

TaPS: A Performance Evaluation Suite for Task-based Execution Frameworks

J. Gregory Pauloski,^{*†} Valerie Hayot-Sasson,^{*†} Maxime Gonthier,^{*†} Nathaniel Hudson,^{*†} Haochen Pan,^{*} Sicheng Zhou,^{*} Ian Foster,^{*†} and Kyle Chard^{*}

^{}University of Chicago, [†]Argonne National Laboratory*

19 September 2024 – Osaka, Japan



Enabling eScience Applications



Better Benchmarking



TaPS: Task Performance Suite



Evaluation *Exploration*



Let's Put the Science in eScience

Carl Kesselman

Information Sciences Institute
University of Southern California

Robert Schuler

Information Sciences Institute
University of Southern California

Ian Foster

University of Chicago, Chicago, USA
Argonne National Lab, Lemont, USA

Abstract—The underlying premise behind eScience is that computational methods and data-driven approaches can contribute to scientific discovery on a par with, or even superior to, traditional experimental methods; that the combination of computers, software, and extant data collections are the modern equivalent to the scientific instruments that have led to our understanding of fundamental laws in physics, chemistry, biology, and other domains. However, a robust methodology for making the results of eScience activities “scientific” is lacking, with significant consequences. In this brief paper we propose a shift in perspective as to what it means to create an eScience-based result and how the scientific validity of eScience experiments might be improved.

Index Terms—eScience, scientific method, reproducability, data management

II. A CHANGE OF PERSPECTIVE

Errors such as those described above persist in spite of the use of widely accepted tools such as shared storage systems, workflow management tools, and software and data repositories. We argue here that these persistent problems are a consequence not of inherent deficiencies in the tools used, but of methodologies that are overly focused on documenting the process that was followed to produce published results (e.g., software stack, the sequence of computational steps taken, the repository where results are placed). This narrow focus

“The underlying premise behind **eScience** is that **computational methods** and **data-driven approaches** can contribute to scientific discovery on a par with, or even superior to, traditional experimental methods”



eScience Paradigms (non-exhaustive)

Machine Learning

Text Summarization towards Scientific Information Extraction

Asynchronous Decentralized Bayesian Optimization for Large Scale Hyperparameter Optimization

FLoX: Federated Learning with FaaS at the Edge

Nikita Kotsehub*, Matt Baughman[†], Ryan Chard[‡], Nathaniel Hudson^{†‡}, Panos Patros[§], Omer Rana[¶], Ian Foster^{†‡}, Kyle Chard^{†‡}

Romain Egele
Université Paris-Saclay, France & Argonne National Laboratory, USA
romain.egele@universite-paris-saclay.fr

Isabelle Guyon
Venkatesh Vishwanath
Leadership Computing Facility Argonne National Laboratory, USA
venkat@anl.gov

Books Part of Computing at University of Illinois Urbana-Champaign
3rd Daniela Rusu
School of Computing DePaul University
4th Peter Hastings
School of Computing DePaul University
5th Roseanne Teeklo
School of Computing DePaul University
Romain Egele
Université Paris-Saclay, France & Argonne National Laboratory, USA
romain.egele@universite-paris-saclay.fr

3rd Daniela Rusu
School of Computing DePaul University
4th Peter Hastings
School of Computing DePaul University
5th Roseanne Teeklo
School of Computing DePaul University
Romain Egele
Université Paris-Saclay, France & Argonne National Laboratory, USA
romain.egele@universite-paris-saclay.fr

Books Part of Computing at University of Illinois Urbana-Champaign
3rd Daniela Rusu
School of Computing DePaul University
4th Peter Hastings
School of Computing DePaul University
5th Roseanne Teeklo
School of Computing DePaul University
Romain Egele
Université Paris-Saclay, France & Argonne National Laboratory, USA
romain.egele@universite-paris-saclay.fr

Distributed/High-Perf. Computing

Lazy Python Dependency Management in Large-scale systems

PSI/J: A Portable Interface for Submitting, Monitoring, and Managing Jobs

Federated Function as a Service for eScience

Yadu Babuji*, Josh Bryan*, Ryan Chard*, Kyle Chard*, Ian Foster*, Ben Galewsky*, Daniel S. Katz*, Zhuozhao Li*

*University of Chicago, +University of Illinois at Urbana-Champaign

Alok Kumar
Department of Computer Science University of Chicago Chicago, IL, USA

Marii Sakaravala
Department of Computer Science University of Chicago Chicago, IL, USA

Valerie Frasconi-Sorenson
Department of Computer Science University of Chicago Chicago, IL, USA

Kyle Chard
Department of Computer Science University of Chicago Chicago, IL, USA

Ian Foster
Department of Computer Science University of Chicago Chicago, IL, USA

Mihael Hategan-Marandiu^{1,4}, Andre Merzky³, Nicholson Collier^{1,4}, Ketan Maheshwari⁶, Jonathan Ozik^{1,4}, Matteo Turlili^{3,5}, Andreas Wilke^{1,4}, Justin M. Wozniak⁴, Kyle Chard^{1,4}, Ian Foster^{1,4}, Rafael Ferreira da Silva⁶, Shanetur Jha^{3,5}, Daniel Lane²

Data-Driven/Provenance

Database Evolution, by Scientists, for Scientists: A Case Study

A Method for Constructing Research Data Provenance in High-Performance Computing Systems

Can Automated Metadata Extraction make Scientific Data More Navigable?

