Università
della
Svizzera
italiana

Faculty
of
Informatics

ClusterBench - a Modular Benchmarking Framework for HPC

Filippo Wang, Leon Ackermann, Matteo Arrigo, Christian Karg

Universitá della Svizzera Italiana, Institute of Computing, Lugano, Switzerland.

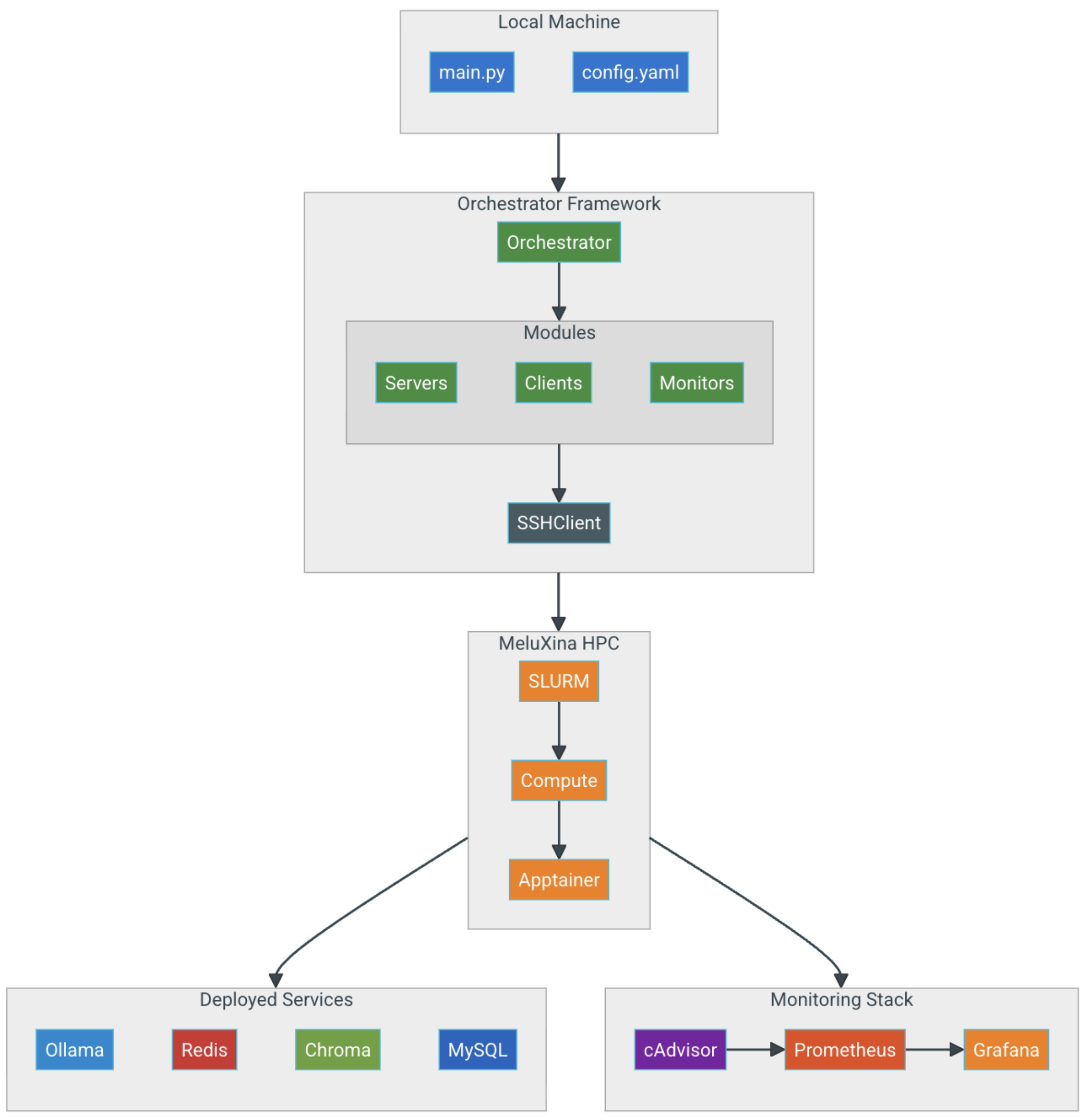
Software Atelier: Simulation, Data Science & Supercomputing SA 2025-2026.

