What can PyArrow do for you

Array interchange, storage, compute and transport

Rok Mihevc, Alenka Frim

Apache Arrow committers

February 2, 2025

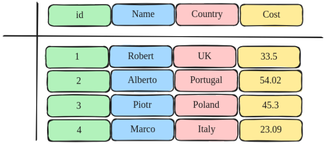
Overview

1. Introduction
2. Array Interchange
3. Storage
4. Compute
5. Transport

What is Arrow

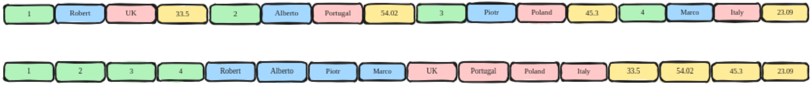
Columnar memory layout

- Tables in memory can be stored as rows or columns
- Columns are better suited for analytical workloads due to cache locality, memory prefetching, SIMD vectorizations and cheap schema manipulation



id	Name	Country	Cost
1	Robert	UK	33.5
2	Alberto	Portugal	54.02
3	Piotr	Poland	45.3
4	Marco	Italy	23.09

Figure: Table composed of arrays



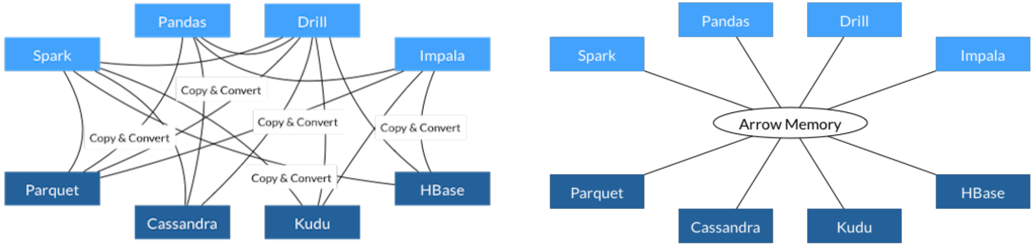
1	Robert	UK	33.5	2	Alberto	Portugal	54.02	3	Piotr	Poland	45.3	4	Marco	Italy	23.09
1	2	3	4	Robert	Alberto	Piotr	Marco	UK	Portugal	Poland	Italy	33.5	54.02	45.3	23.09

Figure: Row major layout (top) vs column major layout (bottom)

What is Arrow

Arrow

- Set of implementations in multiple languages with added components to enable efficient storage, processing and movement of data



(a) Without common memory layout (n^2 connectors)

(b) With Arrow memory layout

What is PyArrow

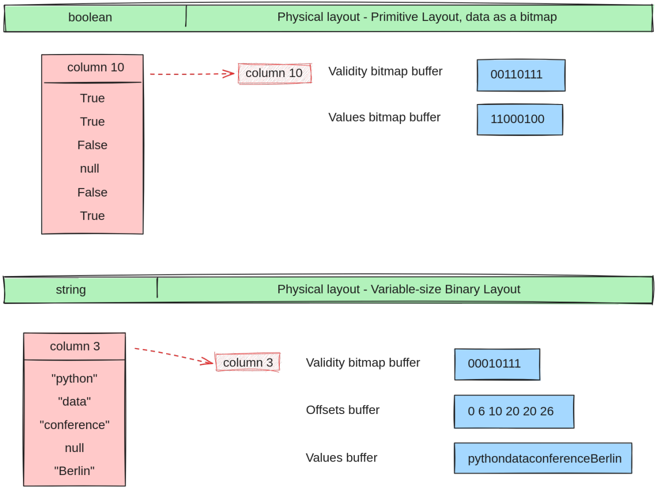
PyArrow

- Python bindings for Arrow C++ implementations
- First-class integration with NumPy, pandas, and built-in Python objects
- Include leaf libraries that add additional functionality such as reading Apache Parquet files into Arrow structures

PyArrow Array

- Central data structure in Arrow
- A contiguous, one-dimensional sequence of values with known length
- All values have the same type

PyArrow Array memory layout



boolean

Physical layout - Primitive Layout, data as a bitmap

column 10

True

True

False

null

False

True

column 10

Validity bitmap buffer

00110111

Values bitmap buffer

11000100

string

Physical layout - Variable-size Binary Layout

column 3

"python"