Tyler J. Skluzacek
National Center for Computational Sciences
Oak Ridge National Lab, Oak Ridge, TN
skluzacek@ornl.gov

Kyle Chard, Ian Foster
Department of Computer Science
University of Chicago, Chicago, IL
Data Science and Learning Division
Argonne National Lab, Lemont, IL

Yuta Namiki^{1*}, Takeo Horie^{1*}, Hideyuki Tanabu¹, Akifumi Yamashita¹
¹Joint Research Laboratory for Integrated Infrastructure of High Performance Computing
Cybermedia Center Osaka University, Osaka, Japan
²NEC Corporation, Tokyo, Japan
Email: yuta.namiki@nec.com

Computational Workflows

Extreme Scale Survey Simulation with Python Workflows

Running Ensemble Workflows at Extreme Scale: Lessons Learned and Path Forward

WfChef: Automated Generation of Accurate Scientific Workflow Generators

Towards Lightweight Data Integration using Multi-workflow Provenance and Data Observability

Kshitiji Mehta¹
14-9981 ● 0000-0001-7809-5546 ● 0000-0003-1902-1955 ● Angelica M. Walker¹
Ashley Cliff^{1†}
Daniel Jacobson^{1*}
Frédéric Suter¹
436 ● 0000-0002-9822-8251 ● Scott Klasky¹
● 0000-0003-3559-5772

Antonia Sierra Villarreal*, Yadu Babuji¹, Tom Uram*, Daniel S. Katz*
(The LSST Dark Energy Science Collaboration)

Taina Coleman*, Henri Casanova^{1,4}
¹Information Sciences Institute, University of Southern California
⁴Information and Computer Sciences, University of California, Irvine
{tcoleman, casanova}@isi.edu, hcasanova@uci.edu

Renan Souza, Tyler J. Skluzacek, Sean R. Wilkinson, Maxim Ziatdinov, Rafael Ferreira da Silva
National Center for Computational Sciences, Oak Ridge National Lab, Oak Ridge, TN, USA
{souzar, skluzacekj, wilkinsonr, ziatdinovm, silvarf}@ornl.gov

Papers featured in eScience 2021–2023



THE UNIVERSITY OF
CHICAGO

Enabling eScience Applications | 4

globus labs

Modern eScience Applications are *Task-centric*

Applications are composed as a set of **discrete tasks** designed to **automate** computational processes to achieve a **scientific goal**

Benefits

- Heterogeneous Resources
- Software Modularity
- Monitoring
- Performance
- Reproducibility
- *and many more!*

Applications [1]

- Bioinformatics
- Cosmology
- High Energy Physics
- Materials Science
- Molecular Dynamics
- *and many more!*

Challenges [2]

- Coupling AI/ML/Quantum
- Cloud and HPC Integration
- Data Flow/Provenance
- Standards/Interoperability
- Performance
- *and many more!*

[1] "Scientific Workflows: Moving Across Paradigms" (<https://dl.acm.org/doi/10.1145/3012429>)

[2] Workflows Community Summit (<https://arxiv.org/abs/2304.00019>)

How do we build and execute task-based eScience applications?



Task Execution Frameworks

Manage the execution of tasks in parallel across arbitrary hardware.

Task Execution Frameworks

Workflow Management Systems

Define, manage, and execute workflows represented by a directed acyclic graph (DAG) of tasks

Explicit

DAG defined via configuration file or domain specific language



Implicit

Task dependencies derived through dynamic evaluation of a procedural script



Concurrent Executors

On-demand asynchronous execution of tasks





Enabling eScience Applications



Better Benchmarking



TaPS: Task Performance Suite



Evaluation *Exploration*

The Status Quo

Ad Hoc Benchmarks

- Extensions of framework-specific examples or demos
- One-off/custom evaluation scripts for a publication
- Forks of real science applications

Problems

- Code is **framework-specific**
- Ad-hoc scripts subject to **code rot**
- Porting applications can be **onerous**
- Subtle **errors** in ported applications can lead to **inaccurate comparisons**

SimGrid: a Generic
Large-Scale Distribut

Developing accurate and scalable sim
management systems with WRENCH

Application skeletons: Construction and use in

Daniel S. Katz^{a,*}, Andre Merzky^b, Zhao Zhang^c, Shantenu Jha^b

^a Computation Institute, University of Chicago & Argonne National Laboratory, Chicago, IL, USA

^b RADICAL Laboratory, Rutgers

^c AMPLab, University of Califor

WfCommons: A Framework for Enabling Scientific Workflow Research and Development

Tainā Coleman^{a,c,*}, Henri Casanova^b, Loïc Pottier^a, Manav Kaushik^c, Ewa Deelman^{a,c}, Rafael Ferreira da Silva^{a,c,*}

WfBench: Automated Generation of
Scientific Workflow Benchmarks

Tainā Coleman*, Henri Casanova[†] Ketan Maheshwari[‡], Loïc Pottier*, Sean R. Wilkinson[‡]

Justin Wozniak[§], Frédéric Suter[‡], Mallikarjun Shankar[‡], Rafael Ferreira da Silva[‡]

University of Hawaii, Honolulu, HI, USA

Oak Ridge National Laboratory, Oak Ridge, TN, USA

Prior work focused on **simulations** and **synthetic workloads**



THE UNIVERSITY OF
CHICAGO

We lack a **standardized** set of real applications/workloads for benchmarking task executors.



