Enabling Julia code to run at scale with artefact caching

JuliaHEP Workshop, 2024

Elvis Aguero[1]
Supervisors: Graeme A Stewart[2] and Pere Mato Vila [2]

[1]School of Engineering, Brown University
[2]CERN

Challenges in computational High-Energy Physics

- Research in High-Energy Physics (HEP) is computationally intensive.
  - Each LHC experiment at CERN has around 6M+ lines of code
  - The WLCG has 1.4M cores distributed across 170 sites.

Challenges in computational High-Energy Physics

- Research in High-Energy Physics (HEP) is computationally intensive.
  - Each LHC experiment at CERN has around 6M+ lines of code
  - The WLCG has 1.4M cores distributed across 170 sites.
- Current paradigm: Performance-critical code is written in C++, with significant use of Python.
  - Leading to the need to interface these two, rather different, languages.

Julia for HEP

- Julia is a high-level, general-purpose language, with dynamically typed characteristics and a REPL.
  - → Ease of use, promoting high productivity workflows.

[1] https://hepsoftwarefoundation.org/workinggroups/juliahep.html
[2] https://arxiv.org/pdf/2306.03675

Julia for HEP

- Julia is a high-level, general-purpose language, with dynamically typed characteristics and a REPL.
  - → Ease of use, promoting high productivity workflows.
- JIT compilation at runtime via LLVM.
  - → Julia benchmarks similar to C/C++, particularly in scientific computing packages

[1] https://hepsoftwarefoundation.org/workinggroups/juliahep.html
[2] https://arxiv.org/pdf/2306.03675

Julia for HEP

- Julia is a high-level, general-purpose language, with dynamically typed characteristics and a REPL.
  - → Ease of use, promoting high productivity workflows.
- JIT compilation at runtime via LLVM.
  - → Julia benchmarks similar to C/C++, particularly in scientific computing packages

→ Julia has drawn considerable attention in the HEP community[1,2].

[1] https://hepsoftwarefoundation.org/workinggroups/juliahep.html
[2] https://arxiv.org/pdf/2306.03675

Distributed computing with Julia

Julia precompiles packages and user files generating local files that are optimized for its current CPU microarchitecture.

Distributed computing with Julia

Julia precompiles packages and user files generating local files that are optimized for its current CPU microarchitecture.

- → Files generated at one node are not directly available to the other nodes.
- → When available, generated precompiled files might not be compatible.
- → Waste of resources when running at scale on the grid

Distributed computing with Julia

Julia precompiles packages and user files generating local files that are optimized for its current CPU microarchitecture.

- → Files generated at one node are not directly available to the other nodes.
- → When available, generated precompiled files might not be compatible.
- → Waste of resources when running at scale on the grid

Our goal:

Distributed computing with Julia

Julia precompiles packages and user files generating local files that are optimized for its current CPU microarchitecture.

→ Files generated at one node are not directly available to the other nodes.

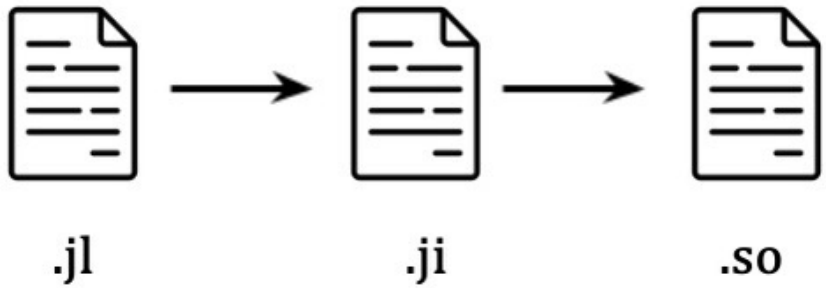
→ When available, generated precompiled files might not be compatible.

