Turbocharging Dask Apps: Accelerating Data Flow with ProxyStore

Klaudiusz Rydzy
Loyola University Chicago
Chicago, Illinois, USA

J. Gregory Pauloski (Advisor)
University of Chicago
Chicago, Illinois, USA

Kyle Chard (Advisor)
University of Chicago
Chicago, Illinois, USA

Abstract

Despite advancements in distributed computing libraries, performance challenges, such as data serialization and transfer, still persist. We focus on understanding data limitations within Dask, a versatile and popular Python library designed for distributed and parallel computing, and then investigate the potential of using the pass-by-proxy paradigm implemented by ProxyStore to address these inefficiencies. By integrating ProxyStore, we streamline data flow in Dask applications, reducing overheads associated with data serialization and scheduler overheads. Our approach evaluates the impact of proxies on data transfer times and overall computational efficiency. We find that our integration reduces task overheads by 5–6× on a real machine learning application.

1 Introduction

Dask [6] is popular in scientific computing for handling large datasets and parallelizing tasks across CPUs or GPUs. Dask's popularity is driven through key attributes, such as it's pure Python implementation and easy-to-use interfaces. In addition, Dask generally performs well; prior work has shown that Dask performs comparably to Apache Spark in most workloads [1, 2].

Dask Distributed extends Dask's capabilities by managing large-scale computations with dynamic scheduling, scalability, and real-time monitoring. Dask describes remote, asynchronous computations requested by a client as a task graph, which contains, at least, the function to execute and its parameters. Task graphs are communicated to the scheduler process, and eventually to the worker that will execute the task. Thus, the scheduler represents a single bottleneck in task execution. Without input/output data, tasks overheads are minimal—between 10 μs and 1 ms. However, these overheads can explode when large Python objects are included in the task graph (e.g., by passing a machine learning model directly as input to a task). Dask provides mechanisms to avoid embedding large objects in the task graph (e.g., Dask Array and DataFrame alternatives, delayed functions, or data chunking), but the viability of these mechanisms is specific to the application.

2 Approach

Here, we (1) investigate Dask inefficiencies, (2) address inefficiencies using the ProxyStore library, (3) improve serialization performance in ProxyStore for common data science types, and (4) demonstrate improved performance in real Dask applications. ProxyStore [4] manages data efficiently in distributed applications using proxy objects as transparent references to target objects in a global store. This pass-by-reference system, with just-in-time resolution, operates across processes and sites, allowing dynamic communication changes without altering application code [4, 5].

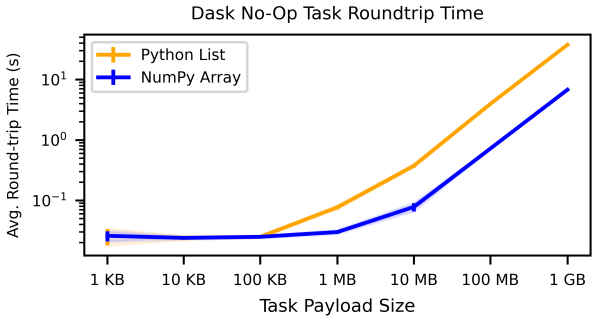

Figure 1: Dask overheads increase exponentially when large objects are embedded directly in the task graph. Dask handles certain data types better (e.g., NumPy arrays), but there remains room for performance improvements.

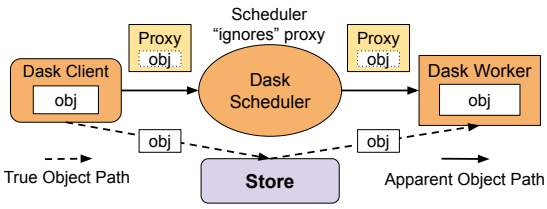

Figure 2: Proxy objects transparently decouple data flow from the Dask Distributed scheduler.

Overheads: In Dask, all task graphs are communicated to the scheduler which incurs costs for task objects that are not needed by the scheduler. In addition, task graphs are serialized with MessagePack [3] which is easy and flexible to use but has known performance limitations. Namely, MessagePack is inefficient for large messages and cannot handle bytestrings larger than 4 GB. Figure 1 shows how task round-trip time degrades in Dask when large objects are embedded in the task graph.

Integration: We created a custom Dask client that automatically proxies all inputs and outputs or can be configured to only proxy objects of a certain type or size. Passing objects that Dask struggles with, such as large or deeply nested objects, as proxies can significantly reduce data flow burdens in the Dask scheduler and minimize the amount of data being MessagePacked. The Dask scheduler effectively ignores proxies, as the proxy itself is very small, while the actual serialized object is located in a `Store` managed by ProxyStore. When a worker receives a task from the scheduler containing a proxy, the proxy is automatically resolved and the original object is reconstructed from the `Store`. Figure 2 illustrates the flow of objects when using ProxyStore with Dask.

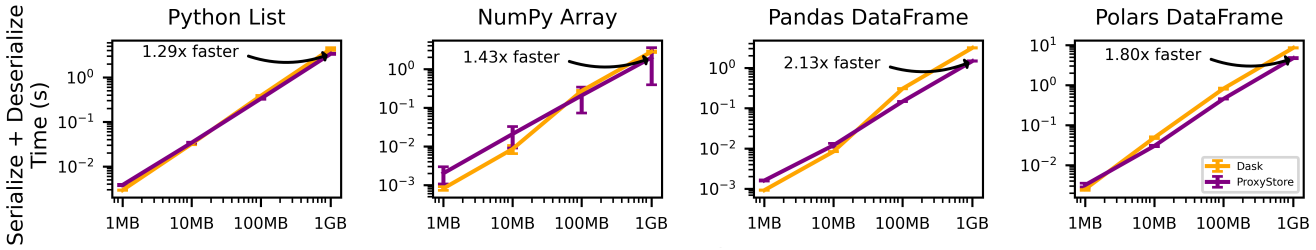

Figure 3: Time to serialize and deserialize messages is faster with ProxyStore for large objects compared to embedding objects directly in the task graph. ProxyStore also produces smaller messages which reduces I/O overheads in the scheduler.