Drawing Inspiration from Other Fields

Systems

The LINPACK Benchmarks are a measure of a system's floating-point computing power. Introduced by Jack Dongarra, they measure how fast a computer solves a dense $n \times n$ system of linear equations $Ax = b$, which is a common task in engineering.



TPC Transaction Processing Performance Council

byte-unixbench

UnixBench is the original BYTE UNIX benchmark suite, updated and revised by many people over the years.

AI/ML



FaaS



FunctionBench

FAASDOM: A Benchmark Suite for Serverless Computing

Pascal Maissen
Department of Computer Science
University of Fribourg, Switzerland
pascal.maissen@unifr.ch

Peter Kropf
Department of Computer Science
University of Neuchâtel, Switzerland
peter.kropf@unine.ch

Pascal Felber
Department of Computer Science
University of Neuchâtel, Switzerland
pascal.felber@unine.ch

Valerio Schiavoni
Department of Computer Science
University of Neuchâtel, Switzerland
valerio.schiavoni@unine.ch

Goals of a Good Benchmark

- Objective metrics
- Facilitate meaningful comparison
- Transparency and reproducibility

- Common ground
- Democratize research
- Accelerate advancements

SeBS: The Serverless Benchmark Suite
Evaluate supported serverless platforms with real-world benchmarks.
Performance & Cost
Invocation Overhead
Container Eviction
Serverless Communication*
Serverless Workflows*
GPU Functions*
*Experimental and work-in-progress features.



THE UNIVERSITY OF
CHICAGO



Enabling eScience Applications



Better Benchmarking



TaPS: Task Performance Suite



Evaluation *Exploration*

TaPS: Task Performance Suite

A standardized framework for evaluating
task execution frameworks with real and synthetic
science applications



TaPS: Task Performance Suite

- Audience
- Architecture
- Applications
- Framework
- Plugin System
- Using TaPS



Audience

Systems Software Developers & Application Builders

Anyone with questions like:

- How do I evaluate my:
 - ◆ distributed execution framework?
 - ◆ data management system?
 - ◆ modifications to existing systems?
- What are the performance characteristics of prior work?
- Which task executor performs best for similar workloads to mine?

