Continuous Benchmarking for Production HPC Centers

AUTOMATING REPRODUCIBILITY, COVERAGE, AND PERFORMANCE REGRESSION ANALYSIS

Parikshit Ardhapurkar
HPC – TECH CDAC
aparikshit@cdac.in

C-DAC

विज्ञान एवं प्रौद्योगिकी मंत्रालय
MINISTRY OF SCIENCE AND TECHNOLOGY
सत्यमेव जयते

NATIONAL SUPERCOMPUTING MISSION
INFRASTRUCTURE | APPLICATIONS | R&D | HRD

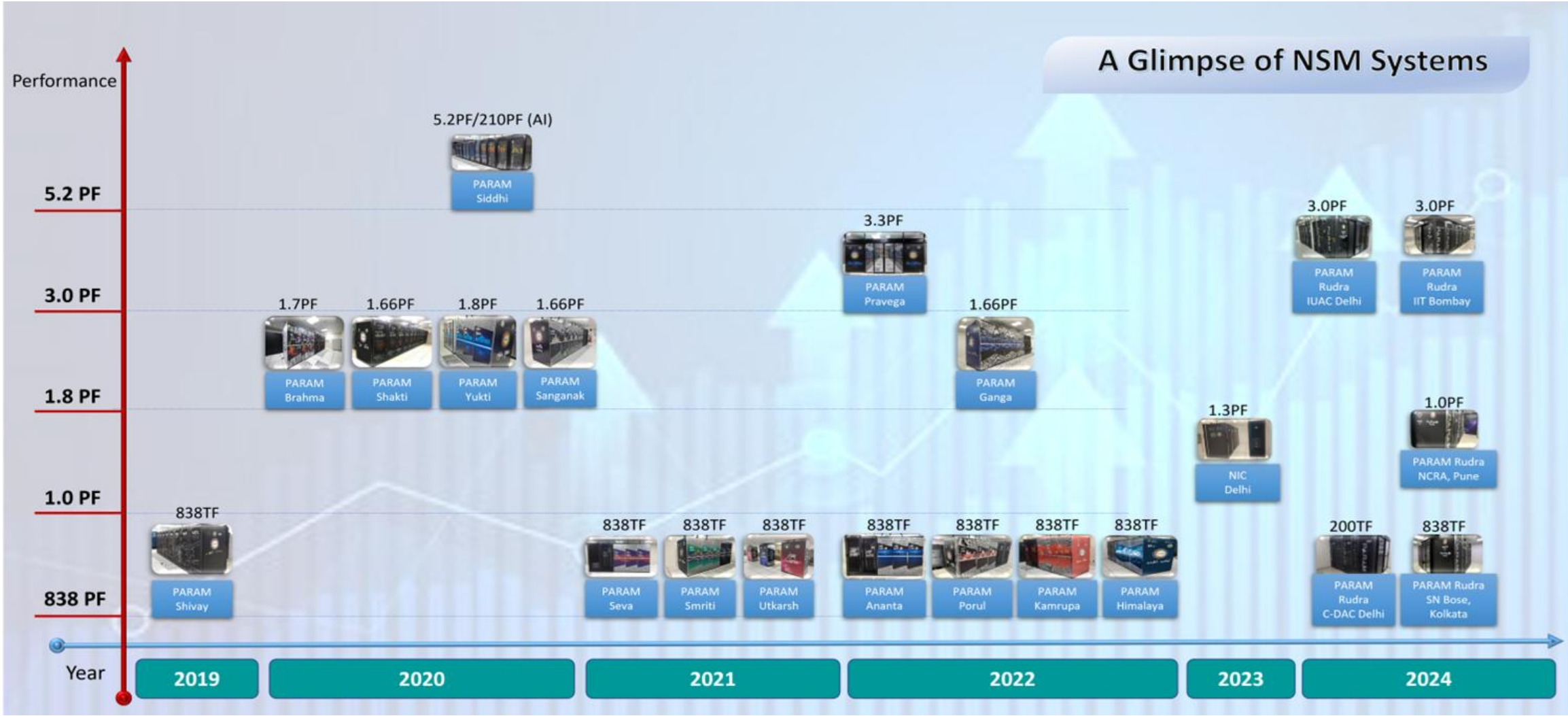
Overview:

- A visionary initiative to strengthen India's **academic and R&D ecosystem** through **high-performance computing (HPC)**.
- Deploying supercomputers across **academic institutions, research organizations, and government bodies**.
- Enables development of applications for **national importance**.
- Focused on **building a skilled HPC workforce** for computational advancements.

Objectives:

- Establish India as a **global leader in supercomputing**.
- Enhance computational power to solve **grand challenge problems**.
- Provide state-of-the-art **HPC infrastructure** for cutting-edge research.
- Optimize investments by **minimizing redundancies** in supercomputing.
- Achieve **self-reliance** in HPC technology & attain **global competitiveness**.