→ Waste of resources when running at scale on the grid

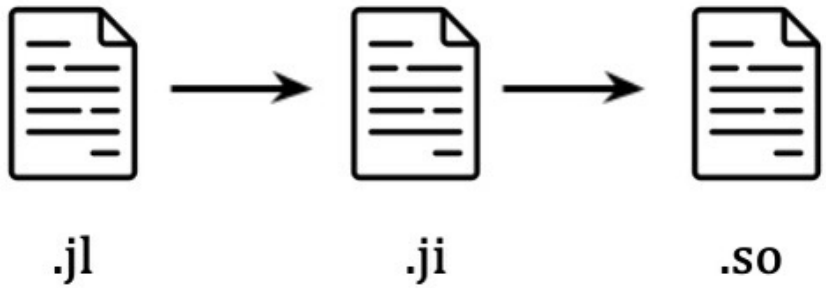
Our goal:

- Leverage Julia's potential to run in distributed contexts

Making cache files relocatable

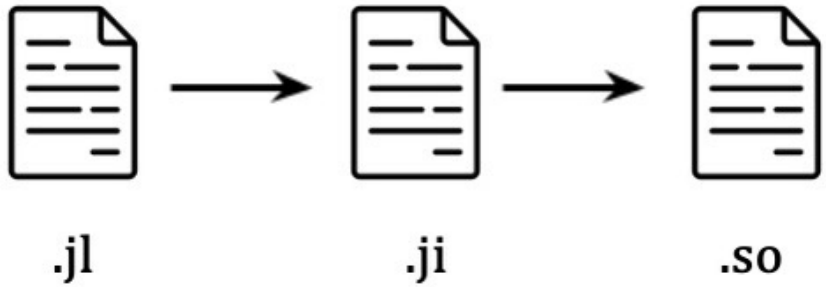


Making cache files relocatable



The variable `Base.DEPOT_PATH` controls where cached compiled package images are stored.

Making cache files relocatable



The variable `Base.DEPOT_PATH` controls where cached compiled package images are stored.

```
[eaguerov@pcphsft90]~% julia -e "println(Base.DEPOT_PATH)"
["/home/eaguerov/.julia", "/home/eaguerov/.julia/juliaup/julia-1.11.0-rc1+0.x64.linux.gnu/loca
l/share/julia", "/home/eaguerov/.julia/juliaup/julia-1.11.0-rc1+0.x64.linux.gnu/share/julia"]
[eaguerov@pcphsft90]~% ls .julia/compiled/v1.11/BitFlags/
[eaguerov@pcphsft90]~% julia -e "using BitFlags"
Precompiling BitFlags...
  1 dependency successfully precompiled in 1 seconds
[eaguerov@pcphsft90]~% ls .julia/compiled/v1.11/BitFlags/
sbYUO_k6ZNm.ji  sbYUO_k6ZNm.so
```

Making cache files relocatable

- → First entry of `DEPOT_PATH` must be writable, while others entries are treated as read-only.
- → Multiple Julia projects (within the same node) use the same local DEPOT_PATH. Julia decides if cache file is stale at runtime.
- → Can be set before startup on the terminal

```
export JULIA_DEPOT_PATH="/foo/bar:\$JULIA_DEPOT_PATH"
```

Making cache files relocatable (2)

Prior to Julia 1.11, the following would invalidate a cache file:

1. The **absolute path** of the generated cache file.
2. The file's modification time (mtime).
3. Incompatible image targets for the host's instruction set architecture (ISA).

Making cache files relocatable (2)

Prior to Julia 1.11, the following would invalidate a cache file:

1. The **absolute path** of the generated cache file.
2. The file's modification time (mtime).
3. Incompatible image targets for the host's instruction set architecture (ISA).