Architecture

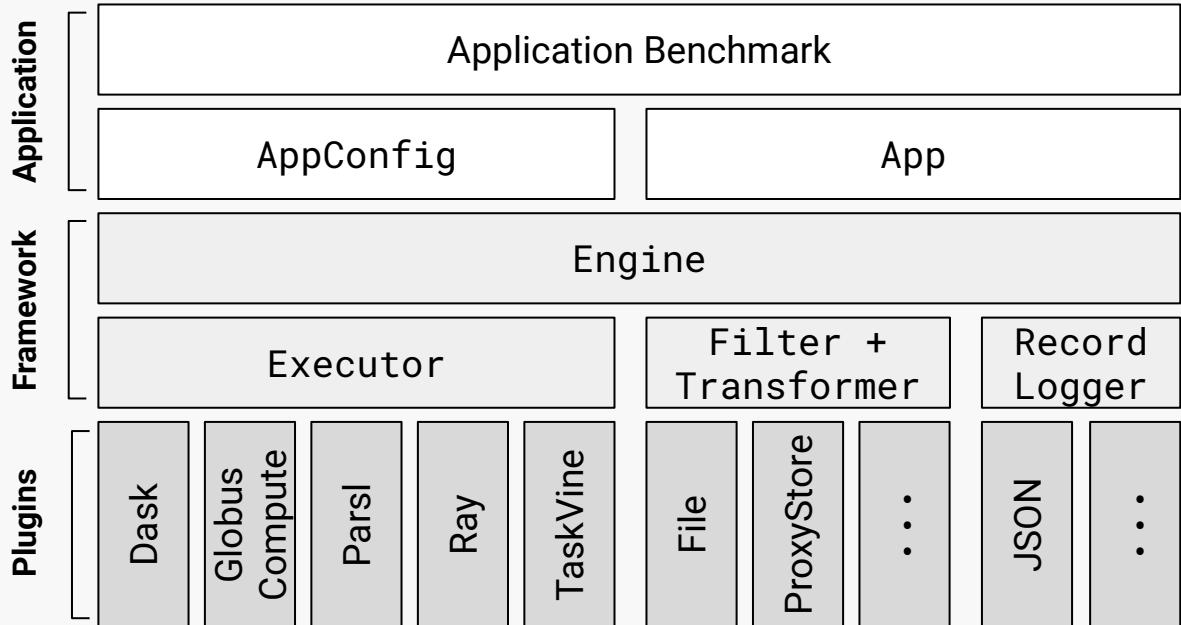
Use reference Apps
or add your own



Engine: Glue that
integrate Apps with
Engine Plugins



Use provided Engine
Plugins or your own



<https://taps.proxystore.dev/latest/api/>



THE UNIVERSITY OF
CHICAGO

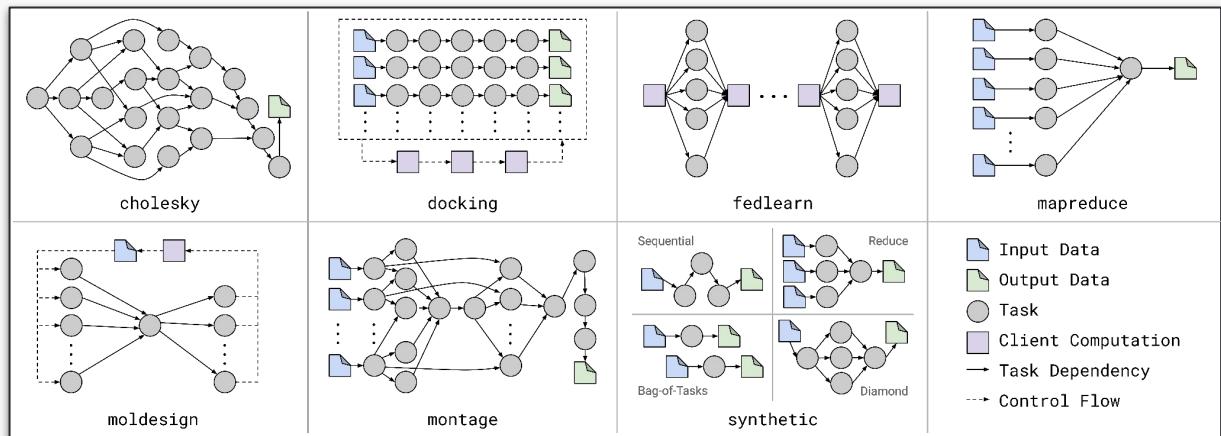
TaPS: Task Performance Suite | 17

globus  labs

Applications

- Six Real Apps
- Two Synthetic
- Diverse Patterns
- Diverse Domains
- Configurable
- Per-App Guides

Type	Name	Domain	Task Type(s)	Data Type(s)
Real	cholesky	Linear Algebra	Python	In-memory
	docking	Drug Discovery	Executable, Python	File
	fedlearn	Machine Learning	Python	In-memory
	mapreduce	Text Analysis	Python	File, In-memory
	moldesign	Molecular Design	Python	In-memory
	montage	Astronomy	Executable	File
Synthetic	synthetic	—	Python	In-memory
	failures	—	Depends on base app	Depends on base app



<https://taps.proxystore.dev/latest/apps/>

How to design an interface **expressive** enough to build these applications but simple enough to **unify** task executors?



Task Execution Frameworks

Workflow Management Systems

Define, manage, and execute workflows represented by a directed acyclic graph (DAG) of tasks

Explicit

DAG defined via configuration file or domain specific language

Implicit

Task dependencies derived through dynamic evaluation of a procedural script

Concurrent Executors

On-demand asynchronous execution of pure tasks

Why are explicit WMSs not Supported?

- Static DAGs not expressive enough for dynamic/procedural applications
- DSLs are tightly coupled/unique to WMS
- Possible but requires complex per-DSL parsing and code generation

Engine

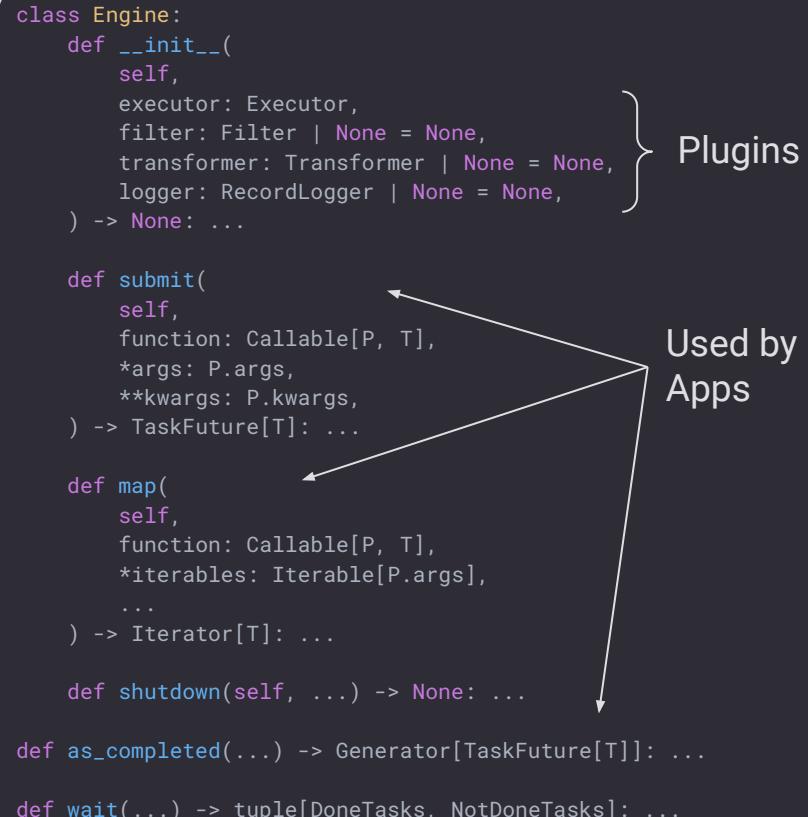
Interface between Apps and Plugins

- Apps submit tasks to Engine and gets back a TaskFuture
- TaskFuture can be an argument for other tasks (implicit data flow dependency)
- Engine invokes plugins (e.g., submit task to Executor)