A Glimpse of NSM Systems

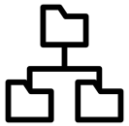


Motivation

Current Practice(Fragile)

Hand-crafted shell script

Manual module load & setup



Ad-hoc results directories

Missing metadata & documentation

VS

What we need (Reproducible)

Automated & standardized workflows

Complete metadata & environment capture

Structured Storage & versioning

Reliable comparison & continuous

Key Challenges

1. **Non-reproducible Environments**
   - Environments setup exists as tribal knowledge
   - Manual module loading settings and undocumented settings
   - Re-running months later with identical conditions is often impossible

2. **Inconsistent Data & Poor Traceability**
   - Results stored in ad hoc directory structures
   - Inconsistent naming conventions
   - Lack of metadata makes comparison meaningless

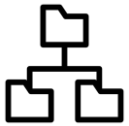
3. **Undetected Regression**
   - Software updates, compiler changes or new MPI libraries may degrade performances
   - No automated flags deviations from baseline

Motivation

Current Practice(Fragile)

Hand-crafted shell script

Manual module load & setup



Ad-hoc results directories

Missing metadata & documentation

VS

What we need (Reproducible)

Automated & standardized workflows

Complete metadata & environment capture

Structured Storage & versioning

Reliable comparison & continuous

Key Challenges

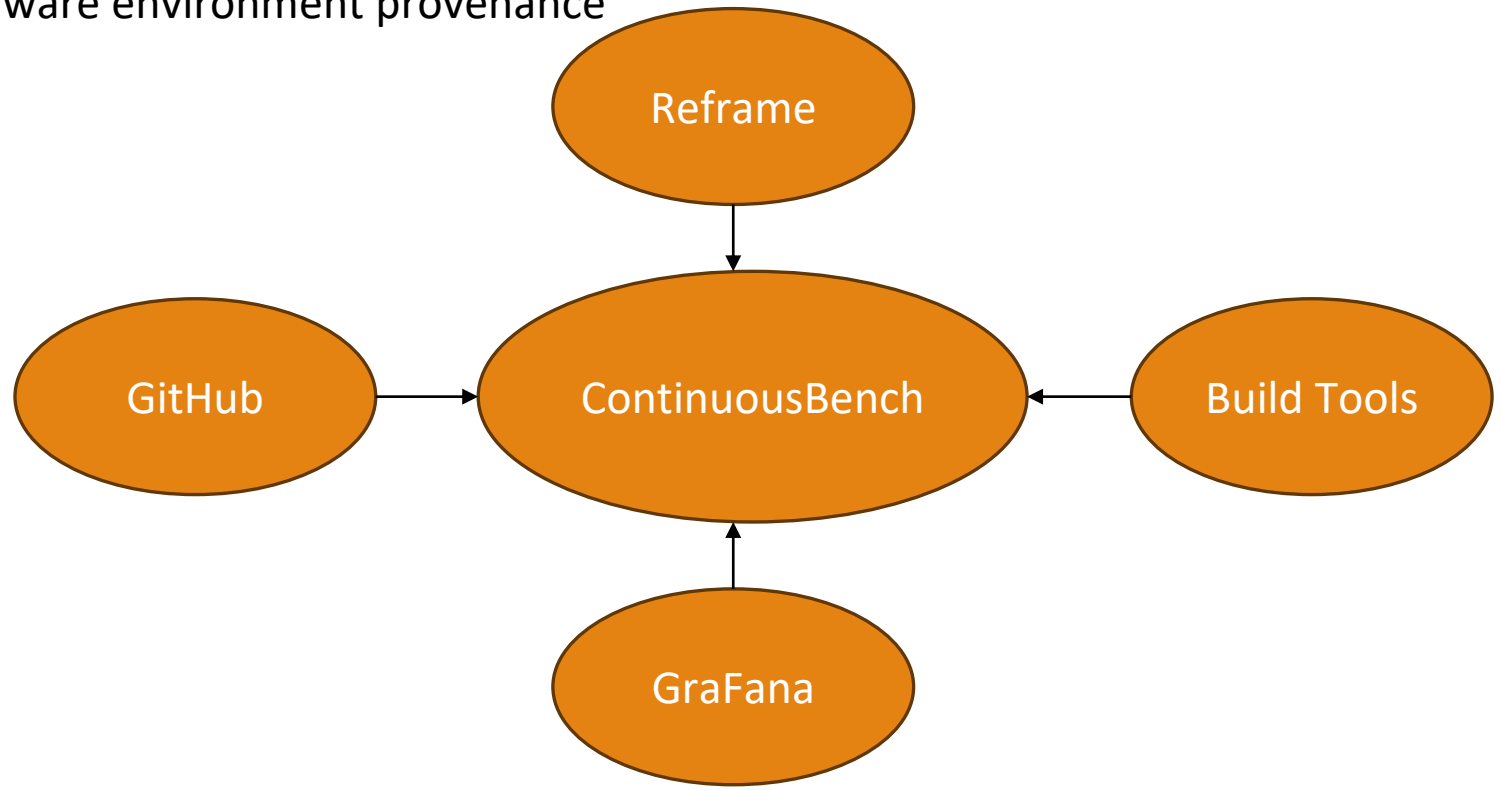
4. **Configuration Errors & Hidden Bias**
   - Misconfigurations introduce systematic biases
   - Issue remain invisible unless detailed metadata is captured

5. **High Manual Efforts & Low Coverage**
   - Manual efforts limit frequency and scope of campaigns
   - Critical period (system commissioning upgrades) has coverage gaps

Introduction

- ContinuousBench, a framework that tightly integrates
  - batch scheduling infrastructure with continuous integration (CI)
  - pipelines to automate benchmark execution
  - capture full software and hardware environment provenance
  - persist results in a structured

- Key Contributions
  - Automated regression detection for shared production environments.
  - Quantitative improvements: 87% reduction in setup time and 4.5× increase in benchmark coverage.
  - Operational deployment on a 64-node CPU/GPU production cluster at C-DAC Pune



