Understanding Python’s pandas source code

D. Gueorguiev 5/26/21

<https://github.com/pandas-dev/pandas>

Contents

[Preliminaries and third-party packages 1](#_Toc73216351)

[Apache Arrow 1](#_Toc73216352)

[Google Flatbuffers 1](#_Toc73216353)

[Numpy 2](#_Toc73216354)

# Preliminaries and third-party packages

## Apache Arrow

<https://github.com/apache/arrow>

<https://arrow.apache.org/docs/format/Columnar.html>

From <https://arrow.apache.org/overview/> :

Apache Arrow defines language-independent columnar memory format for flat and hierarchical data, organized for efficientanalytic operations on modern hardware like CPUs and GPUs. The Arrow memory format supports zero copy reads for fast data access without serialization overhead.

Apache Arrow is software development platform for building high-performance applications that process and transport large data sets. It is designed to both improve the performance of analytical algorithms and the efficiency of moving data from one system or one programming language to another. The defining component of Arrow is its in-memory columnar format which is a standardized, language-agnostic specification for representing structured, table-like datasets in memory. This format has rich data type system including nested and user-defined data types designed to support the needs for analytic database systems, data frame libraries, etc.

The Apache Arrow format allows computational routines and execution engines to maximize their efficiency when scanning and iterating large chunks of data. In particular, the contiguous columnar layout enables vectorization using the latest SIMD operations included in modern processors.

Assuming standard format for the data speeds up execution

Without standard format there are potentially a lot of conversion and transforming of data which is lots of unnecessary operations. Moving data from one system to another involves costly serialization and deserialization. In addition, common algorithms must be transformed / rewritten for each data format.

Arrow’s in-memory columnar format provides a solution to these kinds of problems allowing data transfers between between disparate systems to be achieved with very low cost. Additionally, the standardized format allows for reuse of libraries and algorithms.

The Arrow Columnar Format

The Arrow Columnar Format includes a language-agnostic in-memory data structure specification, metadata serialization, and a protocol for serialization and generic data transport. The new columnar format is created without the aid of existing implementation. For the implementation of the columnar format [Flatbuffers](https://github.com/google/flatbuffers) is used for metadata serialization purposes.

## Google Flatbuffers

<https://google.github.io/flatbuffers/>

<https://github.com/google/flatbuffers>

Flatbuffers is efficient serialization library supporting various languages.

Benefits of using Flatbuffers:

* Access to serialized data without parsing/unpacking – what sets Flatbuffers apart is that it represents hierarchical data in a flat binary buffer in such way that it still can be accessed without parsing / unpacking while also still supporting data structure evolution (forward/backward compatibility)
* Memory efficiency and speed- the only memory needed to access your data. It requires 0 additional allocations in C++. Flatbuffers is also very suitable for use of mmap (or streaming) requiring only a part of the buffer to be in memory. Access is close to the speed of raw struct access with only one extra indirection (a kind of vtable) to allow for format evolution and optional fields. It is aimed at projects where spending time and space (many memory allocations) to be able to access or construct serialized data is undesirable such as in games or any other performance sensitive applications
* Flexible – Optional fields means not only do you get forwards and backwards compatibility (do not have to update all data with each new version). It also means you have a lot of choice in what data you write and what data you don’t and how you design data structures.
* Tiny code footprint – small amounts of generated code and just a single small header as the minimum dependency which is very easy to integrate.
* Strongly typed – Errors happen at compile time rather than manually having to write repetitive and error prone run-time checks. Useful code can be generated for you.
* Convenient to use – generated C++ code allows for terse access and construction code.
* Cross platform code with no dependencies

How to use Flatbuffers:

* Write a schema file that allows you to define the data structures you may want to serialize. Fields can have scalar types (int / floats of all sizes) or they can be : string, array of any type; reference to yet another object; or a set of possible objects (unions). Fields are optional and have defaults so they do not need to be present for every object instance.
* Use flatc (the FlatBuffer compiler) to generate C++ header (or Java/Kotlin/C#/Go/Python classes) with helper classes to access and construct the serialized data. This header (say mydata\_generated.h) only depends on flatbuffers.h which defines the core functionality.

## Numpy

<https://github.com/numpy/numpy>

<https://numpy.org/doc/stable/user/>