Protocol: concurrent.futures Executor

- Closest to “standard” in Python ecosystem
- Easy to port existing apps using an Executor
- Protocol extended to require implicit data flow dependencies via futures

```
class Engine:  
    def __init__(  
        self,  
        executor: Executor,  
        filter: Filter | None = None,  
        transformer: Transformer | None = None,  
        logger: RecordLogger | None = None,  
    ) -> None: ...  
  
    def submit(  
        self,  
        function: Callable[P, T],  
        *args: P.args,  
        **kwargs: P.kwargs,  
    ) -> TaskFuture[T]: ...  
  
    def map(  
        self,  
        function: Callable[P, T],  
        *iterables: Iterable[P.args],  
        ...  
    ) -> Iterator[T]: ...  
  
    def shutdown(self, ...) -> None: ...  
  
    def as_completed(...) -> Generator[TaskFuture[T]]: ...  
  
    def wait(...) -> tuple[DoneTasks, NotDoneTasks]: ...
```



Engine Plugins — Task Execution

Interface: Executor*

```
class Executor:
    def submit(
        self,
        function: Callable[P, T],
        *args: P.args,
        **kwargs: P.kwargs,
    ) -> Future[T]: ...

    def map(
        self,
        function: Callable[P, T],
        *iterables: Iterable[P.args],
        ...
    ) -> Iterator[T]: ...

    def shutdown(self, ...) -> None: ...
```

*Requires support for implicit data via futures. Wrapper provided for implementations that lack this feature.

Purpose: Asynchronously execute functions

Implementations:

- ThreadPool
- ProcessPool
- Dask
- Globus Compute
- Parsl
- Ray
- TaskVine

Future Extensions:

- Cloud FaaS
- New Executors



Engine Plugins — Data Management

Interface: Filter and Transformer

```
class Filter:
    def __call__(self, obj: Any) -> bool:
        ...

class Transformer(Generic[IdentifierT]):
    def is_identifier(self, obj: T) -> bool:
        ...

    def transform(self, obj: T) -> IdentifierT:
        ...

    def resolve(self, id_: IdentifierT) -> Any:
        ...
```

Purpose: Manage task data by filtering and transforming data into/resolve data from intermediate representations

Implementations:

- Shared File Systems
- ProxyStore (DAOS, Globus Transfer, Margo, Redis, UCX, ZMQ, ...)

Future Extensions:

- Cloud Storage



Engine Plugins — Task Logging

Interface: RecordLogger

```
Record: TypeAlias = Dict[str, Any]

class RecordLogger:
    def log(self, record: Record) -> None:
        ...
```

Purpose: Record task execution traces

Implementations:

- JSON

Future Extensions:

- Databases
- WfTrace format



Adding an Engine Plugin

- Config types for each plugin
- Contains all user-controllable parameters (optional defaults)
- `@register(<type>)` decorator
 - ◆ Registers plugin type with TaPS
 - ◆ Plugin name and parameters exposed in CLI choices / config file parser
 - ◆ Parameter validation auto-generated from fields
- `get_<type>()` used by TaPS

```
import globus_compute_sdk
from concurrent.futures import Executor
from pydantic import Field
from taps.executor import ExecutorConfig
from taps.executor.utils import FutureDependencyExecutor
from taps.plugins import register

@register('executor')
class GlobusComputeConfig(ExecutorConfig):
    """Globus Compute Executor plugin configuration."""

    name: Literal['globus'] = Field('globus', description='Name.')
    endpoint: str = Field(description='Endpoint UUID.')
    batch_size: int = Field(128, description='Batch size.')

    def get_executor(self) -> Executor:
        """Initialize an executor from the config."""
        executor = globus_compute_sdk.Executor(
            self.endpoint,
            batch_size=self.batch_size,
        )
        return FutureDependencyExecutor(executor)
```

Utility for adding implicit data flow support to any executor

<https://taps.proxystore.dev/latest/guides/executor/>

Adding an Application

```
from typing import Literal
from pydantic import Field
from taps.apps import App, AppConfig
from taps.plugins import register

@register('app')
class CholeskyConfig(AppConfig):
    """Cholesky application configuration."""

    name: Literal['cholesky'] = Field('cholesky', ...)
    matrix_size: int = Field(description='Matrix size.')
    block_size: int = Field(description='Block/tile size.')

    def get_app(self) -> App:
        """Create an application instance from the config."""
        from taps.apps.cholesky import CholeskyApp

        return CholeskyApp(
            matrix_size=self.matrix_size,
            block_size=self.block_size,
        )
```

