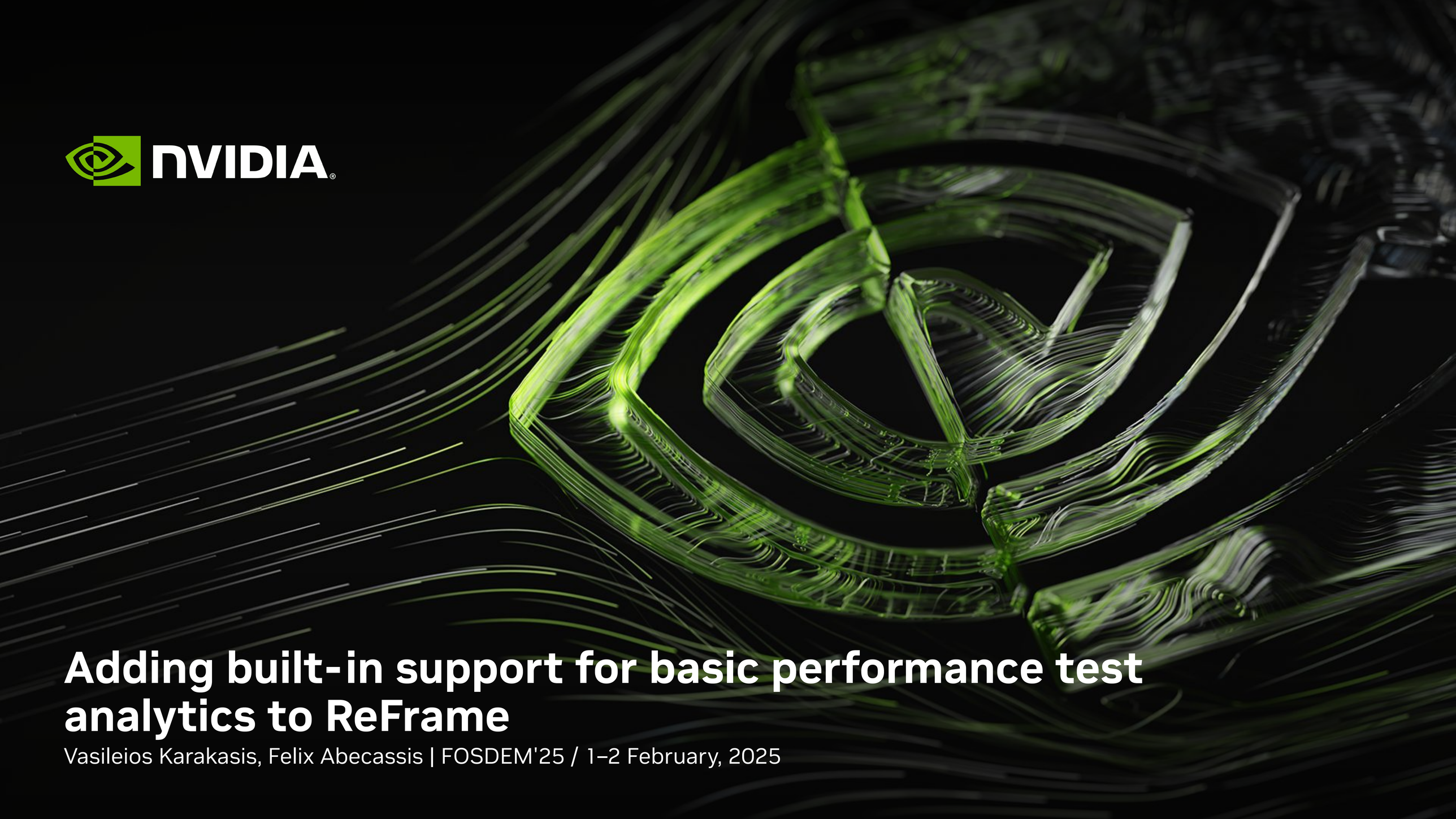


Adding built-in support for basic performance test analytics to ReFrame

Vasileios Karakasis, Felix Abecassis | FOSDEM'25 / 1–2 February, 2025

Applied Systems at NVIDIA

- We bring up the next-gen supercomputers for AI at scale
  - Eos, DGXH100, 2023, #9 in Nov 2023
  - pre-Eos, DGXH100, 2023, #14 in May 2023
  - Selene, DGX A100, #5 in 2020
  - Circe, DGX2H, #61 in 2018
- We enable large scale clusters for internal users and customers.
- We work on new features and advances in the Deep Learning/AI world (e.g. MLPerf, LLMs).

List of large language models

Megatron-Turing NLG	October 2021[28]	Microsoft and Nvidia	530[29]	338.6 billion tokens[29]	38000[30]	Trained for 3 months on over 2000 A100 GPUs on the NVIDIA Selene Supercomputer, for over 3 million GPU-hours.[30]

TOP500 The List.

HOME LISTS STATISTICS RESOURCES ABOUT MEDIA KIT

Home »NVIDIA Corporation »Eos NVIDIA DGX SuperPOD - NVIDIA DGX H100, Xeon Platinum 8480C 56C 3.8...

EOS NVIDIA DGX SUPERPOD - NVIDIA DGX H100, XEON PLATINUM 8480C 56C 3.8GHZ, NVIDIA H100, INFINIBAND NDR400

Site:	NVIDIA Corporation
System URL:	https://www.nvidia.com/en-us/data-center/dgx-superpod/
Manufacturer:	Nvidia
Cores:	485,888
Processor:	Xeon Platinum 8480C 56C 3.8GHz
Interconnect:	Infiniband NDR400
Installation Year:	2023
Performance	
Linpack Performance (Rmax)	121.40 PFlop/s
Theoretical Peak (Rpeak)	188.65 PFlop/s

What is ReFrame?

