Accelerating Communications in Federated Applications with Transparent Object Proxies

J. Gregory Pauloski [1] Valerie Hayot-Sasson [1] Logan Ward [2] Kyle Chard [1 2] Ian Foster [1 2]

[1]University of Chicago [2]Argonne National Laboratory

ProxyStore

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 model enables data producers to communicate data unilaterally, transparently, and efficiently to both local and remote consumers.

Applications of ProxyStore

- **Workflow systems:** reduce overheads in the workflow controller by moving proxies rather than objects.
- **FaaS platforms:** move data directly between clients and executors, bypassing FaaS management systems.
- **Edge computing:** enable peer-to-peer communication across diverse networks.

Proxy Model

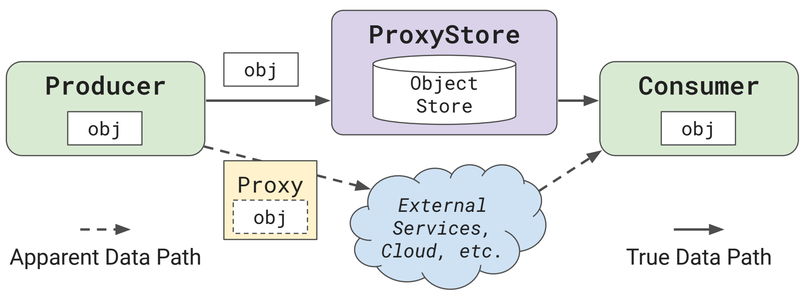

Figure 1. ProxyStore decouples the communication of object data from control flow transparently to the application. Data consumers receive lightweight proxies that act like the true object when used, while the heavy lifting of object communication is handled separately.

Lazy, transparent object proxies act as wide-area object references, giving the illusion that data is moved across processes, while in reality, data is moved via more optimal mechanisms transparently (Figure 1). Key properties of proxies:

- *Transparency*: The proxy behaves identically to its target object by forwarding all operations on itself to the target.
- *Laziness*: The proxy is initialized with a *factory*, an object that is callable like a function and returns the target object. The proxy is *lazy* in that it does not call the factory to retrieve the target until it is first accessed—a process that is referred to as *resolving* the proxy.

Functionally, proxies have both *pass-by-reference* and *pass-by-value* attributes. The eventual user of the proxied data gets a copy, but unnecessary copies are avoided when the proxy is passed among multiple functions.

References

[1] Y. Babuji, A. Woodard, Z. Li, D. S. Katz, B. Clifford, R. Kumar, L. Lacinski, R. Chard, J. M. Wozniak, I. Foster, M. Wilde, and K. Chard. Parsl: Pervasive Parallel Programming in Python. In *28th ACM International Symposium on High-Performance Parallel and Distributed Computing*, 2019.

[2] R. Chard, Y. Babuji, Z. Li, T. Skluzacek, A. Woodard, B. Blaiszik, I. Foster, and K. Chard. funcX: A Federated Function Serving Fabric for Science. In *29th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2020.

[3] Real-Time Defect Identification. `https://github.com/ivem-argonne/real-time-defect-analysis`. Accessed Mar 2023.

[4] L. Ward, J. G. Pauloski, V. Hayot-Sasson, R. Chard, Y. Babuji, G. Sivaraman, S. Choudhury, K. Chard, R. Thakur, and I. Foster. Cloud services enable efficient ai-guided simulation workflows across heterogeneous resources, 2023. `https://arxiv.org/abs/2303.08803`.

[5] L. Ward, G. Sivaraman, J. G. Pauloski, Y. Babuji, R. Chard, N. Dandu, P. C. Redfern, R. S. Assary, K. Chard, L. A. Curtiss, R. Thakur, and I. Foster. Colmena: Scalable machine-learning-based steering of ensemble simulations for high performance computing. In *IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments*. IEEE, 2021.

Proxy Model Benefits

- **Efficiency:** Proxies are lightweight to communicate.
- **Compatibility:** Proxies behave like their target so are interoperable with existing code. The factory contains all logic for data retrieval and manipulation (no more shims/wrappers).
- **Optimization:** Lazy resolution can amortize costs and enable partial resolution of large, nested objects.
- **Security:** Proxies can be moved in place of confidential data while ensuring data are only resolved where permitted.

