may differ by 20% or more in the percentages of execution slots spent in the four categories suggested by Top-Down analysis [29] (retiring i.e. successful, front-end bound, back-end bound and bad speculation).

More generally, HS06 benchmarks are no longer representative of WLCG software and computing today: memory footprints have increased to 2 GB or more [30] per core; 64-bit builds have replaced 32-bit builds; multi-threaded, multi-process and vectorized software solutions are becoming more common; and the hardware landscape is also more and more heterogeneous, with the emergence of non-x86 architectures such as ARM, Power9 and GPUs, especially at HPC centers.

In addition, SPEC CPU2006, on which HS06 is based, was retired in 2018, after the release of a newer SPEC CPU2017 benchmark suite. An extensive analysis [26, 27] of this new suite by the HEPiX BWG, however, pointed out that SPEC CPU2017 is affected by the same problems as HS06. In particular, SPEC CPU2017 scores were found to have a high correlation to HS06 scores, and hence still an unsatisfying correlation to HEP workloads; also, the CPU usage patterns of SPEC CPU2017, as measured by Trident, were found to be similar to those of HS06, and quite different from those of HEP workloads.

The HEP-Benchmarks suite: using containerized HEP workloads as CPU benchmarks

The solution to the issues described above is, in theory, quite simple. Rather than testing real application benchmarks from domains other than HEP (like those in SPEC CPU2006 and CPU2017) or kernel or synthetic benchmarks, and looking for the benchmarks whose score has the highest correlation to the throughputs of typical HEP workloads, and whose CPU usage patterns look most similar to those of HEP workloads, the "obvious" approach to follow is to build a benchmark suite including precisely those typical HEP workloads.

By construction, in fact, a benchmark based on a HEP application is guaranteed to give a score and a CPU usage pattern that are the most representative of that application. This is precisely the approach followed in the new HEP-Benchmarks [31] suite, which we are building within the HEPiX BWG to make it the successor of HS06, as described in the next section. The central package of this project, hep-workloads, is a collection of workloads from the four LHC experiments, covering all of the GEN, SIM, DIGI and RECO use cases.

In retrospective, this is the same approach on which the CERN Unit was based. As discussed in the previous section, the CERN Unit was eventually discontinued because in the early 1990's it seemed no longer possible to capture a complex HEP application with all of its software and data dependencies. A problem which seemed impossible to solve 30 years ago, however, can be much more easily addressed using the technologies available today. The reason why it is now possible to encapsulate HEP workloads in the hep-workloads package, in particular, is the availability of two enabling technologies: first and foremost, Linux containers [32], which allow the packaging and distribution of HEP applications with all of their dependencies, including the full O/S; and, in addition, the cvmfs shrinkwrap utility [33, 34], which makes it possible to selectively capture which specific software and data files are needed to execute a HEP workload in a portable and reproducible way, out of the much larger LHC experiment software installations on the cvmfs (CernVM-FS) filesystem [35].

4 HEP-Benchmarks: a new CPU benchmark suite for HEP

The development of what has now become the HEP-Benchmarks project started in mid-2017 as a proof-of-concept study by one member of the HEPiX BWG, using the ATLAS kit-validation (KV) [36] and a CMS workload as first examples. This work took off on a larger scale towards the end of 2018, when many more collaborators from the BWG and the four LHC experiments joined the effort. The project is maintained on the gitlab infrastructure at CERN [31], which is also used for Continuous Integration (CI) builds and tests, for issue tracking and for documentation. HEP-Benchmarks includes three main components, which are mapped to separate gitlab repositories and are described in the following subsections.

The hep-workloads package: HEP reference workloads

The hep-workloads package is the core of the HEP-Benchmarks suite. It contains all the code and infrastructure, both common and workload-specific, to build a standalone container for each of the HEP software workloads it includes. Individual images are built, tested and versioned in the package's gitlab CI and are then distributed via its container registry [37]. Images are built as Docker containers [32], but they can also be executed via Singularity [38].