- An open-source framework originally developed by CSCS for writing system regression and performance tests, primarily targeted (but not limited) to HPC systems. It essentially provides
  - a powerful and expressive syntax built in Python for writing tests in a more declarative manner
    - The tests express only their logic and constraints
    - Tests are composable and extensible
    - Interactions with the system are handled by the framework (batch schedulers, modules systems, etc.)
  - a runtime to run and manage the tests efficiently either locally or on HPC infrastructure
    - Mapping of tests to systems and environments
    - Parallel execution
    - Dependency and resource management
    - Concurrency control
  - integrations for results reporting (local files, Graylog, Elastic) and CI

- Github page: https://github.com/reframe-hpc/reframe
- Documentation: https://reframe-hpc.readthedocs.io/
  - Tutorials
  - Reference pages

Performance testing in ReFrame

Test syntax and logging

- Performance tests contain specially decorated functions that extract figures of merit
- References are defined by a multi-level dictionary
  - First level: System or system/partition combination
  - Second level: the reference tuple
    - *(target_perf, lower_thres, upper_thres, unit)*
- Test will fail if obtained performance for any of the performance variables is outside bounds
- Test performance is logged to different channels
  - Files called *perflogs*
    - Users control the information to be logged
  - Elastic, Graylog
    - ReFrame sends the full test record to the server
    - Users can control the fields to exclude and the format of the record

```python
import reframe as rfm
import reframe.utility.sanity as sn


@rfm.simple_test
class stream_test(rfm.RunOnlyRegressionTest):
    valid_systems = ['*']
    valid_prog_environs = ['*']
    executable = 'stream.x'
    reference = {
        'generic:default': {
            'copy_bw': (23_890, -0.10, 0.30, 'MB/s'),
            'triad_bw': (17_064, -0.05, 0.50, 'MB/s'),
        }
    }

    @sanity_function
    def validate(self):
        return sn.assert_found(r'Solution Validates', self.stdout)

    @performance_function('MB/s')
    def copy_bw(self):
        return sn.extractsingle(r'Copy:\s+(\S+)', self.stdout, 1, float)

    @performance_function('MB/s')
    def triad_bw(self):
        return sn.extractsingle(r'Triad:\s+(\S+)', self.stdout, 1, float)
```

Performance testing in ReFrame

Examples

- Example ReFrame output:

```
706  [ RUN      ] StreamCUDA %gpu=7 /2f2ee9cc @cluster:default+builtin
707  [       OK ] (183/223) StreamCUDA %gpu=7 /2f2ee9cc @cluster:default+builtin
708  [ RUN      ] StreamCUDA %gpu=6 /83ee8ec0 @cluster:default+builtin
709  [       OK ] (184/223) StreamCUDA %gpu=6 /83ee8ec0 @cluster:default+builtin
710  [ RUN      ] StreamCUDA %gpu=5 /9f10935f @cluster:default+builtin
711  [       OK ] (185/223) StreamCUDA %gpu=5 /9f10935f @cluster:default+builtin
712  [ RUN      ] StreamCUDA %gpu=4 /b4f0328d @cluster:default+builtin
713  [       OK ] (186/223) StreamCUDA %gpu=4 /b4f0328d @cluster:default+builtin
714  [ RUN      ] StreamCUDA %gpu=3 /4279215e @cluster:default+builtin
715  [       OK ] (187/223) StreamCUDA %gpu=3 /4279215e @cluster:default+builtin
716  [ RUN      ] StreamCUDA %gpu=2 /7bc9421f @cluster:default+builtin
717  [       OK ] (188/223) StreamCUDA %gpu=2 /7bc9421f @cluster:default+builtin
718  [ RUN      ] StreamCUDA %gpu=1 /5aea23c7 @cluster:default+builtin
719  [       OK ] (189/223) StreamCUDA %gpu=1 /5aea23c7 @cluster:default+builtin
720  [ RUN      ] StreamCUDA %gpu=0 /1cb37dc2 @cluster:default+builtin
721  [       OK ] (190/223) StreamCUDA %gpu=0 /1cb37dc2 @cluster:default+builtin
```

```
787  [       OK ] (223/223) cufftBench %gpu=7 /e4eb2640 @cluster:default+builtin
788  [----------] all spawned checks have finished
789  [  PASSED  ] Ran 223/223 test case(s) from 223 check(s) (0 failure(s), 0 skipped, 0 aborted)
790  [==========] Finished on Sat Jan 25 21:31:54 2025-0800
```

- Perflog example:

```
'format': (
    '%(check_job_completion_time)s|%(check_display_name)s|%(check_perf_var)s|'
    '%(check_perf_value)s|%(check_perf_unit)s|%(check_perf_ref)s|'
    '%(check_perf_lower_thres)s|%(check_perf_upper_thres)s|%(check_result)s'
),
```

```
2025-01-25T21:10:10|StreamCUDA %gpu=0|Copy|■■■■|GB/s|▲▲▲▲|-0.03|null|pass
2025-01-25T21:10:10|StreamCUDA %gpu=0|Scale|■■■■|GB/s|▲▲▲▲|-0.03|0.03|pass
2025-01-25T21:10:10|StreamCUDA %gpu=0|Add|■■■■|GB/s|▲▲▲▲|-0.03|0.03|pass
2025-01-25T21:10:10|StreamCUDA %gpu=0|Triad|■■■■|GB/s|▲▲▲▲|-0.03|0.03|pass
2025-01-25T21:10:10|StreamCUDA %gpu=0|Read|■■■■|GB/s|▲▲▲▲|-0.03|0.03|pass
2025-01-25T21:10:10|StreamCUDA %gpu=0|Write|■■■■|GB/s|▲▲▲▲|0.0|null|pass
```

