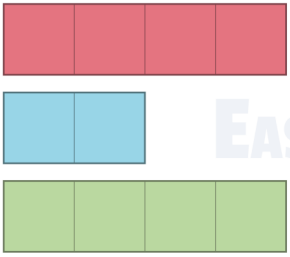


EASYBUILD.io

building software with ease

exaCB

INTEGRATING EXACB WITH EASYBUILD

Why and How?

March 25, 2025 | Jayesh Badwaik | j.badwaik@fz-juelich.de | Jülich Supercomputing Centre

JÜLICH Forschungszentrum

CONTINUOUS BENCHMARKING ON HPC

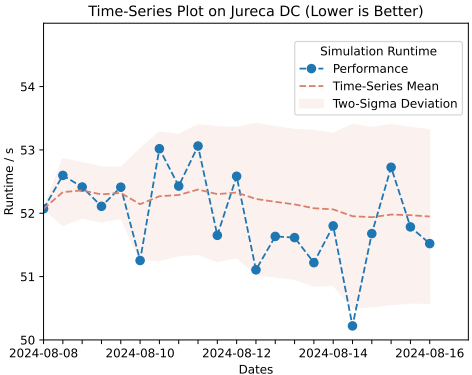
Why now?

- 1 ExaFLOP/s
- NVIDIA Grace-Hopper CG1
  - $\sim$6000 nodes
  - $4\times$ CG1 chips per compute node (ARM + Nvidia)
  - NVLink C2C $450 + 450$ GB/s
  - Smart Unified Memory
  - Power sharing between CPU and GPU
- Mean time between failures $\sim$ hours – days

JÜLICH Forschungszentrum

WHY CONTINUOUS BENCHMARKING ON HPC?

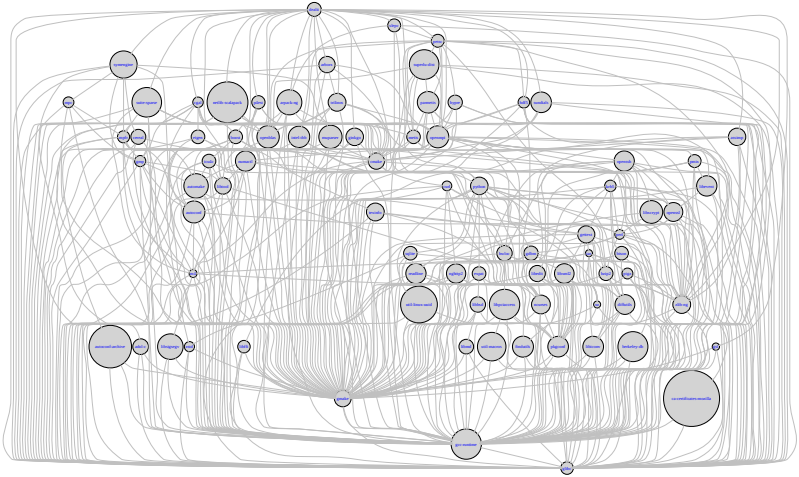
Why now?



- Tracking progress over time
- Tracking progress through changes
- Onboarding troubles
  - Custom versions for applications
  - Custom compiler flags
- Well-formed auditable execution of benchmarks
- Rapid testing of deployment

JÜLICH Forschungszentrum

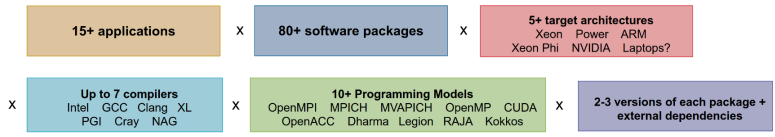
COMPLEX SOFTWARE STACK



JÜLICH Forschungszentrum

COMPLEX SOFTWARE STACK

The ECP software environment is enormously complex



= ~ **1,260,000** combinations!

- Every application has its own stack of dependencies.
- Developers, users, and facilities dedicate (many) FTEs to building & porting.
- Often trade reuse and usability for performance.

Complexity makes software reuse difficult!

