SC24 Atlanta, GA hpc creates.

Accelerating Python Applications with *Dask* and *ProxyStore*

J. Gregory Pauloski,[*†] Klaudiusz Rydzy,[‡] Valerie Hayot-Sasson,[*†] Ian Foster,[*†] and Kyle Chard[*†]

[*]*University of Chicago,* [†]*Argonne National Laboratory,* [‡]*Loyola University Chicago*

18 November 2024 — Atlanta, Georgia

Python Apps for Exascale Systems

Target Apps: Active Learning Workflows

- Dynamic task-based workflows
- Limited optimizations based on DAG
- Diverse data structures and patterns

Requirements:

- Compute Fabric:
  - Deploy workers distributed across cluster
  - Easy-to-use, performant, portable, etc.
- Data Fabric:
  - Leverage hardware features (e.g., DAOS, RDMA)
  - Easy-to-use, performant, portable, etc.

Aurora at ALCF

10K Nodes / 1M Cores / 60K GPUs
20 PB Memory / 230 PB Storage

Can we scale science apps using *Dask* and *ProxyStore*?

dask

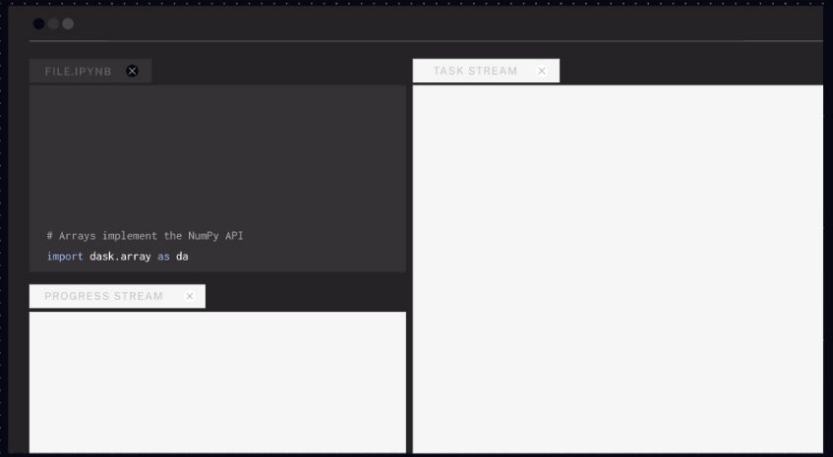
Community Blog Docs

Parallel Python

Fast and Easy

Easy Parallel Python that does what you need

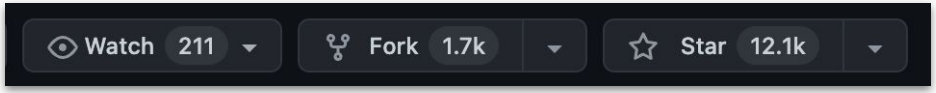
Get started



What you can do with Dask

Big Pandas Parallel For Loops Big Arrays Production ETL ML

Dask Distributed



A lightweight library for distributed computing in Python.

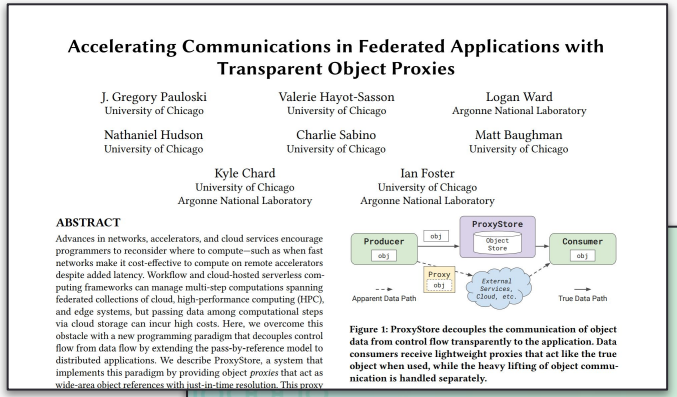
- **Low latency:** Each task suffers about 1ms of overhead. A small computation and network roundtrip can complete in less than 10ms.
- **Peer-to-peer data sharing:** Workers communicate with each other to share data. This removes central bottlenecks for data transfer.
- **Complex Scheduling:** Supports complex workflows (not just map/filter/reduce) which are necessary for sophisticated algorithms used in nd-arrays, machine learning, image processing, and statistics.
- **Pure Python:** Built in Python using well-known technologies. This eases installation, improves efficiency (for Python users), and simplifies debugging.
- **Data Locality:** Scheduling algorithms cleverly execute computations where data lives. This minimizes network traffic and improves efficiency.
- **Familiar APIs:** Compatible with the concurrent.futures API in the Python standard library. Compatible with dask API for parallel algorithms
- **Easy Setup:** As a Pure Python package distributed is `pip` installable and easy to set up on your own cluster.

```python
from dask.distributed import Client

client = Client()

def square(x):
    return x ** 2

def neg(x):
    return -x

A = client.map(square, range(10))
B = client.map(neg, A)
total = client.submit(sum, B)
total.result()  # -285
```