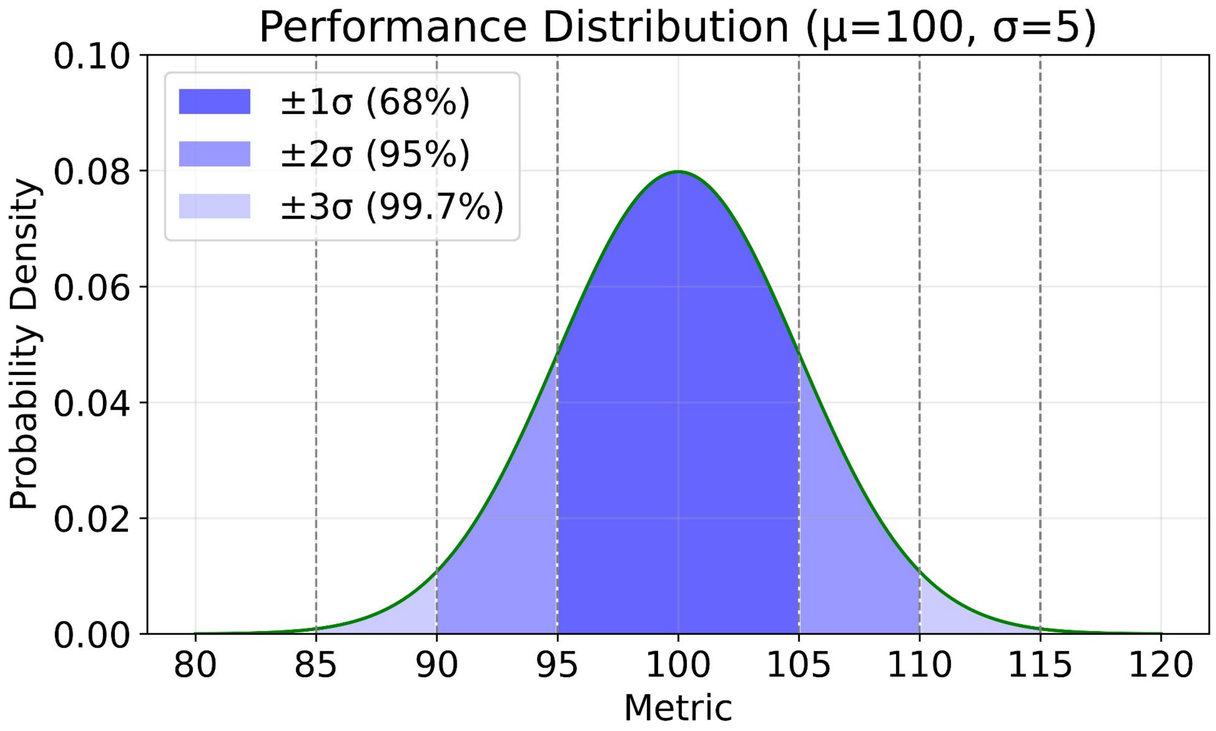
perf value (redacted)

target perf (redacted)

