

HEP Workload Benchmarks: Design/Development

Chris Hollowell <hollowec@bnl.gov>, Andrea Valassi <andrea.valassi@cern.ch>

On behalf of the HEPiX Benchmarking Working Group

HEPiX Fall 2019 – Nikhef Amsterdam, Netherlands



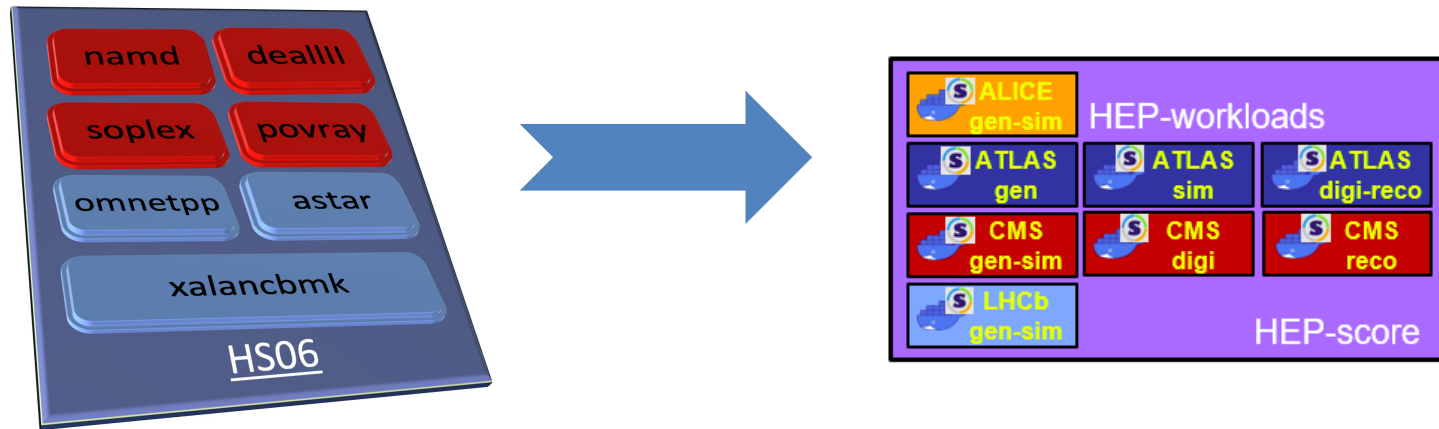
Scientific Data and
Computing Center



HEP Workloads: A Potential Alternative to HS06

The HEPiX Benchmarking Working Group has started the process of creating an alternative to HEP SPEC06

Based on containers from the HEP Workloads project



By using HEP workload benchmarks directly, we're guaranteed to have a score which is highly correlated to the throughput of actual HEP workloads

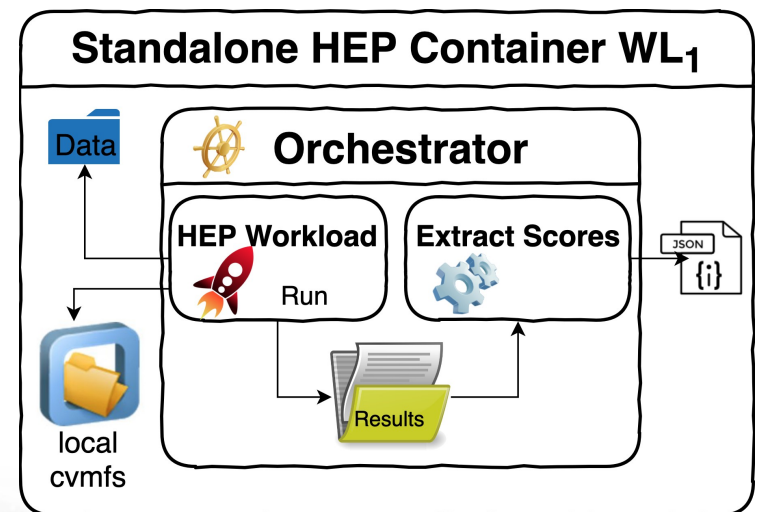
HEP Workloads Containers

Encapsulation: for each HEP workload, build a standalone benchmark container

- Portable and self-contained (no need for network connectivity)
- As small as possible (include all dependencies needed to run the workload, and only those)
- Always do the same thing (results should be as reproducible as possible)

