ProxyStreams: Leveraging Lightweight Python Proxies for Portable Streams

Nazanin Mahmoudi
Wayne State University
USA

Valerie Hayot-Sasson (advisor)
University of Chicago
USA

Kyle Chard (advisor)
University of Chicago
USA

Abstract

A novel streaming approach is introduced for Python, leveraging the ProxyStore system to facilitate the exchange of stream references across distributed systems. This approach utilizes generators to efficiently publish and consume messages from streams. The extensible backend connector interface of ProxyStore enables support for diverse communication mechanisms, such as transitioning from ZMQ to RDMA. Performance results highlight the capability to perform data PUT and GET operations on streams with minimal overhead and high efficiency.

ACM Reference Format:
Nazanin Mahmoudi, Valerie Hayot-Sasson (advisor), and Kyle Chard (advisor). 2026. ProxyStreams: Leveraging Lightweight Python Proxies for Portable Streams. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Scientific data are now produced at alarming rates (e.g., GB/s or more in many cases) and must be processed quickly to provide real-time feedback to experiments or to extract relevant information in situations where data cannot be stored. Data streaming is increasingly used to move data from acquisition to processing environments. Existing streaming methods rely on manual deployment, maintenance of data streaming platforms, and use of wide area transfer protocols built on TCP. We explore a new approach using Python proxies to represent streams, thereby enabling references to streams to be passed around distributed systems. Furthermore, use of different inter- and intra-site data transfer methods (e.g., using TCP for wide area streaming and RDMA for local streaming) is also explored. We implement our solution in ProxyStore [1] and provide a Python-based interface that enables streams to be easily integrated in Python programs.

2 Motivation

Real-time instrument processing and steering. Experiments at the Advanced Photon Source (APS) produce data at up to GB/s [2]. These data must be transferred to HPC resources in near-real-time for reconstruction and analysis. Consumers are dynamically provisioned on HPC resources to process streaming data and results are communicated back to researchers to guide experiments.

Monitoring large-scale parallel programs. Large-scale workflows produce large-quantities of data that are relied upon to understand workflow progression, identify failures, and optimize performance. For example, cosmology workflows [3], running at supercomputer scale (4096 nodes on Theta) produce vast volumes of monitoring data. Sources of information can be represented as streams that can be consumed by logging, fault detection, and steering components in real-time.

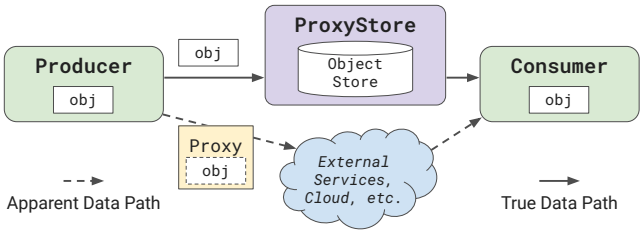

Figure 1. ProxyStore Architecture.

3 Background and Related Work

ProxyStore [1] is a system that manages object proxies across distributed Python applications. These proxies effectively offer pass-by-reference semantics in distributed applications, simplifying data management across different components of the application. ProxyStore has an extensible, plug-in connector architecture via which different data storage and transfer methods can be integrated. ProxyStore includes connectors for wide area transfers (e.g., via Globus and P2P networks) as well as local transfers (e.g., via shared file systems or RDMA). By leveraging ProxyStore, scientists can simplify the complexities associated with streaming data, allowing them to handle their data needs more effectively (see Figure 1).

Apache Kafka is an open-source data streaming platform that is widely used in industry and academia. It provides a high-performance implementation using TCP. Users define topics and publish/consume messages from these topics. Its popularity is due to its performance, low latency, fault tolerance, and throughput. However, While TCP can ensure reliable data transfer, it can lead to increased latency in environments with high speed interconnects.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY
2026. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

4 Approach

We implement a novel data streaming approach using Python proxies. Users can pass around proxies that can then be used to publish or consume data. We implement an intuitive Python-based interface using a generator-like approach and enabling use of ProxyStore's various backend connectors. To highlight the features of ProxyStore's interface for data streaming, we begin by establishing a new ProxyStore data store named 'stream' using the MargoConnector. Within this store, a stream is created to facilitate the seamless management of incoming data. Using the stream object, we assign two data packets, 'DATA' and 'DATA2' to belong to this stream.

5 Evaluation

We evaluate the performance of ProxyStreams by scaling the number of data packets and size of data packets. The following graphs show the time in nanoseconds to perform the GET and PUT functions for various data packet sizes and stream sizes.