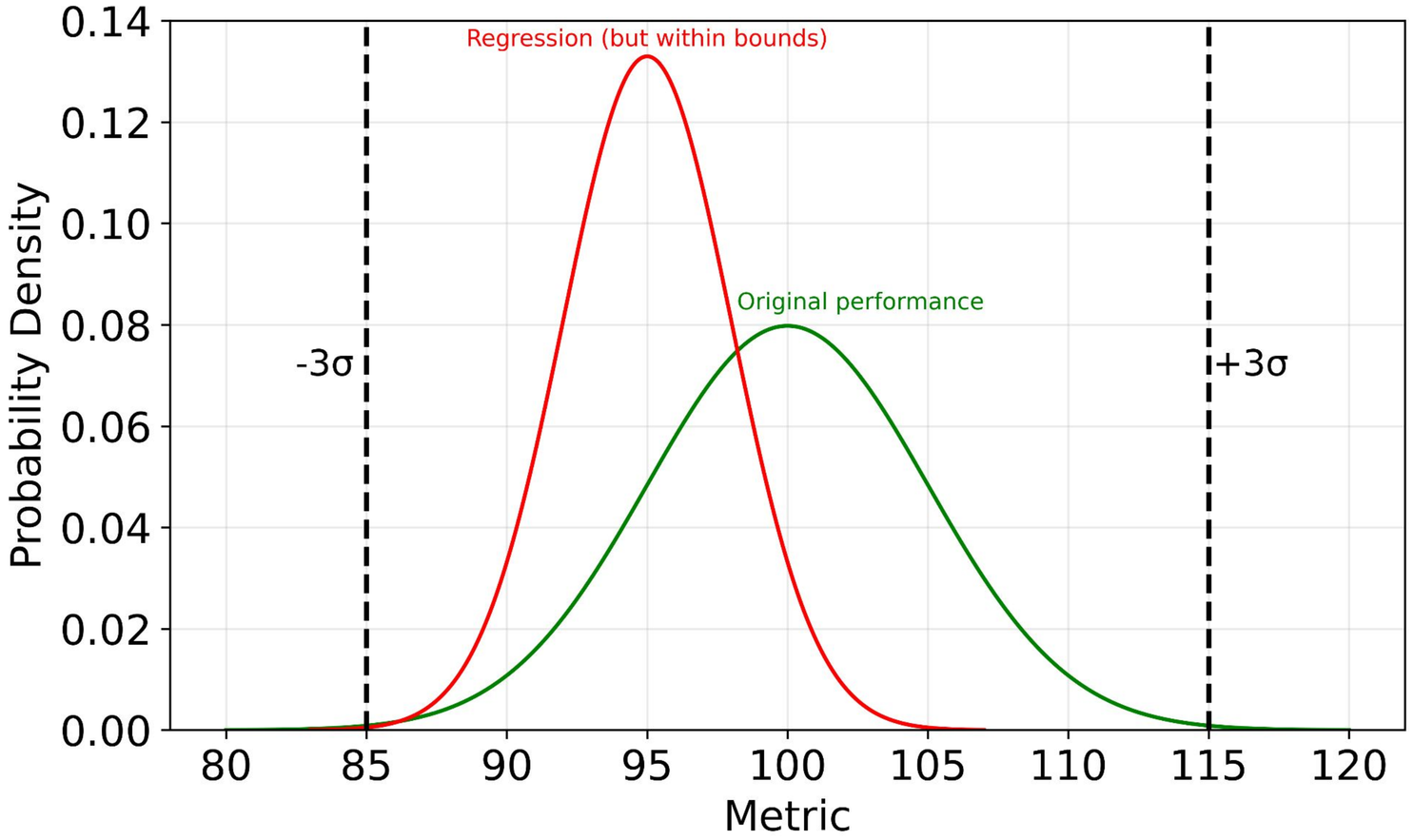
bounds

Deriving performance bounds

- Assuming a normal distribution for the performance metric.
  - **±2σ bounds**: too narrow, spurious failures without an actual regression.
  - **±3σ bounds (or more)**: too large, cannot detect small performance regressions.
- Users of the test suite don’t like false positives, so the bounds tend to grow larger.



Setting bounds ±3σ in ReFrame



Limitations of performance testing in ReFrame

- As cluster owners, we need to validate the performance of a software stack upgrade before deploying it.
  - We need more than fixed performance bounds.
    - Averages, historical trends, comparisons
    - Performance variations within the reference thresholds go undetected
- ReFrame was historically stateless.
  - No way to compare current run with the performance of previous runs

- ReFrame users had to rely on external solutions even for basic analysis.
  - Using external frameworks, such as Splunk, Elasticsearch
  - Using homegrown Pandas script, etc.
  - Usually bound to perflog formatting, which is quite user-specific
  - These solutions are often non-portable and complex to deploy and maintain

Extending ReFrame with performance analytics

Key Goals

- Inspect past test results
- Aggregate test performance across different dimensions
  - Test parameters
  - Nodelists
  - Time periods
  - ...
- Compare performance between runs
  - Current run vs. historical data
  - Runs with different characteristics
  - Runs from different time periods
- Store as much test case information as possible
- Allow external post-processing if needed
- Backward compatible
- Command-line interface

Extending ReFrame with performance analytics

Challenges

Two options considered:

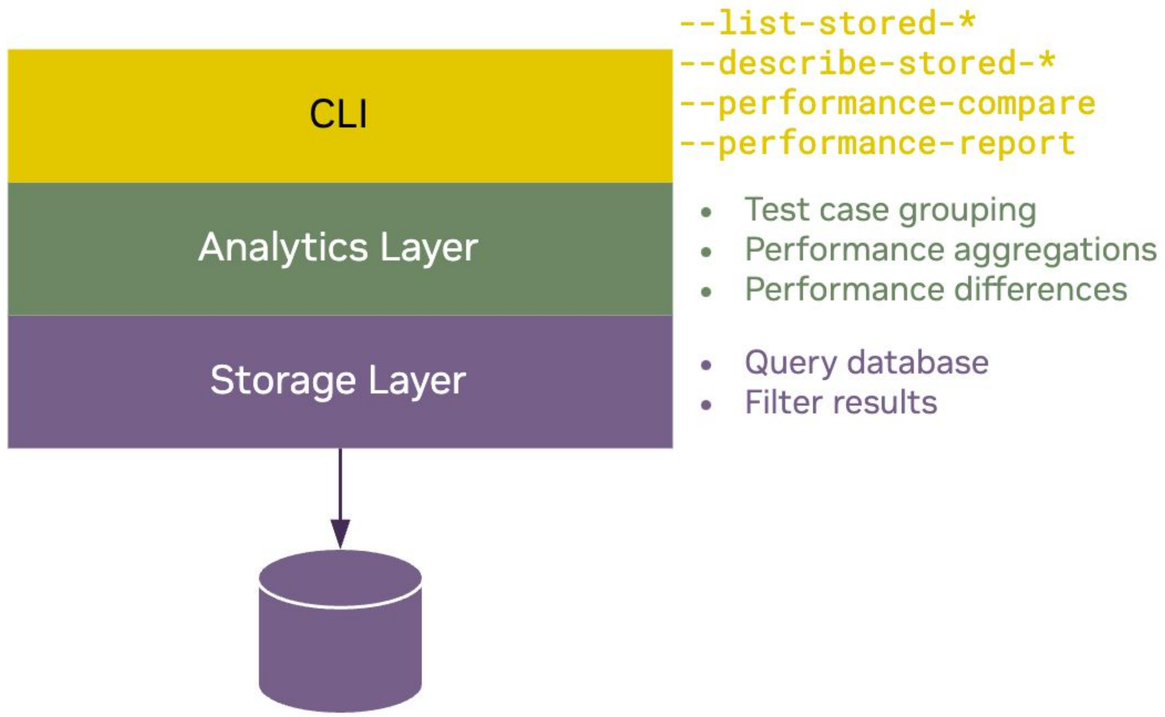
1. Use of perflogs
   - Pros:
     - Simple CSV data format (usually)
     - Compact
   - Cons:
     - Important test information may be lost
     - Information is not context-free (relies on what users deem important to include in the log record)
2. Use of internal JSON report data (see also existing `--report-file` option)
   - Pros:
     - Contains the full session and test case information
     - Information is ReFrame-specific, not user-specific
   - Cons:
     - Much more verbose than perflogs
     - Data is unstructured

