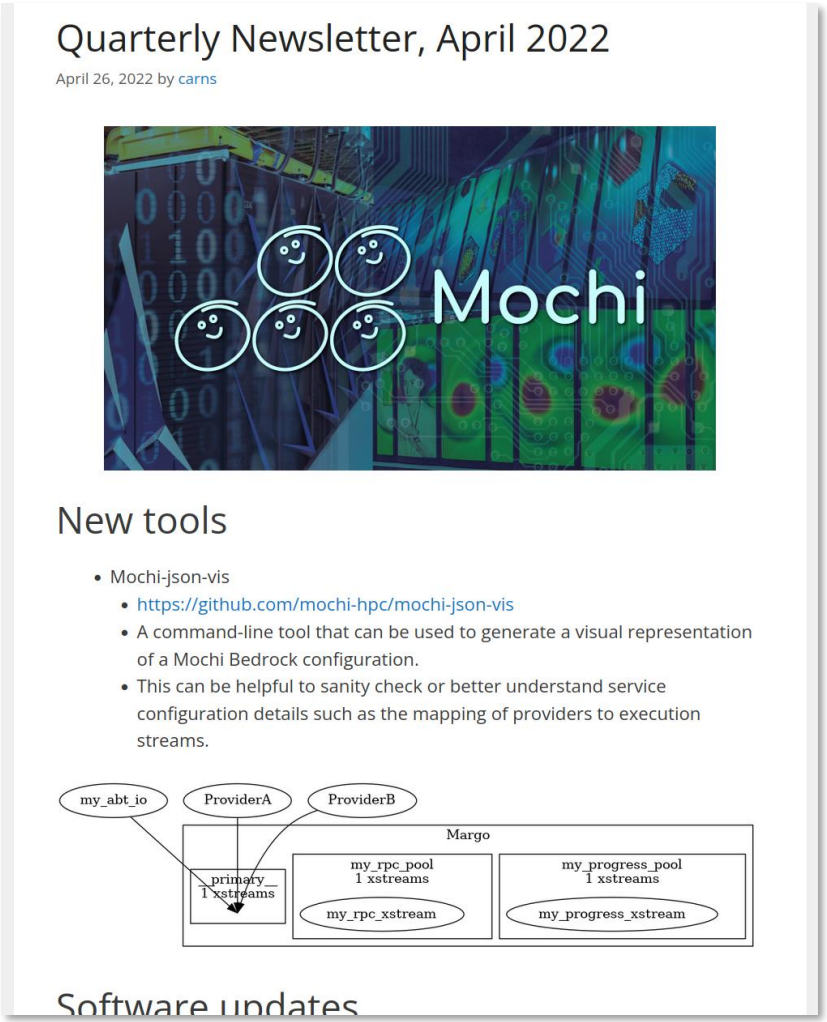
Getting Started with Mochi
&
Recent Updates

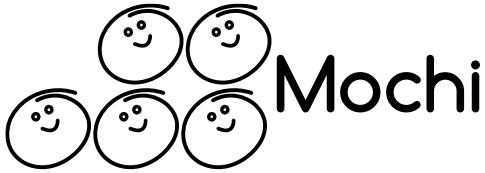
Where to begin

- Start here for documentation:
  - https://mochi.readthedocs.io/

- Don’t hesitate to contact us if you have questions!
  - Ask questions on the mailing list or Slack space.
  - https://www.mcs.anl.gov/research/projects/mochi/
- We also have quarterly meetings for users and developers.
  - Join our mailing list to receive meeting notifications and request agenda items for discussion.
  - We plan to start hosting short presentations on focused topics this year.
  - Each quarterly meeting also has a corresponding newsletter:



Installing Mochi with Spack



We strongly recommend using Spack to install any Mochi components.