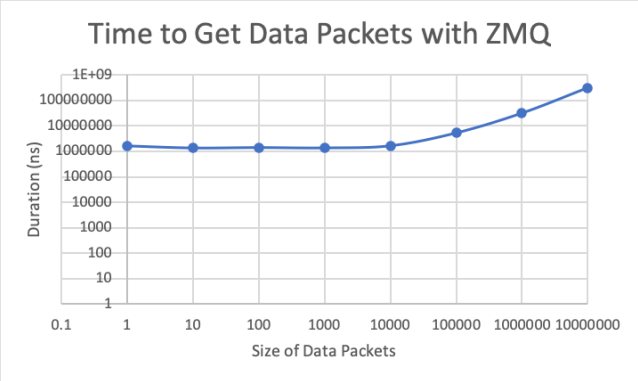

Figure 2. Time to Get Data Packets with ZMQ

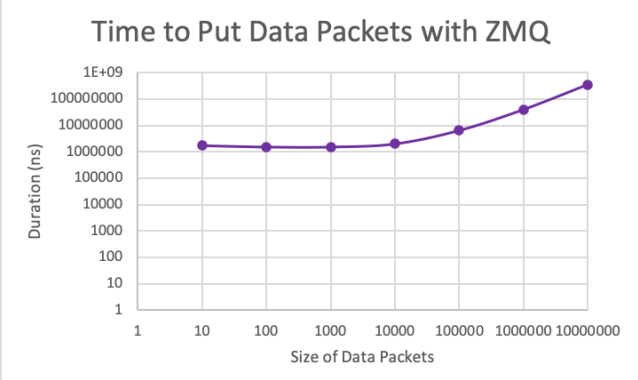

Figure 3. Time to Put Data Packets with ZMQ

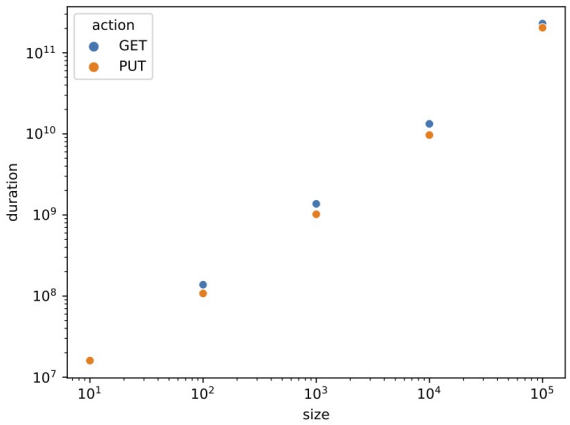

Figure 4. Time to Get/Put Streams with ZMQ

As shown in the graphs, the time for both ZMQ's PUT and GET functions take approximately 1.1 million nanoseconds for data packets up to 10,000 in size. Notably, larger data packets lead to longer processing times, aligning with expected behavior. Just as the GET and PUT functions result in nearly identical graphs when data packet size is altered, so too do the GET and PUT functions when stream size is altered. The time for both increases as stream size increases.

6 Summary

ProxyStreams provides a new approach for use in distributed systems, enabling passing of stream references, use of diverse communication methods, and impressive performance. ProxyStreams also incurs minimal overhead and achieves high-performance results when performing data operations on streams, highlighting its efficiency and practicality. Furthermore, the integration of a Python generator interface enhances usability and ensures seamless compatibility with ProxyStore's backend connectors, encompassing TCP and RDMA protocols.

References

[1] J. Gregory Pauloski, Valerie Hayot-Sasson, Logan Ward, Nathaniel Hudson, Charlie Sabino, Matt Baughman, Kyle Chard, and Ian Foster. 2023. Accelerating Communications in Federated Applications with Transparent Object Proxies. arXiv:2305.09593 [cs.DC]

[2] Rafael Vescovi, Ryan Chard, Nickolaus D. Saint, Ben Blaiszik, Jim Pruyne, Tekin Bicer, Alex Lavens, Zhengchun Liu, Michael E. Papka, Suresh Narayanan, Nicholas Schwarz, Kyle Chard, and Ian T. Foster. 2022. Linking scientific instruments and computation: Patterns, technologies, and experiences. *Patterns* 3, 10 (2022), 100606. https://doi.org/10.1016/j.patter.2022.100606

[3] Antonia Sierra Villarreal, Yadu Babuji, Tom Uram, Daniel S. Katz, Kyle Chard, and Katrin Heitmann. 2021. Extreme Scale Survey Simulation with Python Workflows. In *IEEE 17th International Conference on eScience (eScience)*. 206–214. https://doi.org/10.1109/eScience51609.2021.00031