ServiceX: A distributed, caching, columnar data delivery service

B. Galewsky\textsuperscript{A}, R. Gardner\textsuperscript{B}, M. Neubauer\textsuperscript{A}, J. Pivarski\textsuperscript{C}, L. Gray\textsuperscript{E}
I. Vukotic\textsuperscript{B}, G. Watts\textsuperscript{D}, M. Weinberg\textsuperscript{B}

A. University of Illinois at Urbana–Champaign, B. The University of Chicago,
C. Princeton University, D. University of Washington, E. Fermilab

HSF DAWG -- DOMA Access joint meeting

3/23/2020
Delivering data to new analysis platforms

First introduced in Feb 2018 whitepaper: delivery of data from lakes to clients (insulate from upstream systems, remove latencies, reformat, filter & possibly accelerate)

ServiceX focuses on integration with Rucio and reformatting for pythonic tools & endstage analysis systems

DOMA/AS groups interested in R&D for data delivery for analysis of columnar and other data formats

Supports multiple input types (xAOD, flat ntuples, ...) and common data mgt (Rucio, XCache)

Utilize industry standard tools (GKE, on-prem Kubernetes, Helm, Kafka, Redis, Spark, ...)

Reproducible, portable deployments
Classic analysis workflow

xAOD → DAOD → flat ntuples → skimmed ntuples → histograms/plots

- First two formats are prescribed, but enormous variation in after that

Standard analysis might have “primary ntuple”

- Write ntuplization code to dump xAOD into flat trees with specialized objects
- Submit jobs to HTCondor by hand
- Primary ntuple then skimmed/trimmed; some data replicated (multiple times)
- Selections/cutflows baked into analysis
- Adding new variables means throwing previous skim, replicating everything
Columnar data R&D efforts

Recast data so attributes of physics objects grouped into contiguous columns, rather than grouping by event and then object

- Much more efficient for processing!
- Updating event content (or corrections) can be done by adding columns to existing data
- Can cache only necessary data for computation; No longer need to load entire event in memory

However, this is a significant change for analyzer

- New syntax can be simpler, more expressive
- Imagine analysis code with no for() loops...
ServiceX goals

Adding components to the intelligent data delivery ecosystem. Input-agnostic service to enable on-demand data delivery.

Tailored for nearly-interactive, high-performance array-based analyses

- Provide uniform interface to data storage services; users don’t need to know how or where data is stored
- Capable of on-the-fly data transformations into variety of formats (ROOT files, HDF5, Arrow buffers, Parquet files, ...)
- Pre-processing functionality: Unpack compressed formats, filter events in place, project data columns

Support for columnar analyses. Start from any format, extract only needed columns
Users specify needed events/columns and desired output format
- Use metadata tags (real/sim data, year, energy, run number, ...)
- Any required preselection

**ServiceX**

- Queries backend (Rucio) to find data
- Gives unique token to identify request
- Access data from storage through XCache
- Validates request: Check on small sample, determine message size
- Extract requested columns, perform data transformations
- Send output to message broker for analysis
ServiceX implementation

System designed to be modular
- Can switch out modules to transform different types of input data, swap schedulers, ...

Implemented as central service in Kubernetes cluster on Scalable Systems Lab (SSL) cluster
- Easy to deploy: Just use Helm chart to define plugins to run
- Service can be deployed on variety of systems, including individual laptops
- Reproducible pattern for deployment on Kubernetes clusters (e.g. Tier2s, institutional k8s T3?)

Composed of multiple deployments: REST API server, DID finder, transformer, message broker
- API server: Manages requests via RabbitMQ with Postgres DB
- DID finder: Queries data lake via Rucio, writes ROOT files to XCache
- Transformer: Takes input files from XCache, outputs in various formats (ROOT files with flat trees, Awkward arrays, Parquet, Arrow tables, ...)
- Kafka manager: Receives input from producer (transformer) and makes topics available to consumers (analysis jobs)
ServiceX in the IRIS-HEP ecosystem

Data Lake

Tier 2

XCache

ServiceX

Dataset Finder

Code Generator

Cached Distribution

Analysis Facility

IRIS-HEP Scalable Systems Lab

HSF DAWG -- DOMA Access joint meeting
ServiceX architecture

REST Interface

DID Finder
Pre Flight Check
Code Generator
Transformer Manager
Transformer

RabbitMQ
Minio
Kafka
Kubernetes

Open source
Experiment Specific
Cross Experiment

3/23/2020

HSF DAWG -- DOMA Access joint meeting
Connections to DOMA

**ServiceX is part of DOMA's iDDS**
- feeds data to downstream analysis systems
- enables data transformations developed in individual environments to be scaled up to production-based operations