Config

```
import pathlib
from taps.engine import Engine

class CholeskyApp:
    """Cholesky decomposition application."""

    def __init__(self, matrix_size: int, block_size: int) -> None:
        self.matrix_size = matrix_size
        self.block_size = block_size

    def close(self) -> None:
        """Clean up and close the application."""
        pass

    def run(self, engine: Engine, run_dir: pathlib.Path) -> None:
        """Run the application."""
        future = engine.submit(func, *args, **kwargs)
        future.result()
```

App



Application logic goes inside run() and interfaces with Engine

<https://taps.proxystore.dev/latest/guides/apps/>



THE UNIVERSITY OF
CHICAGO

TaPS: Task Performance Suite | 26

globus labs

Using TaPS

Execute benchmarks with CLI or programmatically via API

```
$ python -m taps.run \
--app cholesky --app.matrix-size 10000 --app.block-size 1000 \
--engine.executor dask --engine.executor.workers 16 \
--engine.transformer proxystore {transformer options} \
--engine.filter object-size {filter options} \
...

[Output Truncated]
RUN  (taps.run) :: Runtime directory: runs/cholesky-dask-2024-09-19-12-00-00
APP  (taps.apps.cholesky) :: Generated input matrix: (10000, 10000)
APP  (taps.apps.cholesky) :: Block size: 1000
APP  (taps.apps.cholesky) :: Output matrix: (10000, 10000)
RUN  (taps.run) :: Finished app (name=cholesky, runtime=13.18s)
```

Run directory:

- Logs for analysis
- Config for reproducibility
- Application outputs



```
+ runs
  |- cholesky-dask-2024-09-19-11-00-00
    \_ cholesky-dask-2024-09-19-12-00-00
      |- config.toml
      |- log.txt
      \_ tasks.jsonl
```

```
[app]
name = "cholesky"
matrix_size = 10000
block_size = 1000

[engine.executor]
name = "dask"
workers = 16

[engine.filter]
name = "object-size"
min_size: 1000

[engine.transformer]
name = "proxystore"
cache_size = 16
extract_target = true
populate_target = true
...

[logging]
level = "INFO"
file_level = "INFO"
file_name = "log.txt"

[run]
dir_format = "runs/{name}-{executor}-{timestamp}"
```

<https://taps.proxystore.dev/latest/guides/config/>

`python -m taps.run --config config.toml`



Enabling eScience Applications



Better Benchmarking



TaPS: Task Performance Suite



Evaluation *Exploration*

~~Evaluation~~ Exploration Goals

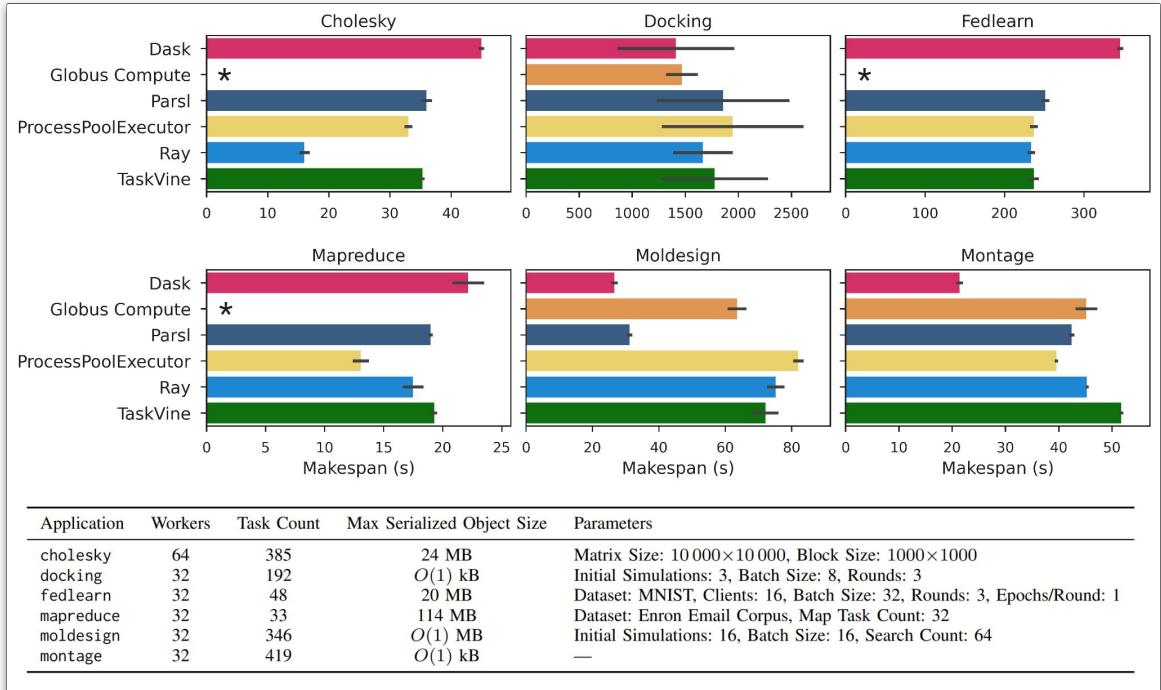
- ✗ Not to determine which executor is best
- ✓ Showcase kinds of evaluations TaPS can support
- ✓ Showcase characteristics of applications and executors
- ✓ Leave with more questions than answers... keep exploring!
- ✓ Encourage more discourse on benchmarking in the community

<https://github.com/proxystore/escience24-taps-analysis>

Application Makespan

No stand-out executor; new questions to pose.