Table 1: Serialization time for 1 GB data structures.

Data Structure	Serialization Time (s)		Improvement (%)
	Before	*After*	
Numpy Array	1.570	0.969	38.34%
Pandas DataFrame	1.573	0.788	49.88%
Polars DataFrame	3.744	2.436	34.94%

Serialization: Dask and, more broadly, scientific applications commonly use array-like data formats, including NumPy arrays, Pandas DataFrames, and Polars DataFrames. ProxyStore serializes objects using pickle/cloudpickle which are not optimized for these data science specific types. Thus, we improved ProxyStore's serialization system to include native support for the PyData stack and Polars and eliminate expensive memory copies where possible. These changes yielded up to 50% reductions in serialization times, as summarized in Table 1.

3 Evaluation

We evaluate our ProxyStore and Dask integration in two ways: low-level message communication and application performance. In Figure 3, we compare Dask's baseline message serialization for various data types and object sizes to message serialization when objects are proxied instead. We observe that Dask's performance degrades with larger objects due to the known inefficiencies of MessagePack. However, proxying objects before serializing messages yields considerable performance improvements.

Machine learning applications commonly require sending large and complex Python objects as task inputs or results. Thus, we use the federated learning benchmark application from the TaPS benchmarking suite [7] to understand the downstream performance implications of using ProxyStore. Federated learning is a collaborative machine learning approach where multiple clients train a shared model using their local data, without transferring the data to a central server. We perform a federated learning workflow using Dask and then extend the workflow to use ProxyStore to communicate models to and from training tasks. Figure 4 shows the overall runtime of the application. ProxyStore reduces the runtime by 30 seconds. To better understand the reasons for performance improvements, we plot the time between submitting a task and the start of execution on a worker in Figure 5. We observe 5–6× reduction in task overheads when using ProxyStore. Thus, ProxyStore can significantly speed up applications that send large objects as task inputs or results, and is beneficial for applications that utilize many shorter tasks.

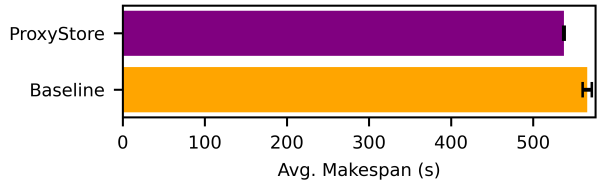

Figure 4: ProxyStore improves the runtime of the federated learning application. Error bars denote standard deviation.

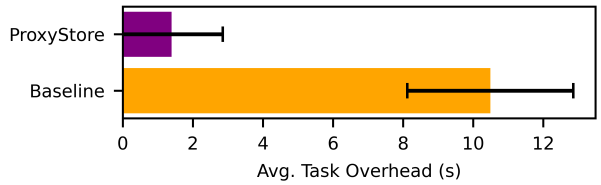

Figure 5: Avg. time between submitting a task to Dask and the task beginning execution on a worker. Pass-by-proxy reduces serialization and scheduling overheads, thereby improving latency. Error bars denote standard deviation.

4 Conclusion

ProxyStore is an effective alternative for efficiently managing objects in distributed Dask applications. ProxyStore provides flexibility to optimize applications without needing to modify task code and is compatible with existing applications via our custom Dask client. We demonstrate the pass-by-proxy model can reduce Dask serialization and scheduling overheads, and we optimize ProxyStore for common data types used in Dask applications. We validate our investigation by applying ProxyStore to a distributed machine learning application using Dask and find that ProxyStore can reduce task overheads by 5–6×. Our scripts, analysis, and steps to reproduce our results are available online at https://github.com/KlaudiuszRydzy/ProxyStore-Dask.

References

[1] Mathieu Dugré, Valérie Hayot-Sasson, and Tristan Glatard. 2019. A performance comparison of dask and apache spark for data-intensive neuroimaging pipelines. In *2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*. IEEE, 40–49.

[2] Mathieu Dugré, Valérie Hayot-Sasson, and Tristan Glatard. 2023. Performance comparison of Dask and Apache Spark on HPC systems for neuroimaging. *Concurrency and Computation: Practice and Experience* 35, 21 (2023), e7635.

[3] MessagePack [n. d.]. MessagePack, It's like JSON. but fast and small. https://github.com/msgpack/msgpack. Accessed July 2024.
[4] J. Gregory Pauloski, Valerie Hayot-Sasson, Logan Ward, Alexander Brace, André Bauer, Kyle Chard, and Ian Foster. 2024. Object Proxy Patterns for Accelerating Distributed Applications. arXiv:2407.01764 [cs.DC] https://arxiv.org/abs/2407.01764
[5] 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. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15.
[6] Matthew Rocklin et al. 2015. Dask: Parallel computation with blocked algorithms and task scheduling.. In *SciPy*. 126–132.
[7] TaPS [n. d.]. TaPS: Task Performance Suite. https://github.com/proxystore/taps. Accessed July 2024.