ProxyStore

Data flow management library for distributed Python workflows

- Proxy ***transparently*** decouples control and data flow
- Best of both ***pass-by-reference*** and ***pass-by-value***
- Use any mediate communication method via plugins
- “Make boring code easy”



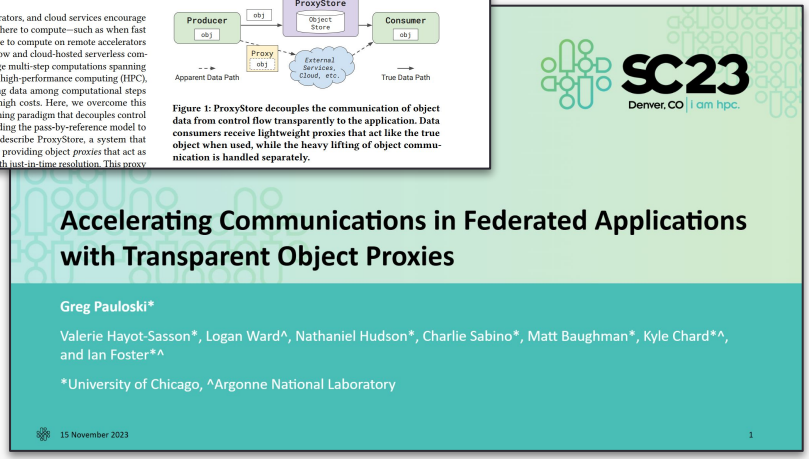
Accelerating Communications in Federated Applications with Transparent Object Proxies

J. Gregory Pauloski, University of Chicago
Valerie Hayot-Sasson, University of Chicago
Logan Ward, Argonne National Laboratory
Nathaniel Hudson, University of Chicago
Charlie Sabino, University of Chicago
Matt Baughman, University of Chicago
Kyle Chard, University of Chicago, Argonne National Laboratory
Ian Foster, University of Chicago, Argonne National Laboratory

ABSTRACT

Advances in networks, accelerators, and cloud services encourage programmers to reconsider where to compute—such as when fast networks make it cost-effective to compute on remote accelerators despite added latency. Workflow and cloud-hosted serverless computing frameworks can manage multi-step computations spanning federated collections of cloud, high-performance computing (HPC), and edge systems, but passing data among computational steps via cloud storage can incur high costs. Here, we overcome this obstacle with a new programming paradigm that decouples control flow from data flow by extending the pass-by-reference model to distributed applications. We describe ProxyStore, a system that implements this paradigm by providing object *proxies* that act as wide-area object references with just-in-time resolution. This proxy

Figure 1: ProxyStore decouples the communication of object data from control flow transparently to the application. Data consumers receive lightweight proxies that act like the true object when used, while the heavy lifting of object communication is handled separately.



SC23 Denver, CO | I am hpc.

Accelerating Communications in Federated Applications with Transparent Object Proxies

Greg Pauloski*

Valerie Hayot-Sasson*, Logan Ward^, Nathaniel Hudson*, Charlie Sabino*, Matt Baughman*, Kyle Chard*^, and Ian Foster*^

*University of Chicago, ^Argonne National Laboratory

15 November 2023

1

SC23 Paper!

Proxy Objects

What is a proxy (in this context)?