ECP EXASCALE COMPUTING PROJECT

14

SYSTEM STUDIES

Energy and Performance

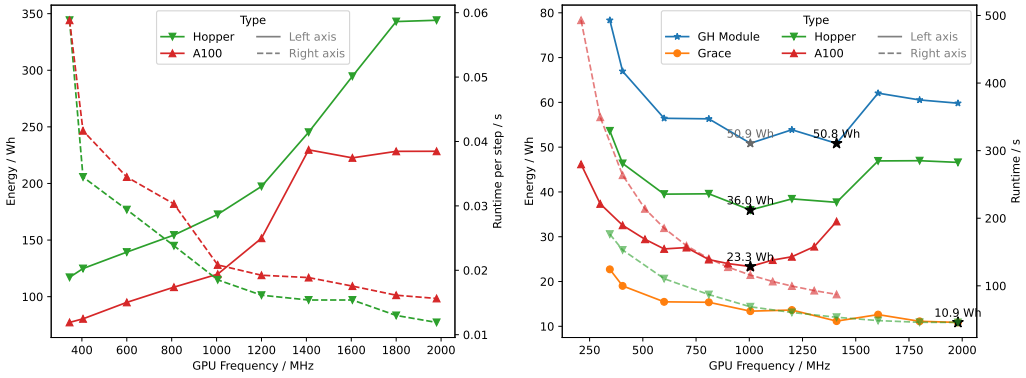


Figure: Energy to Solution for SOMA and MPTRAC

JÜLICH Forschungszentrum

SCHEDULING AND RESOURCE MANAGEMENT

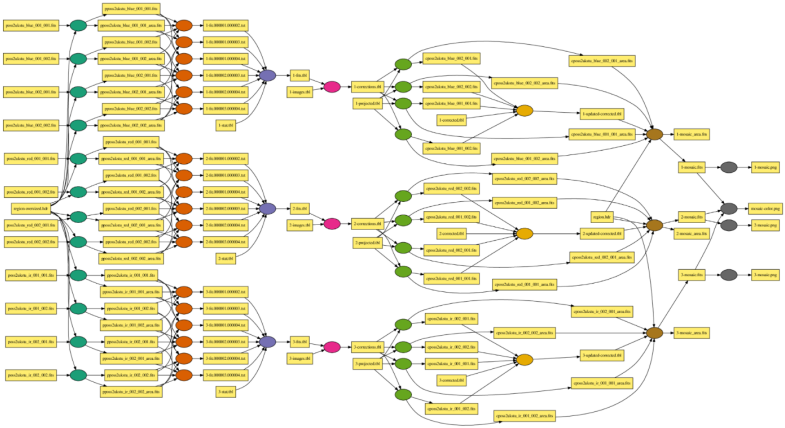


Figure: Workflow Optimization by Smart Scheduling

JÜLICH Forschungszentrum

EXACB

Features

- Continuous Benchmarking for HPC
- Template-based CI/CB (declarative syntax)
- Reduced barrier to entry for CI/CB
- Click-to-run reproducible benchmarks
- Ease of sharing workflows with community

Usecase

- > 70 applications in JUREAP
- Automated report generation for JUREAP

→ Compute time review

- Energy measurements
- System performance evaluation

TEMPLATE-BASED CI/CD

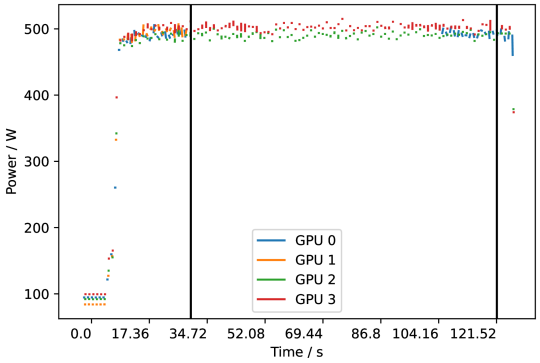
```yaml
include:
 - component: jureap/jube@v3.2
   inputs:
    prefix: "jedi.strong.tiny"
    variant: "strong.tiny"
    machine: "jedi"
    queue: "all"
    project: "cjsc"
    budget: "zam"
    jube_file: "simple.yaml"
```

- Setup compute account and reservations
- Run fixtures (setup and teardown)
- Run benchmark (see below)
- Generate reports
- Upload results into repositories

```
$ jube-autorun -r "--tag $[[inputs....]]" $[[inputs.jube_file]]
|-------------------------------------------------------------------------
|node | slurmid | gitlab | result | runtime | start       | end         |
|-------------------------------------------------------------------------
|1    | 1234    | 5678   | pass   | 200s    | 7548.291648 | 8742.291648 |
|2    | 1235    | 5679   | pass   | 220s    | 4984.336033 | 6284.336033 |
|3    | 1236    | 5680   | pass   | 192s    | 3234.343434 | 4234.343434 |
|-------------------------------------------------------------------------|
```

ENERGY MEASUREMENTS

```yaml
include:
- component: jureap/energy@v3.2
    inputs:
      prefix: "jedi.strong.tiny"
      variant: "strong.tiny"
      machine: "jedi"
      queue: "all"
      project: "cjsc"
      budget: "zam"
      jube_file: "simple.yaml"
```



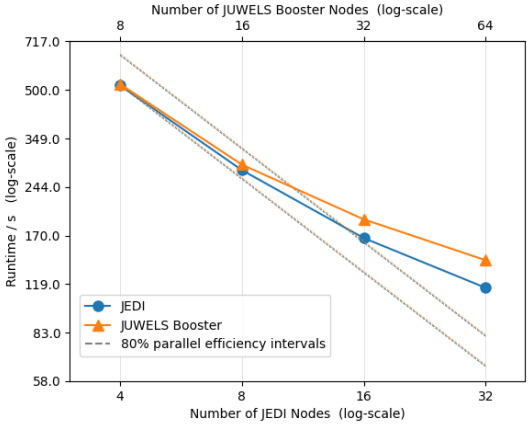

```
srun [options] jpwr [jpwr-options] executable [args]
```