A single command is enough to download a specific benchmark and execute it using the embedded pre-compiled libraries and binaries. A benchmark summary file in json format and more detailed logs are stored in a results directory. For instance, to download the latest version of the LHCb GEN/SIM image, execute it using either Docker or Singularity, and store results in the host /tmp directory, it is enough to run one of the two following commands:

```
docker run -v /tmp:/results \
     gitlab-registry.cern.ch/hep-benchmarks/hep-workloads/lhcb-gen-sim-bmk:latest
singularity run -B /tmp:/results \
     gitlab-registry.cern.ch/hep-benchmarks/hep-workloads/lhcb-gen-sim-bmk:latest
```

The main result in the json file is the benchmark score for the given workload, measured as an absolute throughput of number of events processed per wall-clock time. This is essentially derived from the total wall-clock time to complete the processing of a predefined number of events. The json file also contains all relevant metadata about how the benchmark was run, as well as more detailed results, including memory and CPU time usage.

Some of the HEP workloads, like ALICE and LHCb GEN/SIM, are single-process (SP) and single-threaded (ST); others use parallelism to save memory on multi-core CPUs, via multi-threading (MT) techniques like CMS RECO [39], or via multi-process (MP) techniques involving forking and copy-on-write, like ATLAS RECO [40]. For MT/MP applications, the number of threads/processes is fixed to that used by the experiment in production, and every image is executed in such a way as to fill all available logical cores on the machine that is benchmarked, by launching an appropriate number of identical copies of each application. When more than one copy is executed, the benchmark score is the sum of their throughputs. The number of logical cores, derived from the `nproc` command, is equal to the number of physical cores for machines configured with hyper-threading disabled, but it is higher if this is enabled. On a machine with 16 physical cores and 2x hyper-threading, for instance, by default 32 copies of the ST/SP LHCb GEN/SIM and 8 copies of the 4xMT CMS RECO benchmarks are executed. All of these parameters are, in any case, configurable.

The design of the hep-workloads package relies on the fact that all four LHC experiments install and distribute their pre-compiled software libraries using the cvmfs file system. To add a new workload, experiment experts just need to prepare an orchestrator script, which sets the runtime environment, runs one or more copies of the application, and finally parses the output logs to determine the event throughput, which is used as benchmark score. A common benchmark driver harmonises the control flow and error checking in all workloads, making it easier to debug them. The build procedure in the gitlab CI includes the following four steps.

1. First, an interim Docker container is built, where `/cvmfs` is, as usual, a directory managed by the network-aware cvmfs service [35], which is able to retrieve missing files via http. In addition, the cvmfs shrinkwrap [33, 34] tracing mechanism is enabled.

2. One copy of the workload application from that interim image is then executed: this generates a shrinkwrap trace file specifying which files were accessed from `/cvmfs`.

3. The final standalone Docker container is built, where `/cvmfs` is a local folder, including all files identified by tracing during the previous step. This includes all relevant experiment libraries and executables, pre-compiled for the O/S chosen for this image.

4. This final container is tested, by running the workload using both Docker and Singularity. If tests succeed, the image is pushed to the gitlab registry [37].

A key element of this approach is reproducibility: repeated runs of each workload are meant to always process the same events and produce the same results. This is essential for resource benchmarking, to ensure that timing measurements on two different nodes correspond to the same computational load. It is also important during the build process, where tests of the final container may fail if they need different `/cvmfs` files from those identified while tracing the interim container. Strict reproducibility can be guaranteed for ST (including MP) workloads, but not for the CMS MT workloads, where the sharing of processing across different software threads may lead to small differences in execution paths; however, this is not considered an issue for benchmarking, and no errors have been observed during the build process either.

Workload images vary in size between 500 MB (ATLAS GEN) and 4 GB (CMS DIGI). GEN containers are generally the smallest because event generation is CPU intensive with almost no input data, while DIGI and RECO images are much larger as event digitisation and reconstruction are more I/O intensive and large reference data files must be shipped within the workload containers. Docker images are internally made up of layers (and this structure is maintained when they are converted to Singularity). Taking into account that bug fixing and feature improvements have often led to a rapid development cycle, the hep-workloads CI has been optimized to stack these layers in the order which makes them as cacheable as possible. The bottom layers contain what changes least often, like the O/S and data files, while the higher layers include experiment software and common and workload-specific scripts.

