Benchmarking Cloud Resources for HEP

IOP Conf. Series: Journal of Physics: Conf. Series **898** (2017) 092056 doi:10.1088/1742-6596/898/9/092056

Benchmarking Cloud Resources for HEP

M Alef[2], C Cordeiro[1], A De Salvo[3], A Di Girolamo[1], L Field[1], D Giordano[1], M Guerri[1], F C Schiavi[1] and A Wiebalck[1]

[1] CERN
[2] Karlsruhe Institute of Technology
[3] Università & INFN Roma I

E-mail: domenico.giordano@cern.ch

Abstract. In a commercial cloud environment, exhaustive resource profiling is beneficial to cope with the intrinsic variability of the virtualised environment, allowing to promptly identify performance degradation. In the context of its commercial cloud initiatives, CERN has acquired extensive experience in benchmarking commercial cloud resources. Ultimately, this activity provides information on the actual delivered performance of invoiced resources. In this report we discuss the experience acquired and the results collected using several fast benchmark applications adopted by the HEP community. These benchmarks span from open-source benchmarks to specific user applications and synthetic benchmarks. The workflow put in place to collect and analyse performance metrics is also described.

1. Introduction

Since the early days of computing, benchmarking has represented a powerful method to assess the relative performance of computing systems. Benchmarking applications have evolved together with computing architectures, programming languages and the problems being tackled with the computing resources. To be meaningful, benchmark applications should provide measurements which are compatible with the performance that a real application would experience at runtime. Benchmarks decouple the measurement process from the specific and often difficult task of systematically reproducing features of real workloads.

Benchmarking is a consolidated activity in High Energy Physics (HEP) computing where large computing power is needed to support scientific workloads. In HEP, great attention is paid to the speed of the CPU in accomplishing tasks characterised by a mixture of integer and floating point operations and a memory footprint of few gigabytes. The benchmark widely adopted by the HEP community to measure CPU performance is HEP-SPEC06 (HS06) [1]. It has been defined by the HEPiX Benchmarking Working Group [2] and is based on a subset of the industry standard SPEC CPU2006 benchmark suite. As of 2009, HS06 is the official CPU performance metric used by WLCG sites to describe experiments' computing requirements, assess data centres' computing capacity and procure new hardware. It is distributed under a license that allows site-wide deployment and, on modern architectures, it normally requires several hours for a single run. As a consequence, it is adequate for a limited number of tests executed on individual bare-metal servers or individual server models during hardware acceptance or service commissioning.

With the evolution of computing services towards virtualised IaaS and the increasing adoption of public and private cloud resources in the WLCG, a need for repeated benchmark tests

is emerging, in order to capture performance variations of the VM, caused not only by the configuration of the VM itself and the underlying hardware, but also by changes in load conditions of neighbouring VMs. For this reason, in the HEP community a quest for fast benchmark applications has started with the objective of identifying benchmarks that can run quickly enough to avoid wasting compute resources, but still accurate enough to represent HEP workloads running on the cloud.

The next sections describe the benchmark applications that have been studied and the peculiarities identified, the application areas addressed and the workflow put in place to collect and analyse performance metrics.

2. HEP benchmarks

The benchmark applications included in this study aim to assess the speed of the CPU in accomplishing specific computing tasks. Since pseudo-random number generation is a core task of Monte Carlo simulations (one of the most frequent type of WLCG workload running on commercial clouds), the selected benchmark jobs consist mainly of pseudo-random number generations via synthetic code or using submodules which are part of the experiments' applications. The seed used by the random number generator is fixed at configuration level in order to guarantee the reproducibility of the sequence across different tests. This approach also makes sure that the comparison between tests is not affected by the variation of the sequence, which would result in a decreased resolution of the measurements and higher spread. The synthetic benchmarks used in this study are called ATLAS Kit Validation (KV) [3] , Dirac Benchmark 2012 (DB12) [4] and Whetstone benchmark (WSN).

