OAK RIDGE National Laboratory

Nov 16, 2025 | St. Louis, MO

From Edge to HPC: Investigating Cross-Facility Data Streaming Architectures

Anjus George, Michael Brim, Christopher Zimmer, David Rogers, Sarp Oral, Zach Mayes

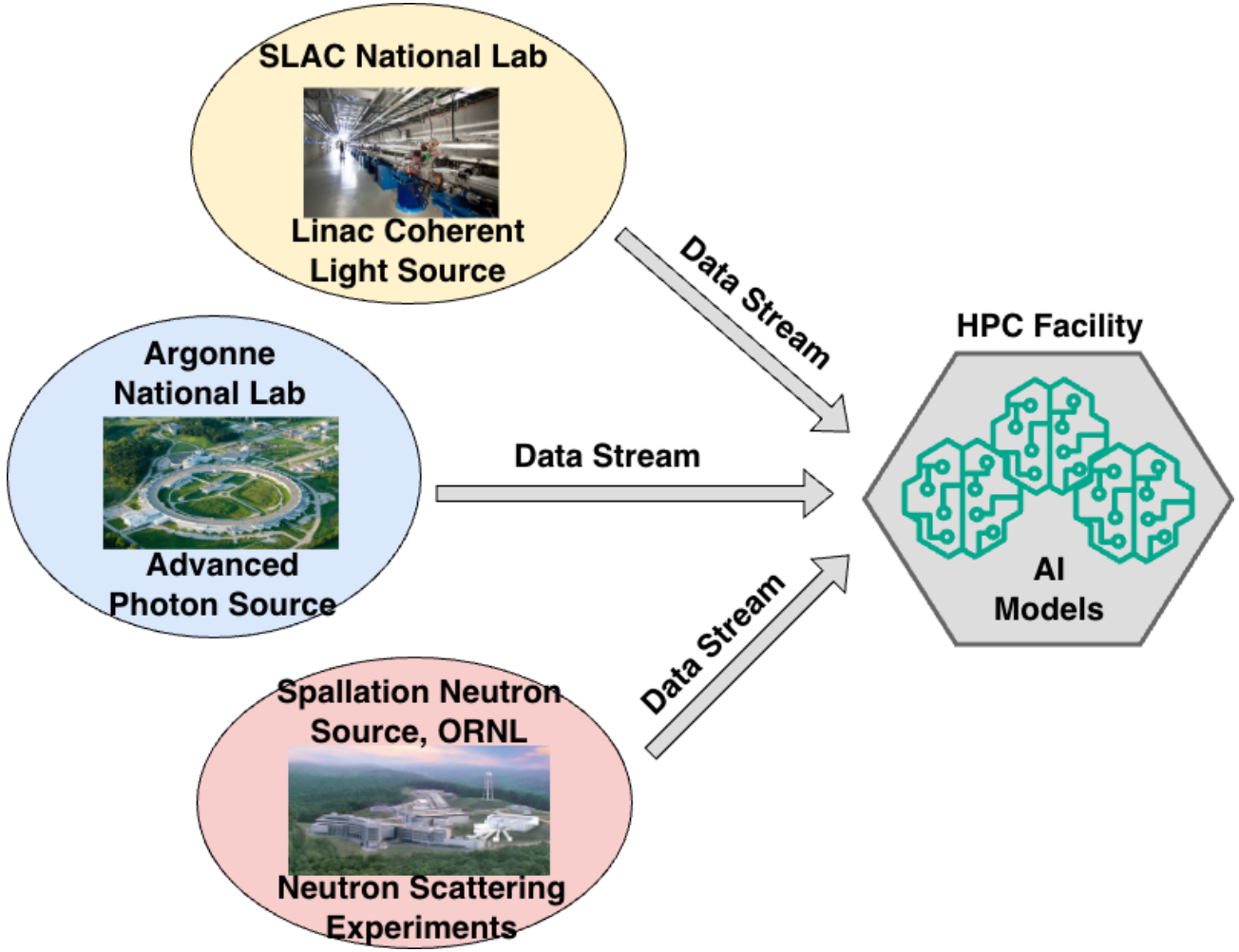
Oak Ridge National Laboratory

U.S. DEPARTMENT of ENERGY

ORNL IS MANAGED BY UT-BATTELLE LLC
FOR THE US DEPARTMENT OF ENERGY

Introduction

- Modern experimental workflows have real-time data analysis requirement
- Many of them send the experimental data to HPC facilities for on-the-fly analysis and timely feedback.
- Some examples are **LCLS at SLAC, APS at ANL,** and **SNS at ORNL**
- Data Streaming allows to send data with minimal delay using direct memory transfers
- At OLCF, we are exploring different data streaming architectures to study,
    - Performance impacts
    - Impact of architectural choices
    - Abstraction trade-offs