The hep-score package: a new CPU benchmark for HEP

The aim of the hep-score package is to combine the benchmark scores derived from the individual HEP workloads into a single number, a "HEPscore". The package is highly configurable, allowing the definition of a combined HEPscore from any combination of specific versions of individual workloads, with specific MT/MP settings. The numbers of events to process can also be tuned, to choose the appropriate compromise between benchmark precision and execution time. The prototypes that are currently being developed, for instance, derive a combined score from a geometric mean of the throughputs of ATLAS (GEN, SIM and DIGI/RECO), CMS (GEN/SIM, DIGI and RECO) and LHCb (GEN/SIM) benchmarks, and take between 6 and 16 hours to complete. This is similar to what was done for HS06, whose combined score was derived from the geometric mean of the 7 individual SPEC CPU2006 C++ benchmarks. A normalization factor can also be added for each individual benchmark, to redefine its relative score on a machine as a ratio, i.e. as the throughput on that machine divided by the throughput on the reference machine. Unlike absolute throughputs, which can take a priori any value and have the dimensions of events processed per second, these relative scores are quite practical because they are adimensional numbers. In particular, the relative scores of individual benchmarks, and a fortiori the combined relative score defined as the geometric mean of a subset of these scores, are all equal to 1 on the reference machine.

Having said that, it should be stressed that it is impossible to characterize a computer system's performance by a single metric. This concept was very well expressed by Kaivalya Dixit, long-time president of the SPEC corporation, who even warned about "the danger and the folly" [10] of relying on either a single performance number or a single benchmark. There are, however, many use cases where a single number is needed, and the accounting of WLCG resources is presently one of them. This is not a technical issue: it is a policy issue, and its discussion is beyond the scope of this paper. On a technical level, our design of the hep-score package takes this into account by allowing the definition of a highly configurable combined score, but also by the fact that the detailed scores of individual workloads are also stored in the report of any HEPscore execution. This is very important because it provides a mechanism to analyse a posteriori the performance of individual HEP workloads.

The hep-benchmark-suite package: a toolkit for benchmark execution and result collection

The hep-benchmark-suite package, finally, is a toolkit to coordinate the execution of several benchmarks, including not only HEPscore, but also HS06, SPEC CPU2017, KV and others. Results are collected in a global json document that can then be uploaded to a database.

This package is used for what are currently the main activities of our team, to demonstrate the readiness of HEP-Benchmarks as a replacement of HS06: first, for testing that individual HEP workloads are reliable and give stable results (typically, within 5% or better) on repeated run on the same system; second, for the study of their correlations to one another, and to HS06 and other benchmarks. To this end, a wide range of x86 worker nodes has been collected, similar to the lxbench cluster that had been used in the initial comparisons of HS06 and SI2K.

5 Outlook: GPUs and non-x86 CPU architectures

To date, WLCG pledged compute resources have essentially consisted only of x86 CPUs. This processor architecture has therefore been the main focus of developments in the HEP-Benchmarks project so far. The design of its components, however, and specifically that of the core hep-workloads package, is quite general and can be easily extended to non-x86 architectures such as ARM or Power9, and even to other compute resources such as GPUs, which are becoming important for WLCG because of their widespread adoption in the latest supercomputers at HPC centers. By and large, the large scale GEN, SIM, DIGI, RECO production workloads of the four LHC experiments are not yet ready [41] to be moved from traditional WLCG x86 resources to GPUs, but we should be ready to benchmark these resources when this happens. Within the HEP-Benchmarks project, a new hep-workloads-GPU package has therefore been added, to prototype the benchmarking of GPU workloads, including a software workload from the LHC accelerator domain, SixTrack [42]. Work is also in progress to integrate a prototype of the CMS RECO workload on heterogeneous resources [43].

