Data Management Package for the novel data delivery system, ServiceX, and Applications to various physics analysis workflows

KyungEon Choi[1,*] and *Peter* Onyisi[1]

[1]The University of Texas at Austin

Abstract. Recent developments of HEP software allow novel approaches to physics analysis workflows. The novel data delivery system, ServiceX, can be very effective when accessing a fraction of large datasets at remote grid sites. ServiceX can deliver user-selected columns with filtering and run at scale. We introduce the ServiceX data management package, ServiceX DataBinder, for easy manipulations of ServiceX delivery requests and delivered data using a single configuration file. We show various practical use cases within analysis pipelines that range from a data delivery of a few columns for machine learning study to a data delivery for full-scale physics analysis.

1 Introduction

ServiceX [1, 2] is a novel data delivery service that has been developed in the area of Data Organization, Management and Access (DOMA) of the Institute of Research and Innovation in Software for High Energy Physics (IRIS-HEP) [3]. It features quick and easy access to large data sets at remote locations by running transformation on a Kubernetes cluster at scale and then delivers user-selected columns with filtering. ServiceX data delivery requests are made via a REST interface. Figure 1 shows a schematic workflow of ServiceX requests to data delivery.

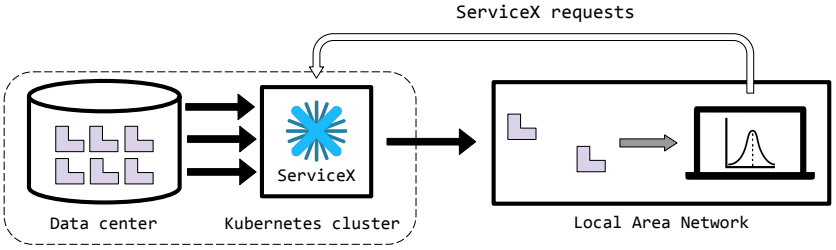

Figure 1. A schematic of ServiceX workflow.

The ServiceX client library [4] is the python library for users to interact with ServiceX. It provides essential features such as submitting a ServiceX data delivery request, downloading

*e-mail: kyungeonchoi@utexas.edu

transformed outputs, handling of ServiceX endpoint access token and local data cache. The python package, `servicex-databinder` [5], presented in this paper was motivated to provide a better user experience than the ServiceX client library, especially when dealing with many ServiceX requests as typical high-energy physics analyses.

2 Data Management Package for ServiceX

The primary feature of `servicex-databinder` is a configuration file which includes general settings and details of each ServiceX request. It is also effortless to run a `servicex-databinder` configuration file and get data delivered from remote as specified in the configuration file.

2.1 Configuration file for `servicex-databinder`

```yaml
General:
  ServiceXName: servicex-uc-af
  Transformer: uproot
  OutputFormat: root
  OutputDirectory: /Users/kchoi/data_for_MLstudy
  WriteOutputDict: fileset_ml_study

Sample:
  - Name: Signal
    RucioDID: user.kchoi:user.kchoi.fcnc_tHq_ML.ttH.v8,
              user.kchoi:user.kchoi.fcnc_tHq_ML.ttW.v8,
              user.kchoi:user.kchoi.fcnc_tHq_ML.ttZ.v8
    Tree: nominal
    FuncADL: DEF_ttH_nominal_query
  - Name: Background1
    XRootDFiles: DEF_ggH_input
    Tree: mini
    Filter: lep_n>2
    Columns: lep_pt, lep_eta
  - Name: Background2
    Transformer: atlasr21
    RucioDID: DEF_Zee_input
    FuncADL: DEF_Zee_query
  - Name: Background3
    LocalPath: /Users/kchoi/Work/data/fcnc

Definition:
  DEF_ttH_nominal_query: "Where(lambda e: e.met_met>150e3). \
              Select(lambda event: {'el_pt': event.el_pt, 'jet_e': event.jet_e, \
              'jet_pt': event.jet_pt, 'met_met': event.met_met})"
  DEF_ggH_input: "root://eospublic.cern.ch//eos/opendata/atlas/OutreachDatasets\
                  /2020-01-22/4lep/MC/mc_345060.ggH125_ZZ4lep.4lep.root"
  DEF_Zee_input: "mc15_13TeV:mc15_13TeV.361106.PowhegPythia8EvtGen_AZNLOCTEQ6L1_Zee.\
                merge.DAOD_STDM3.e3601_s2576_s2132_r6630_r6264_p2363_tid05630052_00"
  DEF_Zee_query: "SelectMany('lambda e: e.Jets(\"AntiKt4EMTopoJets\")'). \
              Where('lambda j: (j.pt() / 1000) > 30'). \
              Select('lambda j: j.pt() / 1000.0'). \
              AsROOTTTree('junk.root', 'my_tree', [\"JetPt\"])"
```

Figure 2. An example `servicex-databinder` configuration file.

A configuration file is written in YAML format. Figure 2 shows an example configuration file which contains many of the available options of `servicex-databinder`. `General` and `Sample` are mandatory blocks and `Definition` block is optional.