Framework Design Goals

Functional Requirements

- **Multi-System & Multi-Partition Support**
  The framework must handle various node types (CPU, GPU, High-Memory) and queue configurations without duplicating code for every system
- **Declarative Specifications**
  Use human-readable, version-controlled YAML for benchmark definitions. This lowers the barrier for administrators who may not be experts in Python
- **Scheduler-Native Integration**
  Use built-in support for schedulers like SLURM to manage job generation, resource limits, and monitoring directly, ensuring consistency across runs
- **Comprehensive Environment Capture**
  Every performance record must include a JSON sidecar capturing the full state: loaded modules, compiler/MPI versions, GPU drivers, and hardware topology
- **Structured Longitudinal Storage**
  Results must be stored in a time-series database (e.g., InfluxDB) to allow for historical reconstruction and regression analysis over months or years

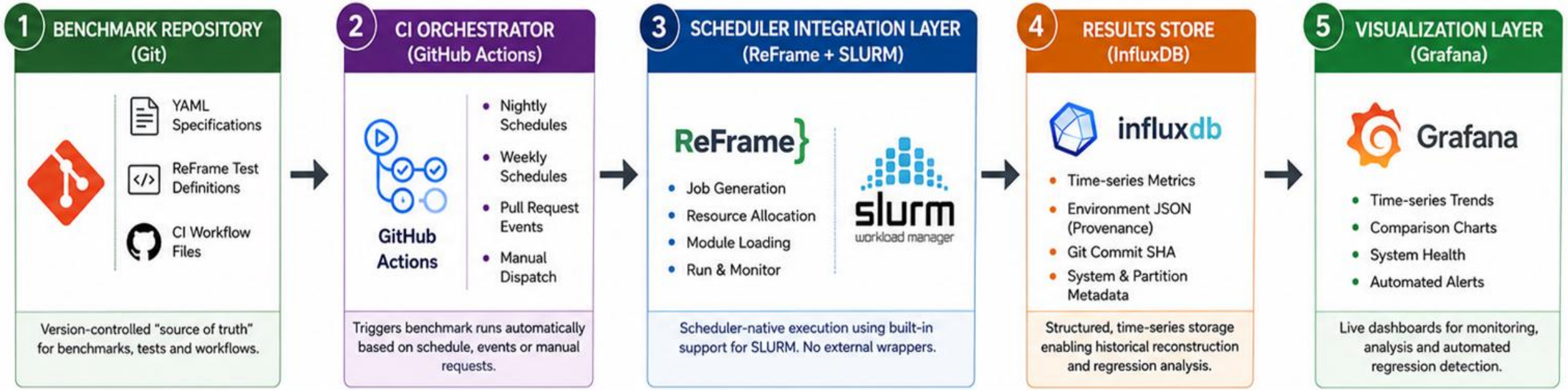
Non-Functional Requirements

- **Reproducibility**
  A stored specification combined with its environment snapshot must allow for re-execution under identical conditions at any future date
- **Portability**
  Deploying at a new facility should only require updating a system-specific configuration file (settings.py), not the framework's core logic
- **Low Operational Overhead**
  Adding a new benchmark should be as simple as editing one YAML file and pushing a commit, without needing to modify CI pipelines
- **Extensibility**
  The architecture must allow for adding new metrics (like energy or carbon intensity) without requiring major changes

Governing Design Principles

- **Scripted & Version-Controlled Workflows:** Every step—from setup to dashboard publication—must be automated and stored in Git.
- **Separation of Concerns:** Keep benchmark semantics (logic), scheduler templates (site policy), and CI logic (orchestration) in independent directories.
- **Explicit Machine-Readable Provenance:** All context required to interpret a result must be captured in structured JSON alongside the measurement.
- **Integrated Post-Processing:** Data parsing, aggregation, and visualization should be automated pipeline stages triggered by job completion, not manual afterthoughts

Architecture



ContinuousBench INTEGRATION LAYER (Our Contribution)