The KV is a toolkit adopted by the ATLAS collaboration for the validation of their software installation in Grid sites. The tests include, among others, the GEANT4 [5] simulation of the ATLAS detector. A KV benchmark simulates N independent events of a single muon particle propagating through the detector. The CPU time needed to simulate each event is recorded and the average over the N events is computed as the final benchmark result. In order to remove from the measurement the overhead coming from the initialization of the software libraries and the configuration of the simulation parameters (detector geometry, list of particles, properties of the materials), the first event in the sequence is excluded from the final average.

DB12 is a Python script that iteratively samples a sequence of pseudo-random numbers from a Gaussian distribution. The number of iterations is fixed at $12.5 \cdot 10^6$, which corresponds to 250 HS06 seconds. Even if rather simple in its conception, it has been proven to scale well with the duration of LHCb simulation jobs [6].

WSN is a synthetic benchmark introduced in 1972 consisting in several modules meant to represent a mix of "typical" operations executed by scientific calculators, including floating point computations, such as sin, cos, sqrt, exp, and log as well as integer calculations, array accesses, conditional branches, and procedure calls. Despite its synthetic mix of operations, the goal of this benchmark is to measure the performance of both integer and floating-point arithmetic and it is mostly representative of small engineering/scientific applications that fit into cache memory. With a given configuration, the benchmarks above-mentioned run very quickly, with execution times around 5 minutes, 2 minutes and 1 minute for KV, DB12 and WSN respectively.

3. Benchmarking suite

In order to allow for a fast scale-out of the benchmarking process and ease the systematic collection of measurements, a benchmark suite has been built, based on a scalable architecture capable of representing HEP workloads on compute resources. The open-source suite features a light set of dependencies, a configurable set of executable benchmarks jobs and a JSON-based measurement report. The JSON document contains not only the benchmark results but also metadata information such as the unique identifier and timestamp of the test, the name of the

machine under test, its IP address, operating system and CPU model. In the implemented data flow (Fig. 1) the JSON documents are transferred through an AMQ [7] transport layer, and sinked with Logstash into Elasticsearch [8], where the data is subsequently monitored with Kibana [9] or extracted and analysed with other analytic tools, such as Jupyter [10] and Pandas [11] libraries. The benchmarks adopted are single threaded as are most of the LHC experiments' software applications. In order to fully load the compute resources and reproduce the worst case load scenario, where all the compute slots are running jobs and CPU idle time is minimized, the benchmarking procedure is, by default, configured to run in parallel with as many threads as the number of logical cores of the server and the benchmark result is calculated as the arithmetic average over the set of threads.

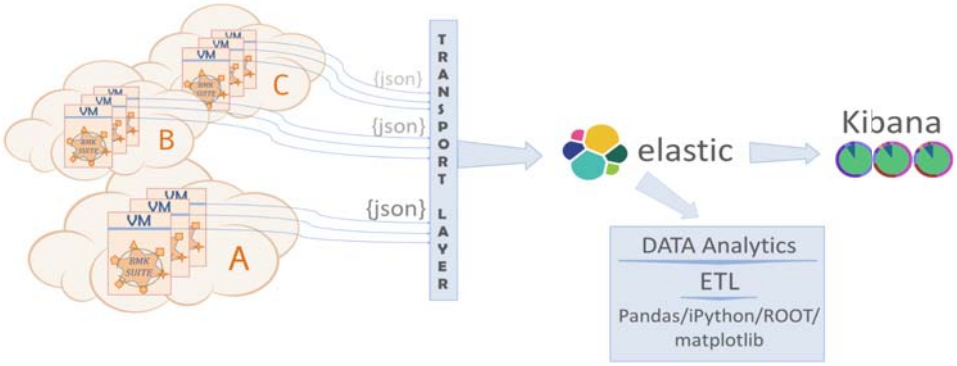


Figure 1: Example of a benchmarking data flow implementation. Applied in several use cases, summing up a total of 4M results collected throughout the last 1.5 years, from over 77k different machines and 89 different CPU architectures.