Components of each HEP workload container

- Input event and conditions data
- OS
- HEP application software from /cvmfs
- A benchmark driver (orchestrator)
 - Runs multiple copies of the workload
 - Support for multi-threaded and multi-process applications
 - Parses the output, and generates a summary JSON file with scores



HEP Workloads: CI & Container Registry

Individual HEP workload container images are built and distributed via CERN's Gitlab

- <https://gitlab.cern.ch/hep-benchmarks/hep-workloads>
- The **Gitlab CI** (continuous integration) builds and tests new images on commit
- If the build is successful, the CI pushes these (versioned) images to the **Gitlab Registry**

The images are built as Docker containers

But they can be executed both via **Docker** and **Singularity**

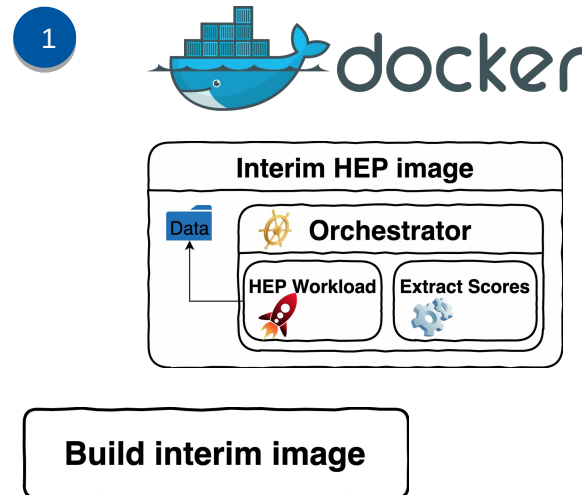
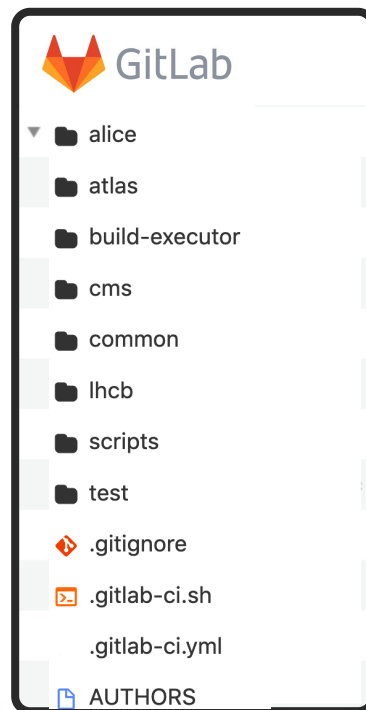
```
$ docker run -v /my_host_path:/results $IMAGE  
$ singularity run -B /my_host_path:/results docker://$IMAGE
```

A JSON summary and detailed logs are then found in /my_host_path on the host system