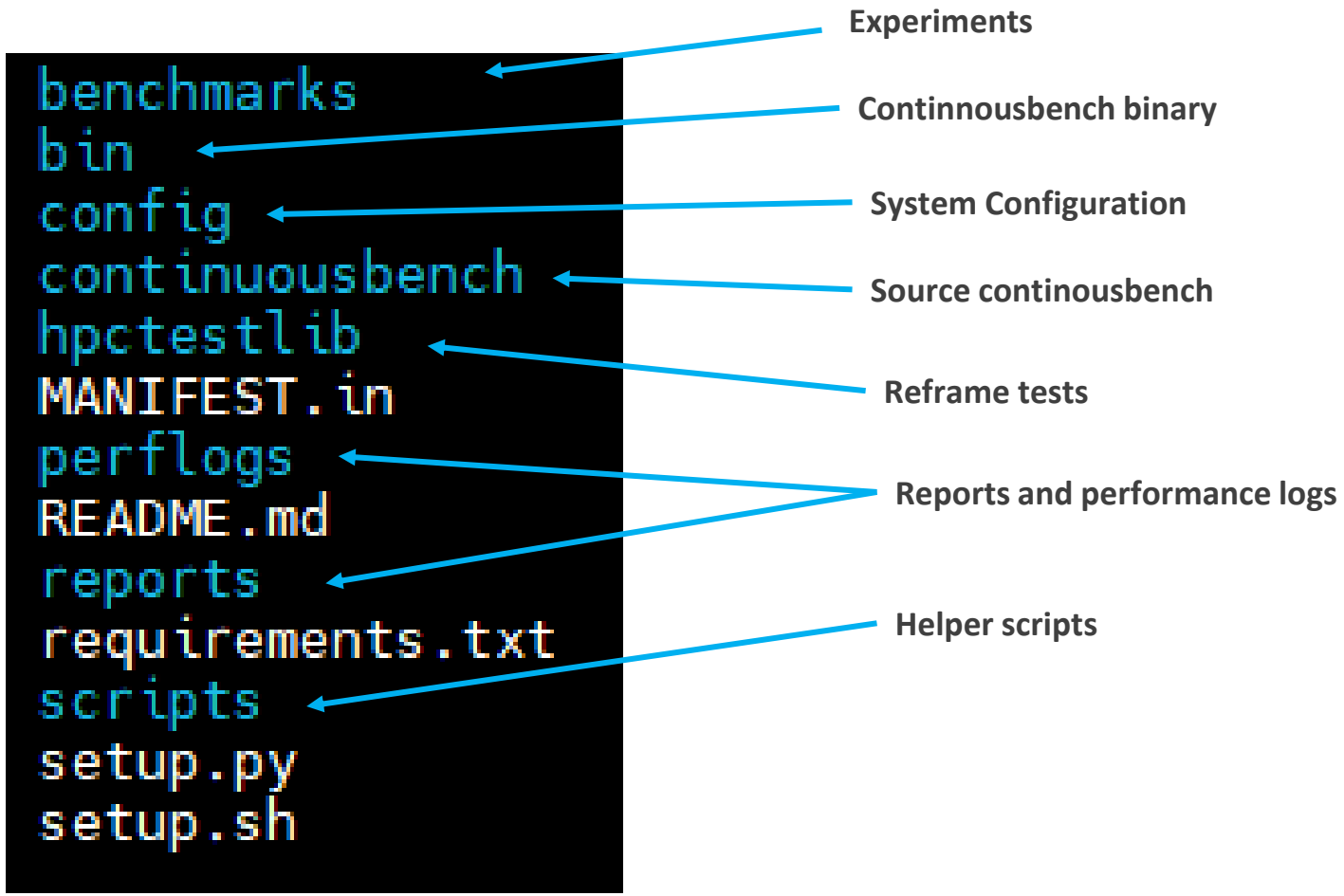
YAML Specification Schema
Human-readable, version-controlled definitions of benchmarks and parameters; independent of underlying Python code.

ReFrame Configuration Layer
settings.py maps abstract requirements (e.g., GPU node) to site-specific SLURM partitions, QOS and policies.

Environment Snapshot Hook (env_snapshot.sh)
Captures the "ground truth" of the execution environment (modules, compiler/MPI, GPU drivers, kernel, topology, etc.) as JSON.

Automated Ingestion Pipeline
Parses performance logs and JSON sidecars after every run and ingests data into InfluxDB automatically.