4. Profiling commercial clouds

CERN has acquired extensive experience in benchmarking cloud resources from commercial providers [12], including Microsoft Azure [13], IBM, ATOS [14], T-Systems and the Deutsche Börse Cloud Exchange. The CERN cloud procurement process has also greatly profited from benchmark measurements to assess the compliance of the bids with the capacity requested in the technical specifications. During the tendering phase, cloud providers are invited to benchmark their resources. The results collected by CERN (Fig. 2) are used not only to verify compliance with the technical requirements but also to evaluate the effective cost, renormalised to the performance, of the different cloud offers.

In addition, recurring benchmarking is performed during the cloud production activity in order to monitor the delivered performance. VMs are generally benchmarked every 8 or 12 hours. In situations where the delivered performance falls under certain pre-defined reference values, penalties may be applied according to the contractual terms. Figure 3 shows how recurring benchmarking can help identifying poorly and well performing providers throughout the delivery period.

4.1. Estimation of WLCG jobs performance

The majority of the workloads executed in the WLCG are simulation jobs, and as these are mainly CPU intensive, a recurring CPU benchmarking can also be used to anticipate their real performance on the delivered resources. Such representative measurements have been tested with several commercial cloud providers, where the KV benchmark results were compared with the

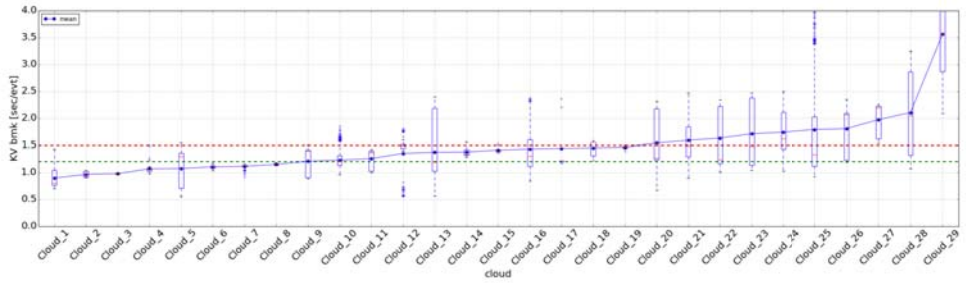
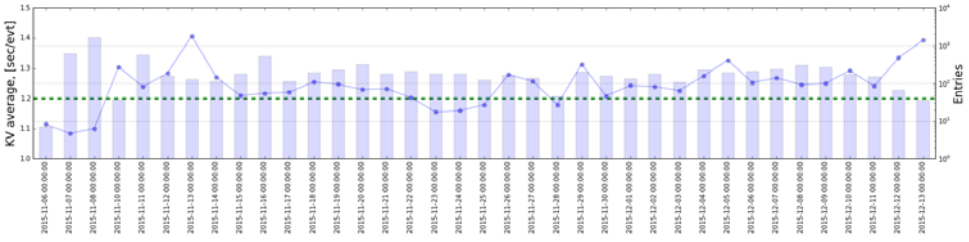
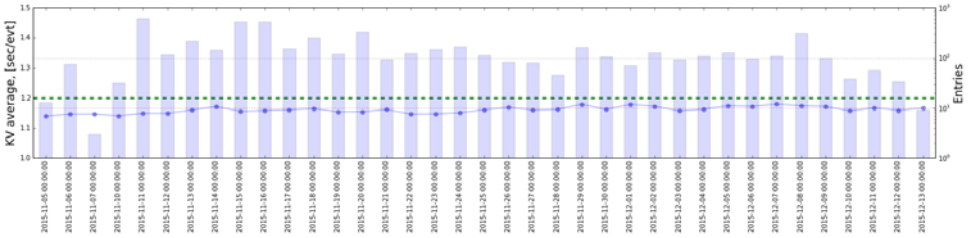


Figure 2: Average CPU performance of VMs provisioned in several commercial cloud providers during a procurement process. The green and red dashed lines identify the requested performance and the rejection threshold respectively. The anonymous identifiers are used for confidentiality reasons.