We selected option (2) since all test information is valuable and user-independent data format is important.

Extending ReFrame with performance analytics

Design and architecture

- Command-line interface
  - **`--list-stored-{testcases/sessions}`**
    - Presents data in tabular form (by default)
  - **`--describe-stored-{testcases/sessions}`**
    - Returns raw data in JSON
  - **`--performance-compare`**
    - Compares past results
  - **`--performance-report`**
    - Optionally compares current run with past results
  - **`--session-extras`**
    - Extra information to be stored with the current session
  - **`--table-format`**
    - Controls format of tabular data (supports CSV output)
- Analytics Layer
  - Groups test cases
  - Aggregates performance
  - Calculates performance differences
  - Returns tabular or JSON data to upper layer
- Storage Layer
  - Responsible for interacting with the results storage
  - Stores and queries results
  - Filters results
  - Returns raw JSON data to be processed by upper layer

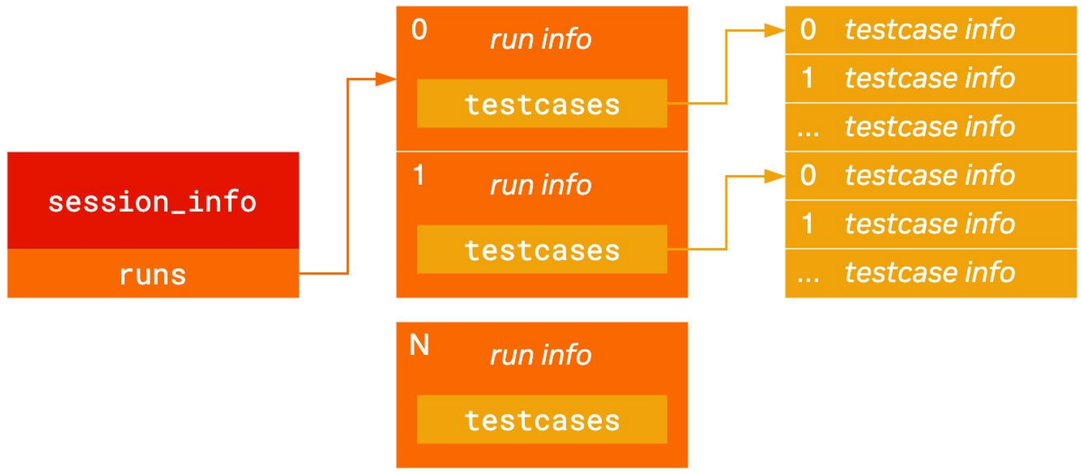


Extending ReFrame with performance analytics

Implementation details

- A report contains a single *session*
- A *session* corresponds to a **`reframe --run`** invocation. It contains:
  - Basic session information:
    - UUID, start/end timestamps, user, hostname, command line, basic statistics etc.
    - Extra user information passed with **`--session-extras`**
  - A session contains one or more *runs*
- A run corresponds to a run cycle of tests in the same session, e.g.,
  - retried tests due to **`--max-retries`**
  - rerun tests due to **`--reruns`** or **`--duration`**
  - A run contains zero or more *testcases*.
- A *testcase* is an instance of a *test* that has executed on a specific *system*, *partition* and *environment* combination. It contains:
  - All test variables and parameters
  - Performance variables with the obtained performance
  - Performance references and thresholds

ReFrame report structure



Extending ReFrame with performance analytics

Implementation details

- We store the full JSON reports in a SQLite database
  - Each report is indexed by its UUID, start and end timestamps.
  - Individual testcases are indexed by their **`name`**, **`system`**, **`partition`**, **`environment`** and **`job_completion_time_unix`**.
  - Each testcase is assigned a pseudo-UUID which has the form: **`<session_uuid>:<run_index>:<testcase_index>`**
    - This contains the exact testcase coordinates in the specific report.
  - We employ file locking to ensure concurrent access to the DB file

- Time-based testcase queries use the index to retrieve the sessions of interest
  - The sessions are decoded and the full testcase info is retrieved
  - Filtering happens on the decoded testcase

- Session queries use the session index to retrieve the sessions of interest
  - For filtering, only **`session_info`** is decoded.

