Benchmarking WLCG resources using HEP experiment workloads

Andrea Valassi (CERN IT-DI)
On behalf of the HEPix CPU Benchmarking WG

CHEP2019, 4 November 2019 – Adelaide, Australia
https://indico.cern.ch/event/773049/contributions/3473809
Outline

• Overview: CPU benchmarking in WLCG
  – Why benchmarking?
  – Current approach (HEP-SPEC06) and its limitations

• New approach: benchmarking using HEP experiment workloads
  – Overview, implementation, status
  – Applicability to HPCs and GPUs

• Conclusions
WLCG (Worldwide LHC Computing Grid): a varied computing landscape

- “CPU cores” are not all equivalent to one another (sites are managed independently):
  - Some CPU cores are able to do more “work” than others per unit time (throughput)
  - Some CPU cores are more expensive than others
Why benchmarking CPU resources in WLCG?

Two main use cases for WLCG:

- **Accounting**
  - Experiments *request* "X" CPU resources to do their computing for one year
  - Funding agencies and sites *provision* "X" CPU resources to the experiments
  - Resource review boards *compare the “X” used to the “X” requested*

- **Procurement**
  - Each site buys the CPU resources providing the *best “X” per CHF/EUR/…*

In addition:

- **Scheduling**
- **Software optimizations**
WLCG accounting: current benchmark is **HEP-SPEC06 (HS06)**

Very approximate rule of thumb: 10 HS06 per core (9M HS06 is ~0.9M cores)
HS06 is derived from SPEC CPU2006®

- Standard Performance Evaluation Corporation: industry standard since 1988
- *Real applications (from domains other than HEP), not a synthetic benchmark*
- After evaluating several subsets of SPEC CPU2006, chose the “all_cpp” subset
  - Seven C++ benchmarks (recompiled for HEP) – HS06 score is their geometric mean
  - Execution time: O(4h)

<table>
<thead>
<tr>
<th>Bmk</th>
<th>Int vs Float</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>444.namd</td>
<td>CF</td>
<td>92224 atom simulation of apolipoprotein A-I</td>
</tr>
<tr>
<td>447.dealll</td>
<td>CF</td>
<td>Numerical Solution of Partial Differential Equations using the Adaptive Finite Element Method</td>
</tr>
<tr>
<td>450.soplex</td>
<td>CF</td>
<td>Solves a linear program using the Simplex algorithm</td>
</tr>
<tr>
<td>453.povray</td>
<td>CF</td>
<td>A ray-tracer. Ray-tracing is a rendering technique that calculates an image of a scene by simulating the way rays of light travel in the real world</td>
</tr>
<tr>
<td>471.omnetpp</td>
<td>CINT</td>
<td>Discrete event simulation of a large Ethernet network.</td>
</tr>
<tr>
<td>473.astar</td>
<td>CINT</td>
<td>Derived from a portable 2D path-finding library that is used in game's AI</td>
</tr>
<tr>
<td>483.xalancbmk</td>
<td>CINT</td>
<td>XSLT processor for transforming XML documents into HTML, text, or other XML document types</td>
</tr>
</tbody>
</table>
HS06 was chosen because (in 2009) it seemed representative of HEP workloads

- **HS06 showed good correlation to the throughputs of HEP workloads**
  - Throughput (events per second) is the most relevant metric for HEP processing