(a)



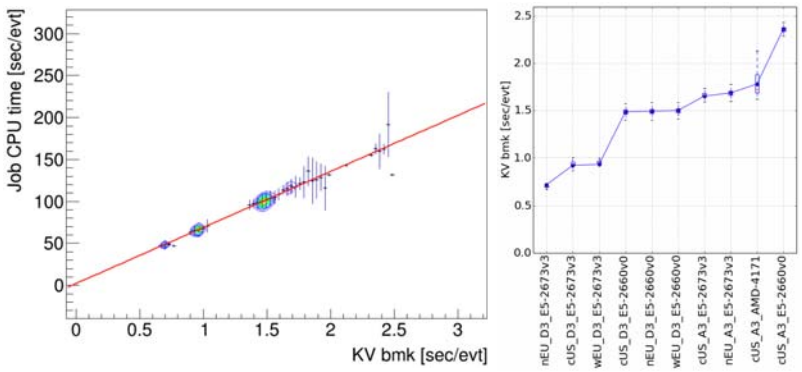
(b)

Figure 3: Example of a poorly performing (a) and well performing (b) cloud provider, based on the recurring KV benchmarking of the delivered resources. The blue dots report the average performance of the running VMs over 24 hours. The number of measurements performed during each day is reported by the histogram bins.The compensation region is defined by the area above the green dashed line (1.2 sec/evt).

average CPU time spent by the VM to simulate ATLAS events, as reported by the experiment's job monitoring framework. Figure 4a shows the correlation between these two independent measurements for a large range of benchmark results obtained from VMs of different flavours running in several Azure data centres, as shown in Figure 4b.

5. A systematic study

A recent benchmark study has examined the compute performance of a new set of 240 physical servers located in the CERN Wigner data centre (Hungary). The servers are part of the CERN OpenStack cloud Infrastructure-as-a-Service, based on KVM [15], and expose compute resources dedicated to the Tier-0 batch system activity as VMs. Each bare-metal server consists of two 8-core Intel Xeon E5-2630v3 processors providing a total number of 32 threads per server with



(a) (b)

Figure 4: Correlation between KV measurements and average CPU time per event of ATLAS jobs (a) running in several Azure data centres within VM flavours of different performance (b).

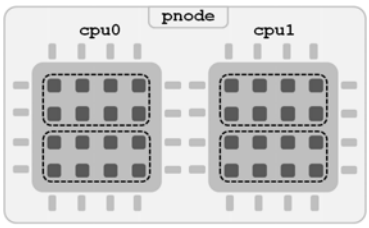


Figure 5: Four VMs of eight vCPUs each (VM-8). Within each physical CPU, software threads were not pinned to the hardware counterparts. The external rectangle, labeled "pnode", represents the physical host and the two internal squares represent the physical CPUs with 16 logical cores. The VMs provisioned are represented with a dotted line, each using a number of threads (black squares) equivalent to the number of vCPUs.

simultaneous multithreading enabled. The server memory adds up to 64 GiB of DDR4 RAM in fully balanced configuration, with each memory channel populated with the same number of dual in-line memory modules (DIMM) of equal capacity.

VMs with different numbers of virtual processors (vCPUs) have been provisioned, keeping fixed the amount of memory (1.875 MiB) and storage (30 GB) per vCPU. A VM with n vCPUs is hereinafter referred to as VM-n. Five configurations have been tested, each consisting of VMs of the same size n, with n={32, 16, 8, 4, 1}. In order to collect performance metrics under similar load conditions, the number of provisioned VMs per server in a given configuration VM-n has been fixed to 32/n. The benchmark jobs are executed synchronously on all the VMs in order to guarantee the full utilization of host resources. Each qemu-kvm [16] thread belonging to the same VM has been configured via libvirt to run on a single Non-Uniform Memory Access (NUMA) node, with the result that each VM, with the exception of VM-32, was confined to a specific socket (Figure 5).