Contributions

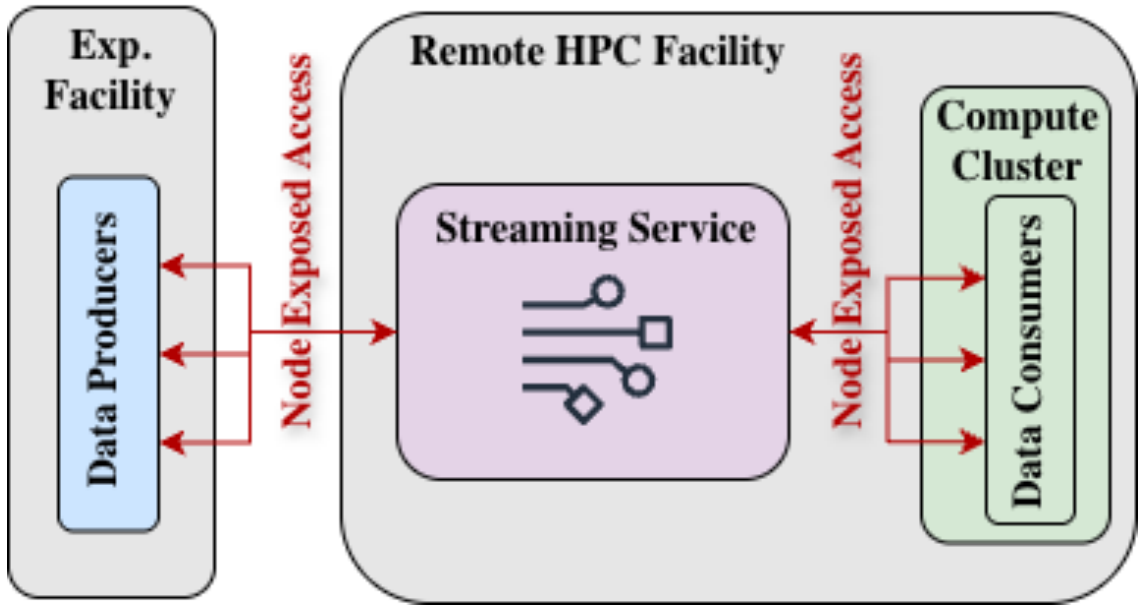
1. Architecture and deployment of 3 cross-facility data streaming architectures:
   - **Direct Streaming (DTS)**
   - **Proxied Streaming (PRS)**
   - **Managed Service Streaming (MSS)**

2. Characterize their streaming **throughput, round-trip time, and overhead** using IRI scientific workflows

3. Evaluated the architectures using 3 types of messaging patterns:
   - **Work sharing, Work sharing with feedback, and Broadcast and Gather**

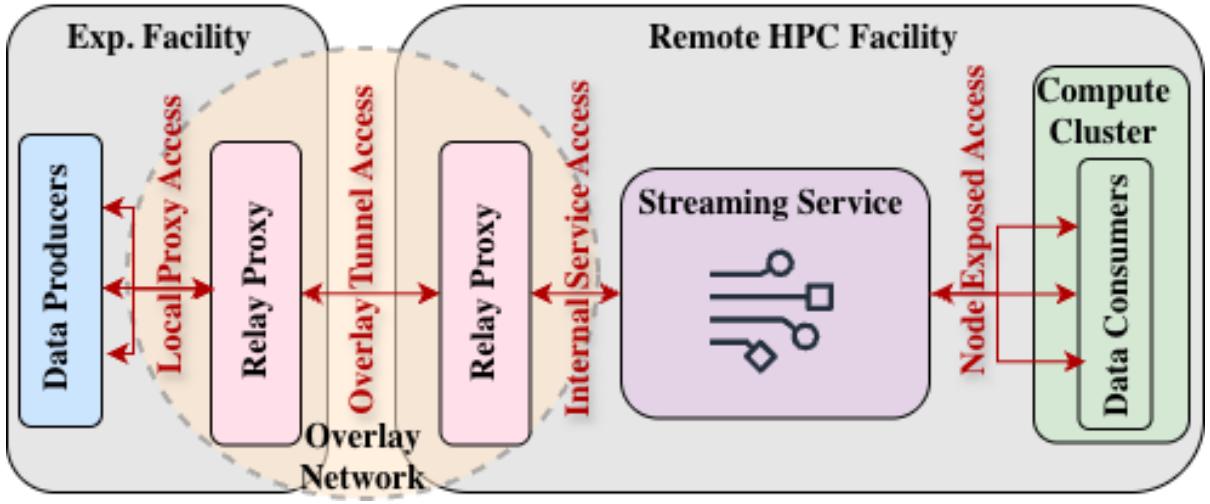
Data Streaming Architectures: Direct Streaming (DTS)

- Deploys streaming service on destination nodes
- Exposed via node-level network ports
- Pros:
  - No intermediate proxies
  - Therefore, minimal latency
- Cons:
  - Requires explicit admin configs
  - Complex because of firewalls and NAT
  - Security risks by node-level exposure
  - Poor scaling