In collaboration with EUMASTER4HPC and MeluXina Supercomputer (Luxembourg) – Supervised by Dr. Farouk Mansouri.

Motivation and System Architecture

Problem: Deploying and benchmarking workloads on HPC clusters involves significant operational complexity. Researchers must manually configure SLURM jobs, build container images, establish SSH tunnels, and coordinate monitoring infrastructure—a process that is error-prone and difficult to reproduce.

Our Solution: We present a modular Python framework that abstracts these complexities through declarative YAML configurations. The system automatically orchestrates containerized services via Apptainer, manages SLURM job submission, and provides integrated real-time monitoring through a Prometheus-Grafana stack.

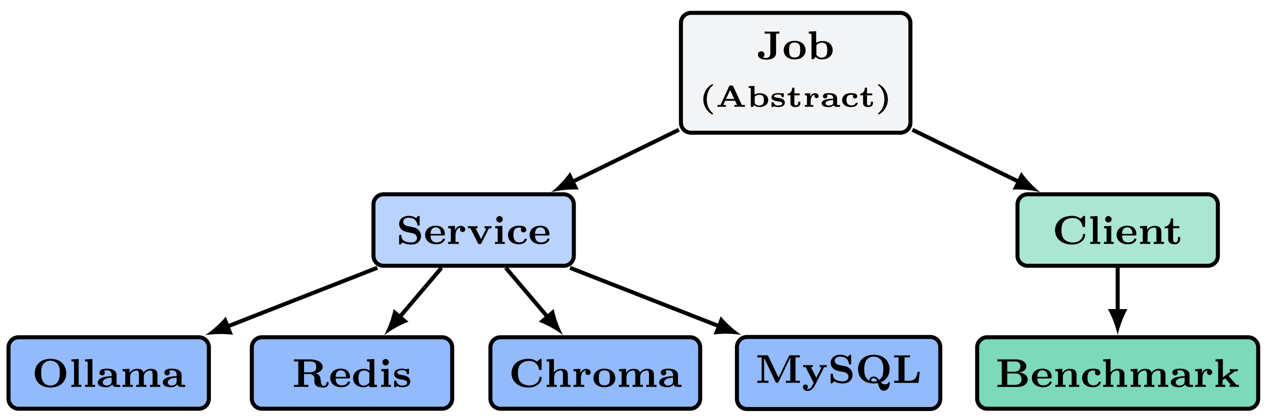


Job Hierarchy and Design Patterns

The framework employs a class hierarchy separating service deployment from benchmark execution. The abstract `Job` base class defines the SLURM script generation template, while subclasses implement service-specific logic.

Adding a Service: Extend the `Service` class, override `get_container_command()` to define the Apptainer execution, and register via `JobFactory.register_service()`.

Adding a Client: Extend the `Client` class, implement benchmark logic in `get_benchmark_commands()`

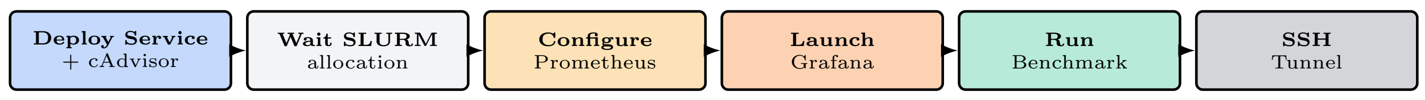


- ➜ **Factory Pattern** `JobFactory` dynamically instantiates Service/Client from YAML recipe `type` field
- ➜ **Template Method** `generate_slurm_script()` defines structure; subclasses override `get_setup_commands()`
- ➜ **Strategy Pattern** Clients receive service endpoint via `target_host:port` and implement workload-specific benchmarks

Automated Deployment Pipeline

Traditional HPC workflows require multiple manual steps: allocating resources, building containers, configuring services, and setting up monitoring. Our framework consolidates this into a single declarative command.

```
python main.py --start-session service.yaml client.yaml monitor.yaml
```



The CLI handles SLURM submission, node discovery, and endpoint resolution. For multi-service workflows, users can compose shell scripts using CLI primitives or use the built-in `--start-session` for full automation.

Composable CLI for Reproducible HPC Workflows

Real-Time Monitoring with Grafana

The monitoring stack provides comprehensive observability. Each service runs alongside a cAdvisor sidecar exposing container metrics. We have implemented some pre-configured Grafana dashboards visualize resource utilization.

