PHYSITE @ BigQuery
Introduction

- Target physics data (100TB) for our project is columnar data
- our data format (PHYSLITE ROOT) is natively convertible into Parquet (on-disk) and Apache Arrow (in-memory)
Introduction

- Our own HEP infrastructure routinely exploits columnar data structure to do partial read of data

- single 32b column of leading Jet pt from 100 TB dataset: 80GB data

- analyses read only a fraction of branches (most likely statically determinable before runtime)

- HEP Analysis benefits from a data serving mechanism that allows us to selectively read columns of our data
  
  - Data Lake (files) + (ROOT + XRootd | S3 + HTTP + multi-range requests)
  
  - Columnar Data-Warehouse + SQL + Arrow

- not supported by Google can we enable this?

- Google has dedicated Infrastructure for this
**BigQuery**

- Dremel + Data Storage. Google PR markets it for PB-scale datasets
- Can ingest Arrow Data
- allows columnar reads + queries
- no file boundaries

Too restrictive for reco data but could be a possible candidate for PHYSLITE analysis $O(1 \text{ PB})$

- allows to work around some of GCS limitations
Ingestion

- PHYSLITE Schema is complicated: deeply nested (up to 4 levels), 1k columns
- Naive ingestion into BQ fails (schema auto-detection does not work)
- Can manually create schema
  - limitations of BQ:
    Arrow/Parquet support List[List[..]] types - but BQ does not (MET)
    Arrow/Parquet support wider range of fields (periods in ELinks) than BQ
    - no show-stopper but requires intervention
- when manually adjusted PHYSLITE data can be loaded into BQ
  - limitation: only JSON load works for me, ingesting parquet still fails ?
    (help from Googlers appreciated)
Ingestion

## Table schema

<table>
<thead>
<tr>
<th>Field name</th>
<th>Type</th>
<th>Mode</th>
<th>Policy Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>MuonSpectrometerTrackParticles</td>
<td>RECORD</td>
<td>REPEATED</td>
<td></td>
</tr>
<tr>
<td>CombinedMuonTrackParticles</td>
<td>RECORD</td>
<td>REPEATED</td>
<td></td>
</tr>
<tr>
<td>BTagging_AntiK4EMPFLow_201903</td>
<td>RECORD</td>
<td>REPEATED</td>
<td></td>
</tr>
<tr>
<td>MET_Core_MET</td>
<td>RECORD</td>
<td>REPEATED</td>
<td></td>
</tr>
<tr>
<td>METAAssoc_MET</td>
<td>RECORD</td>
<td>REPEATED</td>
<td></td>
</tr>
<tr>
<td>AntiK10UFoCSVSSJets</td>
<td>RECORD</td>
<td>REPEATED</td>
<td></td>
</tr>
<tr>
<td>xTrigDecision</td>
<td>RECORD</td>
<td>REQUIRED</td>
<td></td>
</tr>
<tr>
<td>PrimaryVertices</td>
<td>RECORD</td>
<td>REPEATED</td>
<td></td>
</tr>
<tr>
<td>Photons</td>
<td>RECORD</td>
<td>REPEATED</td>
<td></td>
</tr>
<tr>
<td>Electrons</td>
<td>RECORD</td>
<td>REPEATED</td>
<td></td>
</tr>
<tr>
<td>ExtrapolatedMuonTrackParticles</td>
<td>RECORD</td>
<td>REPEATED</td>
<td></td>
</tr>
<tr>
<td>InGetTrackParticles</td>
<td>RECORD</td>
<td>REPEATED</td>
<td></td>
</tr>
<tr>
<td>EventInfo</td>
<td>RECORD</td>
<td>REPEATED</td>
<td></td>
</tr>
<tr>
<td>TauJets</td>
<td>RECORD</td>
<td>REPEATED</td>
<td></td>
</tr>
<tr>
<td>CaloCalTopoClusters</td>
<td>RECORD</td>
<td>REPEATED</td>
<td></td>
</tr>
<tr>
<td>egammaClusters</td>
<td>RECORD</td>
<td>REPEATED</td>
<td></td>
</tr>
<tr>
<td>Muons</td>
<td>RECORD</td>
<td>REPEATED</td>
<td></td>
</tr>
</tbody>
</table>

## Expanded Schema

[Image of Expanded Schema]

## Top Level Schema

[Image of Top Level Schema]
Ingestion

Somehow the same 20k events incur a 10x increase in data size?