`General` block has two mandatory fields, `ServiceXName` and `OutputFormat`. `ServiceXName` is the ServiceX endpoint name specified in the user's ServiceX access file [6]. `OutputFormat` can be either `root` for ROOT `TTree` format or `parquet` for Apache Parquet format. `Transformer` is the field to specify the default transformer for all samples in the configuration. A user can select either `uproot` or `python` or `atlasr21` depending on the input data format and the query language. `OutputDirectory` is the path to deliver files from ServiceX. `WriteOutputDict` is the name of output yaml file containing output file paths. This file is usually handed over to a subsequent analysis framework such as coffea. A user can also specify a field `Delivery`, not shown in the example configuration, to choose one of three delivery options. `LocalPath` is the default value paired with `OutputDirectory`. or `LocalCache` to download from ServiceX but do not copy to the target directory specified in the `OutputDirectory`. or `ObjectStore` not to download any file from ServiceX immediately but only download S3 URIs for later access or direct streaming.

`Sample` block specifies details of each sample. Each sample requires to have a unique `Name` and can have different combinations of fields depending on the input data format, input data protocol, query language. The first sample in Figure 2, `Signal`, has a list of input datasets as Rucio [7] dataset ID, and each input dataset is separated by comma. Since the default transformer is set to `uproot` in the `General` block, the field `Tree` needs to be provided for the inputs. `FuncADL` [8] is the query language to filter events and select branches from the input datasets. The second sample, `Background1`, has an input file read from CERN EOS space using XRootD protocol. Although the input data protocol of the second sample differs from the first sample, the input data format is identical to the first sample. Thus, the same transformer is used and `Tree` needs to be specified for the input of second sample. The TCut syntax [9] can be also used as a query language with `Filter` and `Columns` fields. `Filter` for TCut syntax cut and `Columns` for the requested branches from the input dataset of given tree. One caveat of TCut as query language is `Filter` works only for scalar type of branches. The third sample, `Background2`, has a field `Transformer` which overwrites the default transformer. `atlas21` transformer is for the input format of ATLAS xAOD. The input files are accessed using Rucio protocol as the first sample but the data format is now ATLAS xAOD. The last sample in the example configuration is `Background3`. This sample does not have any field on input data format or data protocol because it simply binds local files into a sample.

`Definition` block keeps a configuration file short and tidy for a better readability. Every value in the configuration starts with `DEF_` are replaced by the ones defined in the `Definition` block.

2.2 Running `servicex-databinder`

The `servicex-databinder` package is published at PyPI, and it can be easily installed using the following command.

```
1 python -m pip install servicex-databinder
```

Figure 3 shows how to run `servicex-databinder` in a Jupyter notebook. The first cell imports the python package. The second cell loads the configuration file. It performs basic syntax checks of configuration file. It also recognize how many samples defined in the configuration file and how many ServiceX requests will be submitted. The third cell submits

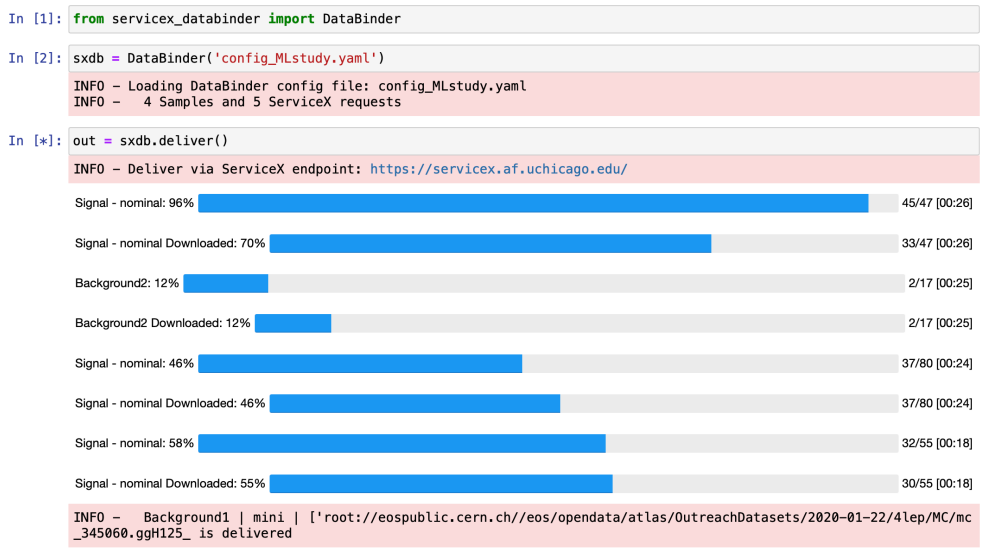

Figure 3. An example of running `servicex-databinder` in a Jupyter notebook.

ServiceX requests and shows two progress bars for each sample; one for transformation happening on the ServiceX and the other one for the status of download. Three ServiceX requests correspond to the three Rucio datasets in the example configuration file shown in Figure 2. All ServiceX transformations and downloads are running in parallel and asynchronously. Each sample is delivered to the final target path as soon as downloaded. Transformation status can be also checked from the web dashboard of ServiceX endpoint shown at the beginning of the third cell. The object `out` in the third cell is the python dictionary containing the file list of each sample. A user can perform subsequent analysis in the Jupyter notebook or can feed that into analysis framework such as coffea.