Architecture: Benchmark Reposiiter

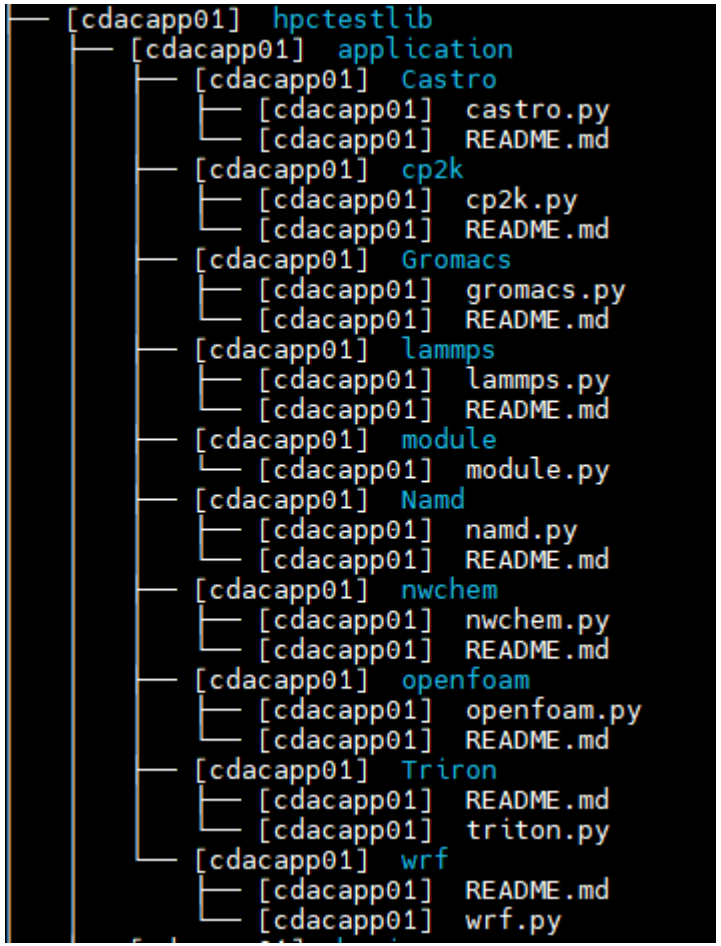


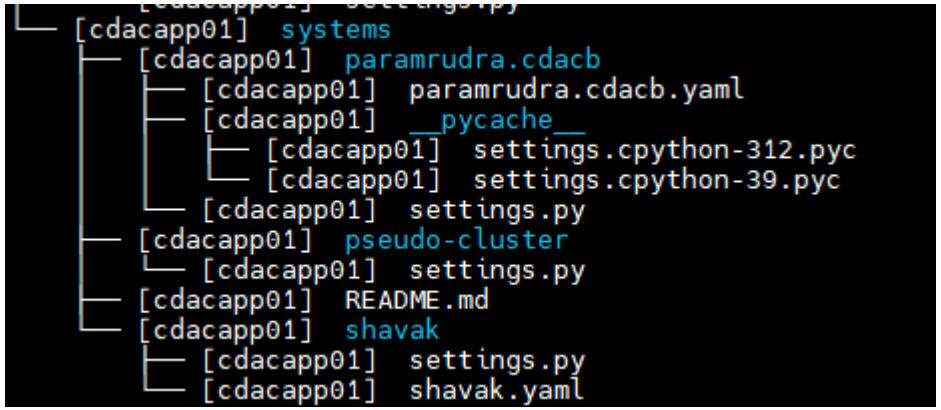
continuousbench/ # Core Continus Bench framework
cli/ # Typer CLI (8 commands)
generators/ # System/experiment/test YAML generators
hooks/ # ReFrame hooks (env_capture, spack_build)
templates/ # ReFrame template base classes
execution/ # Benchmark executor
collectors/ # Result collection and regression detection
reporting/ # Multi-format report engine
validators/ # Benchmark spec validation
utils/ # Config management, YAML helpers

Architecture: CI

```yaml
  on:
    pull_request:
      branches: [main]
    workflow_dispatch:

  jobs:
    smoke-test:
      runs-on: [self-hosted]
      timeout-minutes: 15

      steps:
        - name: Checkout
          uses: actions/checkout@v3

        - name: Install Dependencies
          run: |
            pip install -r requirements.txt
            pip install -e .

        - name: Set ReFrame Environment
          run: |
            export RFM_CONFIG_FILES="${PWD}/config/common/settings.py:${PWD}/config/environments/settings.py:${PWD}/config/pseudo-cluster/settings.py"
            export RFM_CHECK_SEARCH_PATH="${PWD}"
            export RFM_CHECK_SEARCH_RECURSIVE=yes
            export RFM_GENERATE_FILE_REPORTS=true
            echo "RFM_CONFIG_FILES=$RFM_CONFIG_FILES"
            echo "RFM_CHECK_SEARCH_PATH=$RFM_CHECK_SEARCH_PATH"
            reframe --version

        - name: List Smoke Tests
          run: |
            export RFM_CONFIG_FILES="${PWD}/config/common/settings.py:${PWD}/config/environments/settings.py:${PWD}/config/pseudo-cluster/settings.py"
            export RFM_CHECK_SEARCH_PATH="${PWD}"
            export RFM_CHECK_SEARCH_RECURSIVE=yes
            reframe -c . -t smoke -l

        - name: Run Smoke Tests
          run: |
            export RFM_CONFIG_FILES="${PWD}/config/common/settings.py:${PWD}/config/environments/settings.py:${PWD}/config/pseudo-cluster/settings.py"
            export RFM_CHECK_SEARCH_PATH="${PWD}"
```