- Spack simplifies dependency resolution (recall that Mochi is inherently made up of many components).
- No special privileges are needed to install any Mochi components.
- We provide an external package repository that enables rapid integration of new Mochi component releases without updating Spack itself.
  - https://github.com/mochi-hpc/mochi-spack-packages/tree/main/packages
- We also provide build recipes for several platforms.
  - These are expressed as Spack environment configuration files.
  - One unified spack.yaml file contains all preferred build settings (e.g., network transport, compiler, storage backend) for a given platform.
  - The recipes also include job script examples with suggested runtime environment settings where applicable.
  - https://github.com/mochi-hpc-experiments/platform-configurations

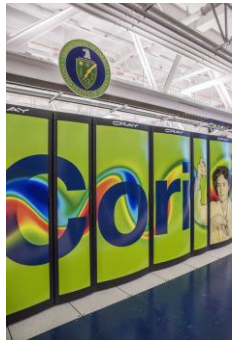
Platform support

```
carns-x1-7g ~/w/s/m/platform-configurations (main=)> find . -name spack.yaml
./ORNL/Crusher/spack.yaml
./ORNL/Summit/spack.yaml
./NERSC/Cori/spack.yaml
./NERSC/Perlmutter/spack.yaml
./ANL/JLSE/spack.yaml
./ANL/Cooley/spack.yaml
./ANL/Theta/spack.yaml
./ANL/Bebop/spack.yaml
carns-x1-7g ~/w/s/m/platform-configurations (main=)> head -n 20 ./ORNL/Crusher/spack.yaml
spack:
  specs:
  - mochi-margo
  concretization: together
  repos:
  - /path/to/mochi-spack-packages
  packages:
    libfabric:
      externals:
      - spec: libfabric@1.15.0.0
        modules:
        - libfabric/1.15.0.0
      buildable: true
      version: []
      target: []
      providers: {}
      compiler: []
    mercury:
      variants: ~boostsys ~checksum
      buildable: true
carns-x1-7g ~/w/s/m/platform-configurations (main=)> tail -n 1 ./ORNL/Crusher/job.sbatch
srun -n 2 --ntasks-per-node=1 /ccs/home/carns/working/install-crusher/bin/margo-p2p-bw -x 8388608 -n "cxi://" -c 8 -D 20
```

Available example configurations

Spack environment file specifying system packages to use and recommended options

Runtime job example



Platform support

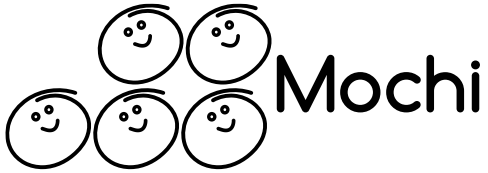
```
carns-x1-7g ~/w/s/m/platform-configurations (main=)> find . -name spack.yaml
./ORNL/Crusher/spack.yaml
./ORNL/Summit/spack.yaml
./NERSC/Cori/spack.yaml
./NERSC/Perlmutter/spack.yaml
./ANL/JLSE/spack.yaml
./ANL/Cooley/spack.yaml
./ANL/Theta/spack.yaml
./ANL/Bebop/spack.yaml
carns-x1-7g ~/w/s/m/platform-configurations (main=)> head -n 20 ./ORNL/Crusher/spack.yaml
spack:
  specs:
  - mochi-margo
  concretization: together
  repos:
  - /path/to/mochi-spack-packages
  packages:
    libfabric:
      externals:
      - spec: libfabric@1.15.0.0
        modules:
        - libfabric/1.15.0.0
      buildable: true
      version: []
      target: []
      providers: {}
      compiler: []
    mercury:
      variants: ~boostsys ~checksum
      buildable: true
carns-x1-7g ~/w/s/m/platform-configurations (main=)> tail -n 1 ./ORNL/Crusher/job.sbatch
srun -n 2 --ntasks-per-node=1 /ccs/home/carns/working/install-crusher/bin/margo-p2p-bw -x 8388608 -n "cxi://" -c 8 -D 20
```

Early access system with HPE Slingshot 11 (CXI transport)

Phase 1 system with HPE Slingshot 10 (Verbs transport)

We will also track Aurora and other platforms as early access opens up.