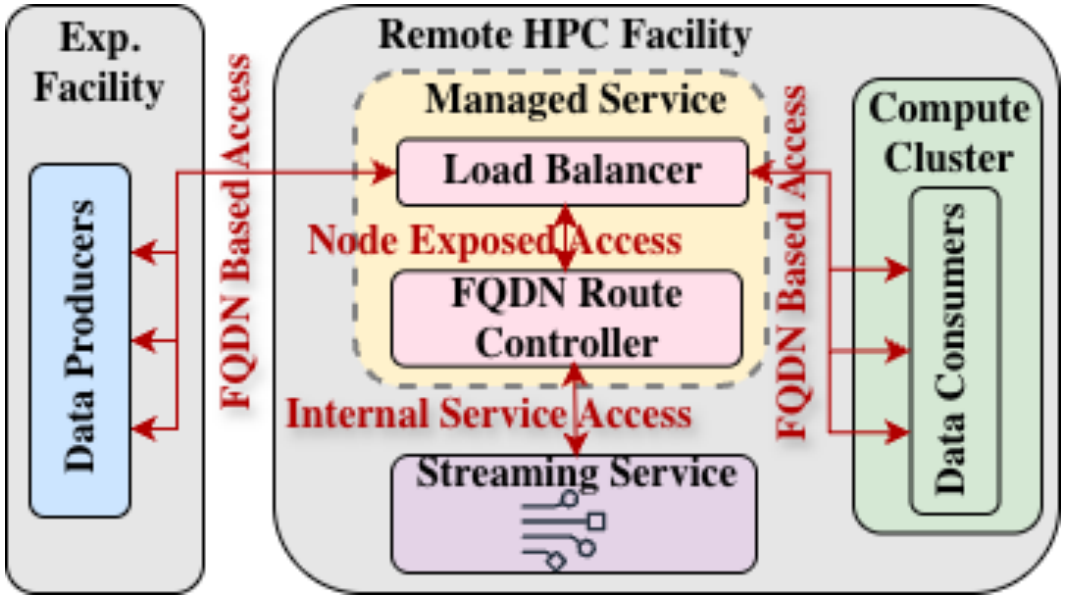
Data Streaming Architectures: Proxied Streaming (PRS)

- Employs intermediary proxies to relay data
- Local proxies deployed on edge/gateway nodes
- Pros:
  - Overcomes NAT and firewall barriers with centralized firewall rules
  - Avoids complexity of managing node-level IPs and ports
  - Admins can pre-authorize stable endpoints
- Cons:
  - Extra layers of proxies can add delays

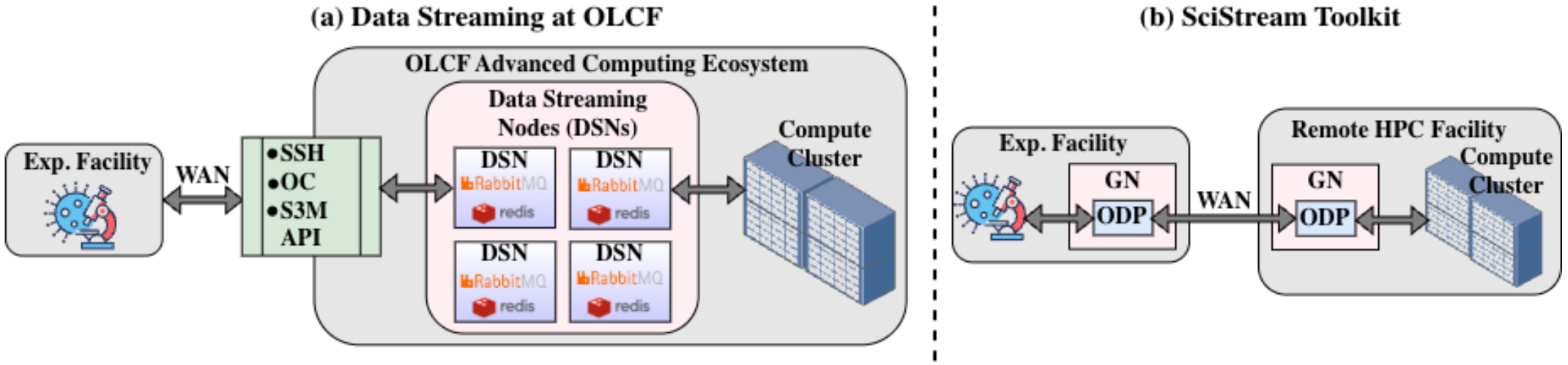


Data Streaming Architectures: Managed Service Streaming (MSS)

- Facilities platform infrastructure manages the data flow
- Data is sent to a stable Fully Qualified Domain Name (FQDN)
- Pros:
  - Abstracts away networking complexities
  - Users don't manage IP address, ports and NAT
  - Facility handles all routing and DNS
  - High convenience for users
- Cons:
  - Abstraction layers and extra components might add to latency



Streaming Deployment: Frameworks and Toolkits



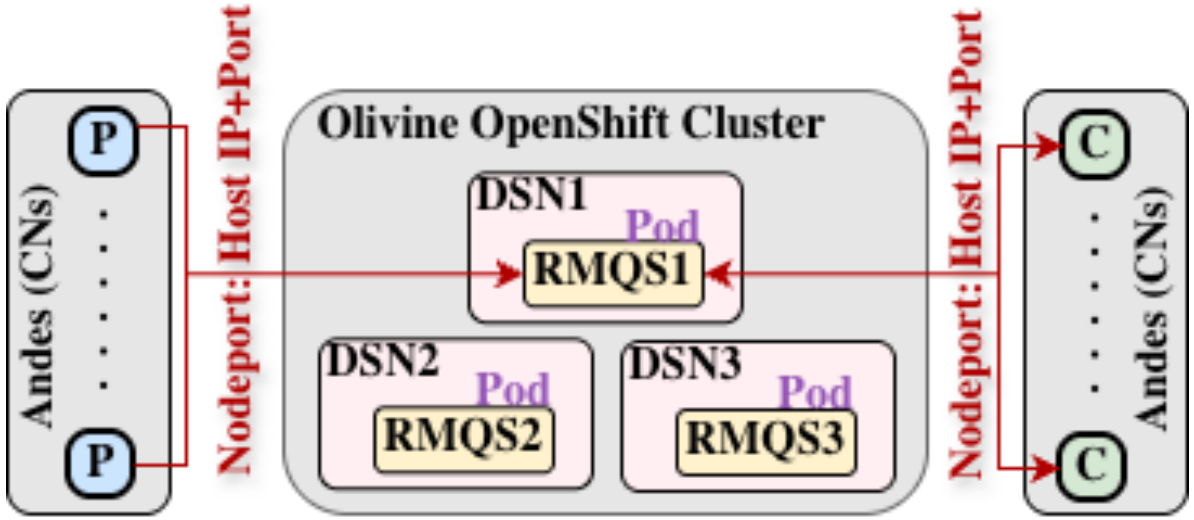
- Used Data streaming infrastructure at OLCF to deploy all architectures
- Consists of DS2HPC framework within OLCF Advanced computing Ecosystem (ACE)
- Data Streaming Nodes (DSNs) managed in RedHat OpenShift environment in the OLCF Olivine cluster
- OLCF Secure Scientific Mesh (S3M) provides authentication and provisions streaming service
- SciStream memory-to-memory toolkit developed by researchers at ANL
    - Consists of Control Server (S2CS), Data Server (S2DS), and User Client (S2UC) components