<table>
<thead>
<tr>
<th>Correlation</th>
<th>Generation</th>
<th>Simulation</th>
<th>Reconstruction</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atlas</td>
<td>0.9969</td>
<td>0.9963</td>
<td>0.9960</td>
<td>0.9968</td>
</tr>
<tr>
<td>Alice pp MinBias</td>
<td>0.9994</td>
<td>0.9832</td>
<td>0.9988</td>
<td></td>
</tr>
<tr>
<td>Alice PbPb</td>
<td>0.9984</td>
<td>0.9880</td>
<td>0.9996</td>
<td></td>
</tr>
<tr>
<td>LhcB</td>
<td>0.9987</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMS HiggsZZ</td>
<td>0.9982</td>
<td>0.9987</td>
<td>0.9983</td>
<td></td>
</tr>
<tr>
<td>CMS MinBias</td>
<td>0.9982</td>
<td>0.9974</td>
<td>0.9974</td>
<td></td>
</tr>
<tr>
<td>CMS QCD 80 120</td>
<td>0.9988</td>
<td>0.9987</td>
<td>0.9988</td>
<td></td>
</tr>
<tr>
<td>CMS Single Electron</td>
<td>0.9987</td>
<td>0.9942</td>
<td>0.9981</td>
<td></td>
</tr>
<tr>
<td>CMS Single MuMinus</td>
<td>0.9986</td>
<td>0.9926</td>
<td>0.9970</td>
<td></td>
</tr>
<tr>
<td>CMS Single PiMinus</td>
<td>0.9955</td>
<td>0.9693</td>
<td>0.9955</td>
<td></td>
</tr>
<tr>
<td>CMS TTbar</td>
<td>0.9965</td>
<td>0.9589</td>
<td>0.9967</td>
<td></td>
</tr>
</tbody>
</table>

Correlation of HEP-SPEC06 with several kinds of applications and different experiments

- **HS06 showed similar CPU usage patterns to those of HEP workloads**
  - Hardware performance counters (FP+SIMD, Load+Store, Mispredicted Branch)
    - Analysis using perfmon on lxbatch (compute nodes of LHC experiments at CERN)

Similar FLOATING POINT fraction (~10%) for HEP workloads on lxbatch and SPEC2006 “all_cpp” (lower for SPEC INT, higher for SPEC FP)
After 10 years, HS06 does not describe HEP workloads well enough any longer

- **HS06 score shows poor correlation to the throughputs of HEP workloads**
  - Issue reported by ALICE, LHCb – somewhat better agreement for ATLAS, CMS
  - Use of 32-bit benchmark for 64-bit applications explains part of the discrepancy

- **HS06 shows different CPU usage patterns from those of HEP workloads**
  - Hardware performance counters (front-end, back-end, retiring, bad speculation)
    - Analysis using the **Trident** toolkit, similar to that done with perfmon in the past

---

[Plot by M. Alef]

D. Giordano, CHEP2018, Sofia, July 2018

D. Giordano, WLCG GDB, May 2019
SPEC CPU 2017 has been evaluated, but it has the same issues as HS06

- **SC17 score shows poor correlation to the throughputs of HEP workloads**
  - Because it is highly correlated, i.e. essentially equivalent, to the HS06 score

- **SC17 shows different CPU usage patterns from those of HEP workloads**
  - Whereas it has very similar CPU usage patterns to those of HS06
We are developing an alternative solution: benchmarking CPUs using HEP workloads

- Why did we choose HS06? Because it seemed representative of HEP WLs!
  - Score correlated to HEP WL throughput, CPU usage similar to HEP WLs

- By construction, using HEP workloads directly is guaranteed to give
  - A score with high correlation to the throughputs of HEP workloads
  - A similar CPU usage pattern to that of HEP workloads

It seems obvious… why did we not do this before?

What allows us to do this now is the availability of container technology!
**“HEP benchmarks” project overview**

- Three repositories under [https://gitlab.cern.ch/hep-benchmarks](https://gitlab.cern.ch/hep-benchmarks):

  - **hep-workloads**
    - runs a single HEP workload
    - includes common and WL-specific infrastructure to build WL containers
    - most active package so far

  - **hep-score**
    - runs several HEP workloads
    - average/combine individual scores to give a single benchmark number

  - **hep-benchmark-suite**
    - runs several benchmarks (hep-score, HS06 and others)
    - collects results in a database

- **Project organization**
  - Team: core development and infrastructure, testing, experiment experts
  - Track work progress via [Jira Project](https://hep-help.cern.ch/jira) and [Twiki](https://twiki.cern.ch/twiki/bin/view/HEP/hep-Site)
From HEP reference workloads to containers: the hep-workloads project

- **Main requirements:**
  - Self-contained (no network), easy to use, fast/small, stable/reproducible...