Extending ReFrame with performance analytics

Query syntax I

- The general syntax of past result queries is: **`<select>/<aggregation>/<columns>`**
  - Not all options accept the **`<aggregation>`** and **`<columns>`** specs
  - The **`--performance-compare`** options requires two **`<select>`** specs

- The **`<select>`** spec defines which results to select:
  - Timestamp form: **`20240125:20240131`**
  - Timestamp form with abbreviations: **`now-7d:now`**
  - Session UUID form: **`eba49e9c-81f2-45b7-8680-34a5c9e08ac2`**
  - Session properties: **`'?driver_version=="570.26" and hostname=="nid0001"'`**
    - Any from the predefined or user-specified properties passed with **`--session-extras`** can be queried
    - Any valid Python expression on session properties is accepted

Extending ReFrame with performance analytics

Query syntax II

- The **`<aggregation>`** spec defines how results will be aggregated: **`<aggr_fn>:<group_by>`**
  - Default grouping is by: **`name`**, **`system`**, **`partition`**, **`environment`**, **`pvar`**, **`punit`**
  - Add more properties to the default group by, e.g.: **`mean:+job_nodelist`**
  - Use a custom grouping, e.g., **`mean:name,pvar,punit`**
  - Available aggregation functions: **`first`**, **`last`**, **`mean`**, **`median`**, **`min`**, **`max`**

- The **`<columns>`** spec defines how the aggregated results will be presented:
  - By default all the grouped properties and the aggregated performance is displayed (along with the performance difference for comparisons)
  - Add more columns to display: **`+jobid+env_vars`**
    - Different property values are joined in a comma-separated list and displayed
  - Use custom column listing: **`name,pvar,pval,punit,psamples`**

- Existing test filtering options can also be used:
  - **`-n|--name`**: filter by test name
  - **`-E|--filter-expr`**: filter by evaluating an expression on test's properties

Extending ReFrame with performance analytics

Examples I

- List the mean performance of a specific benchmark for the last 7 days:
  - `--list-stored-testcases=now-7d:now/mean:/ -n StreamCUDA`

- Assuming a multi-way parameterized benchmark, e.g., `ParamTest %mode=foo %gpu=3`, give me the mean performance across all GPUs for all nodes and all benchmark modes for a specific driver:
  - `--list-stored-testcases='?driver_version=="570.26"'/mean:name,mode,pvar,punit,job_nodelist/+psamples -n ParamTest`

- Compare all benchmark data between two driver versions:
  - `--performance-compare='?driver_version=="570.26"'/'?driver_version=="560.28.03"'/median:/`
  - NB: Assumes `driver_version` has been passed with `--session-extras` during the runs

- Show basic information of all sessions between two timestamps
  - `--list-stored-sessions=20250110T0300:20250110T0500`

- Dump a specific session in JSON:
  - `--describe-stored-sessions=eba49e9c-81f2-45b7-8680-34a5c9e08ac2`
  - You can use `jq` to filter the session info only: `jq .[].session_info`