Design and Implementation

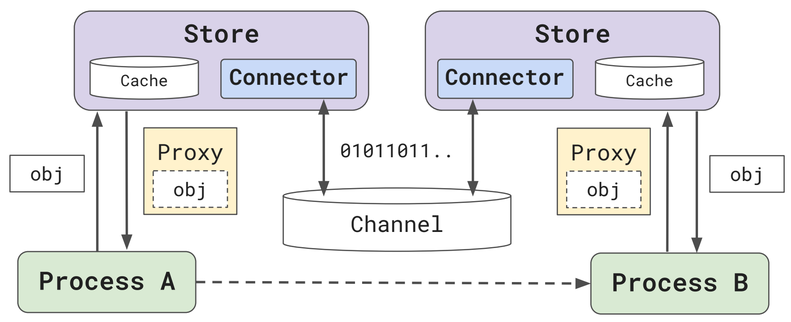

Figure 2. Overview of ProxyStore's design. ProxyStore provides the `Store` interface for creating proxies and a suite of `Connector` implementations for commonly used mediated communication channels.

ProxyStore provides four primary components: the `Proxy` and `Factory`, described previously, the `Store` interface, and a set of `Connectors`. The `Store` is the high level interface for creating proxies, and `Connectors` enable the `Store` to interface with byte-level communication channels (Figure 2).

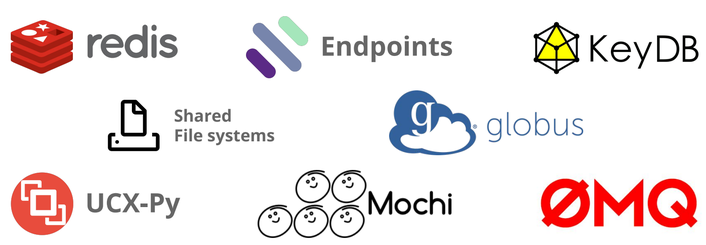

Figure 3. Communication technologies supported out-of-the-box third-party code can easily provide new connectors.

A `Store` is initialized with a `Connector` instance and provides additional functionality on top of the `Connector`. `Store` operations act on Python objects rather than byte strings, (de)serializing objects before invoking the corresponding operation on the `Connector`. The `Store` provides caching of operations to reduce communication costs, asynchronous resolving of proxies, and auto-initialization. The proxy abstraction, provided by the `Store.proxy()` enables a producer to unilaterally (without the agreement of the receiver) choose the best communication channel. The provided `Connectors` support a mix of in-memory and on-disk storage, intra- and inter-site transfer, and persistence goals (Table 1 and Figure 3).

Connector	Storage	Intra-Site	Inter-Site	Persistence
File	Disk	✓		✓
Redis	Hybrid	✓		✓
Margo	Memory	✓		
UCX	Memory	✓		
ZMQ	Memory	✓		
Globus	Disk		✓	✓
Endpoint	Hybrid	✓	✓	✓

Table 1. Attributes of the provided `Connector` implementations. The `MultiConnector` enables intelligent routing across multiple `Connectors` via user-provided policies.

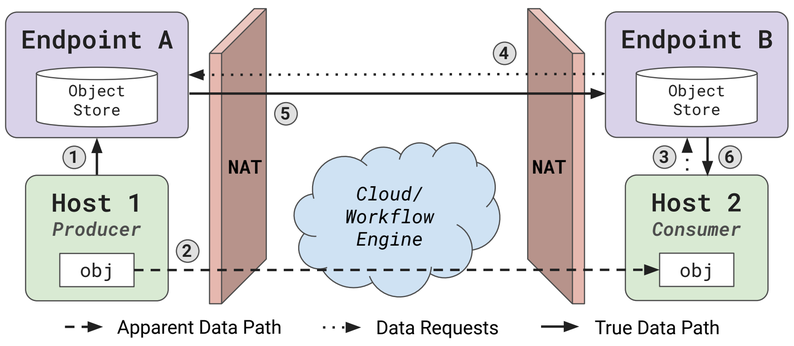

Figure 4. ProxyStore endpoints facilitate direct data transfer between sites by using WebRTC for peer-to-peer communication (NAT traversal via UDP hole-punching and a public relay server).

Accelerating Applications

Here we demonstrate how ProxyStore can accelerate communications in distributed and federated applications by integrating ProxyStore into three real-world scientific applications.