JÜLICH Forschungszentrum

REPORT GENERATION

```
include:
 - component: jureap/report@v3.2
   inputs:
    prefix: "report"
    pipeline: ["245543", "245544"]
    selector: ["jedi", "booster" ]
```

- Fetch data for the pipeline from datastore
- Generate PDF reports with plots and tables



JÜLICH Forschungszentrum

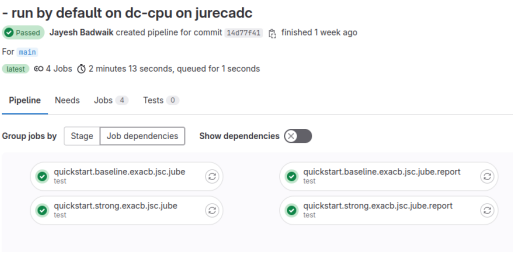
OTHER FACTORS

Technical Factors

- Scheduled runs and report generation
- Data storage (Gitlab Repo, S3)
- Automatic alerts (Gitlab, email)

Community Factors

- Enforce reproducibility
- Compare application with synthentic
- Integrate benchmarks in development repor



JÜLICH Forschungszentrum

WHAT IS REPRODUCIBILITY?

In this context

Site-specific dependencies

Required	Optional	Uncontrollable
Versioned dependencies (including flags)	Bit-wise reproducibility of dependencies	Network effects
Slurm configuration (including defaults)	Same hardware	I/O effects
Environment configuration	Environment Isolation	Driver and OS updates

AUDITABILITY AND REPRODUCIBILITY

Aspect	Should Audit	Can Audit	Tool
Configure Environment	No	No	Fail/Pass
Load data/cache	Yes	Yes (No Uniformity)	Checksum
Load dependencies	Yes	Depends	Dependency manager
Build application	Ideally Yes	Difficult	Hermiticity
Run benchmark	Yes	Yes	exaCB
Upload results	Yes	Yes	exaCB

Why develop for auditability?

- Helps in debugging on support side
- Shared techniques – faster onboarding
- Quicker study for prioritization

AUDITABILITY AND REPRODUCIBILITY

Aspect	Should Audit	Can Audit	Tool
Configure Environment	No	No	Fail/Pass
Load data/cache	Yes	Yes (No Uniformity)	Checksum
Load dependencies	Yes	Depends	Dependency manager
Build application	Ideally Yes	Difficult	Hermiticity
Run benchmark	Yes	Yes	exaCB
Upload results	Yes	Yes	exaCB

CURRENT STATE

Resolution Method	Number of Projects	Explicit Dependencies	Rerunnable
Out-of-band (Spack, Container)	6	20	?
Prebuilt Binary	17	15	?
In-situ/Ad-hoc	26	15	?
JUBE script	1	4	?
Gitlab CI	2	6	?
Total (Outside EasyBuild)	48	20	12
EasyBuild	20	12	16

Undesirable Situation

- Blind spots about important dependencies
- Restricts number of projects that can be integrated
- Restricts ability to help users with potential bugs
- Restricts ability to plan changes
- Prevents building up of shared resource
- Duplicated work

TOWARDS A POTENTIAL SOLUTION

Current problems (and potential solutions?)

Problem	Potential Solution
Long build times	Caching
Complicated build process	Dependency management
Modules not available in EasyBuild	Custom modules
Customized build for each configuration	On-demand builds
Custom version of modules	On-demand builds
Custom compilation options	On-demand builds
Reproducibility at later date	Lockfiles

CURRENT TOOLS IN EASYBUILD ECOSYSTEM

- Easybuild
  - Need variant and toolchain along with suffix as additional actionable parameters
  - `-Wl,-rpath` – extremely useful to separate build and run environments
  - Chaining installations ergonomically
- User installations
  - Manually setup caches
  - Not easy to inject custom dependencies in a given chain
  - Not enough institutional knowledge to advise users
- EESSI
  - S3 is important - JSC is going to offer MinIO
  - Periodic refresh (need on-demand)
  - Unergonomic to use without root
  - Per user filepaths are (not possible?)
  - Multiple caches per user are not possible (independent CI runs)
  - Geared towards broadcasting downstream

TENTATIVE IDEA

Single Point of Usage

```
ebpkg cache add jedi.2025 --type s3 --credentials jedi.2025.yaml
ebpkg source --chainload /p/easysite/jedi/2025 dependency.yaml
ebpkg resolve
```

User driven auditable and reproducible wrapper around easybuild

- Exactly similar workflow on all machines
  - Desktop, HPC Centers, CI Containers
- Ability to chain installations
  - Separation of concerns and workload
  - Leverage the 3000 easybuild recipes
- First class support to download and upload caches
- Support for injecting a specific dependency
- Reproducibility at later date (Lockfiles)
- Institutional knowledge (with users)
- Uniform names for core packages
- Ability to experiment and freeze current state
- Users and admins are equal (mostly?)
- Community contributions
- Community testing

Thank You!