The VM-8 configuration, consisting of four VMs each with 8 vCPUs, offered the opportunity to easily compare the performance of a setup where a single VM is allocated to one socket instead of two: this new configuration, referred to as VM-8_3, is achieved by destroying one VM per hypervisor in a VM-8 configuration. The resulting setup consists of 3 VMs: two VMs (labeled VM-8_{3T}) hosted by one of the two sockets and the remaining VM (labeled VM-8_{3A}) hosted by the other socket and potentially making use of the complete set of CPU resources as it would be in the case of simultaneous multithreading (SMT) disabled.

Figure 6 shows for each benchmark the relative performance across configurations defined as $Ratio_{(bmk_A,VM_X)} = 2 \cdot \frac{\mu_{(bmk_A,VM_X)}}{\mu_{(bmk_A,VM-8_{3A})}}$, where the performance of VM-8_{3A} is taken as reference and the factor of 2 takes into account the double number of running threads when SMT is enabled. In addition to the fast benchmarks, the relative performance obtained running HS06 is also reported for the configuration VM-8 and VM-16.

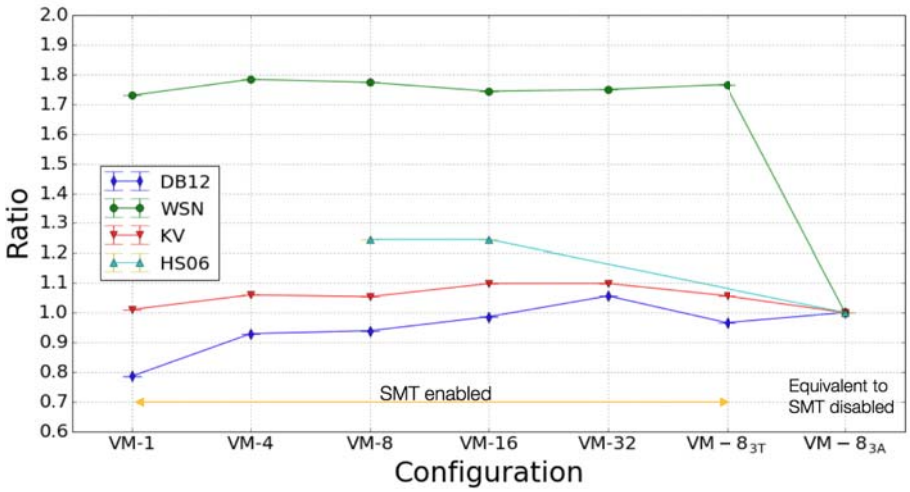


Figure 6: Relative performance of DB12 (diamond), WSN(circle), KV (triangle down) and HS06 (triangle up) across different configurations. The performance of VM-8_{3A} is used as reference.

The results show that SMT enabled implies a benefit of 24% in throughput for HS06 when all hardware threads are in use. On the contrary KV has a benefit in throughput smaller than 10% and DB12 results in practically little or no benefit from running multiple hardware threads because it is penalised for most of the configurations up to the 20%. Finally WSN shows remarkable results with a throughput improvement of 70%. These different performance profiles can be explained with an analysis of the peculiarities of the three workloads under test.