HEP Workloads: CI Container Build Procedure

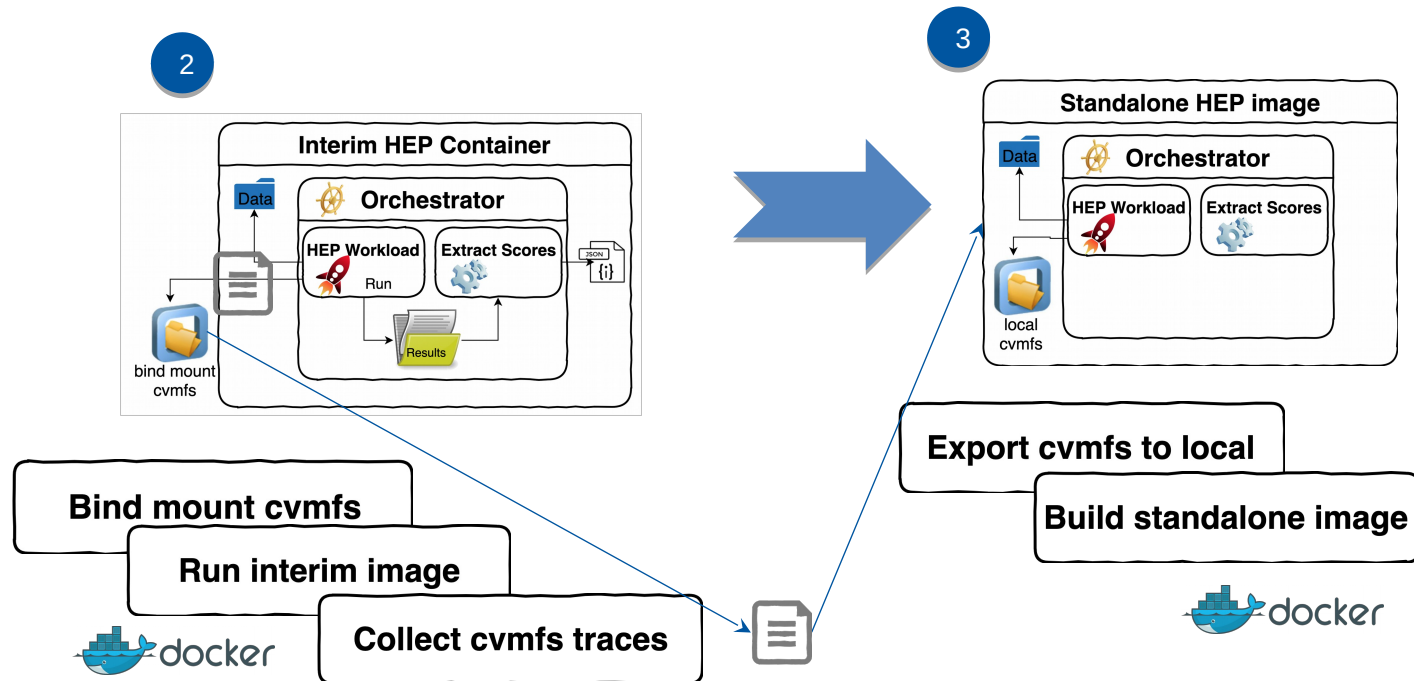
Main idea: experiment software is on /cvmfs, discover what is needed in a dry run

- Enabling technology: cvmfs tracing mechanism



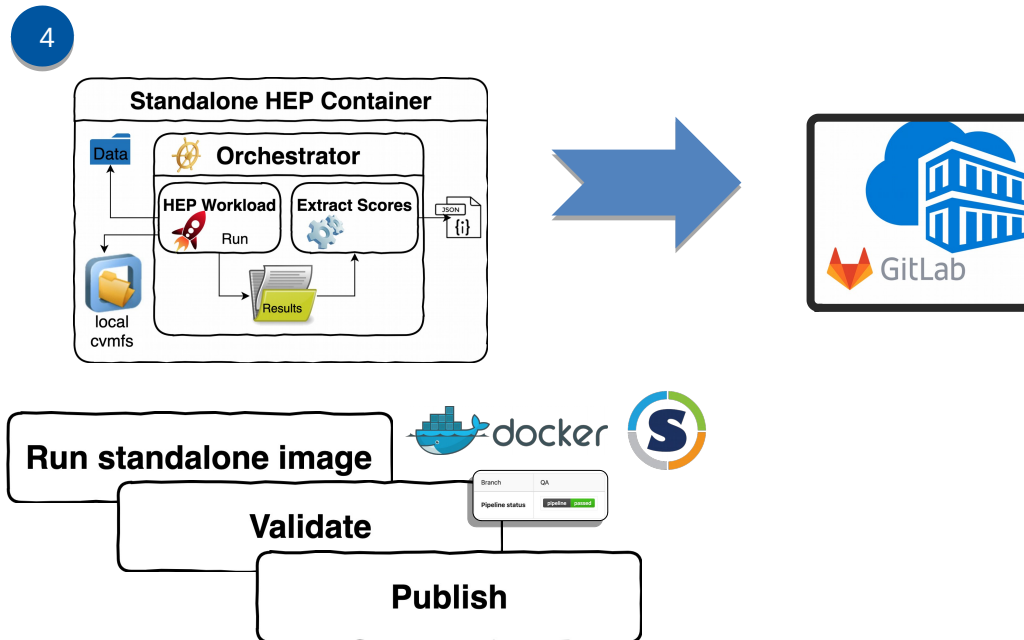
- Starting from a Gitlab repo containing only the CI and WL orchestrator scripts:
1. Build an interim image, where /cvmfs is the standard network-connected service

HEP Workloads: CI Container Build Procedure (Cont.)