Streaming Deployments: Specifications

- Used RabbitMQ as the streaming service for the deployments
  - Provides message queues to store messages
  - Producers to send messages to queues
  - Consumers consume messages from queue
- A 3 node RabbitMQ cluster was deployed on the DSNs
- Producers and consumers are deployed on OLCF Andes compute cluster
- Current network connectivity is limited to 1Gbps on the DSNs

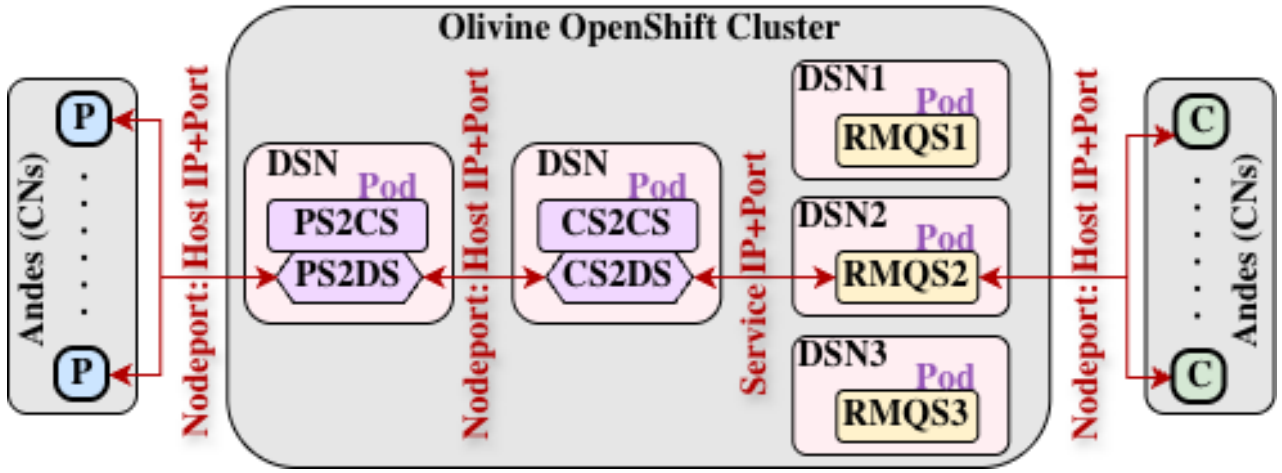
Streaming Deployments: DTS Deployment

- Used RabbitMQ Helm Chart to deploy 3 RabbitMQ server pods on 3 separate DSNs
- Resource limits are set to use 12 CPUs, 32 GiB memory and 15 GiB persistent storage
- RabbitMQ service is exposed via NodePort 30671 on each DSN
- DTS is feasible only between sites with direct connectivity
- However, it helps to quantify the streaming overhead of other architectures