"data"

"conference"

null

"Berlin"

column 3

Validity bitmap buffer

00010111

Offsets buffer

0 6 10 20 20 26

Values buffer

pythondataconferenceBerlin

https://arrow.apache.org/docs/format/Intro.html

From NumPy to PyArrow

Examples

```
>>> import numpy as np
>>> numpy_array = np.arange(5)
>>> numpy_array
array([0, 1, 2, 3, 4])
>>> import pyarrow as pa
>>> pa.array(numpy_array)
<pyarrow.lib.Int64Array object at ...>
[
  0,
  1,
  2,
  3,
  4
]
```

From NumPy to PyArrow

Pointing to the same data → zero-copy!

Zero-copy

```
>>> numpy_array.__array_interface__['data'][0]
5568689760

>>> pa.array(numpy_array).buffers()[1].address
5568689760
```

From PyArrow to Pandas

Examples

```
>>> pyarrow_array = pa.array(numpy_array)
>>> pyarrow_array.to_pandas()
0    0
1    1
2    2
3    3
4    4
dtype: int64

>>> pyarrow_array.to_pandas().to_numpy().__array_interface__["data"][0]
5568689760
```

From Pyarrow to Polars

Examples

```
>>> import polars as pl
>>> pl.from_arrow(pyarrow_array)
shape: (5,)
Series: '' [i64]
[
        0
        1
        2
        3
        4
]

>>> pl.from_arrow(pyarrow_array)._get_buffer_info()[0]
5568689760
```

From Pyarrow to DataFusion and DuckDB

Examples

```
>>> pyarrow_table = pa.table({"arr": pyarrow_array})

>>> from datafusion import SessionContext
>>> ctx = SessionContext()
>>> ctx.from_arrow(pyarrow_table)

>>> import duckdb
>>> duckdb.query("SELECT * FROM pyarrow_table")
```

.. and nanoarrow, narwhals, arro3, ibis, quak, ...

From PyArrow to PyTorch via DLPack

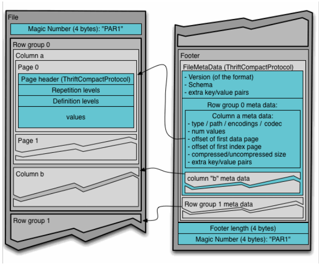
Examples

```
>>> import torch
>>> torch.from_dlpack(pyarrow_array)
tensor([0, 1, 2, 3, 4])
>>> torch.from_dlpack(pyarrow_array).data_ptr()
5568689760
```

Parquet

Parquet file format

- Column-oriented file format designed for efficient data storage and retrieval
- Provides efficient data compression and encoding schemes with enhanced performance to handle complex data in bulk
- Optional encryption
- Use of metadata allows for reading subsets of columns or even rows


File
Magic Number (4 bytes): "PAR1"
Row group 0
Column a
Page 0
Page header (ThriftCompactProtocol)
Repetition levels
Definition levels
values
Page 1
Column b
Row group 1
Footer
FileMetaData (ThriftCompactProtocol)
- Version (of the format)
- Schema
- extra key/value pairs
Row group 0 meta data:
Column a meta data:
- type / path / encodings / codec
- num values
- offset of first data page
- offset of first index page
- compressed/uncompressed size
- extra key/value pairs
column "b" meta data
Row group 1 meta data
Footer length (4 bytes)
Magic Number (4 bytes): "PAR1"

Read and write images to Parquet

Examples

```
>>> import pyarrow as pa
>>> import pyarrow.parquet as pq
>>> from PIL import Image
>>> from urllib.request import urlopen
>>> urls = [f"https://picsum.photos/seed/{i}/1/1" for i in range(3)]
>>> images = np.stack([Image.open(urlopen(url)) for url in urls])
>>> image_array = pa.FixedShapeTensorArray.from_numpy_ndarray(images)
>>> table = pa.table({"idx": range(len(urls)), "url": urls,
>>>                   "image": image_array})
>>> pq.write_table(table, "example.parquet", compression=None)
>>> read_table = pq.read_table("example.parquet")
>>> read_table == table
True
```

Read subset of rows

Examples

```
>>> import pyarrow.parquet as pq
>>> pq.read_table("example.parquet",
>>>               columns=["image"],
>>>               filters=[
>>>                   ("idx", ">", 1),
>>>                   ("idx", "<", 3),
>>>               ])
pyarrow.Table
image: extension<arrow.fixed_shape_tensor[value_type=uint8,
shape=[1,1,3], permutation=[0,1,2]]>
----
image: [[[93,93,91]]]
```

Write and read from cloud object storage (S3, GCS, HDFS, ..)