**ServiceX is being prototyped using IRIS-HEP's Scalable Systems Lab**
- includes reproducible pattern for deployment
- entire project implemented as central service in Kubernetes cluster on SSL
- takes advantage of SSL infrastructure support to develop new features quickly.
10TB across 7794 files (~1.3GB/file)

10-column test: reads ~ 10% of file
- Time for transformer to process file: ~ 13 seconds
- ~1 second each for file access and output to MinIO/Kafka
- Output size ~ 3MB (~400 reduction from input)
- 400-transformer test completes in <5 minutes

100-column test: reads ~ 30% of file
- Total time to process file: ~ 31 seconds
- ~1 second for file access (unless high load)
- Output size ~ 38MB (~35 reduction from input)
- 400-transformer test completes in <25 minutes
ServiceX demo

Sample transform request including dataset to be transformed and output columns

Development version; some of these decisions will be hidden from the user

User receives unique request ID

Rudimentary updates on the progress of the transformation

To be augmented with status plots

3/23/2020

HSF DAWG -- DOMA Access joint meeting
Output cached in message broker
Can be read out asynchronously, re-run

User analysis code

Time to go from xAOD or DAOD to analysis plot only a couple of minutes
Current work

- Transformers now use compiled C++ to speed up read/write
  - Implementing via SQL-like declarative selection statements to permit in-place filtering for requests
- Service scales well with multiple large concurrent requests
- Sharing swapable components: transformers, message brokers (Kafka and Redis), ...
  - Coordinating development of transformers with iDDS/ESS production use cases
- Adding multiple transform input formats
  - Available for xAOD, DAOD, flat TTree formats
- Stability testing: Ensure output robust vs randomly killing transformer nodes
- Ramping up to v1.0
Backup slides
ServiceX running on SSL: RIVER 2

- Currently deployed on Kubernetes cluster on SSL RIVER 2
- Performance testing: Running with hundreds of transformers on cluster
- Has Prometheus monitoring dashboard
IRIS-HEP R&D efforts in DOMA & Analysis Systems

- DOMA/AS groups interested in R&D for data delivery for analysis of columnar and other data formats
- Supports multiple input types (xAOD, flat ntuples, ...) and common data mgt (Rucio, XCache)
- Utilize industry standard tools (GKE, on-prem Kubernetes, Helm, Kafka, Redis, Spark, ...)
- Reproducible, portable deployments
Loop-less array programming

Can do all kinds of stuff with optimized linear algebra computations

- Multidimensional slices of data
- Element-wise operations (e.g. muons_pz = muons_pt * sinh(muons_eta))
- Broadcasting (e.g. muon_phi - 2 * pi)
- Event masking, indexing, array reduction, etc.

But we don’t have a simple rectangular arrays

- Nested variable-size data structures everywhere in HEP
- Jagged arrays handle this with two 1D arrays:
  - First array contains long list of values, one per object
  - Second array contains breaks that give event boundaries
Loop–less array programming

- But this is shown to the user as a list containing lists of various lengths:

```python
import uproot
tf = uproot.open("HZZ-objects.root")
t = tf["events"]

in [4]:
    a = t.array("muoniso")
    # muon isolation variable; multiple per event
    a
    a

    in [5]:
    a[Event 1]
    Event 2
    ...

    out [5]:
    <JaggedArray [4.2001534 2.1510613 2.1880474 1.4128217 3.3835042 ... [3.7629452], [0.5508107], [0.]] at 7b2291313248>

    The implementation is a façade: these are not millions of list objects in memory but two arrays with methods to make them behave like nested lists.

    In [6]:
    a.offsets

    out [6]:
    array([0, 2, 3, ..., 3823, 3824, 3825])

    In [7]:
    a.content

    out [7]:
    array([4.2001534, 2.1510613, 2.1880474, ..., 3.7629452, 0.5508107, 0.], dtype=float32)
```
Open source technologies used

- Flask
- Apache Arrow
- Kafka
- Helm
- Kubernetes
- Docker
- RabbitMQ
- SQLite
- PostgreSQL
- Pytest