References

[1] S. Campana, *Computing challenges of the future*, Update of European Strategy for Particle Physics, Grenada (2019). https://indico.cern.ch/event/808335/contributions/3365192

[2] I. Bird, *LHC computing (WLCG): Past, present, and future*, Proc. Int. School of Physics "Enrico Fermi", Varenna (2014). https://doi.org/10.3254/978-1-61499-643-9-1

[3] I. Bird, *WLCG status report*, CERN-RRB-2019-123, WLCG RRB, October 2019. https://indico.cern.ch/event/843657/contributions/3542198

[4] P. K. Sinervo, *Computing resources scrutiny group report*, CERN-RRB-2019-080, WLCG RRB, October 2019. https://indico.cern.ch/event/843657/contributions/3542201

[5] J. Dongarra, J. L. Martin, J. Worlton, *Computer benchmarking: paths and pitfalls*, IEEE Spectrum **24**, 38-43 (1987). https://doi.org/10.1109/MSPEC.1987.6448963

[6] HEP Software Foundation, *A Roadmap for HEP Software and Computing R&D for the 2020s*, Comput. Softw. Big Sci. **3**, 7 (2019). https://doi.org/10.1007/s41781-018-0018-8

[7] HEPiX Benchmarking WG web site. https://w3.hepix.org/benchmarking.html

[8] P. Charpentier, *Benchmarking worker nodes using LHCb productions and comparing with HEP-SPEC06*, Proc. CHEP2016, San Francisco, J. Phys. Conf. Ser. **898**, 082011 (2017). https://doi.org/10.1088/1742-6596/898/8/082011

[9] E. McIntosh, *Benchmarking computers for HEP*, 15th CERN School of Computing, L'Aquila (1992), CERN-CN-92-13. https://doi.org/10.5170/CERN-1993-003.186

[10] K. M. Dixit, *Overview of the SPEC Benchmarks*, in Jim Gray (Ed.), *The Benchmark Handbook for Database and Transaction Systems* (2nd Edition), Morgan Kaufmann 1993. https://jimgray.azurewebsites.net/benchmarkhandbook/toc.htm

[11] J. Dongarra, P. Luszczek, A. Petitet, *The LINPACK Benchmark: past, present and future*, Conc. Comp. Pract. Exper. **15**, 803-820 (2003). https://doi.org/10.1002/cpe.728

[12] H. J. Curnow, B. A. Wichmann, *A synthetic benchmark*, The Computer Journal **19**, 43-49 (1976). https://doi.org/10.1093/comjnl/19.1.43

[13] R. P. Weicker, *Dhrystone: a synthetic systems programming benchmark*, Comm. ACM **27**, 1013-1030 (1984). https://doi.org/10.1145/358274.358283

[14] Standard Performance Evaluation Corporation (SPEC) web site. https://spec.org

[15] M. Michelotto et al., *A comparison of HEP code with SPEC benchmarks on multi-core worker nodes*, Proc. CHEP2009, Prague, J. Phys. Conf. Ser. **219**, 052009 (2010). https://doi.org/10.1088/1742-6596/219/5/052009

[16] ALICE Coll., *ALICE Computing TDR* (2005). https://cds.cern.ch/record/832753

[17] ATLAS Coll., *ATLAS Computing TDR* (2005). https://cds.cern.ch/record/837738

[18] CMS Coll., *CMS Computing TDR* (2005). https://cds.cern.ch/record/838359

[19] LHCb Coll., *LHCb Computing TDR* (2005). https://cds.cern.ch/record/835156

[20] J. L. Henning, *SPEC CPU2006 benchmark descriptions*, ACM SIGARCH Comp. Arch. News **34**, 1-17 (2006). https://doi.org/10.1145/1186736.1186737

[21] M. Wong, *C++ benchmarks in SPEC CPU2006*, ACM SIGARCH Comp. Arch. News **35**, 77-83 (2007). https://doi.org/10.1145/1241601.1241617

[22] G. Benelli et al., *The CMSSW benchmarking suite: using HEP code to measure CPU performance*, Proc. CHEP2009, Prague, J. Phys. Conf. Ser. **219**, 052016 (2010). https://doi.org/10.1088/1742-6596/219/5/052016

[23] S. Eranian, *Perfmon2: a flexible performance monitoring interface for Linux*, Proc. OLS2006, Ottawa. https://www.kernel.org/doc/ols/2006/ols2006v1-pages-269-288.pdf