- **One workload ↔ One standalone Docker container (with all dependencies)**
  - Operating system
  - Input data (event and conditions data)
  - Experiment-specific software (on cvmfs)
  - Orchestrator script (benchmark driver)
    - Sets environment
    - Runs application (many copies)
      - Each copy may be multi-process/threaded
    - Parses output to generate WL score (json)

Docker WL images are made up of layers
Workload containers are built in the hep-workloads gitlab CI

- **Main idea:** experiment software is on \( /\text{cvmfs} \), discover what is needed in a dry run
- **Enabling technology:** cvmfs tracing mechanism

Starting from gitlab repo containing only CI and WL orchestrator scripts:
1. Build interim Docker image: \( /\text{cvmfs} \) is the standard network-connected service
2. Run WL in interim Docker image: generate cvmfs traces listing which files were accessed
3. Build standalone Docker image: \( /\text{cvmfs} \) is a local folder, copy all relevant files
4. Test standalone Docker image (both in Docker and Singularity)
5. Push standalone Docker to gitlab registry
The hep-workloads container registry: available images

- The following images can currently be downloaded and tested:
  - GEN and SIM workloads are available for all four LHC experiments
  - DIGI and RECO workloads are available for CMS (work in progress for ATLAS)
  - Available from https://gitlab.cern.ch/hep-benchmarks/hep-workloads/container_registry

- Executing one specific workload benchmark is a one-liner:
  - Example for CMS DIGI, both via Docker and Singularity:
    ```
docker run -v /tmp/results:/results $IMAGE
    singularity run -B /tmp/results:/results docker://$IMAGE
    ```
  - A json summary and detailed logs are then found in /tmp/results on the host system
The hep-score benchmark: many degrees of freedom, one number

- Each HEP workload stresses different components of a computer system
  - Some are I/O intensive, others not; some are vectorized, others not...

- Using a single metric to characterize performance is difficult (and dangerous)
  - But this is what we often need for accounting and/or procurement
  - Presently, HEP score is the geometric mean of a small subset of HEP workloads
    - But the json output also keeps a record of each individual WL score independently!

HEPscore19 prototype
(use the currently most stable and best understood workloads):

- atlas-gen-bmk v1.1
- atlas-sim-bmk v1.0
- cms-gen-sim-bmk v1.0
- cms-digi-bmk v1.0
- cms-reco-bmk v1.0
- lhcb-gen-sim-bmk v0.12
The hep-benchmark-suite toolkit

- A single toolkit to coordinate execution and result collection for several benchmarks
  - Example: execute HS06, SPEC2017 and HEP-SCORE on a set of reference machines
  - Collect results of all benchmarks in a global JSON document and upload it to a database

- Work in progress:
  - Identify a set of reference machines (like the “lxbench” cluster used for HS06 studies)
  - Systematically study the correlation of individual HEP WL’s to one another and to HS06
Outlook: heterogeneous resources (HPCs, GPUs…)

- All of the work on hep-workloads described so far refers to x86 architectures

- WLCG computing is expected to go well beyond x86 in the medium term future
  - Non-x86 HPC supercomputers (ARM, Power9, GPUs…) will probably play a large role

- By and large, the software of the experiments is not yet production-ready for this
  - Porting and validating it (and having the people to do that) is one of the first priorities
  - But our new benchmarks must be ready in time to do the accounting for these resources!

- **Specifically: a HEP workload container involving GPUs is ready and is being tested**
  - CMS event reconstruction, with optional GPU offload of pixel tracking (*Patatrack*)
  - The container build approach described earlier applies also in this case
Conclusions

• After 10 years, HEP-SPEC06 no longer describes well enough HEP workloads

• Our solution: build a new benchmark directly from HEP workload throughputs
  – Enabling technologies: Docker containers and cvmfs tracing mechanism

• Status: individual containers exist for GEN-SIM workloads of all four experiments
  – And for the DIGI and RECO of CMS and ATLAS

• WIP: a first HEP-SCORE prototype built from a subset of those workloads exists
  – It presently defines the new overall benchmark as the geometric mean of individual scores

• WIP: the basic infrastructure to run the WLs on a set of reference machines exists
  – We will then systematically collect the data to compare WL’s to one another and to HS06

• Outlook: can extend the idea and implementation to HPCs and non-x86 resources
  – A container for a workload with optional GPU offload (CMS Patatrack) is being tested

Bonus: HEP workload containers can be useful also beyond benchmarking!
Example: simulate a realistic load on the batch system of a WLCG site

See CHEP 2019 contribution #52 “Managing the CERN batch system with kubernetes”
Backup slides
Validation studies for hep-workloads

• Careful analysis of robustness and accuracy of individual HEP benchmarks
  – Are the results of multiple measures reproducible?
  – How do results change as a function of input parameters (#events, #threads, #copies)
  – Are the results obtained with singularity and docker containers compatible?