Architecture: ReFrame

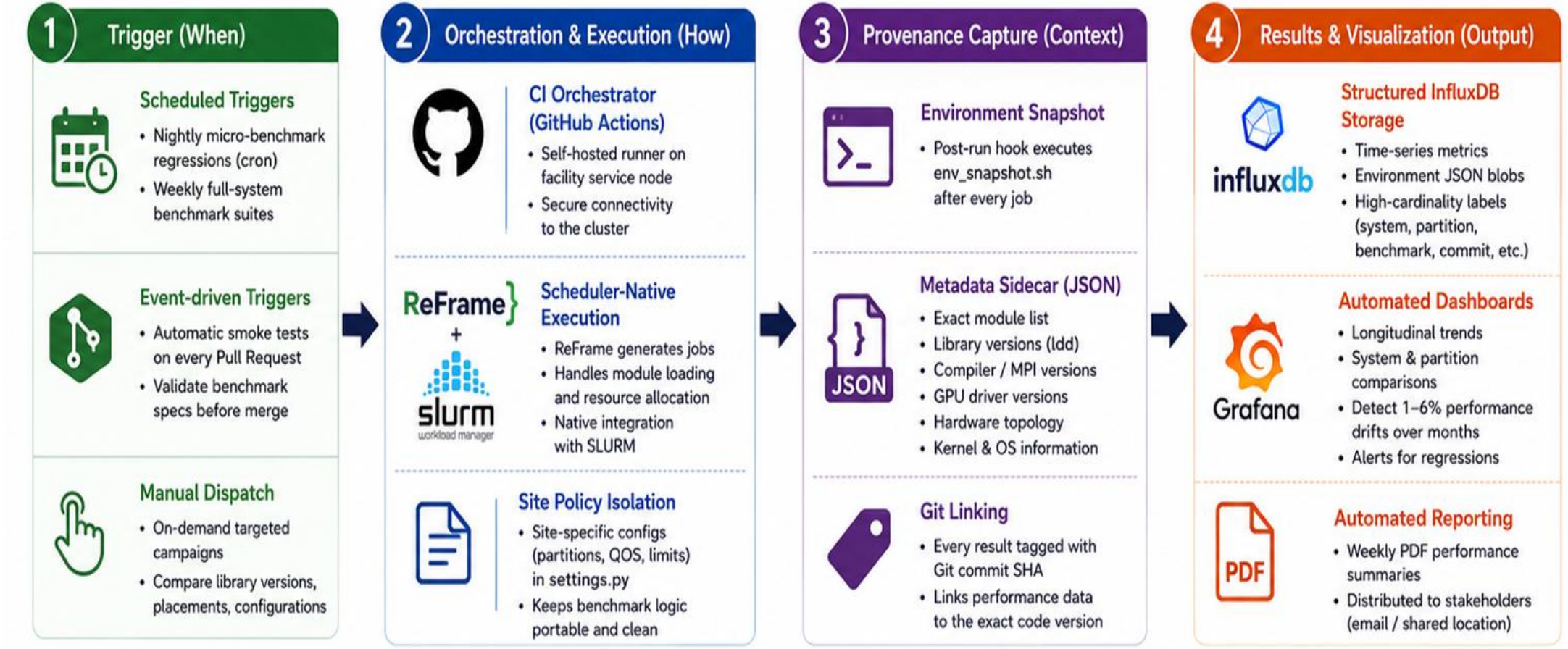




Architecture: InfluxDB

name	sysenv	pvar	punit	pval (max)
stream_test	PARAMSYSTEM:cpu+gnu	copy_bw	MB/s	19538.5
stream_test	PARAMSYSTEM:cpu+gnu	triad_bw	MB/s	15299.1
stream_test	PARAMSYSTEM:default+gnu	copy_bw	MB/s	25944.8
stream_test	PARAMSYSTEM:default+gnu	triad_bw	MB/s	13701.7

Workflow


1 Trigger (When)
Scheduled Triggers
• Nightly micro-benchmark regressions (cron)
• Weekly full-system benchmark suites
Event-driven Triggers
• Automatic smoke tests on every Pull Request
• Validate benchmark specs before merge
Manual Dispatch
• On-demand targeted campaigns
• Compare library versions, placements, configurations
2 Orchestration & Execution (How)
CI Orchestrator (GitHub Actions)
• Self-hosted runner on facility service node
• Secure connectivity to the cluster
ReFrame
+
slurm
workload manager
Scheduler-Native Execution
• ReFrame generates jobs
• Handles module loading and resource allocation
• Native integration with SLURM
Site Policy Isolation
• Site-specific configs (partitions, QOS, limits) in settings.py
• Keeps benchmark logic portable and clean
3 Provenance Capture (Context)
Environment Snapshot
• Post-run hook executes env_snapshot.sh after every job
JSON
Metadata Sidecar (JSON)
• Exact module list
• Library versions (ldd)
• Compiler / MPI versions
• GPU driver versions
• Hardware topology
• Kernel & OS information
Git Linking
• Every result tagged with Git commit SHA
• Links performance data to the exact code version
4 Results & Visualization (Output)
influxdb
Structured InfluxDB Storage
• Time-series metrics
• Environment JSON blobs
• High-cardinality labels (system, partition, benchmark, commit, etc.)
Grafana
Automated Dashboards
• Longitudinal trends
• System & partition comparisons
• Detect 1–6% performance drifts over months
• Alerts for regressions
PDF
Automated Reporting
• Weekly PDF performance summaries
• Distributed to stakeholders (email / shared location)

Supported Systems and Architectures

Deployment Baseline

- **CPU Nodes:**
  48 nodes featuring dual-socket Intel Xeon Platinum 8358 processors (24 cores per socket) and 192 GB DDR4 RAM
- **GPU Nodes:**
  16 nodes featuring Intel Xeon Platinum 8358 processors and dual NVIDIA A100-80GB GPUs with 192 GB DDR4 RAM
- **Interconnect:**
  High-speed **HDR InfiniBand** at 200 Gb/s, supporting both intra-rack and cross-rack communication monitoring
- **Storage & Management:**
  The system uses a dedicated service node in the facility's DMZ for CI orchestration via SSH access to the login node and the Lustre file system for storage

Software Stack and Environment

- **Operating System:**
  AlmaLinux 8
- **Workload Manager:**
  SLURM 22.05, utilizing native integration for job generation, resource limits, and monitoring
- **Environment Management:**
  Lmod for module handling and **Spack** as the primary build system
- **CI/CD Integration:**
  GitHub Actions with self-hosted runners

Portability and Extensibility

- **Site-Specific Mapping:**
  Porting to a new facility primarily requires updating a single settings.py file to map abstract requirements (like "gpu-partition") to local site policies
- **Component Substitution:**
  The modular design allows facilities to substitute core components such as different CI systems, schedulers, or databases while maintaining the same reproducibility and automation properties

Framework Flexibility & Node Types

- **Multi-Partition Support:**
  Natively supports different node configurations, including CPU-only, High-Memory (HM), and GPU-accelerated nodes
- **Queue Management:**
  Designed to navigate complex queue policies using dedicated service accounts and **Quality of Service (QOS)** limits to prevent monopolising production resources
- **Programming Models:**
  The benchmark suite supports multiple programming models, including **OpenMP**, **CUDA**, and **MPI,** through its underlying integration with ReFrame

Experimental Setup

1. TWO-TIERED BENCHMARK SUITE

Tier 1: Micro-Benchmarks (Subsysten Health)

Memory
STREAM (CPU/GPU) for bandwidth

Network
OSU Micro-Benchmarks MPI latency & bandwidth (2–64 nodes)

Compute
DGEMM/BLAS for raw CPU & GPU FP throughput

Reference
HPL (Linpack) for monthly full-system peak performance

Tier 2: Application Kernels (Production Workloads)

Molecular Dynamics
- LAMMPS (Lennard-Jones & Protein-folding)
- GROMACS (Water-box & STMV virus capsid)

MPI Application
Finite Element (FE) mini-app for CPU-bound MPI

2. EVALUATION METHODOLOGY

Timeline
Longitudinal evaluation over six months (Oct 2024 – Mar 2025)