3 Use-cases in Physics Analysis Workflows

In a case of when a user wants to perform a machine learning study, and needs just few branches from all signal samples and background samples on the grid. A user can retrieve the data with less effort using `servicex-databinder` than by submitting grid jobs. The example configuration in Figure 2 accesses about 300 GB of data on the grid using ServiceX, and delivers few columns to my laptop in about 1 minute.

A more interesting use case is when ServiceX plays data reduction and delivery roles in a full scale physics analysis as shown in Figure 4. An example ATLAS Run-2 physics analysis uses more than 600 Rucio datasets which amount to more than one TBytes. An `servicex-databinder` configuration file defines all samples and preselections applied to each sample. Running `servicex-databinder` for this configuration spawns 140 Uproot transformer pods and automatically scale up/down to extract up to 70 columns from 130 trees out of input datasets. Transformation took about 30 minutes with about 400 transformer pods on average. Total wall time including transformation and download is about 50 minutes to the University of Chicago Analysis Facility. Downloads take longer when outputs are delivered to the local machine at University of Texas at Austin.

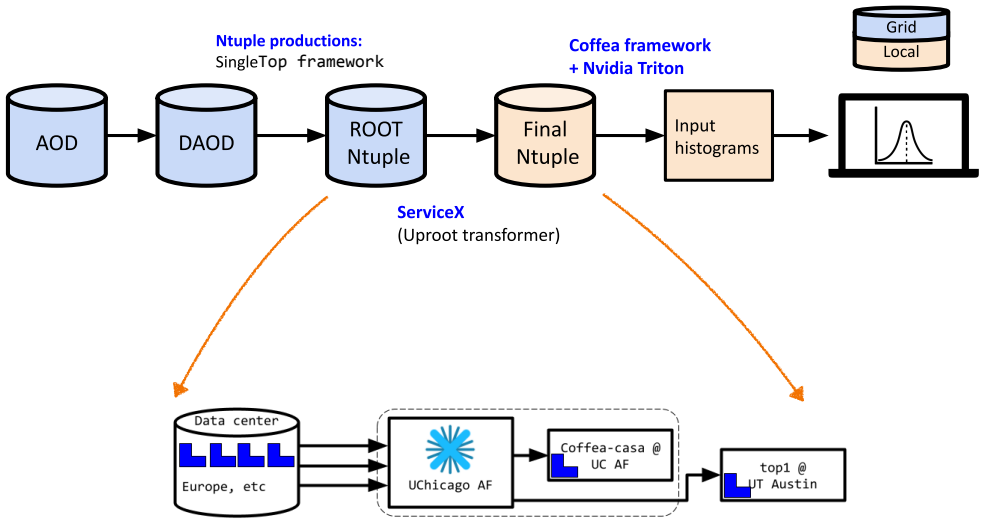

Figure 4. An example use case of ServiceX in a ATLAS Run-2 analysis workflow. Delivered data by ServiceX is processed using coffea framework and machine learning inference is calculated using NVidia Trition. Identical `servicex-databinder` configuration file to deliver data at coffea-casa at the University of Chicago Analysis Facility and local machine at the University of Texas at Austin.

4 Summary and Outlook

`servicex-databinder` provides a straightforward user experience when dealing with multiple and complicated ServiceX requests. A single configuration file which manages all input datasets and preselections and binds them together in one place allows consistent control of an analysis.

The current version of `servicex-databinder` is primarily for the handling of ServiceX requests and datasets. A more generic data management package is envisioned to include a more extensive information of an physics analysis.

References

[1] B. Galewsky, R. Gardner, L. Gray, M. Neubauer, J. Pivarski, M. Proffitt, I. Vukotic, G. Watts, and M. Weinberg, EPJ Web of Conferences 245, 04043 (2020)

[2] K. Choi, A. Eckart, B. Galewsky, R. Gardner, M. Neubauer, P. Onyisi, M. Proffitt, I. Vukotic and G. Watts, EPJ Web of Conferences 251, 02053 (2021)

[3] *IRIS-HEP*, https://iris-hep.org (2023), accessed: 2023-09-20

[4] *ServiceX frontend (version 2.6.2)*, https://github.com/ssl-hep/ServiceX_frontend (2023), accessed: 2023-09-20

[5] *ServiceX DataBinder (version 0.5.0)*, https://github.com/kyungeonchoi/ServiceXDataBinder (2023), accessed: 2023-09-20

[6] *ServiceX documentation*, https://servicex.readthedocs.io/en/latest/user/getting-started/ (2023), accessed: 2023-09-20

[7] *Rucio's documentation*, https://rucio.cern.ch/documentation/ (2023), accessed: 2023-09-20

[8] *Functional ADL*, https://iris-hep.org/projects/func-adl.html (2023), accessed: 2023-09-21
[9] *ROOT TCut Class Reference*, https://root.cern.ch/doc/master/classTCut.html (2023), accessed: 2023-09-21