• Use the most stable and best understood workloads for the first hep-score prototype!
The hep-workloads output report

JSON document with the essential information

– Configuration parameters
  • #copies, #threads, #events, status
– Benchmark score: **total node throughput**
  • **Events per wall second** (sum over all copies)
    – Or events per CPU second in some cases
  • Details for each application copy
    – Statistics: mean, median, max, min…
– Additional metrics for performance studies:
  • Memory and CPU utilization
– Workload metadata
  • Description, version, checksum
HEP software and computing evolves… so do HEP CPU benchmarks!

1980’s
MIPS (M Instr Per Sec)  
VUPS (VAX units)  
CERN units

1990’s – 2000’s
SI2k (SPEC INT 2000)  
INTEGER benchmarks  
200 MB footprint

2009
HS06 (SPEC CPU 2006 all_cpp)  
INTEGER + FP benchmarks  
1 GB footprint  
32-bit  
x86 servers  
single-threaded/process on multi-core

2019
2 GB footprint (or more)  
64-bit  
multi-threaded, multi-process  
multi-core, many-core  
vectorization (SSE, … AVX512)  
x86 servers, HPCs  
ARM, Power9, GPUs…?

• As time goes by, **WLCG computing is becoming more and more heterogeneous**

• One of the challenges is how to summarize performance using a single number  
  – Unfortunately, this is needed at least for accounting purposes
Docker layers in hep-workloads images

- Docker container images are always made up of layers
  - Translating Docker images to Singularity also keeps this layer structure unchanged
  - From the bottom up, these layers can be cached until the first difference is found

- The hep-workloads CI builds these layers to make them as cacheable as possible
  - The bottom layers contain what is expected to change least often
  - The top layers may change more frequently (across different workloads or versions)
  - Advantage in the CI: faster builds/tests, save storage space (both Docker and Singularity)
    - Advantage for users: faster tests, save storage space (if Docker and Singularity caches are set up)

<table>
<thead>
<tr>
<th>Changes less often: caching more likely</th>
<th>Changes more often: caching less likely</th>
</tr>
</thead>
<tbody>
<tr>
<td>O/S: Scientific Linux CERN 6</td>
<td>O/S add-ons via yum install</td>
</tr>
<tr>
<td>Experiment WL data files: /data</td>
<td></td>
</tr>
<tr>
<td>Experiment WL software: /cvmfs</td>
<td></td>
</tr>
<tr>
<td>Common and WL scripts: /bmk</td>
<td></td>
</tr>
</tbody>
</table>
Redistribution: software and data licensing

- The software stacks of the four LHC experiments are all Open-Sourced
  - With slightly different licenses and copyright clauses

- Data-intensive experiment workload benchmarks (e.g. digi-reco) use Open Data
  - Approved by the relevant experiments

- We are in the process of clarifying the details of how to Open-Source our benchmark suite, while allowing its use by all interested parties
Fast benchmarks

• Two fast benchmarks have been found useful by the LHC experiments
  – ATLAS KV (KitValidation)
    • Essentially GEANT4 detector simulation (default: 100 single muon events)
    • Agreement with ATLAS and CMS simulation (M. Alef et al, Proc. CHEP2016)
  – DIRAC Benchmark 2012 (DB12)
    • Agreement with ALICE and LHCb (when DB12 runs at job time)

• Not robust enough to replace long-running benchmarks
  – DB12 dominated by the front-end call and branch prediction unit
  – SMT is not beneficial at all for DB12 when loading all CPU threads
  – KV shortness and event simplicity affected by systematics
    • Found in the performance studies of Meltdown/Spectre patches
    • Can be improved extending the test duration and event complexity

M. Guerri, CERN ITTF, April 2017
D. Giordano, CHEP2018, Sofia, July 2018