- Self-contained wide-area **reference** to a **target** object
- Transparently resolve target **just-in-time** when first used

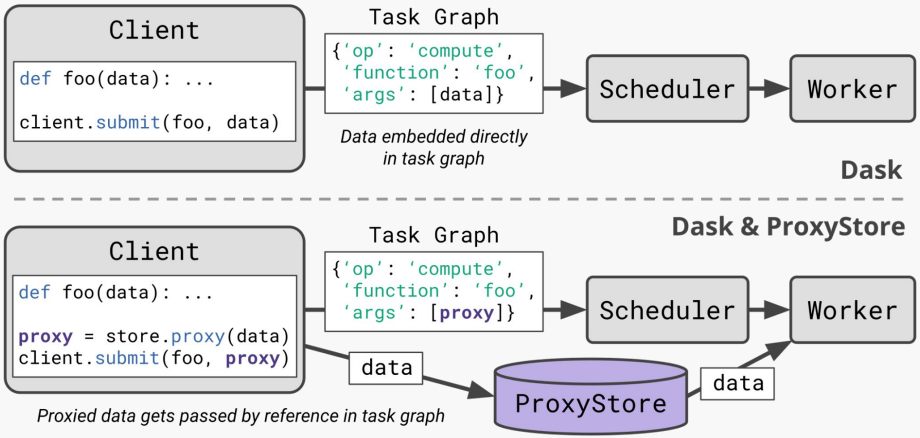
What are the benefits?

- Performance (pass-by-reference, async resolve, skip unused objects)
- Reduce code complexity
- Partial resolution of complex objects
- Access control

```python
from proxystore.connectors import RedisConnector
from proxystore.store import Store
from proxystore.proxy import Proxy

def foo(x: Bar) -> ...:
    # Resolve of x deferred until use
    assert isinstance(x, Bar)
    # More computation...

with Store('demo', RedisConnector(...)) as store:
    x = Bar(...)
    p = store.proxy(x)  # Anything can be proxied
    assert isinstance(p, Proxy)
    foo(p)  # Proxies can be passed-by-ref anywhere
```

Why use ProxyStore with Dask?

- Better data transfer mechanisms
  - *Storage*: KeyDB, Redis, Lustre
  - *Transfer*: Grid FTP, TCP, RDMA, WebRTC
- Use Dask anti-patterns without fear!
  - *Large objects in task graphs*: avoid scheduler and MessagePack overheads
  - *Frequent calls to future.result():* common in active learning apps



Engineering Challenges

Learn more in the short paper!

Integration Model → Today's Demo

Code Quality → MyPy plugin for type-checking duck-typed proxies

Compatibility → Rabbit hole on proxies, Dask hashing, and Python descriptors

Serialization → See poster at end on serialization optimization in ProxyStore

DAOS Support → Evaluation TBD due to Aurora/DAOS unavailability

Integration Model

Manual

- Fine-grain control over what is proxied
- **Okay** for **simple** apps
- **Bad** for **complex** apps

Custom Dask Client

- **Automatically proxy** task inputs/outputs according to user spec
- **Easy** for existing **Dask** apps
- **Limited** access to ProxyStore features

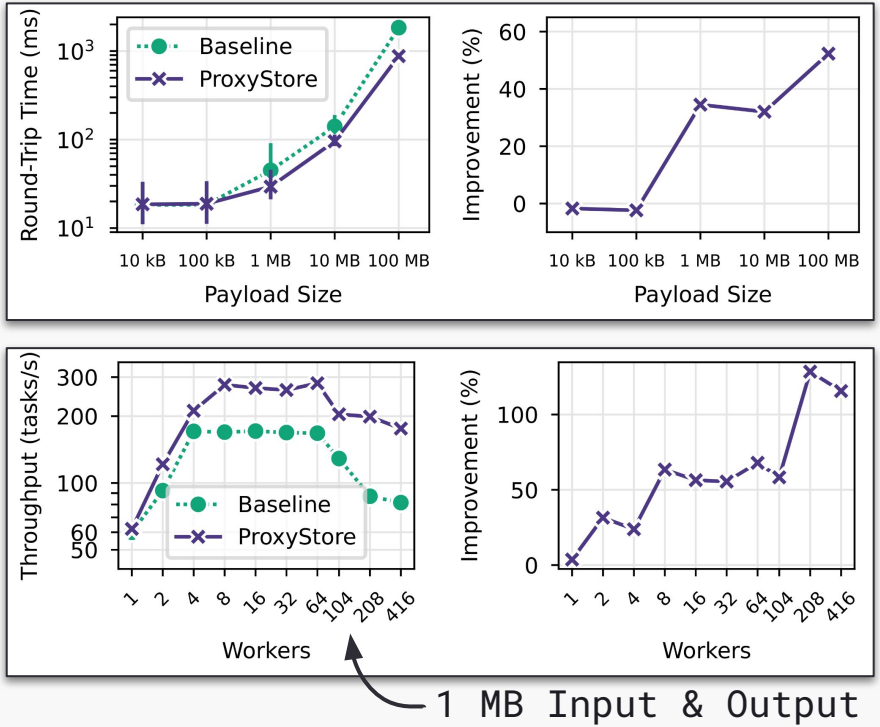
StoreExecutor

- ProxyStore wrapper for any concurrent futures Executor (incl. Dask)
- **Automatically proxy** task inputs/outputs
- **Better** memory management
- **Not** always compatible with existing Dask apps

