TWO SIGMA

www.twosigma.com

Memory Interoperability for Analytics and Machine Learning

Wes McKinney @wesmckinn

ScaledML @ Stanford

March 25, 2017

Me

- Currently: Software Architect at Two Sigma Investments
- Creator of Python pandas project
- PMC member for Apache Arrow and Apache Parquet
- Author of *Python for Data Analysis*
- Other Python projects: Ibis, Feather, statsmodels

Important Legal Information

The information presented here is offered for informational purposes only and should not be used for any other purpose (including, without limitation, the making of investment decisions). Examples provided herein are for illustrative purposes only and are not necessarily based on actual data. Nothing herein constitutes: an offer to sell or the solicitation of any offer to buy any security or other interest; tax advice; or investment advice. This presentation shall remain the property of Two Sigma Investments, LP ("Two Sigma") and Two Sigma reserves the right to require the return of this presentation at any time.

Some of the images, logos or other material used herein may be protected by copyright and/or trademark. If so, such copyrights and/or trademarks are most likely owned by the entity that created the material and are used purely for identification and comment as fair use under international copyright and/or trademark laws. Use of such image, copyright or trademark does not imply any association with such organization (or endorsement of such organization) by Two Sigma, nor vice versa.

This talk

- Benefits of interoperable data and metadata
- Challenges to sharing memory between runtime environments
- Apache Arrow: Purpose and C++ architecture
- Opportunities for collaboration
- Example application: pandas 2.0

Changing hardware landscape

- Intel has released first production 3D Xpoint SSD
  - Reported 1000x faster than NAND, less expensive than RAM

- Convergence between RAM vs. shared memory / mmap performance

Changing software landscape

- Next-gen ML / AI frameworks (TensorFlow, Torch, etc.)
- DIY open source architectures for machine learning in production
  - Streaming / batch data processing pipelines
  - Data cleaning and feature engineering
  - Model fitting / scoring / serving

“Zero-copy” memory interfaces

- Enables computational tools to process a dataset **without any additional serialization**, or transfer to a different memory space
- Can do random access on a dataset that does not fit in RAM

- Another interpretation: reading a dataset is a **metadata-only conversion**

Challenges to zero-copy memory sharing

- Cross-language issues
  - Type metadata + logical types
  - Byte/bit-level memory layout

- Language-specific issues
  - In-memory data structures
  - Memory allocation and sharing constructs

What is pandas?

- Popular in-memory data manipulation tool for Python
  - Focused on tabular datasets (“data frames”)
- Sprawling codebase spanning multiple areas
  - IO for many data formats
  - Array manipulations / data preparation
  - OLAP-style analytics
- Internals implemented using NumPy array objects

NumPy

- Tensor memory model ("ndarray") for numeric data
  - Strided, homogeneously-typed, byte-addressable memory
  - APL-inspired semantics
  - Zero-copy construction from compatible memory layouts

- Computational tools support both strided and contiguous memory access

pandas: Technical debt + Architectural issues

- Tensor library like NumPy awkward fit for pandas use cases
  - Multidimensionality + strided memory access complicated algorithms
  - Lack of built-in missing value support
  - Weak on native string, variable length, or nested types

- pandas at core a “in-memory columnar” problem, similar to analytical SQL engines

Thesis: Tensors and Tables

- 2 data structures best suited for zero-copy sharing
  - **Tensors**: N-dimensional, homogeneously-typed arrays
  - **Tables**: Column-oriented, heterogeneously typed

- These data structures can be defined using common memory and metadata primitives

Observations

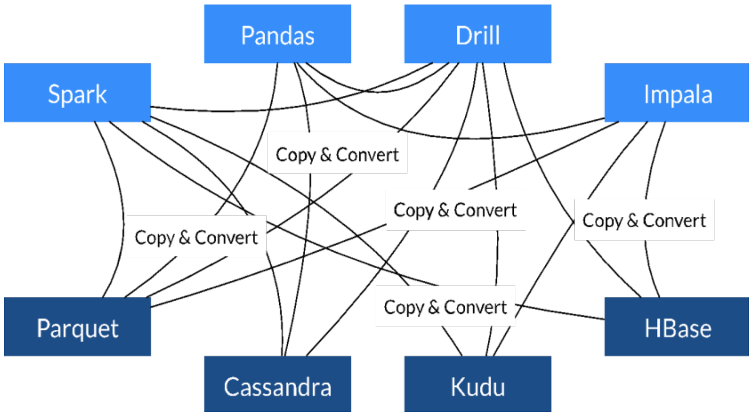
- A Tensor is semantically a multidimensional view of a 1D block of memory
- Writing computational code targeting arbitrary tensors is much more difficult than 1D contiguous arrays
- Tensors of non-fixed size types (e.g. strings) occur less frequently

Apache Arrow

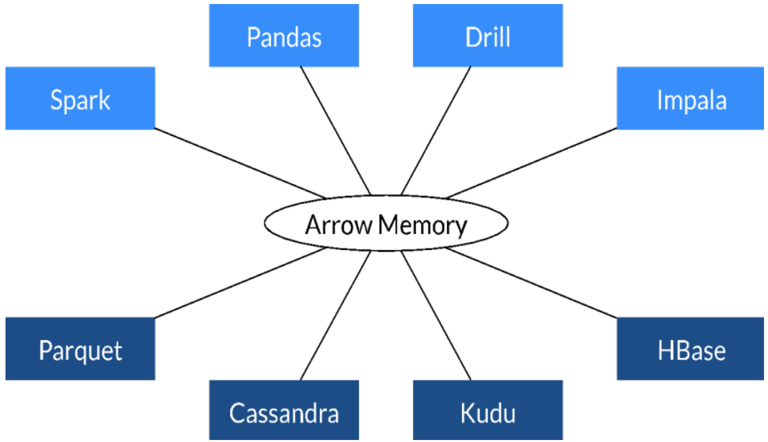
- github.com/apache/arrow
- Collaboration amongst broad set of OSS projects around language-agnostic shared data structures
- Initial focus
  - In-memory columnar tables
  - Canonical metadata
  - Interoperability between JVM and native code (C/C++) ecosystem

High performance data interchange

Today



With Arrow



Source: Apache Arrow

What does Apache Arrow give you?