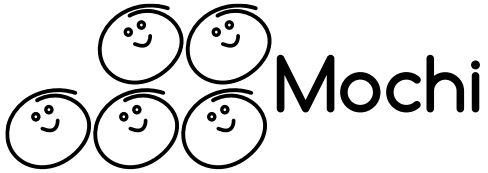
Mochi source code



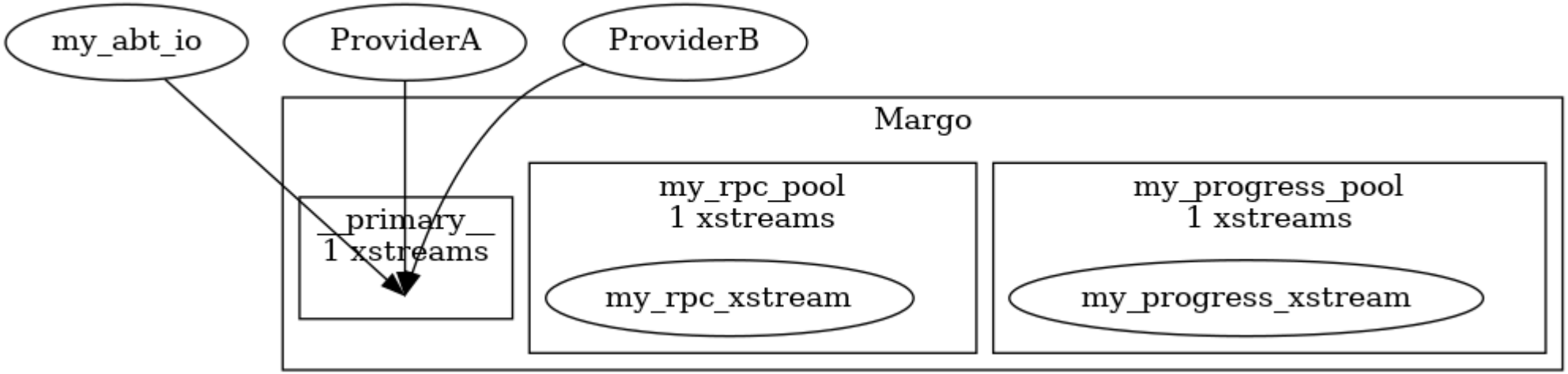
- All Mochi source code is available on github.com.
  - https://github.com/mochi-hpc/
  - The Mochi software is actually a collection of components maintained in separate repositories.
  - Bug reports and contributions are welcome!
  - PR contributors will be prompted automatically for contributor licensing information.

What should you do once you have a Mochi service up and running and you want to understand it better?

Things to do once your service is running: Validate service layout



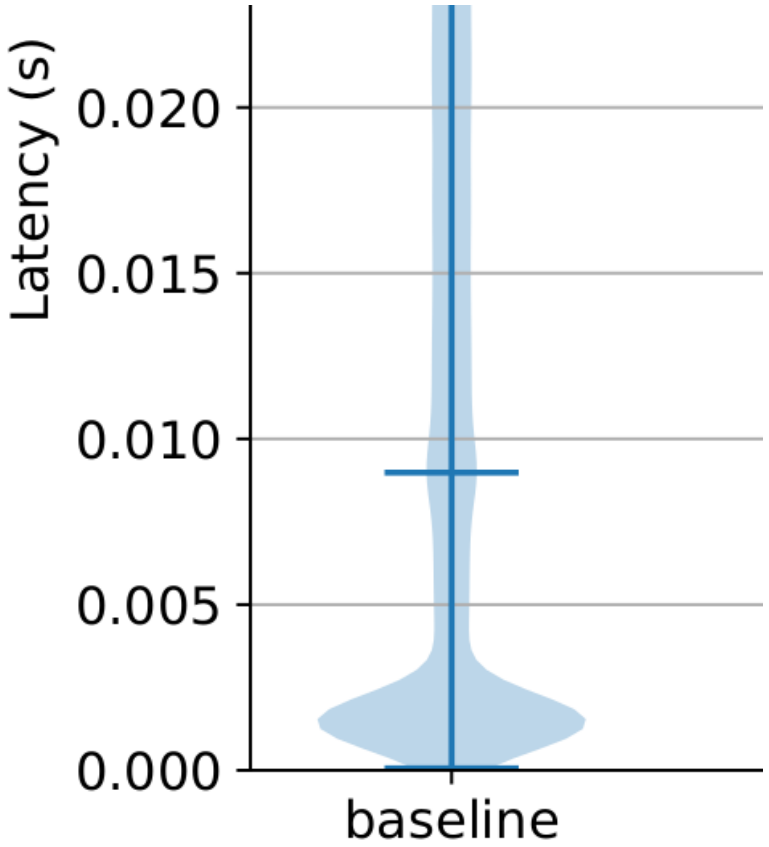
- The Mochi “Bedrock” component can be used to configure and bootstrap combinations of services with json or jx9 configuration files; we’ll see examples later in this BoF.
- How to do you validate / visualize a configuration, though?
- Try mochi-json-vis (https://github.com/mochi-hpc/mochi-json-vis)



It translates the json configuration into a visual mapping of services (i.e., Mochi providers) to execution resources. See README.md for an explanation of the figures.