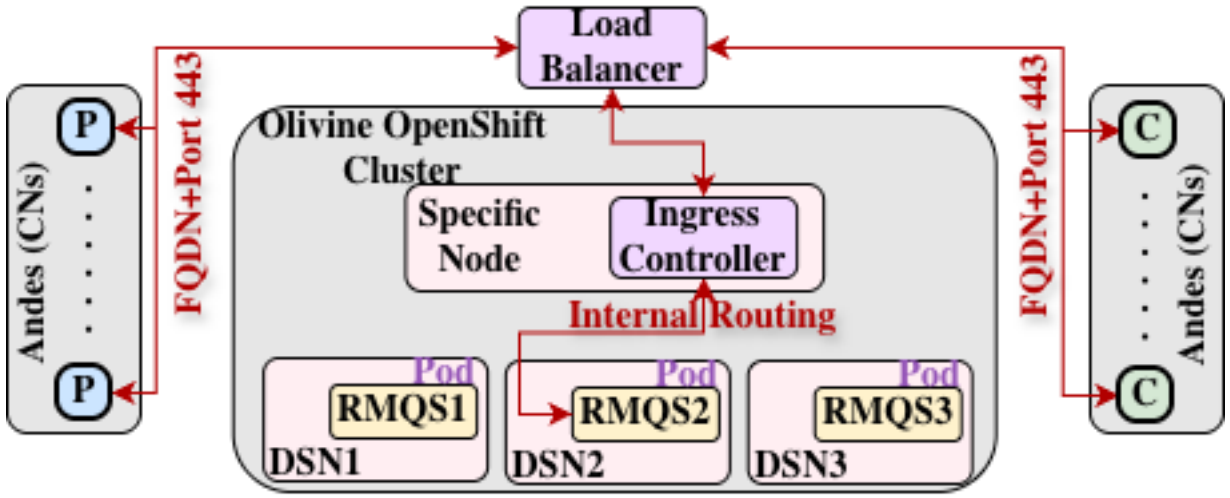
Streaming Deployments: PRS Deployment

- Utilized the SciStream toolkit to deploy PRS
- SciStream producer and consumer S2CS were deployed on 2 separate DSNs
- Evaluated two tunneling methods – Stunnel and HAProxy
- Exposed each hop using node-IP:NodePort
- Streaming service is a 3-node RabbitMQ cluster
- S2CS pods and RabbitMQ server pods exposed via node-level access
- S2UC is deployed on Andes login node
- Sends inbound and outbound requests to the S2CSs to create proxies (S2DSs)



Streaming Deployments: MSS Deployment

- 3 node RabbitMQ cluster is provisioned via the facility's S3M API
- Requires an authentication token generated via S3M
- Once provisioned, provides an AMQPS URL that can be used in RabbitMQ client API
- Load balancer is a dedicated hardware located outside OpenShift cluster
- Load balancer forwards traffic to cluster's ingress controller



Evaluation: Messaging Patterns

- Messaging patterns found in AI and HPC workloads
  - **Work sharing:-** hyperparameter searches and Monte Carlo ensembles
    - Thousands of tasks are launched without any post-dispatch communication
  - **Work sharing with feedback:-** DL frameworks like TensorFlow-PS and MXNet
    - Weight shard is sent to each worker, and each worker returns its gradient to the same shard
  - **Broadcast and gather:-** distributed data parallel training frameworks like NCCL and Gloo
    - Performs fan-out, reduce-scatter and all-gather

Evaluation: Streaming Workloads

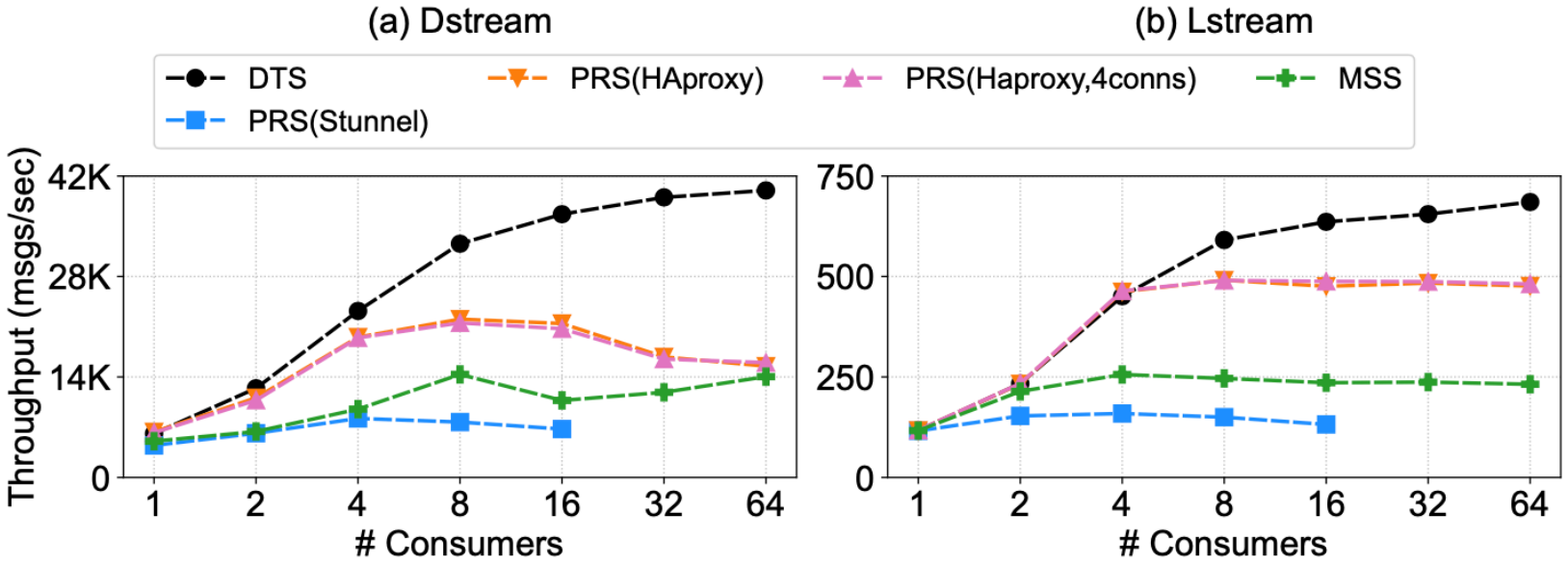
- Streaming Workloads:
  - **Workloads based on IRI workflows:**
    - **Deleria (Dstream), and LCLS (Lstream)**
  - **Generic workload**

Characteristics	Deleria	LCLS	Generic
Payload size	≈KiB range	≈1 MiB	4 MiB
Payload format	Binary	HDF5	Binary
Payload element	Events	Events	Variables
Data packaging	Variable # events/msg	Variable # events/msg	One item/msg
Data rate	32 Gbps	30 Gbps	25 Gbps
Consumption parallelism	Parallel (non-MPI)	Parallel (MPI-based)	Parallel (MPI-based)
Production parallelism	Parallel (non-MPI)	Parallel (MPI-based)	Parallel (MPI-based)