[24] A. Hirstius, *CPU-level performance monitoring with Perfmon*, HEPiX Spring 2008, CERN. https://indico.cern.ch/event/27391/contributions/613843

[25] A. Nowak, *An update on perfmon and the struggle to get into the Linux kernel*, Proc. CHEP2009, Prague, J. Phys. Conf. Ser. **219**, 042048 (2010). https://doi.org/10.1088/1742-6596/219/4/042048

[26] D. Giordano et al., *Next Generation of HEP CPU Benchmarks*, Proc. CHEP2018, Sofia, EPJ Web of Conf. **214**, 08011 (2019). https://doi.org/10.1051/epjconf/201921408011

[27] D. Giordano, E. Santorinaiou, *Next Generation of HEP CPU Benchmarks*, Proc. ACAT2019, Saas Fee. https://indico.cern.ch/event/708041/contributions/3276257

[28] S. Muralidharan, D. Smith, *Trident: An Automated System Tool for Collecting and Analyzing Performance Counters*, Proc. CHEP2018, Sofia, EPJ Web of Conf. **214**, 08024 (2019). https://doi.org/10.1051/epjconf/201921408024

[29] A. Yasin, *A Top-Down method for performance analysis and counters architecture*, Proc. 2014 IEEE ISPASS, Monterey. https://doi.org/10.1109/ISPASS.2014.6844459

[30] J. Elmsheuser et al., *ATLAS Grid Workflow Performance Optimization*, Proc. CHEP2018, Sofia, EPJ Web of Conf. **214**, 03021 (2019). https://doi.org/10.1051/epjconf/201921403021

[31] HEP-Benchmarks project, https://gitlab.cern.ch/hep-benchmarks.

[32] Docker, *What is a container?*, https://www.docker.com/resources/what-container

[33] CernVM-FS Shrinkwrap, https://cvmfs.readthedocs.io/en/stable/cpt-shrinkwrap.html

[34] P. S. M. Teuber, *Efficient unpacking of required software from CERNVM-FS*, CERN Openlab Report (2019). https://doi.org/10.5281/zenodo.2574461

[35] J. Blomer et al., *Distributing LHC application software and conditions databases using the CernVM file system*, Proc. CHEP2010, Taipei, J. Phys. Conf. Ser. **331**, 042003 (2011). https://doi.org/10.1088/1742-6596/331/4/042003

[36] A. De Salvo, F. Brasolin, *Benchmarking the ATLAS software through the Kit Validation engine*, Proc. CHEP2009, Prague, J. Phys. Conf. Ser. **219**, 042037 (2010). https://doi.org/10.1088/1742-6596/219/4/042037

[37] HEP-Benchmarks project: hep-workloads container registry, https://gitlab.cern.ch/hep-benchmarks/hep-workloads/container_registry

[38] G. M. Kurtzer, V. Sochat, M. W. Bauer, *Singularity: Scientific containers for mobility of compute*, PLoS ONE **12**, e0177459 (2017). https://doi.org/10.1371/journal.pone.0177459

[39] E. Sexton-Kennedy et al., *Implementation of a Multi-threaded Framework for Large-scale Scientific Applications*, Proc. ACAT2014, Prague, J. Phys. Conf. Ser. **608**, 012034 (2015). https://doi.org/10.1088/1742-6596/608/1/012034

[40] P. Calafiura et al., *Running ATLAS workloads within massively parallel distributed applications using Athena Multi-Process framework*, Proc. CHEP2015, Okinawa, J. Phys. Conf. Ser. **664**, 072050 (2015). https://doi.org/10.1088/1742-6596/664/7/072050

[41] A. Valassi, *Overview of the GPU efforts for WLCG production workloads*, Pre-GDB on benchmarking, CERN (2019). https://indico.cern.ch/event/739897/contributions/3559134

[42] R. De Maria et al., *SixTrack Version 5*, Proc. IPAC2019, Melbourne. J. Phys. Conf. Ser. **1350**, 012129 (2019). https://doi.org/10.1088/1742-6596/1350/1/012129

[43] A. Bocci, *Heterogeneous online reconstruction at CMS*, to appear in Proc. CHEP2019, Adelaide. https://indico.cern.ch/event/773049/contributions/3474336