Extending ReFrame with performance analytics

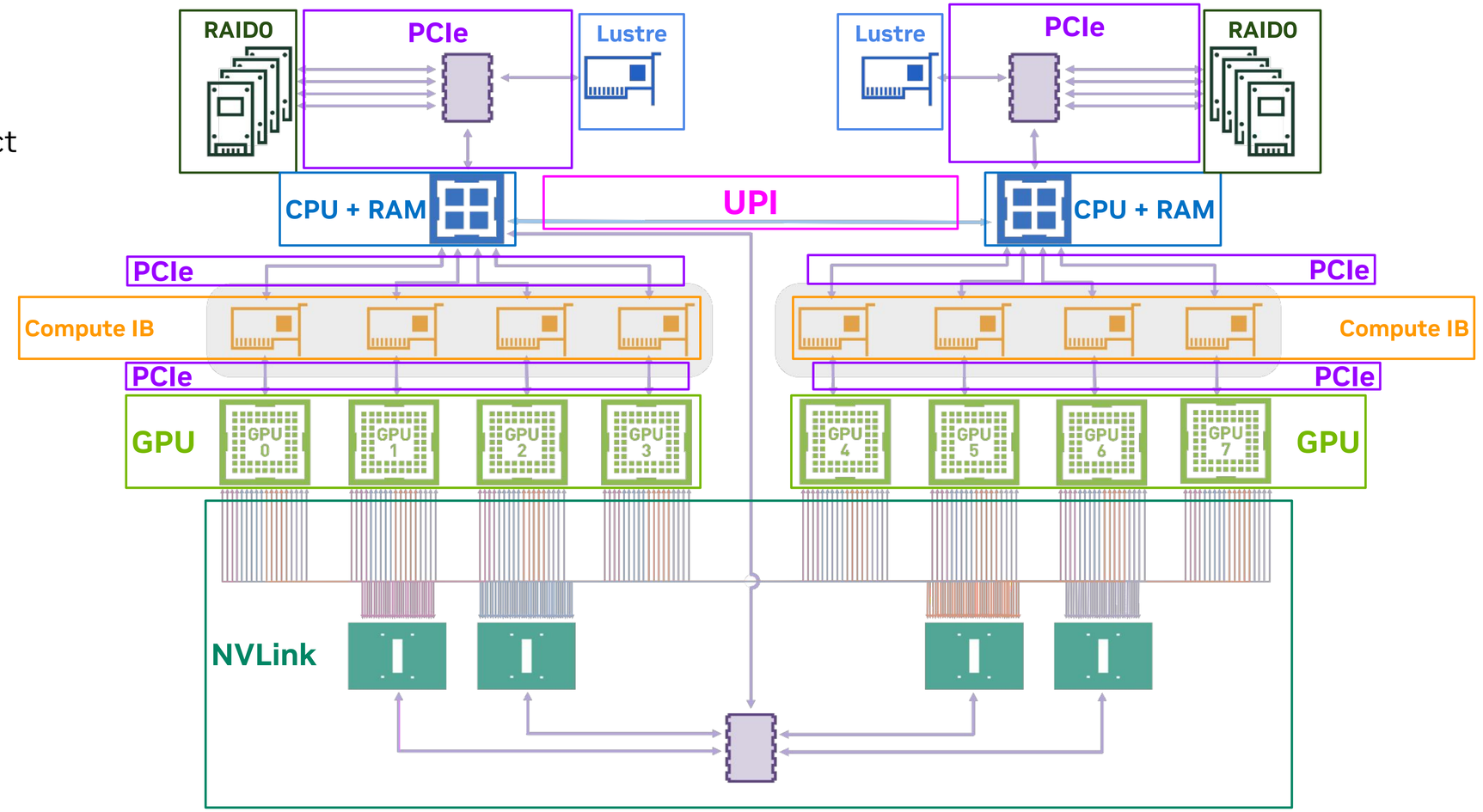
Examples II

- Feature is available in ReFrame >= 4.7
- Enable with **`RFM_ENABLE_RESULTS_STORAGE=y`**
- Optionally set the database file with **`RFM_SQLITE_DB_FILE=/path/to/results.db`**
  - Default: **`$HOME/.reframe/reports/results.db`**

```
karakasis@cluster-abc:~$ reframe --table-format=plain --performance-compare='?"nightly" in tag and "XXX" in gpu_part_no'/'?"nightly" in tag
and "YYY" in gpu_part_no'/mean:name,pvar,punit/ -n FooTestHost

name         pvar              punit      pval_A    pval_B  pdiff

-----------  ----------------  -------  --------  --------  -------

FooTestHost  foo_bandwidth_1   GB/s      41.2099   40.6905  +1.28%

FooTestHost  foo_bandwidth_2   GB/s      46.0236   45.6234  +0.88%

FooTestHost  foo_bandwidth_3   GB/s      50.1933   50.1117  +0.16%

FooTestHost  foo_bandwidth_4   GB/s      55.3926   55.3876  +0.01%

FooTestHost  foo_bandwidth_5   GB/s      52.7173    52.709  +0.02%

FooTestHost  foo_bandwidth_6   GB/s      55.5225   55.5224  +0.00%

FooTestHost  foo_bandwidth_7   GB/s      51.5333   51.5287  +0.01%

FooTestHost  foo_bandwidth_8   GB/s      49.3288   49.6312  -0.61%

FooTestHost  foo_bandwidth_9   GB/s      55.5412   55.5424  -0.00%

FooTestHost  foo_bandwidth_10  GB/s      51.5311   51.5329  -0.00%
```

Using ReFrame on our clusters

- We have to test each hardware component:
  - Each GPU
  - NVLink
  - Each InfiniBand HCA
    - Storage (Lustre)
    - Compute
  - Each CPU + RAM + CPU interconnect
  - Each PCIe link
  - Each NVMe SSD
  - Each network link / switch

