ProxyStore
Decoupling Control and Data Flow in Workflows

Greg Pauloski
20 October 2023
Motivation

Avoid cloud transfer but keep the benefits?

Peer-to-peer transfer between sites?
Control Flow ≠ Data Flow
Proxies + Object Stores = ProxyStore

- *Elegant* pass-by-reference in distributed Python apps
- Mechanism for *transparently* decoupling control and data flow
- Support for *any* (via plugins) object storage method
Proxy Objects

- Transparently wrap **target** objects
- Acts like a wide-area reference
- Initialized with a **factory**
- Just-in-time **resolution**

```python
import numpy as np
from proxystore.proxy import Proxy

x = np.array([1, 2, 3])

# Proxy(Callable[[], T]) -> Proxy[T]
p = Proxy(lambda: x)

# A proxy is an instance of its wrapped object
assert isinstance(p, Proxy)
assert isinstance(p, np.ndarray)

# The proxy can do everything the numpy array can
assert np.array_equal(p, [1, 2, 3])
assert np.sum(p) == 6
y = x + p
assert np.array_equal(y, [2, 4, 6])
```
from proxystore.connectors.redis import RedisConnector
from proxystore.store import Store

store = Store(
    name='my-store',
    connector=RedisConnector('localhost', 6379),
    # extra options
)

my_object = MyData(...)

p = store.proxy(my_object)

from proxystore.proxy import Proxy

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

assert isinstance(p, Proxy)
my_function(p)

Why lazy resolution with proxies?
- Performance (pass-by-reference, async resolve, skip unused objects)
- Avoid writing shims/wrapper functions
- Partial resolution of large objects with nested proxies
- Access control (only resolve data where permitted)
Camera

github.com/proxystore | docs.proxystore.dev | 7

(1) Producer puts object in Store and gets back Proxy.

(2) Store gives object to Connector and generates a Proxy with metadata/Factory.

(3) Producer sends Proxy to consumer.

(4) Consumer uses Proxy like a normal object.

(5) Object resolution happens transparently to consumer.
What’s changed since ParslFest 2022?

- v0.3.3 → v0.6.0
- New architecture
- New data transfer methods
  - Peer-to-peer Endpoints
  - RDMA
  - DAOS
  - Bring your own
- SC23 paper

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Storage</th>
<th>Intra-Site</th>
<th>Inter-Site</th>
<th>Persistence</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Disk</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Redis</td>
<td>Hybrid</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Margo</td>
<td>Memory</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UCX</td>
<td>Memory</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZMQ</td>
<td>Memory</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Globus</td>
<td>Disk</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DAOS</td>
<td>Disk*</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Endpoint</td>
<td>Hybrid</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Multi</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>
ProxyStore in Parsl
import parsl
from parsl.serialize.facade import register_method_for_data
from parsl.serialize.proxystore import ProxyStoreSerializer
...

parsl.load(htex_config())

@parsl.python_app
def my_sum(data: list[int]) -> int:
    from proxystore.proxy import Proxy
    assert isinstance(data, Proxy)
    return sum(data)

with Store('my-store', FileConnector('/tmp/proxystore')) as store:
    serializer = ProxyStoreSerializer(store=store, should_proxy=lambda o: isinstance(o, list))
    register_method_for_data(serializer)  # Note: some extra boilerplate excluded
    assert my_sum([1, 2, 3]).result() == 6
ProxyStore in Globus Compute
Avoiding Globus Compute Cloud Transfer Costs

```python
from globus_compute_sdk import Executor
from proxystore.connectors.endpoint import EndpointConnector
from proxystore.store import Store

ENDPOINT_UUID = '5b994a7d-8d7c-48d1-baa1-0fda09ea1687'

def compute(obj: MyData) -> Result:
    # Computation ...
    return Result(...)

with Store('my-store', EndpointConnector(...)) as store:
    with Executor(endpoint_id=ENDPOINT_UUID) as gce:
        proxy = store.proxy(MyData(...))
        future = gce.submit(compute, proxy)
        print(future.result())
```

$ globus-compute-endpoint configure demo
$ proxystore-endpoint configure demo

https://docs.proxystore.dev/main/guides/globus-compute
https://docs.proxystore.dev/main/guides/endpoints
Questions?

Contact:
jgpauloski@uchicago.edu
github.com/proxystore/proxystore/issues

SC Paper:
docs.proxystore.dev/main/publications