2. Run the WL from that image, generating cvmfs traces listing which files were accessed
3. Build the final standalone image, where /cvmfs is a local folder, copying all relevant files

HEP Workloads: CI Container Build Procedure (Cont.)



4. Test the WL from that image (both in Docker and Singularity), push it to the Gitlab registry

Similar process could potentially be utilized to produce non-LHC workload benchmark containers

Can also be used to create benchmark containers for GPU-based software

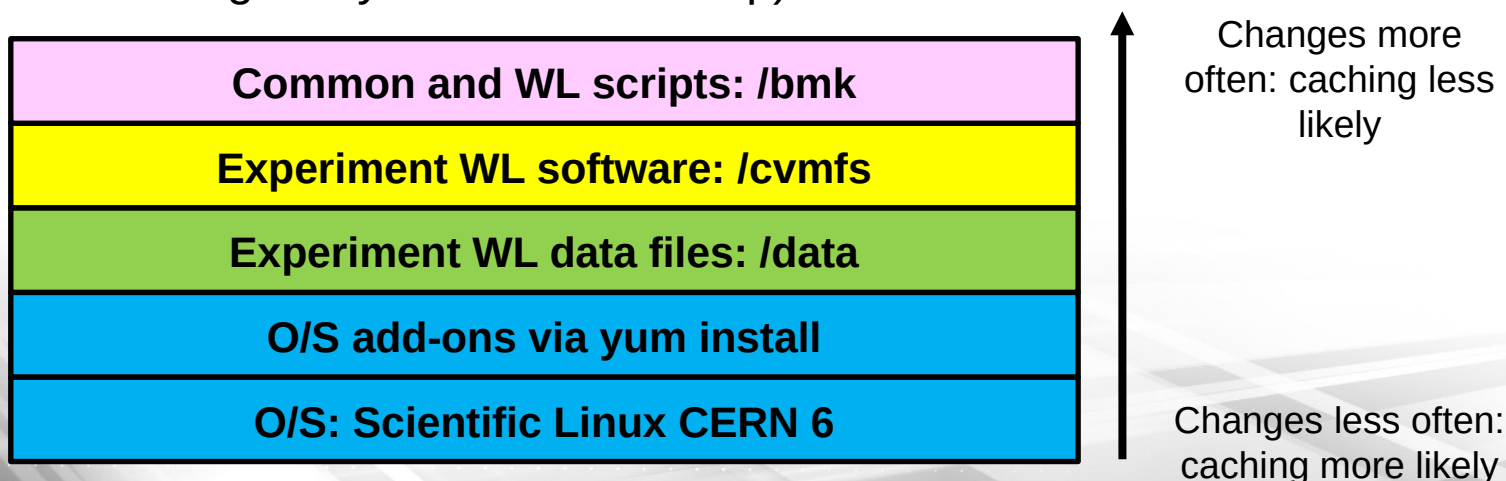
HEP Workloads: Docker Layers

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)



HEP Workloads: Singularity Support

HEP Workloads containers initially developed for use with Docker

Given the popularity of Singularity on HTC/HPC compute hosts, various tests were performed, and it was shown that they were also usable with Singularity's Docker image compatibility functionality



- More recently formalized in the project's CI testing
 - Any changes to the containers must preserve Singularity compatibility

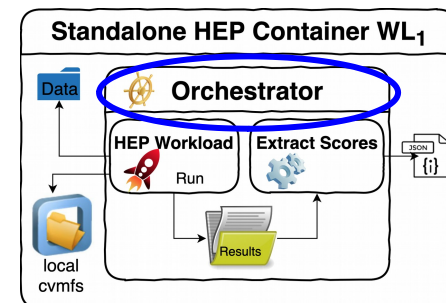
Some testers wanted to try the workload containers on SL6