Evaluation: Streaming Simulator

- Developed using Golang by incorporating RabbitMQ AMQP client APIs
- **Simulator Inputs:**
  - Streaming characteristics of workloads
  - Type of streaming architecture (DTS, PRS or MSS)
  - Streaming service specific parameters (type of acknowledgements, number of queues, prefetch count)
  - Experiment configurations (# producers and consumers, message count, experiment duration)
  - Infrastructure or toolkit specific options (URL for connection, number of connections, TLS)
- RabbitMQ is configured with specific parameters and queue model for each pattern

Evaluation: Work Sharing Pattern

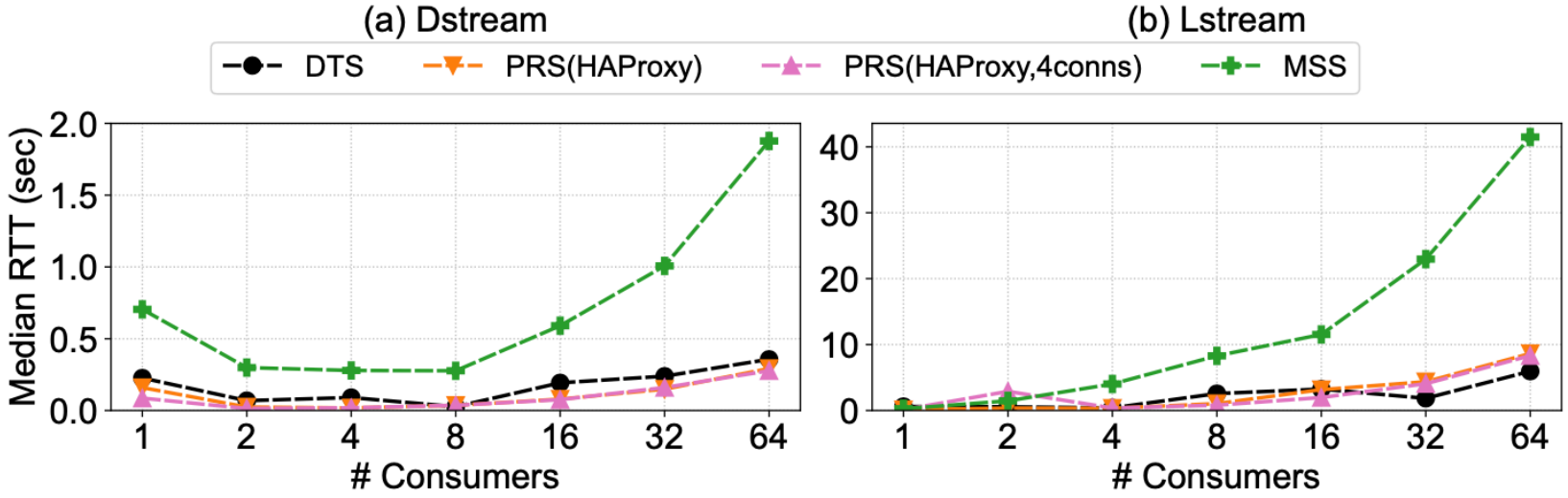


- **Dstream:**
  - 1 prod and 1 cons, PRS with HAProxy achieved highest throughput
  - As prods and cons increase, DTS improved, max of 39K
  - PRS with Stunnel suffered most, and HAProxy scaled better
  - MSS has lower throughput than PRS-HAProxy

- **Lstream:**
  - DTS saturates beyond 8 consumers
  - PRS-HAProxy scales well upto 4 consumers
  - MSS saturated beyond 4 cons