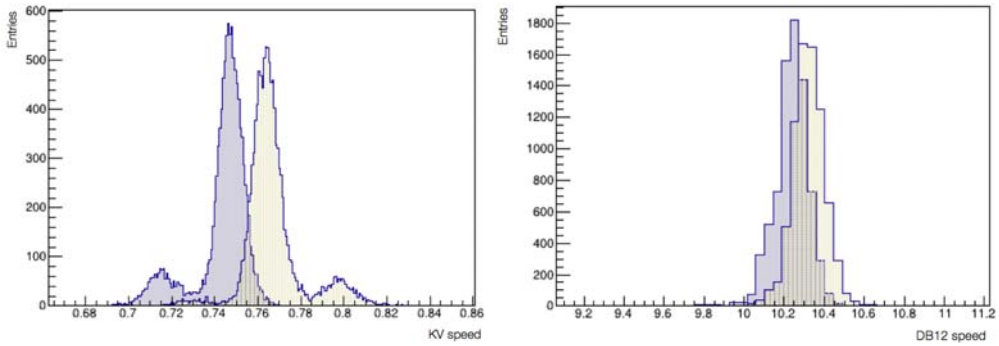
DB12 is a Python benchmark and as a consequence it is significantly affected by the behavior of the interpreter, CPython in this case. The core component of CPython is a large *switch* statement in function *PyEval_EvalFrameEx* that is used to dispatch bytecode instructions to the corresponding *case* branches. This construct highly benefits from a performant branch target prediction unit, in order to correctly feed the processor pipeline and avoid as much as possible wasted/stalled cycles on indirect jumps. On the Haswell architecture, the front-end of the pipeline does an excellent job in speculatively fetching instructions upon encountering the indirect jump that dispatches Python bytecode. This reduces drastically the number of stalled cycles due to unpredictable or mispredicted branches compared to previous architectures, e.g. Sandy Bridge and Ivy Bridge, reducing the time during which a single hardware thread can make forward progress without having to share resources with the neighbouring thread. With the pipeline constantly serving two hardware threads, the per-thread throughput drops by 50%.

KV is a complex tool consisting of a very large code and data segments and leveraging tens of shared libraries from Geant4 framework: caches and Translation Lookaside Buffers (TLB) are a critical resource for this type of workload, especially Instruction Translation Lookaside Buffers (iTLB), and when shared between hardware threads, conflicts might lead to a severe performance hit. This is especially true for iTLB, since even though part of the address space is shared between software processes, e.g. page frames hosting code pages from shared libraries, entries for virtual to physical translation are necessarily distinct. WSN performance instead reflects

an excellent use of hardware multithreading, without highlighting any of the aforementioned bottlenecks.

5.1. Single Thread Performance

The analysis of the performance profiles of the three benchmarks revealed a bimodal gaussian distribution of ATLAS Kit Validation samples, with a 2% difference between the mean of the two gaussians (Figure 7a) . This effect was negligible with DB12 and WSN, showing respectively a 0.8% (Figure 7b) and 0.4% delta.



(a) (b)

Figure 7: Distribution of the execution speed of KV (a) and DB12 (b) for VM-8. The two small bumps in (a) are due to the different behaviour of few servers.

After an in-depth analysis of the samples, it was observed that the slower results were systematically collected from the VM running on the second physical CPU of the dual-socket server. Further single-thread runs highlighted the presence of two hardware threads performing worse than the others, i.e. bare-metal threads number 8 and 24, both belonging to the first physical core of the second CPU. When testing a VM-8 configuration with pinning of qemu-kvm threads to hardware counterparts, a single-thread KV run on either one of the virtual processors corresponding to hardware threads 8 or 24 was showing a 16% increase in the final average simulation time, which could easily explain the 2% spread highlighted by the statistical distribution when averaged over 8 virtual processors. The presence of a slower hardware thread is normally attributed to additional workload, either in userspace or in kernel space (e.g. interrupt service routines, softirqs), that due to scheduling affinity is assigned to one of the threads belonging to the same physical core. Further investigations disproved this hypothesis, making it necessary to collect performance counters as close as possible to the microarchitecture. In order to do so, *perf* was attached to KV running on the bare-metal server, where the same performance asymmetry could be easily reproduced. The resulting profile showed an unexpectedly high number of cycles spent on the initial instructions of the functions constituting the major contributors to the additional runtime required by KV when executed on hardware threads 8 or 24. A first hypothesis pointed in the direction of instruction cache or iTLB misses: even though this could not explain the presence of two hardware threads slower than the others, the large code footprint of KV seemed a good candidate for a high number of such events compared to other workloads. The first attempt to validate this hypothesis involved the creation of a synthetic benchmark aimed at generating instruction cache misses. The idea behind the benchmark is to generate a large number of functions carrying out the same computation, which are then called randomly at runtime. A Python script was used to automatically produce C code compiled with gcc 4.8.5