- Docker is not really usable on SL6
- Use with Singularity requires OverlayFS support in the kernel
 - OverlayFS is not available in SL6
 - Solution: Underlay config option, as implemented in newer (or patched) versions of Singularity

HEP Workloads: Common Driver

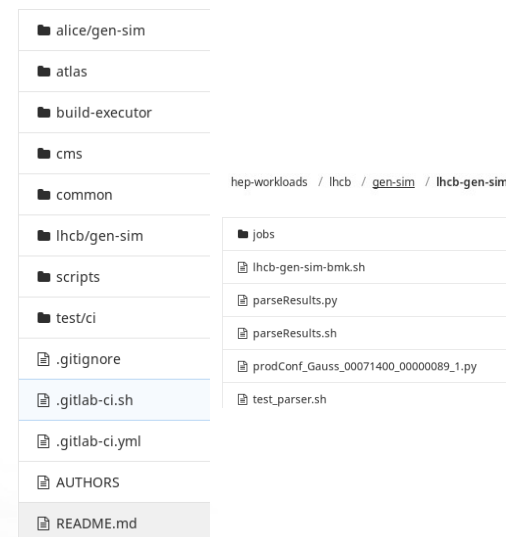
General idea: enforce some commonality between different WLs

- Common naming convention on scripts, images, directories...
- Derive each WL orchestrator from a common bmk driver



Consistent approach within the common bmk driver

- Command line argument interpretation
- Spawn several WL copies and check status code
- Parse results and produce consistent JSON summaries
- Work in progress in finalizing the JSON schema
- Validate JSON summaries (e.g. linting)



HEP Workloads: Output Report

JSON output with 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

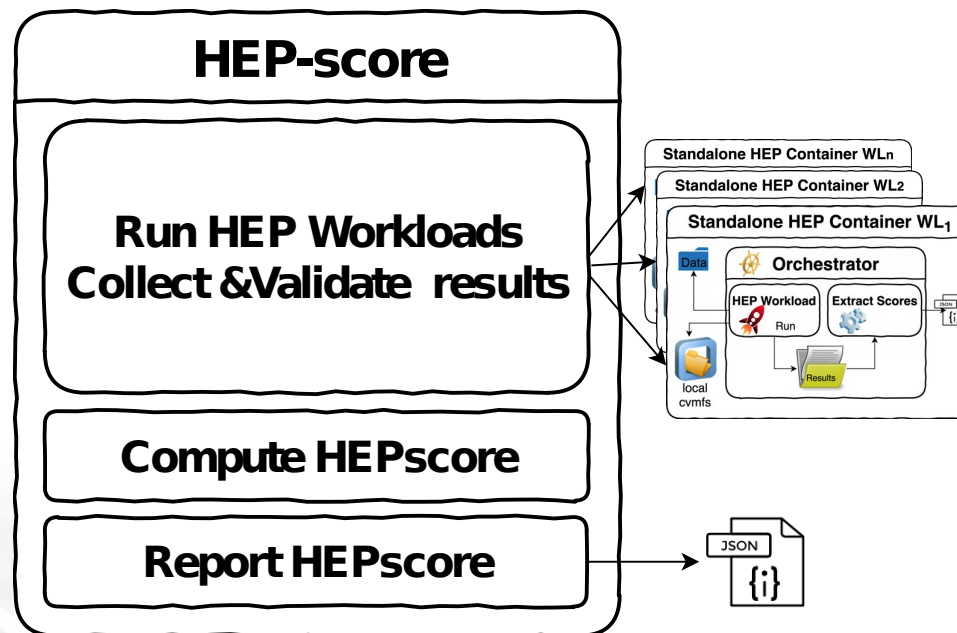
```
"report": {
  "wl-scores": {
    "gen-sim": 0.4438
  },
  "wl-stats": {
    "CPU_score": {
      "max": 0.0226,
      "score": 0.1123,
      "median": 0.0225,
      "avg": 0.0225,
      "min": 0.0222
    },
    "throughput_score": {
      "max": 0.0892,
      "score": 0.4438,
      "median": 0.089,
      "avg": 0.0888,
      "min": 0.088
    }
  },
  "log": "ok",
  "app": {
    "bmkdata_checksum": "e57b3ad19144b7e9574b97056fb35d11",
    "cvdfs_checksum": "b2ab0e3bd4ba1333ebfc7dc49a024536",
    "bmk_checksum": "fc73ae9f18c4ef90791f097cd31b45dc",
    "version": "v1.0",
    "description": "CMS GEN-SIM of ttbar events, based on CMSSW_10_2_9"
  },
  "threads_per_copy": 4,
  "copies": 5,
  "events_per_thread": 100
},
```