- **Cache-efficient columnar memory:** optimized for CPU affinity and SIMD / parallel processing, O(1) random value access
- **Zero-copy messaging / IPC:** Language-agnostic metadata, batch/file-based and streaming binary formats
- **Complex schema support**: Flat and nested data types

- **Main implementations in C++ and Java**: with integration tests
  - Bindings / implementations for C, Python, Ruby, Javascript in various stages of development

Arrow in C++

- Reusable memory management and IO subsystem for native code applications
- Layered in multiple components
  - Memory management
  - Type metadata / schemas
  - Array / Table containers
  - IO interfaces
  - Zero-copy IPC / messaging

Arrow C++: Memory management

- `arrow::Buffer`
  - RAII-based memory lifetime with `std::shared_ptr<Buffer>`
  - `arrow::MemoryMappedBuffer:` for memory maps
- `arrow::MemoryPool`
  - Abstract memory allocator for tracking all allocations

Arrow C++: Type metadata

- `arrow::DataType`
  - Base class for fixed size, variable size, and nested datatypes
- `arrow::Field`
  - Type + name + additional metadata
- `arrow::Schema`
  - Collection of fields

Arrow C++: Array / Table containers

- `arrow::Array`
  - 1-dimensional columnar arrays: `Int32Array, ListArray, StructArray,` etc.
  - Support for dictionary-encoded arrays
- `arrow::RecordBatch`
  - Collection of equal-length arrays
- `arrow::Column`
  - Logical table “column” as chunked array
- `arrow::Table`
  - Collection of columns

Arrow C++: IO interfaces

- `arrow::{InputStream, OutputStream}`
- `arrow::RandomAccessFile`
  - Abstract file interface
- `arrow::MemoryMappedFile`
  - Zero-copy reads to `arrow::Buffer`
- Specific implementations for OS files, HDFS, etc.

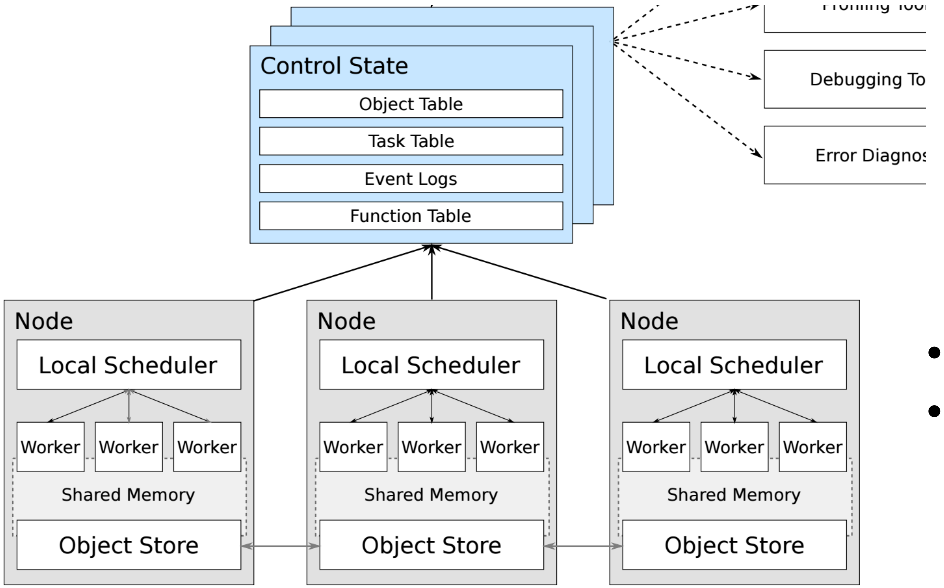
Arrow C++: Messaging / IPC

- Metadata read/write using Google's Flatbuffers library
- Encapsulated Message type
  - Write record batches, read with zero-copy
- `arrow::{FileWriter, FileReader}`
  - Random access / "batch" binary format
- `arrow::{StreamWriter, StreamReader}`
  - Streaming binary format

In development: `arrow::Tensor`

- Targeting interoperability with memory layouts as used in NumPy, TensorFlow, Torch, or other standard tensor-based frameworks
    - data: `arrow::Buffer`
    - shape: dimension sizes
    - strides: memory ordering
- Zero-copy reads using Arrow’s shared memory tools
- Support Tensor math libraries for C++ like **xtensor**

Example use: Ray ML framework from Berkeley RISELab



- **Shared memory-based object store**
- **Zero-copy tensor reads using Arrow libraries**

Source: https://arxiv.org/abs/1703.03924

Example use: pandas 2.0

- In-planning rearchitecture of pandas's internals
  - libpandas — largely Python-agnostic C++11 library
  - Decoupling pandas data structures from NumPy tensors
- Support analytics targeting native Arrow memory
  - Multicore / parallel algorithms
  - Leverage latest SIMD intrinsics
- Lazy-loading DataFrames from primary input formats
  - CSV, JSON, HDF5, Apache Parquet

Other examples

- Spark integration (SPARK-13534)
- Weld integration (ARROW-649)

Thank you

- Building code and community around
  - IO subsystems
  - Metadata
  - Data structures and in-memory formats