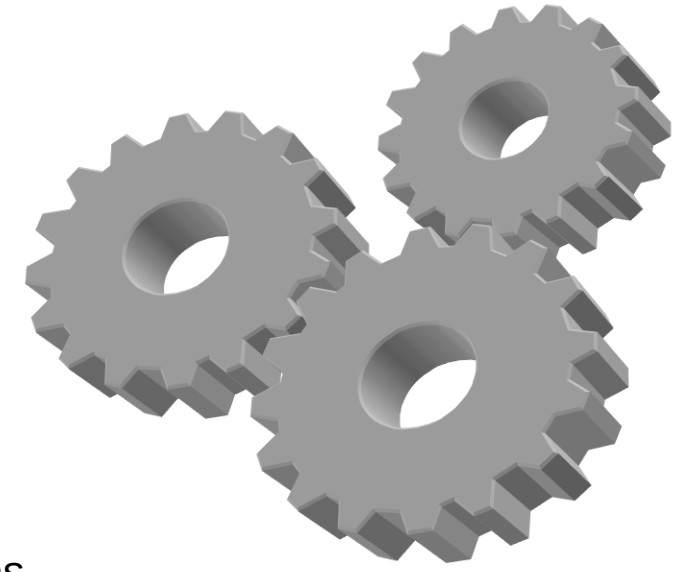
Things to do once your service is running: Integrate self-diagnostics

- The Mochi-quintain provider is a vehicle for adding synthetic workload processing to an existing service.
- https://github.com/mochi-hpc/mochi-quintain
- You can then run parameterized benchmarks against the Quintain component to help isolate performance phenomena in your system.
- The example on the right shows the distribution of response times for RPCs to a single server under load.
- Work in progress! We will improve Quintain benchmarking and visualization capabilities over time.



Things to do once your service is running: investigate performance

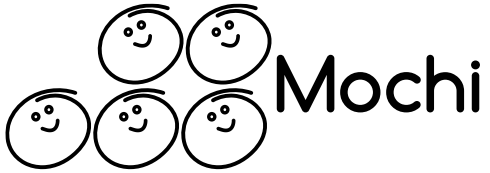
- How do you tune the performance of a Mochi service?
  - Step 1: Use the best (native) network transport for your platform
  - Step 2: Use Mochi diagnostic and profiling tools* to understand where service time is spent
- Basic performance diagnostic and profiling capability is already built into *any* Mochi service.
  - No need to modify or recompile application or service
  - Automatically tracks Mochi RPCs
  - Automatically tracks RPC dependencies
  - Includes intra-node, inter-node, and inter-process calls



* Functionality developed by Srinivasan Ramesh of U. Oregon, see:

SYMBIOSYS: A Methodology for Performance Analysis of Composable HPC Data Services
Srinivasan Ramesh, Allen D. Malony, Philip Carns, Robert B. Ross, Matthieu Dorier, Jerome Soumagne, and Shane Snyder (to appear in IPDPS 2021)

Enable profiling of an existing service



- Set environment variables to enable profiling
- Run your service / application
- Generate profile summary
- Render RPC dependency graph
- Look at the results