Examples

```
>>> # This requires ~/.aws/credentials file for authentication
>>> import pyarrow as pa
>>> import pyarrow.parquet as pq
>>> table = pa.table({"idx": [0, 1, 3], "letter": ["a", "b", "c"]})
>>> s3_uri = "s3://<bucket>/<filepath>"
>>> pq.write_table(table, s3_uri)
>>> read_table = pq.read_table(s3_uri)
>>> print(read_table.to_pandas().iloc[:3, 0:3].to_markdown())
|    |   idx | letter    |
|---:|------:|:----------|
|  0 |     0 | a         |
|  1 |     1 | b         |
|  2 |     3 | c         |
```

Write encrypted

Examples

```python
import pyarrow.parquet.encryption as pe
import pyarrow.parquet as pq
import pyarrow as pa
from pyarrow.tests.parquet.encryption import InMemoryKmsClient

def kms_factory(kms_connection_configuration):
    return InMemoryKmsClient(kms_connection_configuration)
FOOTER_KEY = b"0123456789112345".decode("UTF-8")
FOOTER_KEY_NAME = "footer_key"
COL_KEY = b"1234567890123450".decode("UTF-8")
COL_KEY_NAME = "image_key"
kms_connection_config = pe.KmsConnectionConfig(
    custom_kms_conf = {
        FOOTER_KEY_NAME: FOOTER_KEY,
        COL_KEY_NAME: COL_KEY,
    }
)
encryption_config = pe.EncryptionConfiguration(
    footer_key=FOOTER_KEY_NAME,
    column_keys={COL_KEY_NAME: ["idx"]},
    encryption_algorithm="AES_GCM_V1",
)
crypto_factory = pe.CryptoFactory(kms_factory)
encryption_properties = crypto_factory.file_encryption_properties(kms_connection_config, encryption_config)

table = pa.table({"idx": [0, 1, 3], "letter": ["a", "b", "c"]})
with pq.ParquetWriter("encrypted.parquet", table.schema, encryption_properties=encryption_properties) as writer:
    writer.write_table(table)
```

Read encrypted

Examples

```python
import pyarrow.parquet.encryption as pe
import pyarrow.parquet as pq
from pyarrow.tests.parquet.encryption import InMemoryKmsClient

def kms_factory(kms_connection_configuration):
    return InMemoryKmsClient(kms_connection_configuration)
FOOTER_KEY = b"0123456789112345".decode("UTF-8")
FOOTER_KEY_NAME = "footer_key"
COL_KEY = b"1234567890123450".decode("UTF-8")
COL_KEY_NAME = "image_key"
kms_connection_config = pe.KmsConnectionConfig(
    custom_kms_conf = {
        FOOTER_KEY_NAME: FOOTER_KEY,
        COL_KEY_NAME: COL_KEY,
    }
)
crypto_factory = pe.CryptoFactory(kms_factory)
decryption_properties = crypto_factory.file_decryption_properties(kms_connection_config)

result = pq.ParquetFile("encrypted.parquet", decryption_properties=decryption_properties)
result_table = result.read()
```

Compute

Compute operations like filtering or transforming data in arrays and tables are provided by the `pa.compute` module.

- Standard Compute Functions
- Grouped Aggregations
- Table and Dataset Joins
- Filtering by Expressions

Last three classes of operations are supported on Tables and/or Datasets. Standard compute functions also support Arrow arrays.