- Why are some combos so much faster? (Ray in Cholesky, Dask/Parsl in Moldesign, and Dask in Montage)
- Which benefit more from warm-starts?
- How does performance correlate to average task duration or data flow volume?
- How do they handle resource contention with nested parallelism (e.g., OpenMP tasks)



<https://github.com/proxystore/escience24-taps-analysis>

*Task data exceeds Globus Compute 10 MB payload limit.

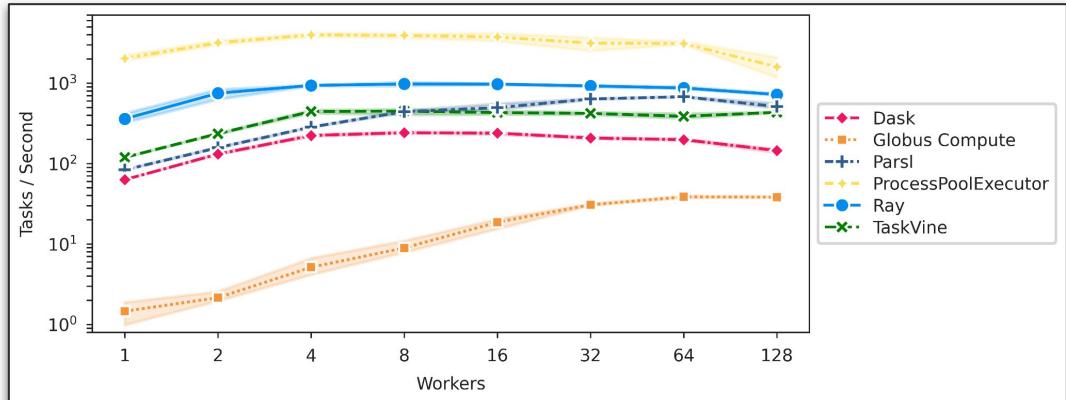
Scheduler Performance – Scaling Workers

Workload

- Synthetic App – Bag of Tasks
- Vary n workers. Submit n initial tasks
- Submit tasks as prior complete
- Record task throughput

Hardware

- Single CHI@TACC compute-zen-3 node
- 2x AMD EPYC 7763 64-Core CPU
- 256 Logical Cores / 256 GB RAM



- **ProcessPool** (yellow) is high-water mark (no scheduler)
- **Ray** (light blue) has lowest task latency but does not scale well
- **Dask** (pink) and **TaskVine** (green) plateau between 4–8 workers
- **Parsl** (dark blue) scales best but has higher individual task latency
- **Globus Compute** (orange) does better when batching more tasks

<https://github.com/proxystore/escience24-taps-analysis>

Scheduler Performance – Data Transfer

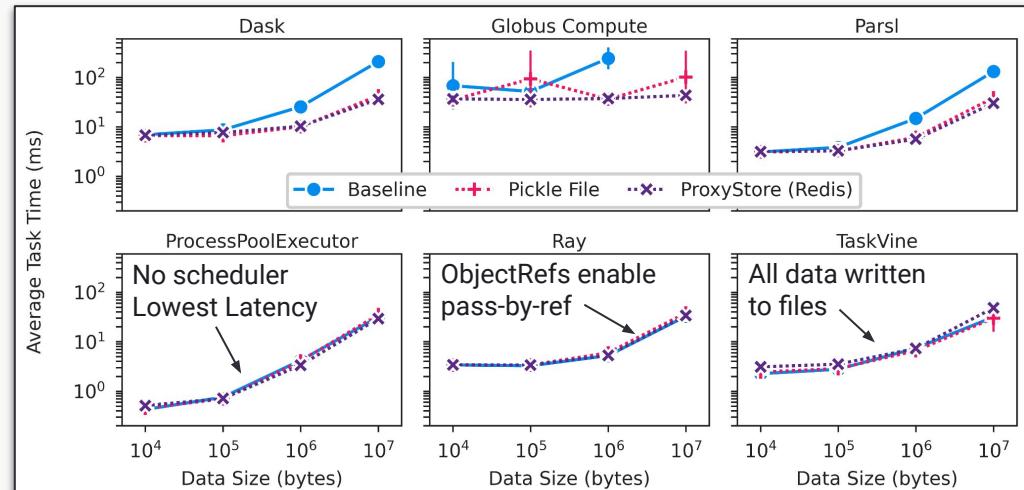
Workload

- Synthetic App – Bag of Tasks
- 32 workers and 32 concurrent tasks
- Vary input/output data size
- Record task round-trip time

Methods

- **Baseline:** Executor handles serialization and transfer
- **File:** Data pickled, written to file, and replaced with file path
- **ProxyStore:** Data is proxied, stored in Redis, and replaced with proxy object

Central **schedulers** enable advanced features but are a **bottleneck** for data transfer



<https://github.com/proxystore/escience24-taps-analysis>



THE UNIVERSITY OF
CHICAGO

Keep Exploring — Give TaPS a Try!