These issues are partially resolved by:

1. The candidate release Julia 1.11-rc only considers relative path of package files
2. Copying with a procedure that respects mtime `rsync` vs `cp` when cache files are generated from a non-host machine.
3. Julia's built-in cross-compilation capabilities, by setting the environment variable `JULIA_CPU_TARGET`

Julia's Cross-compilation

Julia interfaces to the LLVM compiler to set image targets.

```
export JULIA_CPU_TARGET="generic;sandybridge;haswell,clone_all"
```

You could see your current image and cpu targets with `JLOptions()` and `Base.current_image_targets()`

```
[eaguerov@pcphsft90]~% export JULIA_CPU_TARGET=generic
[eaguerov@pcphsft90]~% export JULIA_DEPOT_PATH=/eos/user/e/eaguerov/julia:
[eaguerov@pcphsft90]~% julia -q
(v1.11) pkg> add BitFlags
   Resolving package versions...
  No Changes to `/eos/home-e/eaguerov/julia/environments/v1.11/Project.toml`
  No Changes to `/eos/home-e/eaguerov/julia/environments/v1.11/Manifest.toml`
Precompiling project...
  1 dependency successfully precompiled in 2 seconds

julia>
[eaguerov@pcphsft90]~% ls /eos/user/e/eaguerov/julia/compiled/v1.11/BitFlags/
sbYUO_SBlWU.ji  sbYUO_SBlWU.so
[eaguerov@pcphsft90]~% julia -e "println(Sys.CPU_NAME)"
ivybridge
```

Practical implications

At CERN we could leverage the use of the Cern Virtual machine File System (CVMFS) to make precompiled files available.

- All projects can see the same node, effectively sharing an entry in the `DEPOT_PATH`.
  - → We prepare precompiled artifacts for each workflow in turn, "rsyncing" the cache files to that node.

[1] This would make publishing to CVMFS for Julia $\leq$ 1.10 tricky

Practical implications

At CERN we could leverage the use of the Cern Virtual machine File System (CVMFS) to make precompiled files available.

- All projects can see the same node, effectively sharing an entry in the `DEPOT_PATH`.
  - → We prepare precompiled artifacts for each workflow in turn, "rsyncing" the cache files to that node.
- Then execute the publication:
  - Open a transaction
  - Copy files to the correct path in `/cvmfs/`
  - Publish and close transaction[1]

[1] This would make publishing to CVMFS for Julia $\leq$ 1.10 tricky

Practical implications

At CERN we could leverage the use of the Cern Virtual machine File System (CVMFS) to make precompiled files available.

- All projects can see the same node, effectively sharing an entry in the `DEPOT_PATH`.
  - → We prepare precompiled artifacts for each workflow in turn, "rsyncing" the cache files to that node.
- Then execute the publication:
  - Open a transaction
  - Copy files to the correct path in `/cvmfs/`
  - Publish and close transaction[1]
- Use this cache directory in CVMFS as read-only in the `DEPOT_PATH` list