- **Overhead:**
  - Upto 2.5x overhead compared to DTS

Evaluation: Work Sharing with Feedback Pattern

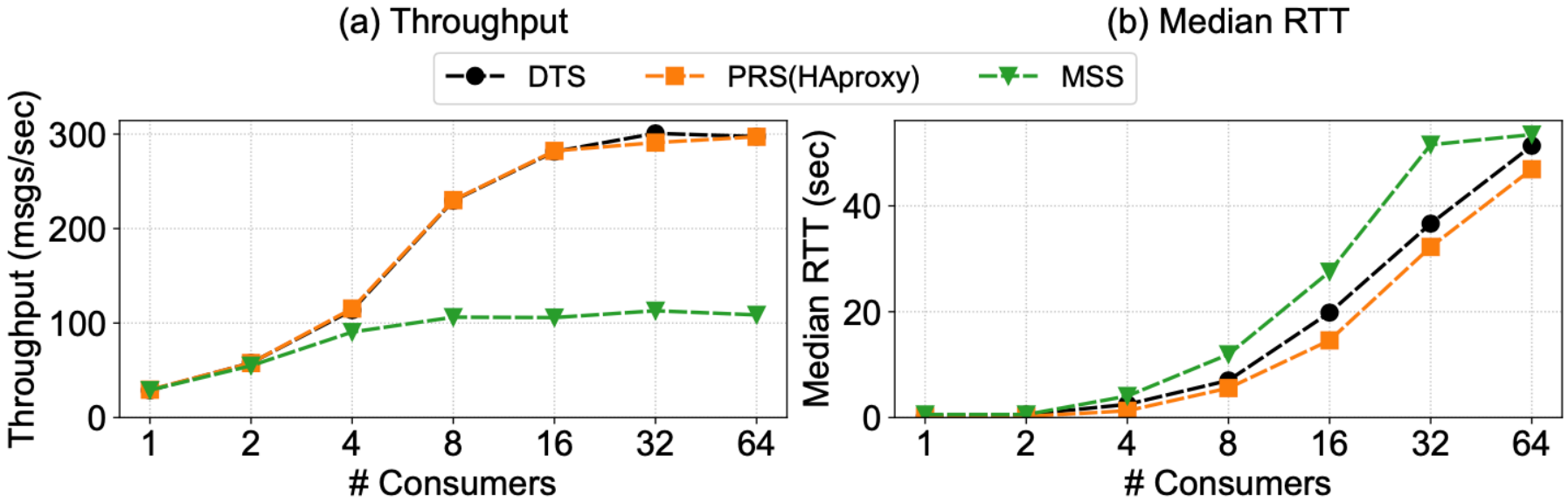


- **Dstream:**
  - Both PRS and DTS has median RTT < 0.5 sec
  - RTT increased for both DTS and PRS for cons > 8
  - MSS showed higher RTTs with sharp increase at 64 cons

- **Lstream:**
  - RTTs were under 200ms for DTS and PRS
  - 600ms (3x) for MSS
  - MSS RTT spiked after 8 cons

- **Overhead:**
  - PRS has equivalent performance as DTS
  - MSS has around 6.9x overhead

Evaluation: Broadcast and gather pattern



- **Broadcast:**
  - PRS scales equivalently to DTS
  - MSS has bottlenecks >4 consumers
  - DTS and PRS has bottlenecks only >32 consumers

- **Broadcast and Gather:**
  - All 3 architectures have comparable RTTs
  - Upto 4 consumers, RTTs are under 5 seconds
  - Beyond this, RTTs increase sharply

Conclusion and Future Work

- Architecturally DTS offers lowest hop path but requires explicit firewall/iptables rule configurations
- PRS reduces such requirements via stable, pre-authorized endpoints
- MSS fully abstracts networking by letting facility's platform manage data flow
- Deployment wise DTS is simplest, PRS adds moderate complexity, and MSS is most streamlined
- Work sharing: PRS and MSS experience significant overhead
- Work sharing with feedback: PRS performs as well or better than DTS, MSS has overhead
- Broadcast and gather: PRS scales equivalently to DTS, and MSS eventually shows lower latency
- **Future work:**
    - Other streaming architectures: EJFAT, Banana Pepper
    - Usage of high-speed network: 100 Gbps connectivity for the DSNs
    - Usage of network and gateway load balancers

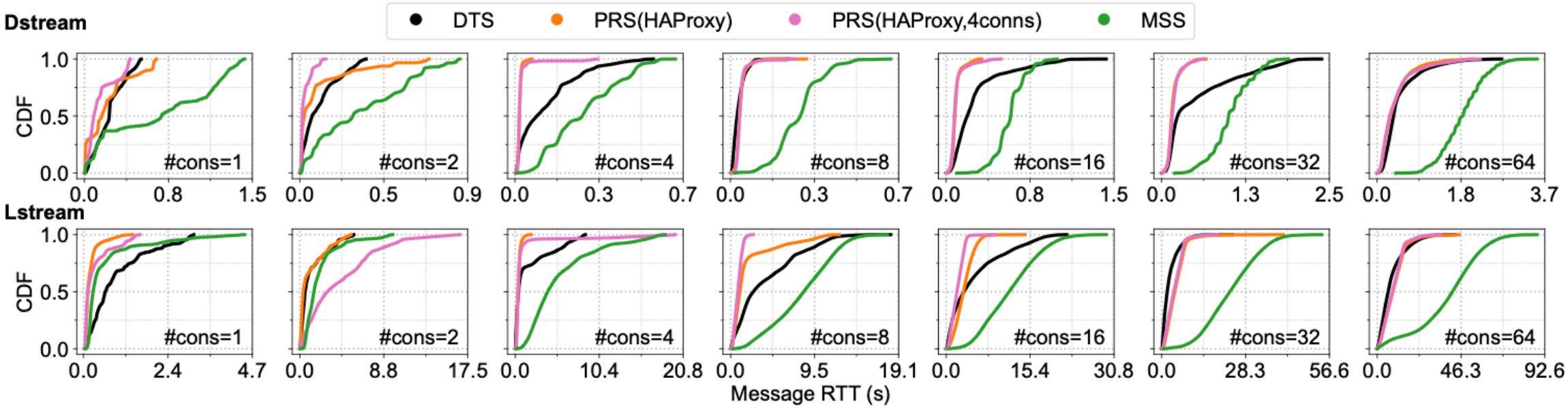
Acknowledgements

- Grateful to my colleagues: Nick Schmitt, Ethan O'Dell, Steven Lu, Tyler Skluzacek, A.J. Ruckman, Paul Bryant, and Gustav Jansen
- Thanks to SciStream team at ANL: Flavio Castro and Rajkumar Kettimuthu
- This research used resources of the Oak Ridge Leadership Computing Facility located at Oak Ridge National Laboratory, which is supported by the Office of Science of the Department of Energy under contract No. DE-AC05-00OR22725.

Thank you!