Initial results highlighted the same performance asymmetry when running on hardware threads 8 or 24, with the performance counters still pointing in the direction of a slower front-end. An additional synthetic benchmark was written to identify a possible correlation between this effect and the size of the "hot" portion of the code segment being executed. Interestingly enough, this benchmark reported identical performance for all cores on the machine up to a code segment size of around 512 KiB, with hardware threads 8 and 24 starting to show higher runtime beyond this point. This number is very interesting, as it is the portion of code that can be addressed using 128 iTLB entries and 4 KiB pages. It does not come as a surprise that the iTLB on Haswell and Broadwell processors can store exactly 128 entries. A further attempt was aimed at reproducing this asymmetry using large pages, 2 MiB in this case. After compiling the benchmark with support for huge pages and preloading libhugetlbfs.so, the results did not highlight any performance difference: by using 2 MiB pages the load on the iTLB is drastically reduced when executing the relatively small-sized code segment of the synthetic benchmark, supporting the hypothesis that the iTLB is behaving differently on the first physical core of the second CPU. After additional tests, the same asymmetry was observed on servers fitted with Broadwell-EP CPUs and on quad-sockets Haswell-EP systems, but it was not possible to reproduce on SandyBridge-EP and IvyBridge-EP processors. The final explanation to this performance asymmetry is to date still unknown.

6. Conclusion

A study of candidate applications to quickly benchmark virtual computing resources running HEP workloads has been presented. Each benchmark highlights peculiarities that result in different behaviours under high CPU load and simultaneous multithreading enabled. The systematic collection, under controlled conditions, of a large amount of benchmark results has revealed an unexpected performance profile of the first physical core on the second socket on Intel Haswell and Broadwell architectures. The adoption of a toolkit to configure and manage the benchmark tests proved to be effective to enable the prompt monitoring and analysis of the results.

References

[1] M Michelotto et al. A comparison of HEP code with SPEC benchmarks on multi-core worker nodes. *J. Phys. Conf. Ser.*, 219:052009, 2010.
[2] HEPiX benchmarking working group. `www.hepix.org/e10227/e10327/e10325`.
[3] Alessandro De Salvo and Franco Brasolin. Benchmarking the ATLAS software through the Kit validation engine. *Journal of Physics: Conference Series*, 219(4):042037, 2010.
[4] Dirac benchmark 2012. `gitlab.cern.ch/mcnab/dirac-benchmark/tree/master`.
[5] G Cosmo and the Geant4 Collaboration. Geant4 - towards major release 10. *Journal of Physics: Conference Series*, 513(2):022005, 2014.
[6] P Charpentier et al. Benchmarking worker nodes using lhcb simulation productions and comparing with hep-spec06. *to appear in J. Phys.: Conf. Ser.*, 2016.
[7] Apache activemq. `activemq.apache.org/`.
[8] Elastic. Elasticsearch reference. `https://elastic.co/guide/en/elasticsearch/reference/current/index.html`.
[9] Elastic. Kibana user guide. `https://elastic.co/guide/en/kibana/current/index.html`.
[10] Jupyter. `http://jupyter.org/`.
[11] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.
[12] D Giordano et al. CERN computing in commercial clouds. *to appear in J. Phys.: Conf. Ser.*, 2016.
[13] D Giordano et al. CERN-IT evaluation of Microsoft Azure cloud iaas. March 2016.
[14] D Giordano et al. Accessing commercial cloud resources within the European Helix Nebula cloud marketplace. *Journal of Physics: Conference Series*, 664(2):022019, 2015.
[15] KVM documentation. `help.ubuntu.com/community/KVM`.
[16] QEMU documentation. `wiki.qemu.org/Main_Page`.