[1] This would make publishing to CVMFS for Julia $\leq$ 1.10 tricky

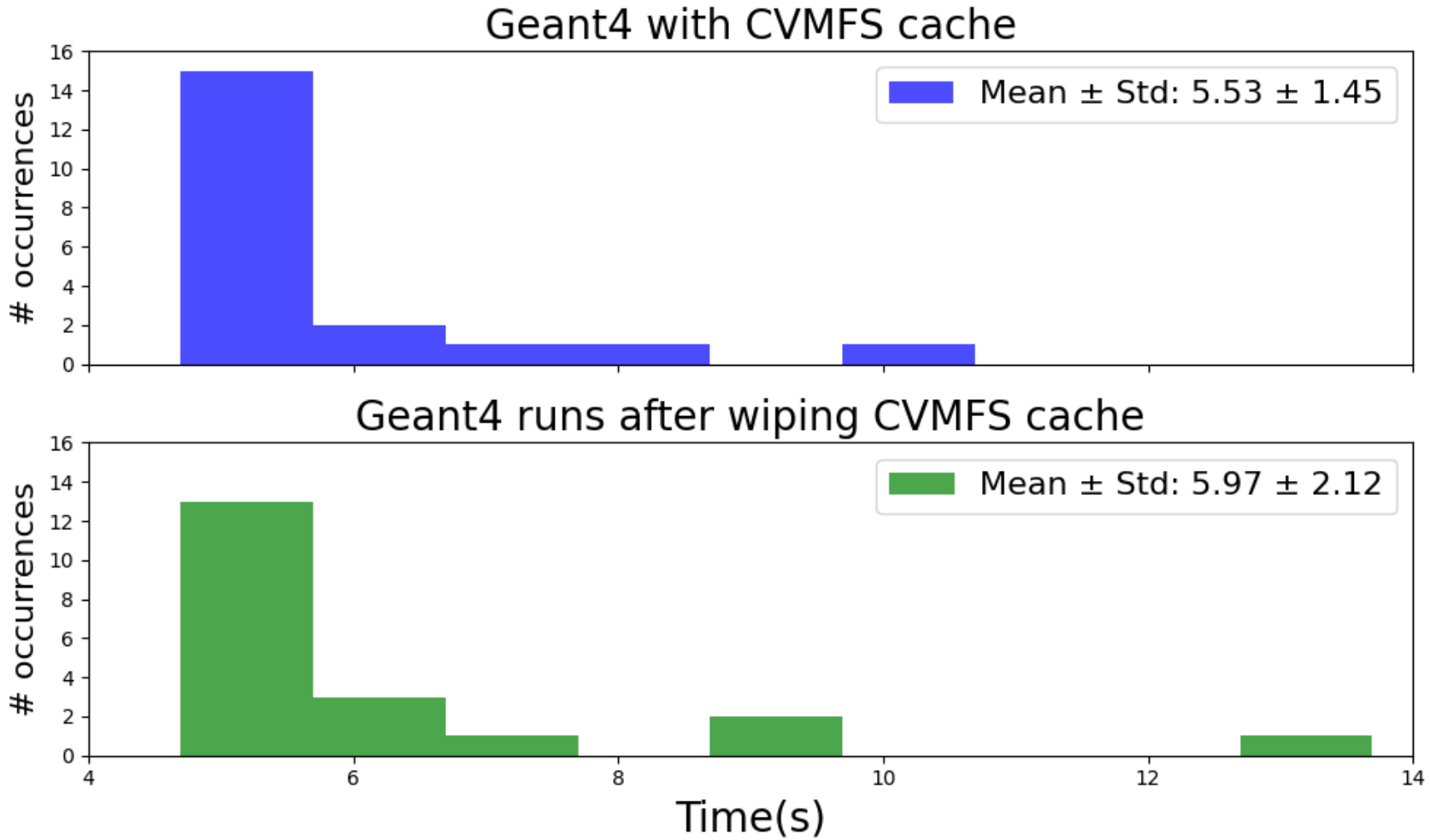
Testing this framework with two example workflows

We selected two use cases to test the influence of precompilation caching:

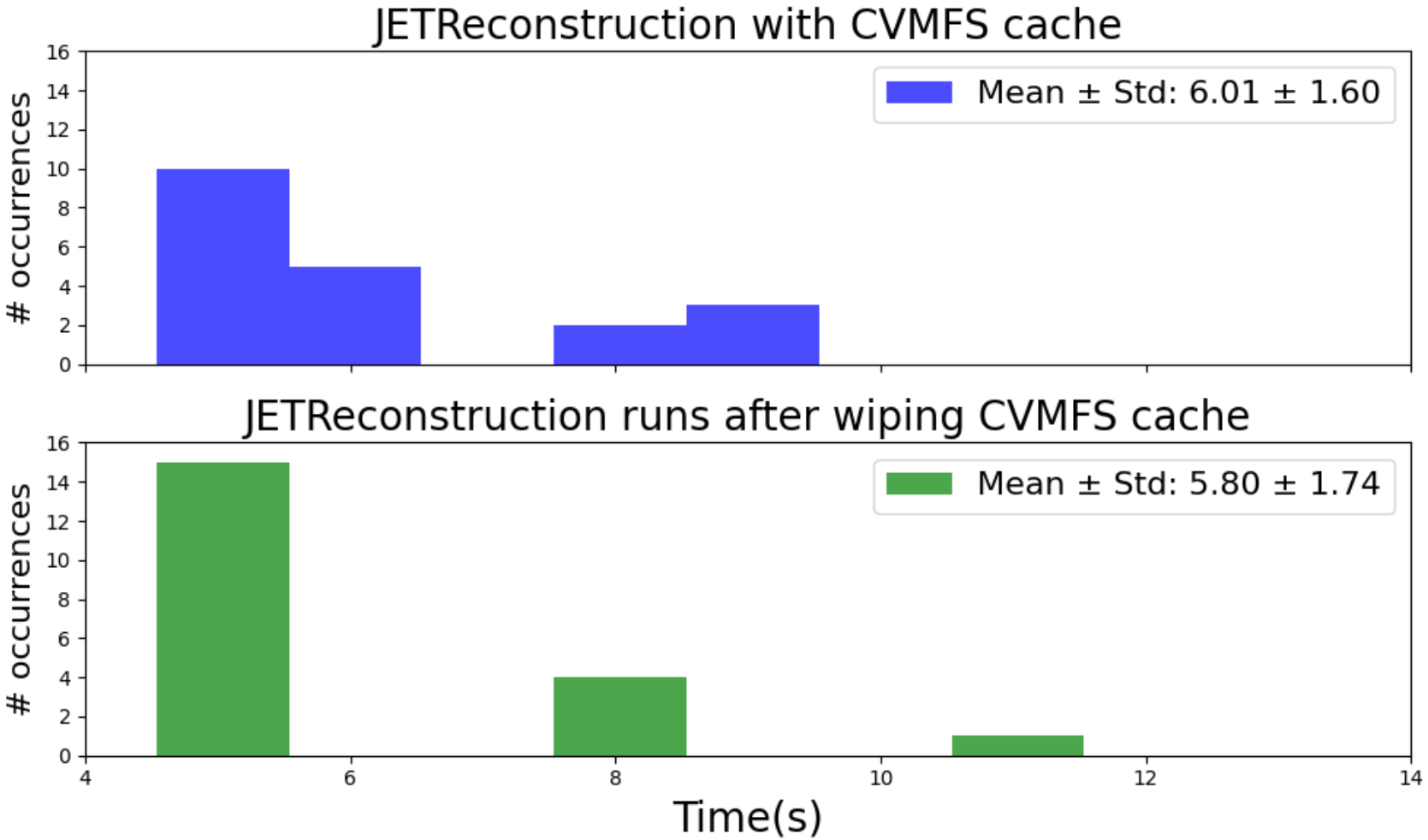
	JET	Geant4
Without cache	90 ± 7	270 ± 80
With CVMFS Cache	6 ± 2	6 ± 1

Table: Time in seconds it takes for Julia to start running the workflow.

Testing Julia's performance integrated with CVMFS



Results (3): Testing Julia's performance integrated with CVMFS



Automating relocation of cache files

Thanks to JuliaCon2024:

: DepotDelivery bundles a Julia project into a standalone depot that can run in air-gapped environments.

Automating relocation of cache files

Thanks to JuliaCon2024:

: DepotDelivery bundles a Julia project into a standalone depot that can run in air-gapped environments.

We proceeded with a PR for our use case:

- Added support for multiple Project.toml files.
- Added support for precompilation of workflows.

Summary

- ✓ Julia was found to greatly reduce startup time by making use of precompiled objects for multiple micro-architectures.
- ✓ Julia is able to integrate to CernVM-FS with virtually no cost in performance.
- ✓ We contributed to a ready-to-use julia package to automatically populate different applications into directory.


CERN