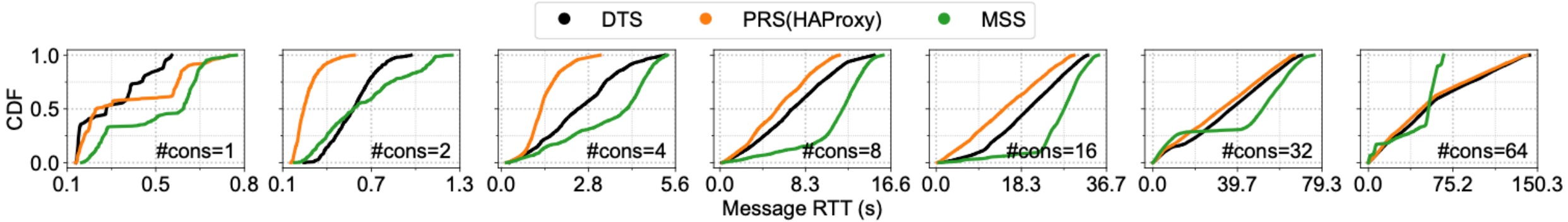
Questions?

CDF of Individual Message RTTs for Work Sharing with Feedback Pattern



Beyond eight consumers, all architectures exhibit a noticeable rightward shift in the CDF, particularly in the MSS architecture under the Lstream workload. This is likely due to the direct feedback path from consumers to producers, which introduces significant RTT bottlenecks. In contrast, the PRS architecture consistently maintains tighter RTT distributions, with less variation than MSS, and often performs comparably to DTS. Notably, for the 64-consumer case, PRS keeps 80% of message RTTs under 0.7 seconds for Dstream and 12.5 seconds for Lstream, demonstrating uniformity in latency. However, increasing PRS connections from one to four does not yield observable improvements in RTT.

CDF of Individual Message RTTs for Broadcast and Gather Pattern



The CDF trends of per-message RTT in this pattern show that, in some cases, particularly with 2 to 16 consumers, PRS exhibits lower RTTs than DTS. MSS still shows comparatively higher RTTs, though not as high as in the work sharing with feed-back pattern. Another notable trend is that as consumer count reaches or exceeds 32, the CDFs of all three architectures converge, and MSS begins to outperform the other two.

More on LCLS

The Linac Coherent Light Source (LCLS) at SLAC National Accelerator Laboratory provides X-ray scattering for molecular structure analysis and streams experimental data to enable rapid analysis and decision-making between experiment runs. The LCLStream pilot project trains a generalist AI model using streamed detector data, from both archived and live LCLS/LCLS-II experiments, to support tasks like hit classification, Bragg peak segmentation, and image reconstruction. This AI-driven approach serves as a shared backbone for various downstream data analysis tasks. With the new LCLS-II producing data at 400x the rate of its predecessor, streaming up to 100 GB/s to HPC systems will be essential for responsive analysis and experiment steering. LCLStream aims to support online streaming and real-time analysis during experiment execution, eliminating delays associated with waiting for data to be written to file storage systems before processing.

More on GRETA

GRETA (Gamma-Ray Energy Tracking Array) is a gamma-ray spectrometer currently being deployed at the Facility for Rare Isotope Beams at Michigan State University. It enables real-time analysis of gamma-ray energy and 3D position with up to 100x greater sensitivity than existing detectors. The associated workflow software, Deleria, continuously streams experimental data over ESNet to hundreds of analysis processes on an HPC system, processing up to 500K events per second. Deleria supports time-sensitive streaming and has been deployed across ESNet and ACE to demonstrate a distributed experimental pipeline. Recent emulation experiments on ACE scaled to 120 simulated detectors, achieving sustained bidirectional streaming rates of~35 Gb/s.

More on Experimental-HPC Workflows

Recent examples of such experimental-HPC workflows include the Linac Coherent Light Source (LCLS) workflow at SLAC National Accelerator Laboratory, which streams diffraction frames from the LCLS light source over Esnet directly to HPC systems at OLCF. There, AI models identify Bragg peaks and recommend parameter changes while the sample is still in the beam. At Argonne's Advanced Photon Source (APS), an AI enabled workflow pushes 2 kHz ptychography data to embedded GPUs for edge inference, while HPC nodes retrain the model on the fly, enabling dose-efficient imaging and real-time experimental steering. Another example is an edge-to-exascale workflow using Frontier at OLCF, where a Temporal Fusion Transformer (TFT) is employed at the Spallation Neutron Source (SNS) to predict 3D scattering patterns and adjust beam settings within minutes instead of hours.

More on Messaging Patterns

These patterns align closely with communication motifs found in many AI and HPC workloads. The work sharing pattern is used in embarrassingly parallel workloads such as hyperparameter searches or Monte Carlo ensembles, where thousands of short, independent jobs are launched without any post-dispatch communication. Schedulers like Slurm job arrays or GNU Parallel distribute tasks once, allowing each node to compute in isolation, matching the work sharing model.

The work sharing with feedback pattern appears in data-parallel deep learning (DL) frameworks like TensorFlow-PS and MXNet, where a weight shard is sent to each worker, and each worker returns its gradient to the same shard. This push–pull interaction forms a classic distribute-with-reply loop. Another example is master–worker task farms that require immediate results, such as real-time inference on micro-batches, where each reply must be routed back to the originating producer.

The broadcast and gather pattern is common in distributed data-parallel (DDP) training frameworks like NCCL and Gloo, which perform a fan-out of weights, a reduce-scatter of gradients, and an all-gather of aggregated values every iteration. Another instance is large-scale metric aggregation, where a single node issues a collect request and all workers send back metrics to be reduced at the initiator.