HEPscore Application – Overview and Design

In order create a multi-benchmark suite from the different HEP Workloads containers, it was necessary to develop an application which:

1. Orchestrates the sequential execution of the benchmark containers
2. Collects the results
3. Computes a final overall score

Functionality implemented in the “HEPscore” utility



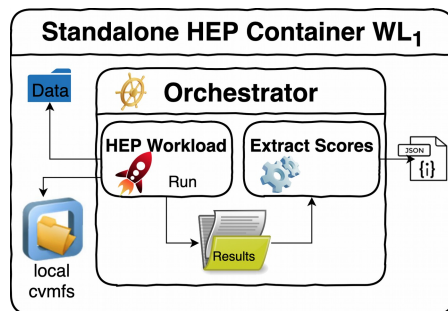
HEPscore Application – Overview and Design (Cont.)

Desired features/requirements

- Decouple the workload applications from the orchestration and overall score computation
 - Allow users to create their own container benchmark suites using custom YAML configuration files
 - Same application could be used to create future versions of the recommended HEPiX/WLCG benchmark
- Develop in Python
- Function with both Singularity and Docker
- Attempt to minimize 3rd party software requirements
 - Ideally, allow users to run without installing any non-stock software on CentOS7/SL7 OS systems (other than Docker or Singularity)
- Traceability of the configuration and workload code versions

HEPscore Application – Overview and Design (Cont.)

- Produce JSON/YAML output
- Include individual benchmark container JSON output data in overall HEPscore JSON output, so that overall score can be validated



```
"report": {
  "wl-scores": {
    "gen-sim": 0.4438
  },
  "wl-stats": {
    "CPU_score": {
      "max": 0.0226,
      "score": 0.1123,
      "median": 0.0225,
      "avg": 0.0225,
      "min": 0.0222
    },
    "throughput_score": {
      "max": 0.0892,
      "score": 0.4438,
      "median": 0.089,
      "avg": 0.0888,
      "min": 0.088
    }
  },
  "log": "ok",
  "app": {
    "bmkdata_checksum": "e57b3ad19144b7e9574b97056fb35d11",
    "cvmfs_checksum": "b2ab0e3bd4ba1333ebfc7dc49a024536",
    "bmk_checksum": "fc73ae9f18c4ef90791f097cd31b45dc",
    "version": "v1.0",
    "description": "CMS GEN-SIM of ttbar events, based on CMSSW_10_2_9"
  },
  "threads_per_copy": 4,
  "copies": 5,
  "events_per_thread": 100
},
```

HEPscore Application – Status

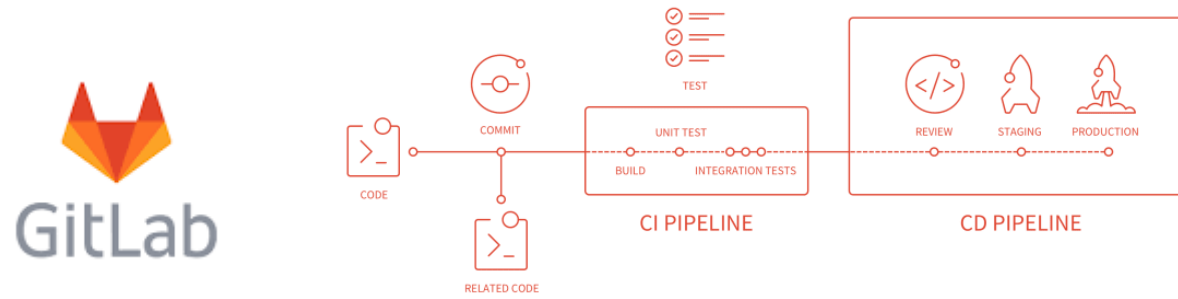
Functioning beta application available for testing