A screenshot of a GitHub repository page for 'proxystore/taps'. The repository is public and was generated from 'gpauloski/python-template'. It shows the main branch with 314 commits. Recent commits include updates to GitHub templates, logos, and engine fixtures. The sidebar includes links for Code, Issues, Pull requests, Discussions, Actions, Security, Insights, and Settings.

github.com/proxystore/taps

A screenshot of the TaPS project website at 'taps.proxystore.dev'. The main page features a navigation bar with Home, Apps, Guides, API Reference, Contributing, and Publications. Below the navigation is a section titled 'TaPS: Task Performance Suite' with a 'Table of contents' and 'Citation' link. It highlights that TaPS is a standardized framework for evaluating task-based execution frameworks and data management systems using real and synthetic scientific applications. The 'About' sidebar on the left provides details about the benchmarking suite for distributed/parallel task executors, including links to the repository, Readme, MIT license, code of conduct, security policy, activity, custom properties, and social metrics (7 stars, 3 watching, 4 forks).

taps.proxystore.dev

Want to collaborate? Reach out if you have...

- an application that could be a benchmark,
- a new execution framework,
- a data management system,
- and more!



THE UNIVERSITY OF
CHICAGO

Evaluation Exploration | 33

globus labs



A Performance Evaluation Suite for Task-based Execution Frameworks



J. Gregory
Pauloski



Valerie
Hayot-Sasson



Maxime
Gonthier



Nathaniel
Hudson



Haochen
Pan



Sicheng
Zhou



Ian
Foster



Kyle
Chard

Questions?

Contact:

jgpauloski@uchicago.edu

github.com/proxystore/taps/issues

Reference:

<https://github.com/proxystore/taps>

<https://taps.proxystore.dev>

Acknowledgements:

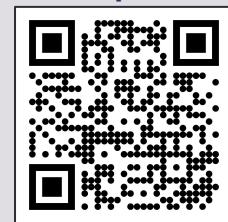
- Argonne National Laboratory under U.S. Department of Energy Contract DE-AC02-06CH1135
- National Science Foundation under Grant 2004894 and Grant 2209919
- Chameleon Cloud testbed supported by the National Science Foundation

GitHub



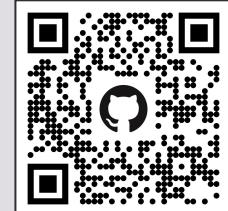
github.com/proxystore/taps

Preprint



arxiv.org/abs/2408.07236

Slides



gregpauloski.com/#presentations



Any Python Executor



Comprehensive Reference Applications



ProxyStore



A Performance Evaluation Suite for Task-based Execution Frameworks

CLI

```
python -m taps.run \
--app cholesky \
{args}
```

Reproducible Configurations

```
[app]
name = "cholesky"
matrix_size = 10000
block_size = 1000
```

```
[engine.executor]
name = "dask"
workers = 16
```

```
[engine.filter]
name =
"object-size"
min_size: 1000
```

Plugin System

Open Source



Get Started



THE UNIVERSITY OF
CHICAGO

Computational Workflows

An **application** composed of a structured set of **discrete tasks** designed to **automate** computational processes to achieve a **scientific goal**.

Applications [1]

- AI/ML
- Bioinformatics
- Climate Science
- Cosmology
- Data Science
- High Energy Physics
- Materials Science
- Molecular Dynamics
- Neuroscience
- Social Sciences

Benefits

- Automation
- Error Handling
- Usability
- Modularity
- Monitoring
- Resource Optimization
- Portability
- Reproducibility
- Scalability
- Software Coupling

Challenges [2]

- Coupling AI/ML/Quantum
- Cloud and HPC Integration
- Computing Continuum
- Data Management
- Data Provenance
- FAIR Principles
- Heterogeneous Hardware
- Interoperability
- Performance and Scalability
- Standards and APIs

[1] "Scientific Workflows: Moving Across Paradigms" (<https://dl.acm.org/doi/10.1145/3012429>)

[2] Workflows Community Summit (<https://arxiv.org/abs/2304.00019>)



Engine Plugins

	Task Execution	Data Management	Logging
Purpose	Asynchronously execute functions	Manage task data by filtering and transforming data into/resolve data from intermediate representations	Record task execution traces
Interfaces	Executor*	Filter + Transformer	RecordLogger
Out-of-the-Box Implementations	ThreadPool, ProcessPool, Dask, Globus Compute, Parsl, Ray, TaskVine	Shared File Systems ProxyStore (DAOS, Globus Transfer, Margo, Redis, UCX, ZMQ, ...)	JSON
Future Extensions	Cloud FaaS New Executors	Cloud Storage	Databases WfTrace

*Requires implicit data flow support via futures. Wrapper provided for implementations that lack this feature.