CLI Commands Reference

Command	Description
`--recipe <file>`	Deploy service/client from YAML recipe
`--target-service <id>`	Connect benchmark client to running service
`--status`	Display all SLURM jobs and their states
`--stop-all-services`	Terminate all running services
`--start-session`	Automated: service + client + monitoring
`--create-tunnel <id>`	Generate SSH tunnel command for UI access
`--download-results`	Fetch benchmark JSON results to localhost

Framework Advantages

Local-First Control: All operations execute from the user's local machine. After initial SSH credential setup, users never need to manually log into cluster nodes—the framework handles remote execution transparently.

Portable Across HPC Systems: YAML recipes abstract cluster-specific details. Migrating benchmarks to a different HPC requires only updating `config.yaml` with new credentials and paths.

Reduced Complexity: Eliminates common error sources in HPC workflows:

- No manual SLURM script writing or module loading commands
- Automatic service discovery and endpoint resolution
- Pre-configured monitoring without Prometheus/Grafana expertise

Target Users:
(1) *Beginners* learning HPC can focus on workloads, not infrastructure;
(2) *Experienced users* can rapidly prototype and compare services across clusters with minimal setup overhead.

Benchmark Results

Redis KV-store

100 parallel clients, 256B payload

Op	Throughput	Latency
GET	45-100K/s	0.5-2.0 ms
SET	40-95K/s	0.5-2.5 ms
LPUSH	35-85K/s	0.6-3.0 ms

MySQL

Sysbench, 300s run

Metric	Value
Transactions	34,418
Throughput	114.67 TPS
Query Rate	2,293 QPS
Avg Latency	139.5 ms

Chroma Vector DB

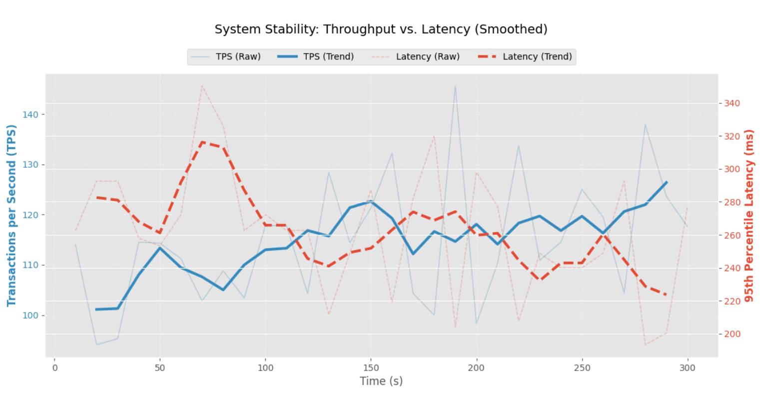
384-dim embeddings

Docs	Insert	Query
1K	250+/s	3.5 ms
100K	252/s	5.25 ms
1M	200/s	8-10 ms

Ollama LLM

llama2, 1× A100 GPU

Conc.	Lat.	Tok/s
1 req	1-2 s	150+
5 req	2.5-5 s	120+
10 req	4-8 s	100+



MySQL Sysbench Time series showing the inverse "saw-tooth" pattern.

Conclusions

We presented ClusterBench, a modular orchestration framework that simplifies the deployment and benchmarking of containerized workloads on HPC clusters.

Key Contributions:

- Declarative YAML-based service configuration eliminating manual SLURM scripting
- Integrated monitoring stack with automatic Prometheus target discovery
- Extensible architecture supporting diverse workloads (LLM, vector DB, cache, RDBMS)

The framework reduces operational complexity while ensuring experiment reproducibility across heterogeneous HPC environments.

Future Work

- **Multi-node MPI:** Extend service deployment across multiple compute nodes with automatic rank coordination
- **GPU Metrics:** Integrate NVIDIA DCGM for GPU utilization, memory bandwidth, and thermal monitoring
- **Auto-scaling:** Dynamic resource allocation based on real-time workload metrics from Prometheus
- **Kubernetes Hybrid:** Support K8s orchestration alongside SLURM for containerized microservices

References

[1] docs.lxp.lu
[2] slurm.schedmd.com
[3] docs.ollama.com
[4] redis.io/docs
[5] trychroma.com
[6] dev.mysql.com

EUMaster4HPC MELUXINA-Q LUXPROVIDE EuroHPC