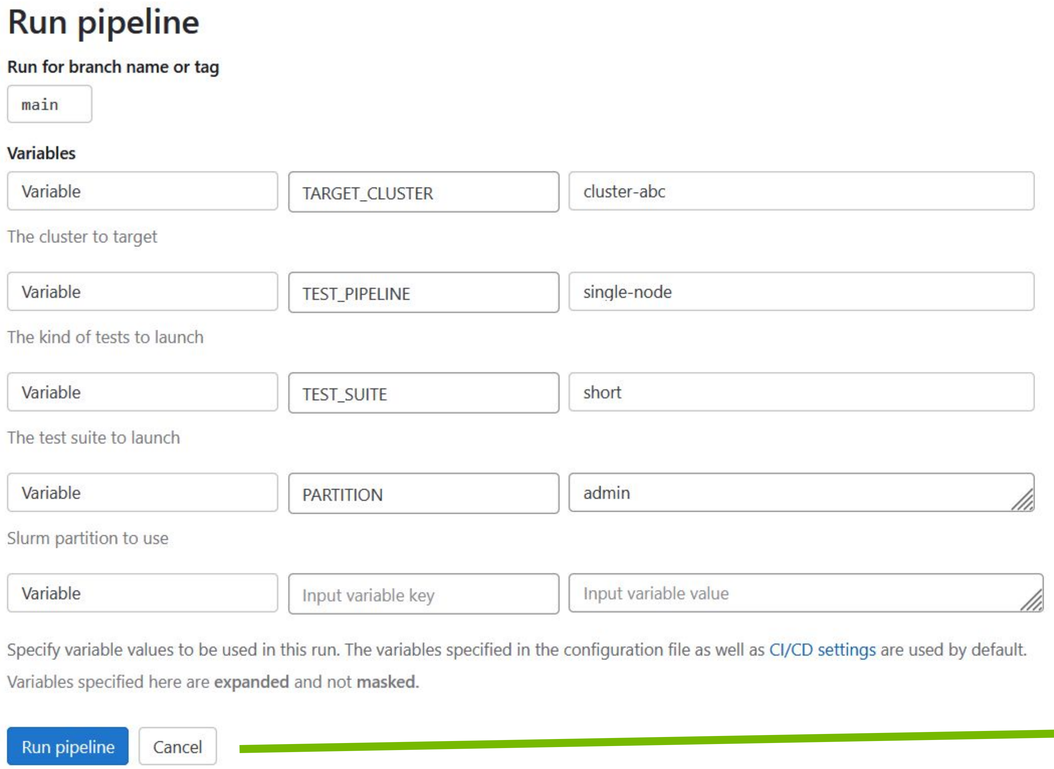


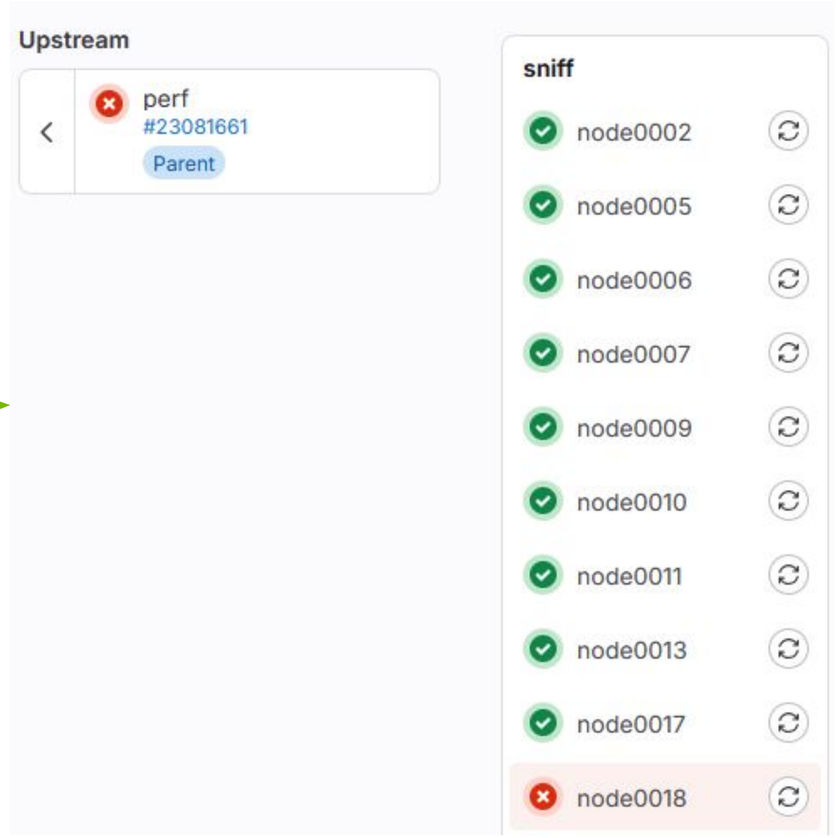
Using ReFrame on our clusters

- **Test setup**
  - Run-only tests using containers launched with Enroot+Pyxis (container runtime) over Slurm

- **Single node ReFrame performance tests:** running automatically on **every node** every few days.
  - NCCL NVLink: https://github.com/NVIDIA/nccl-tests
  - nvbandwidth: https://github.com/NVIDIA/nvbandwidth
  - perftest: https://github.com/linux-rdma/perftest
  - STREAM: https://www.cs.virginia.edu/stream
  - fio: https://github.com/axboe/fio
  - ...
- **Multi-node ReFrame performance tests:** running once a week, or as needed for validation of new software.
  - Distributed Pytorch training: https://github.com/pytorch/pytorch
  - OSU Benchmarks: https://mvapich.cse.ohio-state.edu/benchmarks/
  - NVIDIA NeMo: https://github.com/NVIDIA/NeMo
  - MLPerf Training: https://github.com/mlcommons/training_results_v4.1
  - NVIDIA HPC Benchmarks container (HPL, HPL-MxP, HPCG)
  - NCCL InfiniBand / Multi-Node NVLink

Gitlab CI





- Gitlab CI is the interface used by our cluster admins to launch ReFrame validation and performance checks
- For single node tests, we spawn CI jobs on every available node and launch ReFrame locally collecting useful node information for later queries:
  - Driver version, VBIOS version, GPU and Board part numbers
  - CI pipeline and job IDs
  - CI branch name
  - Test pipeline and test suite type
- We use a single results database per cluster

GitLab CI: inspecting results

Summary

26097 tests | 6 failures | 0 errors | 99.98% success rate

Jobs

Job	Duration	Failed	Errors	Skipped	Passed	Total
node0017	1h 40m 14s	1	0	0	222	223
node0002	1h 39m 45s	0	0	0	223	223
node0005	1h 39m 53s	0	0	0	223	223

/project/admin/fabecassis/gitlab-runner/TbRA22Kr_/13/dcse-appsys/perf/reframe/cpu_test.py

Name	CPU_Test %cpu_node=1 %dtype=fp64 @cluster:singlenode+builtin
Execution time	22.40s
System output	

```
performance: performance error: failed to meet references: mean=1621.21 gflops, expected 3110.2
(l=2954.6899999999996, u=3234.6079999999997)
```

Close

- JUnit report generated by ReFrame with **`--report-junit`**
- Perf reference was:
  - `'mean': (3110.2, -0.05, 0.04, 'gflops'),`

Conclusions & Future Work

- Support for basic performance analytics in ReFrame is a substantial improvement that helps users get insights quickly on their performance data
- It's a feature orthogonal to existing performance logging and does not exclude external processing, rather facilitates it
- Modular design that allows alternative implementations for both the storage and analytics layers

Next steps:

- Collect and present more statistics over results at once (percentiles, mean, stddev etc.)
  - This will allow users to derive quickly performance references and bounds for tests
- Extend session selection syntax to support time periods and property filtering at the same time
  - This will optimize session queries on large databases as it will limit the filtering span
- Improve presentation of results
  - Support of filtering in/out columns for sessions
  - Allow users to name performance columns for A/B testing
- Import existing results (perflogs, reports) to the results DB
- Make the performance comparison feature easily accessible across our teams


NVIDIA