Compute functions reference

arrow.apache.org/docs/python/api/compute.html

Example 1

Computing `Sum` and `Mean/Min/Max` values of an array

Examples

```
>>> import pyarrow as pa
>>> import pyarrow.compute as pc
>>> a = pa.array([1, 1, 2, 3])
>>> pc.sum(a)
<pyarrow.Int64Scalar: 7>
>>> pc.min(a)
<pyarrow.Int64Scalar: 1>
```

Example 2

Data transformations with `pa.equal`

Examples

```
>>> import pyarrow as pa
>>> import pyarrow.compute as pc
>>> a = pa.array([1, 1, 2, 3])
>>> b = pa.array([4, 1, 2, 8])
>>> pc.equal(a, b)
<pyarrow.lib.BooleanArray object at 0x112a721a0>
[
  false,
  true,
  true,
  false
]
```

Example 2

Data transformations with `pa.multiply` array ans scalar

Examples

```
>>> y = pa.scalar(9.3)
>>> pc.multiply(a, y)
<pyarrow.lib.DoubleArray object at 0x112a721a0>
[
  9.3,
  9.3,
  18.6,
  27.900000000000002
]
```

Example 2

Data transformations with `pa.multiply` two arrays

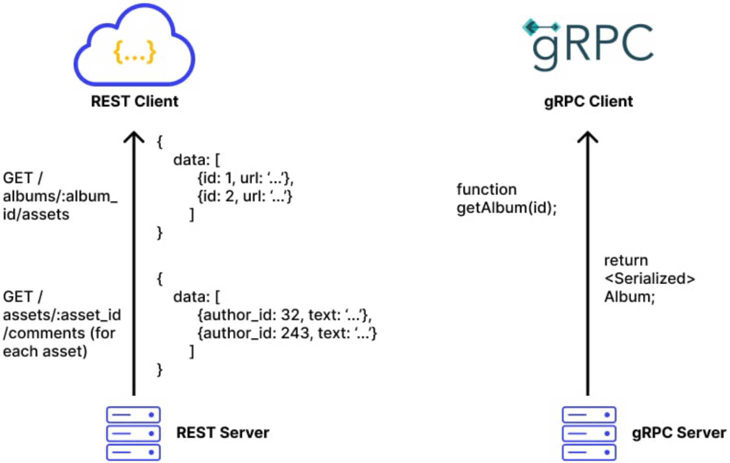
Examples

```
>>> pc.multiply(a, b)
<pyarrow.lib.Int64Array object at 0x114af31c0>
[
  4,
  1,
  4,
  24
]
```

Flight RPC

- A RPC framework for high-performance data services based on the Arrow columnar format
- Defines a generic RPC vocabulary for building custom applications and services
- Data transfer is streaming, both client-side and server-side
- Example: Arrow-native storage servers, execution engines. . .
- Arrow Flight SQL enables sending requests as SQL

Flight RPC



REST Client

GET / albums/:album_id/assets

```
{
   data: [
      {id: 1, url: '...'},
      {id: 2, url: '...'}
   ]
}
```

GET / assets/:asset_id /comments (for each asset)

```
{
   data: [
      {author_id: 32, text: '...'},
      {author_id: 243, text: '...'}
   ]
}
```

REST Server

gRPC

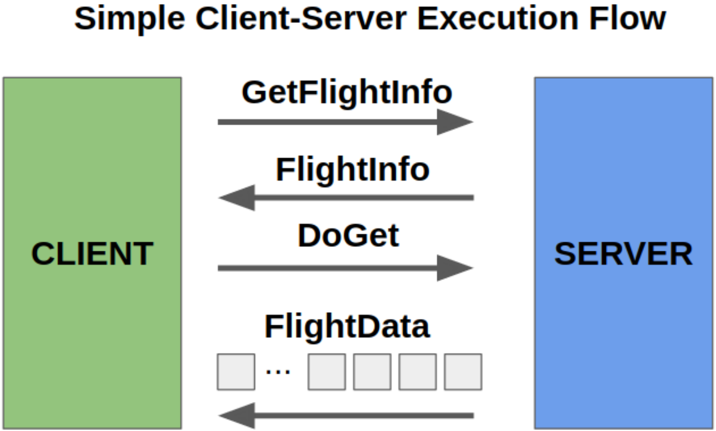
gRPC Client

function getAlbum(id);

return <Serialized> Album;

gRPC Server

Flight RPC


Simple Client-Server Execution Flow
CLIENT
GetFlightInfo
FlightInfo
DoGet
FlightData
...
SERVER

For more ideas see cookbook

https://arrow.apache.org/cookbook/py

PRs welcome!

The End