- how does this affect storage costs?
Once the data is in, we get something nice

- can directly instantiate awkward-array for actual user analysis from data served directly from BQ

- in principle this could be the full PHYSLITE dataset (except we don't have local machines w/ O(10 TB) RAM)

```python
[76]: import awkward as ak
    from google.cloud import bigquery
    ak.from_arrow(bigquery.Client().query(f'''
            SELECT Jets
            FROM personalstuff-309012.testatlas.testit
            ''').to_arrow())

[76]: <Array [{Jets: [{pt: 6.88e+04, ... }] type='20012 * {"Jets": var * ?{"pt": ?float...'}}>
Possible Analysis Model

Rucio S3 → BQ → Query Processing (presel) → User

- Ingest Parquet
- all columns all rows
- distributed across N streams
- some columns some rows
- user reduce operations
- visualization

Data Lake

Big Query

Visualization
Data Throughput

We can read the full dataset (all columns) in three ways. Test on 20k events

**BQ SQL**

- In principle can select columns (nested fields maybe, but expensive JOINs)
- Parallel disjoint reads NOT possible
- Pre-selection possible (WHERE clause)

```python
def read_with_bq_query():
    from google.cloud import bigquery
    client = bigquery.Client()
    r = client.query(""
        SELECT *
        FROM personaltstuff-300812.tests.tables
    ""
    ).result()
    start = time.time()
    ra = r.to_array()
    data = ak.fromArrow(ra)
    return data
```

**BQ Storage API**

- In principle can select columns (but no control over nested fields)
- Parallel disjoint reads possible (streams)
- Pre-selection possible (WHERE clause, how complex?)

```python
def read_with_bq_storage():
    from google.cloud import bigquery_storage
    client = bigquery_storage_client()
    project_id = "personaltstuff-300812"
    table = "projects/{}/datasets/{}/tables/".format(
        project_id, "testatlas", "test")
    requested_session = types.ReadSession()
    requested_session.table = table
    requested_session.data_format = types.DataFormat.AROW
    client = BigQueryReadClient()
    parent = "projects/{}/locations/".format(project_id)
    session = client.create_read_session(
        parent=parent,
        read_session=requested_session,
        max_stream_count=1,
    )
    session.read_options.selected_fields = ['Jets']
    reader = client.read_rows(session.streams[0].name)
    data = ak.fromArrow(reader.to_array(session))
    return data
```

**GCS S3 APIs**

- Forced to download all columns

```python
def read_with_s3():
    from google.cloud import storage
    cl = storage.Client()
    bk = cl.bucket('testparquet')
    b1 = bk.get_blob('data_20k.pq')
    b1.download_to_filename('ts.pq')
    data = ak.from_parquet('ts.pq')
    return data
```

Not great throughput but maybe too small dataset? Overhead / request?

- 6s / 20k events
- 28s / 20k events
- 3s / 20k events
Data Throughput

We can read the full dataset (all columns) in three ways. Test on 20k events

### BQ SQL

- In principle can select columns
- Parallel disjoint reads possible (streams)
- Pre-selection possible (WHERE clause)

```python
def read_with_bq_query():
    from google.cloud import bigquery
    client = bigquery.Client()
    query = 'SELECT * FROM testtable'
    query_job = client.query(query)
    results = query_job.result()
    return results
```

### BQ Storage API

- In principle can select columns (but no control over nested fields)
- Parallel disjoint reads possible (streams)
- Pre-selection possible (WHERE clause, how complex?)

```python
def read_with_bq_storage():
    from google.cloud.bigquery_storage import types
    client = bigquery_storage.Client(project='testproject', location='us')
    request = bigquery_storage.agents.BqStorageAgent()
    request.search(project='testproject', dataset='testdataset', table='testtable',
                    fields=['field1', 'field2'], mode='streaming')
    results = request.execute()
    return results
```

### GCS S3 APIs

- Forced to download all columns
- Parallel disjoint reads NOT possible
- Pre-selection possible (WHERE clause)

```python
with open('testfile.csv', 'rb') as f:
    data = f.read()
```

### Performance

- 28s / 20k events
- 6s / 20k events
- 3s / 20k events

Not great throughput but maybe too small dataset? Overhead / request?
BQ -> Dask on k8s

With BQ Storage API we can distribute events across N streams (limited to 1k parallel streams, could match well to Dask sweet spot of 1k workers)
Summary

• BigQuery seems like an interesting choice (no file handling, uniform batches, pre-selection at the infrastructure layer, columnar reading)

• but hard to judge performance for small-scale tests (on my personal account)

• Suggestion: 1% data set conversion to Parquet and ingestion to BQ?
  • ok $$\text{\$\$\$-wise?}$$

• What throughput can we expect from BQ? (internally BQ claims Pbps, but unclear what BQ -> K8s throughput is, so far we see 1Gbps (1000 workers Tbit?))