- The application itself is currently a single Python script – simplifies downloading/installation
 - No need to install an RPM, or run installation scripts to execute
- If a configuration file is not passed to the benchmark, the built-in configuration is used
 - Currently, the built-in configuration is for an early prototype “HEPscore19” benchmark

Fully integrated into the HEP Benchmark Suite project

- Simplifies performance comparison with other benchmarks supported by the suite, such as HS06
 - Results searchable in Elasticsearch

HEPscore Application – Status (Cont.)



Various Gitlab continuous integration tests in place

hep-score project in CERN's Gitlab:

<https://gitlab.cern.ch/hep-benchmarks/hep-score>

- After cloning, possible to run “pip install .” to install as “hep-score”, along with all needed dependencies

To download the “hepscore.py” script directly:

<https://gitlab.cern.ch/hep-benchmarks/hep-score/raw/master/hepscore/hepscore.py?inline=false>

HEPscore Application – YAML Configuration

Example HEPscore benchmark YAML configuration:

```
hepscore_benchmark:  
  name: HEPscoreTest  
  version: 0.35  
  repetitions: 3 # number of repetitions of the same benchmark  
  reference_machine: 'Intel Core i5-4590 @ 3.30GHz - 1 Logical Core'  
  method: geometric_mean # how to calculate overall score  
  registry: gitlab-registry.cern.ch/hep-benchmarks/hep-workloads  
  scaling: 10.0  
  benchmarks:  
    atlas-sim-bmk:  
      version: v0.18  
      scorekey: wl-scores  
      ref_scores:  
        sim: 0.0052  
    cms-reco-bmk:  
      version: v0.11  
      scorekey: wl-scores  
      refs_scores:  
        reco: 0.1625  
      events: 2  
      threads: 4  
      copies: 1  
      debug: true
```

HEPscore Application – YAML Configuration (Cont.)

Individual benchmarks specified under the “benchmarks” parameter

- Various flags for the benchmark containers implemented in the config: debug, events, copies and threads

The docker registry used for the benchmark containers is also changeable in the configuration

Allows for multiple executions of the same benchmark

- Median is taken in this case
- Allows for the same behavior as implemented in SPEC CPU2006
 - 3 runs of the same sub-benchmark with median reported
- Helps eliminate anomalous run data

```
benchmarks:  
  cms-reco-bmk:  
    version: v0.11  
    scorekey: wl-scores  
    ref_scores:  
      reco: 0.162  
    events: 2  
    threads: 4  
    copies: 1  
    debug: true
```

```
registry: example.com/reg
```

```
repetitions: 3
```

HEPscore Application – YAML Configuration (Cont.)

“geometric_mean” is supported to compute the overall score for all of the benchmark containers – most suitable function to utilize in this context

```
method: geometric_mean
```

$$\left(\prod_{i=1}^n x_i \right)^{\frac{1}{n}} = \sqrt[n]{x_1 x_2 \cdots x_n}$$

- Standard statistical method to summarize unrelated data
 - Also utilized in HEPSPEC06
- Considering adding a weighted geometric mean function in the future

Implementation of benchmark workload reference scores in the configuration

```
reference_machine: E5-2660
benchmarks:
  atlas-sim-bmk:
    ref_scores:
      sim: 0.162
```

- The score of each workload on the defined reference machine becomes “1”
- Specified via the “reference_machine” and “ref_scores” parameters
- Also similar to SPEC CPU2006

Support for scaling the final overall score

```
scaling: 10.0
```

HEPscore Application – Execution

```
HEPscore Benchmark Execution - Version 0.64
hepscore.py {-s|-d} [-v] [-V] [-y] [-o OUTFILE] [-f CONF] OUTDIR
hepscore.py -h
hepscore.py -p [-f CONF]
Option overview:
-h          Print help information and exit
-v          Display verbose output, including all component benchmark scores
-d          Run benchmark containers in Docker
-s          Run benchmark containers in Singularity
-f          Use specified YAML configuration file (instead of built-in)
-o          Specify an alternate summary output file location
-y          Specify output file should be YAML instead of JSON
-p          Print configuration and exit
-V          Enable debugging output: implies -v
```