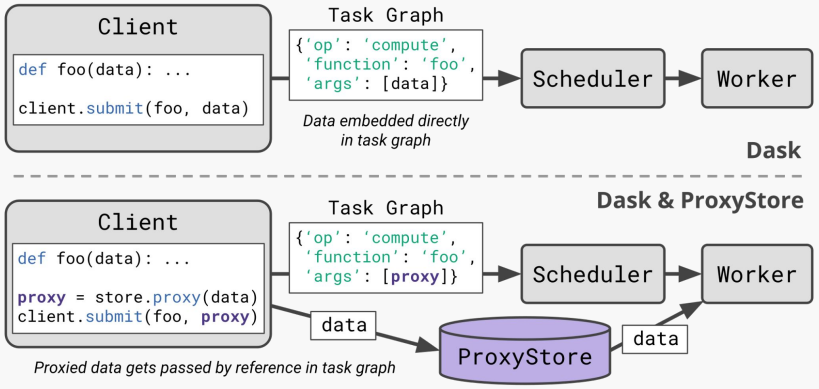
Demo

https://doi.org/10.5281/zenodo.13328934

Eval: Client/Worker Transfer Overheads



Benchmark Configs in TaPS



ProxyStore

Related activities at SC24!

Turbocharging Dask Apps: Accelerating Data Flow with ProxyStore

- *Presenter* — Klaudiusz Rydzy
- *SRC Poster* — Tu 5–7 PM in B302

Accelerating Communications in High-Performance Scientific Workflows

- *Presenter* — Greg Pauloski
- *Doctoral Poster* — Tu 5–7 PM in B302
- *Doctoral Showcase* — Th 11–11:15 AM in B306

Questions?

Contact:
J. Gregory Pauloski
jgpauloski@uchicago.edu

Short Paper on ArXiv

Reference:
docs.proxystore.dev
github.com/proxystore
github.com/proxystore/hppss24-demo

Acknowledgements:

- Argonne National Laboratory under U.S. Department of Energy Contract DE-AC02-06CH1135
- National Science Foundation under Grant 2004894 and Grant 2209919

Integration Model: *Manual*

Manual

- Fine-grain control over what is proxied
- **Okay** for **simple** apps
- **Bad** for **complex** apps

```
from dask.distributed import Client
from proxystore.ex.connectors.daos import DAOSConnector
from proxystore.store import Store

client = Client()
connector = DAOSConnector(pool=..., container=...)

with Store('example', connector) as store:
    proxy = store.proxy([1, 2, 3])
    future = client.submit(sum, proxy)
    assert future.result() == 6
```

Integration Model: *Dask Client*

Custom Dask Client

- **Automatically proxy** task inputs/outputs according to user spec
- **Easy** for existing **Dask** apps
- **Limited** access to ProxyStore features

```python
from proxystore.ex.plugins.distributed import Client
from proxystore.ex.connectors.daos import DAOSConnector
from proxystore.store import Store

connector = DAOSConnector(pool=..., container=...)

with Store('example', connector) as store:
    client = Client(ps_store=store, ps_threshold=1000)
    future = client.submit(sum, [1, 2, 3])
    assert future.result() == 6
```

Integration Model: *StoreExecutor*

StoreExecutor

- ProxyStore wrapper for any concurrent futures Executor (incl. Dask)
- **Automatically proxy** task inputs/outputs
- **Better** memory management
- **Not** always compatible with existing Dask apps

```python
from dask.distributed import Client
from proxystore.ex.connectors.daos import DAOSConnector
from proxystore.store import Store
from proxystore.store.executor import StoreExecutor

client = Client()
connector = DAOSConnector(pool=..., container=...)
store = Store('example', connector)

with StoreExecutor(client, store, ...) as executor:
    future = executor.submit(sum, [1, 2, 3])
    assert future.result() == 6
```