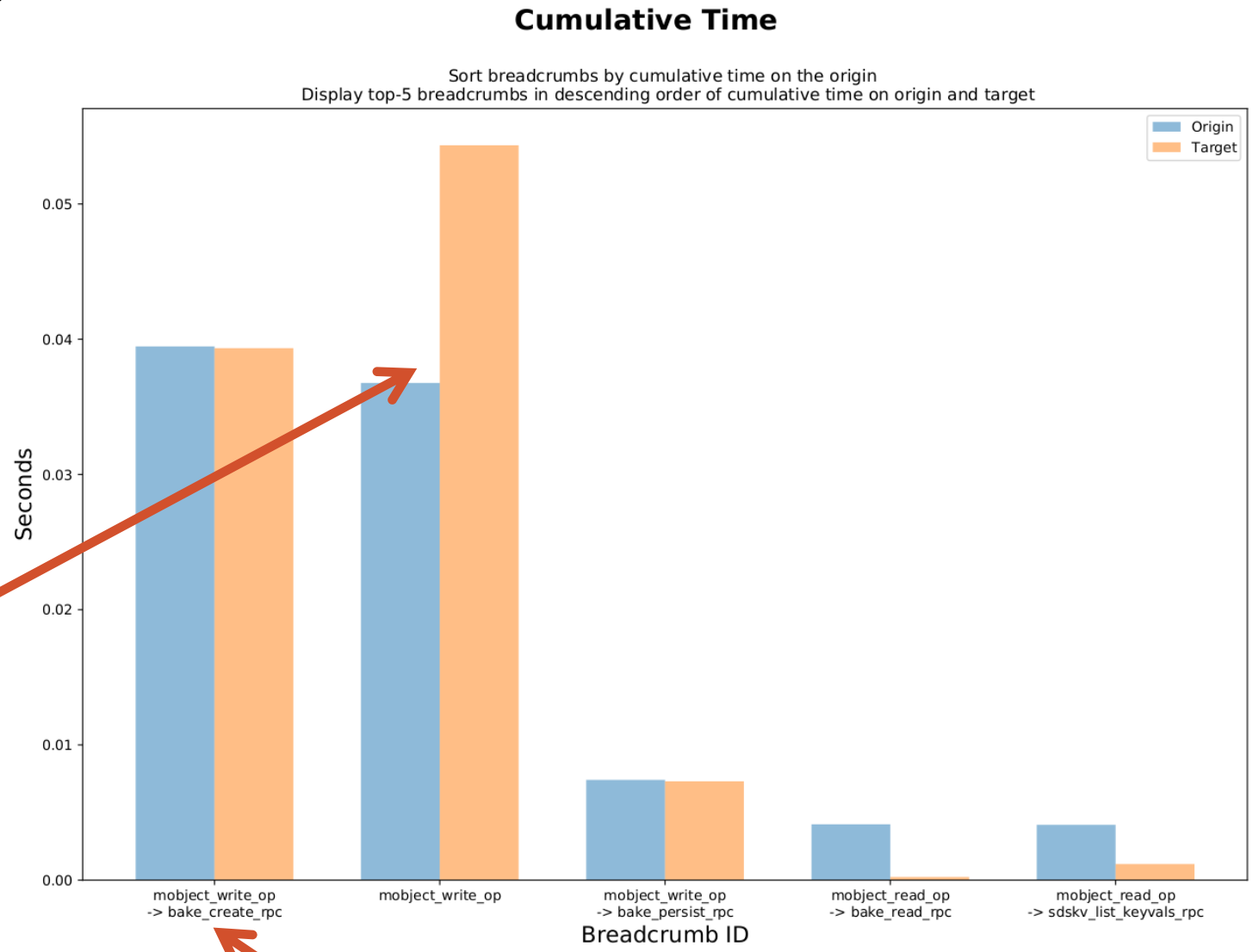
```
> export MARGO_ENABLE_PROFILING=1
> export MARGO_ENABLE_DIAGNOSTICS=1

# run example

> margo-gen-profile
********************MARGO Profile Generator*******************
Reading CSV files from: /home/carns/
Done.
************************************************************

> dot -Tpdf graph.gv -o graph.pdf

> ls *.pdf
graph.pdf  profile.pdf
```

See README.md in mochi-margo for more information

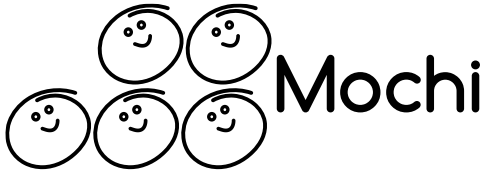
Example: How much cumulative time was spent in each RPC?

This graph shows the top 5 RPCs in terms of cumulative time.

Unusual example: target (server) side of this RPC consumed more time than clients, indicating presence of completion delay after sending ack.



RPCs are labeled by name, with a chain showing call provenance



Thank you!