Requires the PyYAML module be installed (available in stock CentOS/SL)

The default built-in configuration can be displayed with “-p”

HEP Workloads containers attempt to utilize all logical cores by default

Both Docker (“-d”) and Singularity (“-s”) execution of the containers is supported

OUTDIR must be specified and must exist

- Subdirectory named HEPscore_DATETIME is created underneath and shared with the containers as the “/results” area
- Container stdout/stderr redirected to BMKSUITENAME.log in subdir

HEPscore Application – Execution Example

```
$ cat ./test.yaml
hepscore_benchmark:
  benchmarks:
    atlas-kv-bmk:
      scorekey: wl-scores
      version: cil.1
      ref_scores:
        sim: 1.0
  method: geometric_mean
  name: TestBMK
  reference_machine: Intel Core i5-4590 @ 3.30GHz - 1 Logical Core
  registry: gitlab-registry.cern.ch/hep-benchmarks/hep-workloads
  repetitions: 3
  version: 0.1
$ ./hepscore.py -svyf ./test.yaml /tmp/hepscore
Using custom configuration: ./test.yaml
TestBMK Benchmark
Version: 0.1
System: Linux fedora.localdomain 5.2.11-100.fc29.x86_64 #1 SMP Thu Aug 29 12:52:22 UTC 2019 x86_64
Container Execution: singularity
Registry: gitlab-registry.cern.ch/hep-benchmarks/hep-workloads
Output: /tmp/hepscore/HEPscore_30Sep2019_115348
Date: Mon Sep 30 11:53:48 2019

Executing 3 runs of atlas-kv-bmk
Running ['singularity', 'run', '-B', '/tmp/hepscore/HEPscore_30Sep2019_115348:/results',
'docker://gitlab-registry.cern.ch/hep-benchmarks/hep-workloads/atlas-kv-bmk:cil.1'] ...
1.6129
1.9531
1.996
Median: 1.9531
Final result: 1.9531

$ ls /tmp/hepscore/HEPscore_30Sep2019_115348
atlas-kv-cl-e100-1569858866_7638 atlas-kv-cl-e100-1569859231_8193 TestBMK.yaml
atlas-kv-cl-e100-1569859047_4810 TestBMK.log
```

HEPscore19 Benchmark

A prototype “HEPscore19” benchmark is included as the default configuration in the HEPscore utility

- Presently it runs the following benchmark containers 3 times each (taking the median result) and reports a final score based on the geometric mean:

- 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 above benchmarks were chosen, as from previous studies, they have been shown to be robust and reproducible
- Reference scores in the configuration based on the performance of a single socket Intel Xeon E5-2640 v3 @ 2.60 GHz CPU

HEPscore19 Benchmark (Cont.)

Presently takes ~16+ hours to complete (with each benchmark run 3 times) on modern hardware

- We are studying systematics to reduce the number of events processed per benchmark

Ideally we would include at least one benchmark for each LHC experiment

Additional discussion, work and testing necessary to determine the final set of benchmarks to include in HEPscore19

Conclusions

The HEP Workloads project has developed benchmarks based on software from the LHC experiments

- <https://gitlab.cern.ch/hep-benchmarks/hep-workloads>
- Enabling technologies: containers and CVMFS tracing mechanism

A beta version of the HEPscore benchmark utility is available for testing

- <https://gitlab.cern.ch/hep-benchmarks/hep-score>
- Most of the desired functionality has been implemented

The HEPscore application includes a built-in configuration for an early prototype multi-workload “HEPscore19” benchmark

- Being evaluated/tuned as a potential alternative to HEP SPEC06
- Final set of benchmark containers to include under discussion

HEPscore has been integrated into the HEP Benchmark Suite

- <https://gitlab.cern.ch/hep-benchmarks/hep-benchmark-suite>