Reproducibility Metric
Coefficient of Variation (CoV) across 10 repeated runs under identical conditions

Coverage Metric
Count of distinct benchmark configurations executed per month (unique combinations of benchmark, node count, and GPU allocation)

Efficiency Metric
Structured time log maintained by administrators to track total hours spent on campaign setup, monitoring, and reporting

3. SYSTEMS & SOFTWARE ENVIRONMENT

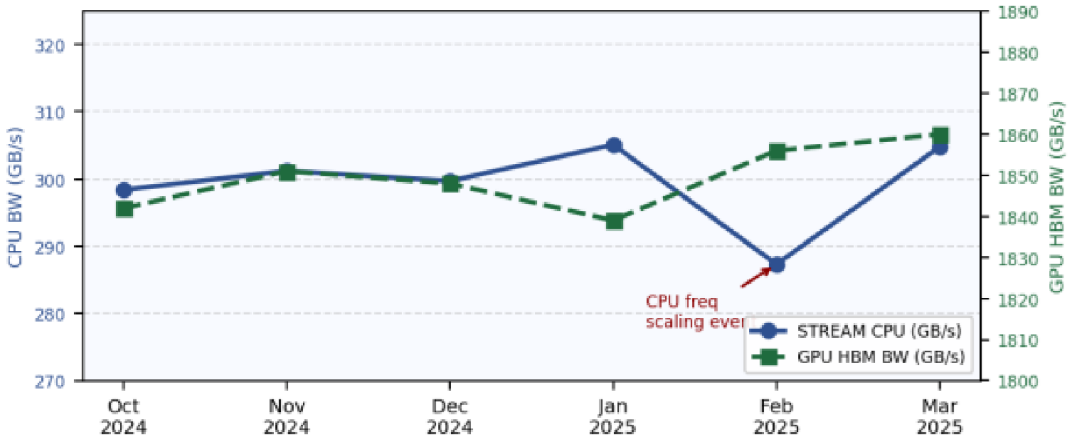
Systems
Evaluated on multiple production HPC clusters with heterogeneous architectures:
- Intel x86 (CPU-only)
- NVIDIA GPU nodes (A100 / V100)
- High-Memory nodes

Scheduler
SLURM (25.x) with per-partition QoS

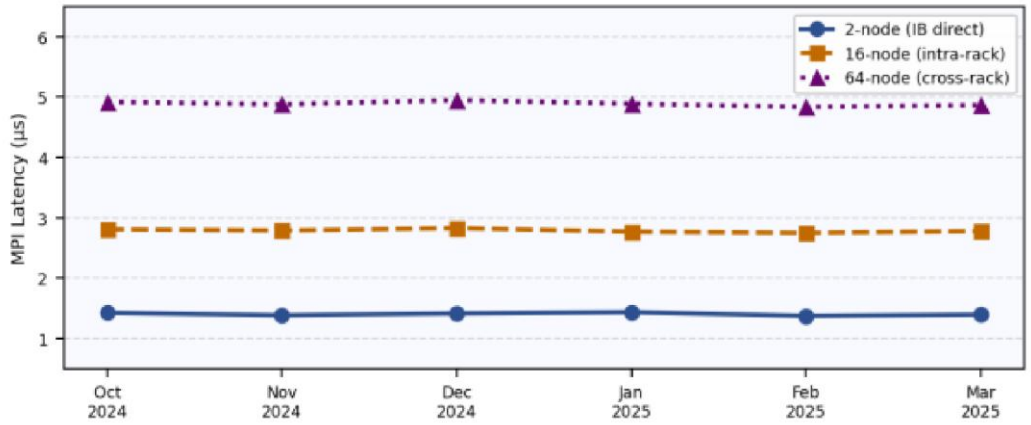
Benchmarking Framework
ReFrame (4.x) integrated with ContinuousBench

Key Software
Compilers: GCC, Intel
MPI: OpenMPI, Intel MPI, MVAPICH2
CUDA, ROCm, cuDNN, NCCL
OS: Rocky Linux / AlmaLinux

Results



STREAM CPU and GPU memory bandwidth over six months. The February CPU dip (−6%) was automatically flagged by a threshold alert and identified as an inadvertent CPU frequency-scaling policy change. The framework's environment capture confirmed the change in /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor, enabling root-cause diagnosis within two hours.

