HyperArrow

Will Ayd

Mar 17, 2022

CONTENTS

CHAPTER

ONE

PYTHON

1.1 Quick Install

```
python -m pip install hyperarrow
```

You may also want to test your installation with pytest

```
python -m pip install pytest
python -c "import hyperarrow as hal; hal.test()"
```

1.2 Usage Example

1.2.1 Writing to a Hyper file

```
import pyarrow as pa
import hyperarrow as hal

schema = pa.schema(
    [
        ("int16", pa.int16()),
        ("int32", pa.int32()),
        ("int64", pa.int64()),
        ("float64", pa.float64()),
        ("boolean", pa.bool_()),
        ("string", pa.utf8()),
        ("date", pa.date32()),
        ("datetime", pa.timestamp("us")),
    ]
)

tbl = pa.Table.from_pydict(
    {
        "int16": range(3),
        "int32": range(3),
        "int64": range(3),
        "float64": [1.0, 2.0, 3.0],
        "boolean": [True, False, True],
        "string": list("abc"),
```

(continues on next page)

(continued from previous page)

```
        "date": [
            datetime.date(1900, 8, 15),
            datetime.date(1970, 1, 2),
            datetime.date(2021, 12, 23),
        ],
        "datetime": [
            datetime.datetime(1900, 8, 15, 12, 36, 0),
            datetime.datetime(1970, 1, 2, 22, 42, 17, 0),
            datetime.datetime(2021, 12, 23, 14, 32, 59, 0),
        ],
    },
    schema=schema,
)

hal.write_to_hyper(tbl, str(tmp_hyper), "schema", "table")
```

1.2.2 Reading from Hyper

```
import hyperarrow as hal

hal.read_from_hyper(str(tmp_hyper), "schema", "table")
```

1.3 Comparison to pantab

The TLDR of HyperArrow's advantage over pantab pantab is that it provides more native type resolution, while also future-proofing and using an Arrow library that is expected to grow in usage and importance in the data space.

1.3.1 Data Types

Pandas was built using the NumPy type system. While this has served open source analytics communities well, it has some shortcomings that are worth noting.

For starters, integer types are not nullable in the NumPy type system. In the presence of `NULL`, arrays are coered to a float type. Due to the nature of floating point arithmetic, this can lead to imprecise calculations after coercion. This also has undesirable effects when working with smaller types (say `uint8_t`) that occupy a much smaller address space than `double_t` types in memory.

Pandas attempts to work around this with its own extended type system. *Nullable types* are often represented in pandas with capital letters (ex: Int64 is a nullable integer, int64 is not). While this may prevent issues with floating point inaccuracies, it introduces memory overhead, design complexity, and to date is not as natively optimized as standard NumPy arrays.

By contrast, Arrow has a simplified type system. Primitive types are all supported natively with or without nullability. Fortunately enough, this type system is also closer to what Hyper expects, requiring less ambigous rules for converting between the two formats.

1.3.2 Leveraging the Arrow Format

pantab is built off of pandas, which has its own internal memory representation that isn't widely compatible with other processes or langauges. Arrow as a project solves this amongst other things. You can read more about what Arrow does in the Apache Arrow Documentation.

CHAPTER

TWO

C++ CORE

The core language for HyperArrow is C++, matching what is offered upstream by the Arrow library.

2.1 Build Dependencies

HyperArrow relies on the follow components:

- A c++ compiler
- Apache Arrow
- CMake as a build system
- Boost (for testing)
- The Tableau Hyper API

Refer to your OS on how to install the components, or alternately use conda to create a virtual environment.

```
conda create -n hyperarrow-dev
conda activate hyperarrow-dev
conda install -c conda-forge "arrow-cpp>=6.0" boost-cpp compilers
```

Due to the nature of how the Tableau Hyper API is distributed, you will need to download that directly from the Tableau website and place somwhere on your computer where CMake can find it. If CMake is unable to resolve the location, you should set an environemnt variable for `TABLEAU_CMAKE_PATH` to point to the CMake file provided by Tableau. On Unix-based systems this may look something like:

```
export TABLEAU_CMAKE_PATH=<SOME_DIRECTORY_PATH>/tableauhyperapi/share/cmake
```

2.2 How to Build

We suggest doing an out-of-core compliation of the libraries. Assuming you are at the root of the HyperArrow project you can perform the following steps to accomplish this.

```
mkdir build
pushd build

cmake -S ..
cmake --build .
```

(continues on next page)

(continued from previous page)

```
ctest
popd
```

CHAPTER THREE

CHANGELOG

3.1 January 29, 2022

- Initial development release of Python bindings to PyPi

HyperArrow is a collection of libraries in different langauges that gives users an easy way to convert Arrow tables to Tableau's Hyper format. The core library is written in C++.

Currently support languages are C++ and Python. Contributions to add any other language would be very welcome. Please refer to each language's section on installation and usage notes.

CHAPTER

FOUR

WHY HYPERARROW?

HyperArrow looks to build off of the advantages of the Apache Arrow library. While the current implementation is limited in scope to just a C++ and Python library dealing with singular arrow `Table` objects, in the future we envision:

- Reading / Writing Hyper files from a large variety of langauges
- Serializing and Deserializing data that is larger than memory (i.e. “out-of-core”)
- Support streaming data into Hyper files