Real-time Defect Analysis

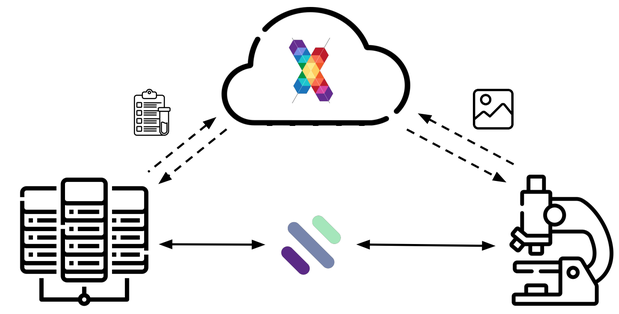

Figure 5. A microscopy facility uses FuncX [2] to invoke ML models to quantify defects in acquired images, dispatching this computation to HPC facilities for fast GPU inference. We modify the application [3] to create and send proxies of images, rather than the actual images, to utilize superior communication mediums.

Transfer Method	Time (ms)	Improvement
Cloud Baseline	3411 ± 389	–
`FileConnector`	2160 ± 46	36.6%
`EndpointConnector`	2280 ± 107	33.2%

Table 2. Round-trip task time improvement when images and inference results are communicated via ProxyStore rather than via cloud services.

Federated Learning (FL)

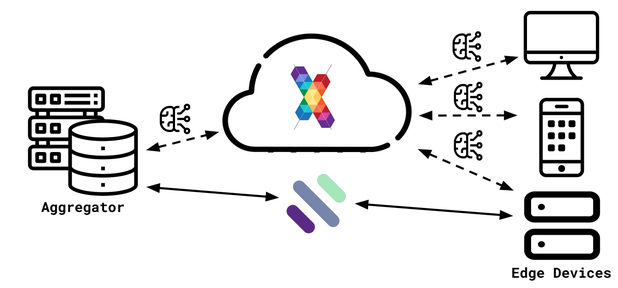

Figure 6. We demonstrate the benefit of ProxyStore for FL use cases and, more generally, edge computing. ProxyStore enables direct communication between central aggregator services and edge nodes, bypassing additional overheads in the FaaS/cloud systems.

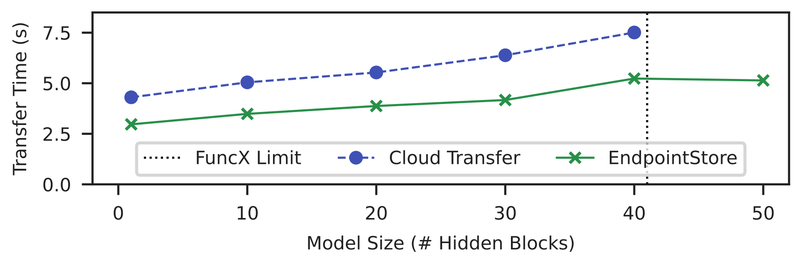

Figure 7. Average model weights transfer time between edge nodes. ProxyStore both reduces transfer time and also enables use of larger models by bypassing the FaaS system which enforces payload limits for data transfer.

Molecular Design

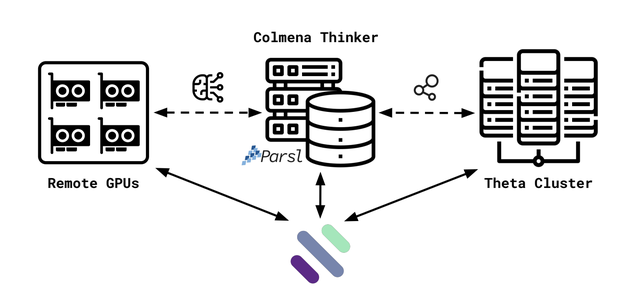

Figure 8. An AI-guided simulation workflow for electrolytes discovery deployed across heterogenous systems at different sites [4].

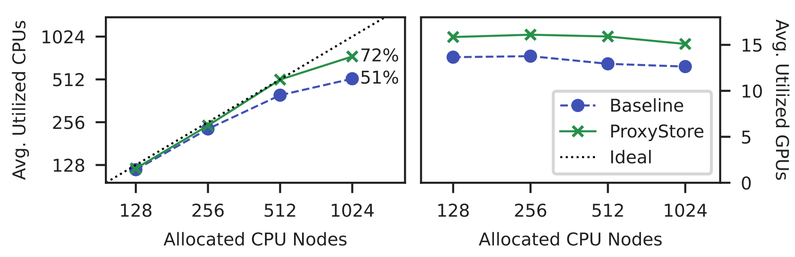

Figure 9. ProxyStore removes data movement burdens from the workflow system and improves scaling, particularly at 512 and 1024 nodes.