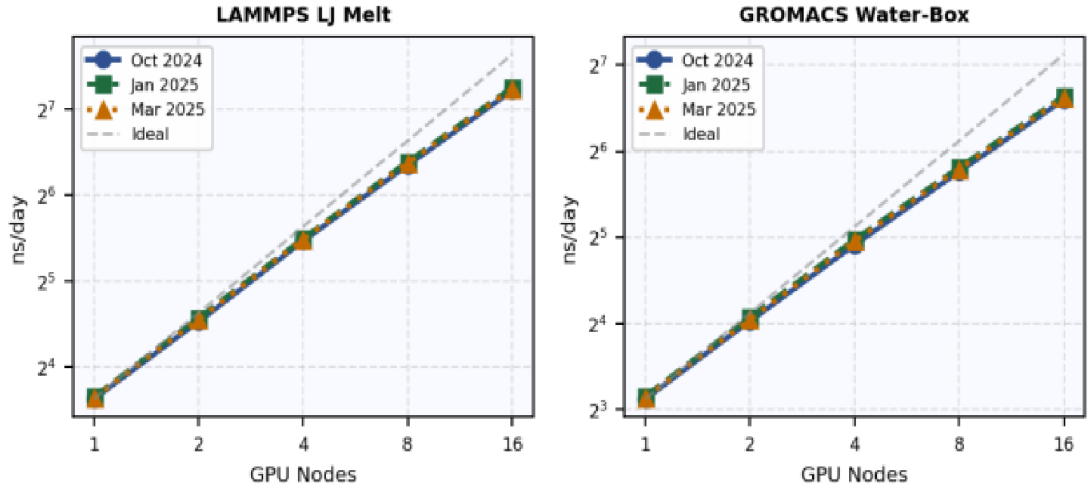


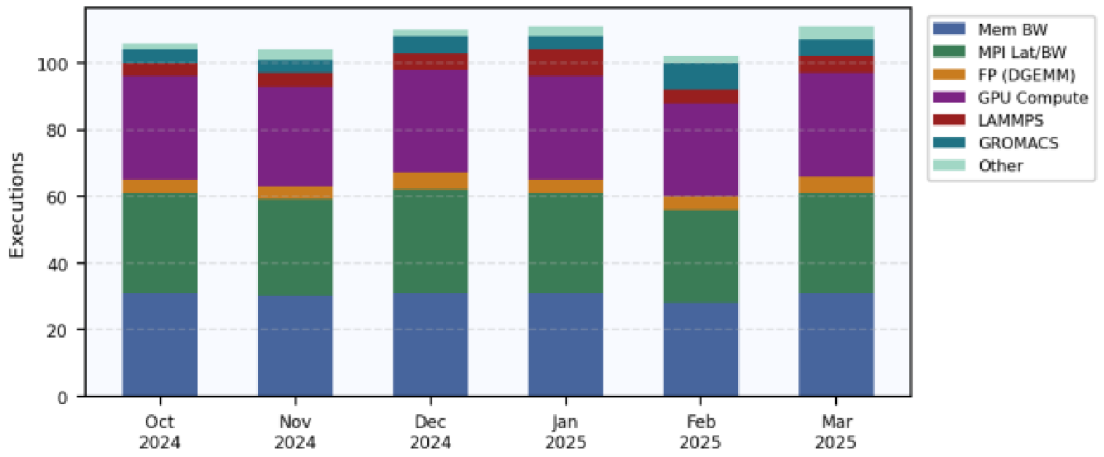
OSU MPI point-to-point latency at 2-, 16-, and 64-node scales. Consistent sub-3 µs cross-rack latency confirms InfiniBand fabric health. The gap between intra-rack and cross-rack is expected; any narrowing would indicate misconfigured routing.

Results

LAMMPS and GROMACS strong-scaling performance at three monitoring snapshots (Oct 2024, Jan 2025, Mar 2025). Dashed lines indicate ideal linear scaling. Both applications sustain >75% parallel efficiency at 16 GPU nodes. Improvements from Oct to Mar reflect CUDA driver updates in November and a GROMACS PME tuning fix in December.

Monthly benchmark execution volume by category. The LAMMPS and GROMACS spikes in January and February correspond to ad hoc campaigns triggered by CUDA driver and GROMACS version updates respectively demonstrating the on-demand workflow in practice.





Future Work

1. Energy and Carbon Metrics

- **Hardware Integration:** There are plans to integrate the framework with **RAPL** for CPU power measurement, **NVML** for GPU power, and available **PDU-level monitoring APIs**.
- **Sustainability Reporting:** ContinuousBench will combine these power measurements with real-time **grid carbon intensity data** from electricity map APIs to calculate derived carbon metrics.
- **Pareto Analysis:** This will enable **multi-objective performance/energy Pareto analysis**, allowing facilities to report on sustainability metrics alongside traditional performance data.
- **Scaling Characteristics:** To extend the tool to specifically analyze power and performance metrics based on different **HPC application categories** and their unique scaling characteristics.

2. Multi-Site Federation and Open Source

- **Open-Source Release:** To foster community adoption, the plan to release ContinuousBench as an **open-source project**.
- **Federation Protocol:** A core goal is to develop a protocol that allows different HPC facilities to **share benchmark specifications** and anonymised result aggregates through a common repository.
- **Cross-Site Comparison:** By moving toward a federated model, the framework aims to enable meaningful **cross-site performance comparisons** once adopted by multiple facilities

Conclusion

- **Integrated Automation:**
  ContinuousBench successfully **integrates CI infrastructure with HPC batch scheduling** to automate the full lifecycle of benchmark execution, environment capture, and result management in production environments.

- **Significant Efficiency Gains:**
  The deployment at C-DAC Pune demonstrated an **87.1% reduction in campaign setup time** (falling from 6.2 to 0.8 hours) and a **350% increase in the volume of benchmark configurations** executed monthly.

- **Enhanced Reproducibility:**
  By automating environment provenance through JSON sidecars and Git linking, the framework **reduced performance variance from 8.3% to 1.1%**, nearly eliminating errors caused by inconsistent module loading or undocumented system changes.

- **Rapid Return on Investment:**
  The framework proved its operational value by reaching a **break-even point in less than one month**, ultimately saving administrators approximately **218 hours of manual effort** over the six-month evaluation period.

- **Architectural Flexibility:**
  Designed as a **portable template rather than a rigid product**, its modular five-component architecture allows different facilities to substitute their preferred CI systems, schedulers, or databases while maintaining core reproducibility.

- **Community and Future Vision:**
  To enable broader adoption and multi-site federation, with future development targeting **energy/carbon metrics** and adaptive automated regression detection

Question?