Proceedings of Statistics Canada Symposium 2018
Combine to Conquer: Innovations in the Use of Multiple Sources of Data

Open Data Standards for Administrative Data

Ryan Mackenzie White, Ph.D.

Abstract

The Statistics Canada Artemis data science framework is a record batch based data processing framework, powered by the Apache Arrow open source data format standard, for the production of high-quality administrative data for analytical purposes. Many statistical organizations are shifting to an administrative data first approach for producing official statistics. The production of high-quality, fit-for-use administrative data must preserve the raw state of the data throughout the data life cycle (ingestion, integration, management, processing and analysis). Data formats and production frameworks must support changes to analytical workloads which have different data access patterns than traditional survey data, efficient iteration on the data at any stage in the data cycle, and statistical tools for continuous data quality and fit-for-use assessment. The Statistics Canada Artemis prototype data science framework at the core relies on a well-defined, cross-language, data format that accelerates analytical processing of the data on modern computing architecture.

Key Words: Administrative data, open source, data formats

1. Introduction

1.1 Description

At the core of any data enterprise is a consistent, well-defined data model. The data model will underpin the long-term data management strategy and provide the requirements for computing, storage, and data analysis. The choice of the data model must reflect the analysis patterns of the end-user, and support the analytical workloads for data production.

The Statistics Canada Artemis prototype data processing framework demonstrates the use of the Apache Arrow (Arrow, 2017) in-memory columnar data format and modern computing techniques for data ingestion, management and analysis of very large datasets. The framework supports analytical workloads for administrative data sources that follow a pattern of write once, read many times, and analytical queries with common data access patterns based on a tabular data structure. The framework relies on streaming record-batch based processing, inspired by event-based data processing frameworks used in high-energy particle physics (Berger, et. Al. 2015), for efficient use of computing resources with horizontal and vertical scaling capability.

The Statistics Canada Artemis prototype leverages Apache Arrow from the start, working with Arrow buffers to produce high-quality datasets. The prototype accomplishes this by providing a framework to execute operations on Arrow tables (an execution engine) through user-defined algorithms and tools. The prototype core functionality is configurable control flow for producing datasets consisting of one or more Arrow Tables. The language-neutral Arrow data format allows data sharing to/from other processes or libraries, in-memory, with zero-copy and no serialization overhead.

The primary objectives of the prototype framework are to:

1. Produce logical datasets of a single consistent data format that enable efficient interactions with very large datasets in a single-node, multicore environment.
2. Support analysis on streams of record batches that do not necessarily reside entirely in-memory.
3. Execute complex business processes on record batches to transform data.
4. Incorporate data quality and fitness-for-use tools as part of the data production process.

1.2 General requirements for data processing

The design of data production frameworks, which supports the end-user analyst needs, must focus on four key features:

Performance – The typical performance indicator is the turnaround time required to run the entire processing chain when new requirements or data are introduced. Runtime is limited by transfer of data, loading of data between successive stages (or even successive pipelines), and conversion between various formats to be used across fragmented systems. The software must minimize the number of read/write operations on data and process as much in memory as possible. Distinct stages should be modular, such that changes in stages can be rerun quickly.

Maintainability – Modular software with a clear separation between algorithmic code and configuration facilitate introduction of new code which can be integrated without affecting existing processes through configuration. Modular code also enforces code structure and encourages re-use. Common data format separates the I/O and deserialization from the algorithmic code, and provide computing recipes and boiler-plate code structure to introduce new processes into the system.

Reliability – The system is built on well-maintained, open-source libraries with design features to easily introduce existing libraries for common analysis routines.

Flexibility – Re-use of common processes is facilitated through configurable algorithmic code. Use of a common in-memory data format simplify introducing new features, quantities and data structures into the datasets.

2. Apache Arrow industry-standard columnar data

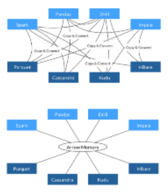
2.1 Open data standards

Open standards enable systems, processes and libraries to communicate with each other. Direct communication using standard protocols and data formats simplifies system architecture, reduces ecosystem fragmentation, improves interoperability across processes, and eliminates dependency on proprietary systems. Most importantly, common data formats facilitate code reuse, sharing, effective collaboration and data exchange, resulting in algorithms and libraries which are supported by a large open community. Common data format that defines the data primitive types that occur in data science, social science and business data will ensure that the raw state of the data can be preserved when consumed by organizations. Tabular data organized in a columnar memory layout allows applications to avoid unnecessary IO and accelerates analytical processing on modern CPUs and GPUs.

The data science and social science community typically deal with tabular data which manifests itself in various forms, most commonly referred to as *DataFrames*. The *DataFrame* concept and the semantics found in various systems are common to the various *DataFrames*. However, the underlying byte-level memory representation varies across systems. The difference in the in-memory representation prevents sharing of algorithmic code across various systems and programming languages. No standard exists for in-memory tabular data, however, tabular data is ubiquitous. Tabular data is commonly found in SQL, the "Big Data" community developed Spark and Hive, and in-memory *DataFrames* are found across popular data science languages. R, python and Julia all have a *DataFrame* in-memory tabular data which is commonly used by analysts.

2.2 Apache Arrow Key Benefits

Figure 2.2-1
Arrow in-memory



The Apache Arrow project (Arrow, 2017) solves the non-portable *DataFrame* problem by providing a cross-language development platform for in-memory data which specifies a standardized language-independent columnar memory format for flat and hierarchical data, organized for efficient analytic operations on modern hardware. Arrow provides computational libraries and zero-copy streaming messaging and inter-process communication. Arrow is a common data interface for inter-process and remote processes.

The Apache Arrow objective is to provide a development platform for data science systems which decouples the vertical integration of data processing components: performant serialization / deserialization and I/O, standard in-memory storage, and embedded computational engine. Apache Arrow deconstructs the typical data architecture stack that is vertically integrated, providing public APIs for each component.

Fast – enables execution engines to take advantage of the latest SIMD (Single input multiple data) operations in modern processors, for native vectorized optimization of analytical data processing. Columnar layout is optimized for data locality for better performance. The Arrow format supports zero-copy reads for fast data access without serialization overhead.

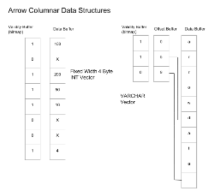
Flexible – Arrow acts as a high performance interface between various systems and supports a wide variety of industry specific languages, including native implementations in C++, Java, Javascript, Rust and Go, with bindings from C++ libraries for Python, Ruby, and R.

Standard – Apache Arrow is backed by key developers from 13 major open-source projects, scientists working at the data frontier, deep learning, and GPU-based software developers, including Calcite, Cassandra, Drill, Hadoop, HBase, Ibis, Impala, Kudu, Pandas, Parquet, Phoenix, Spark, and Storm and CERN making it the de-facto standard for columnar in-memory analytics.

2.3 Apache Arrow Data Format

Apache Arrow comprehensive data format defines primitive data types for scalars of fixed and variable length size as well as complex types such as unions (dense and sparse), structs and lists. Variable width data supports UTF8 or varchar as well as varbinary. Complex types can support nested hierarchical data, for example, to represent nested JSON data.

Figure 2.3-1
Arrow Buffers



- Fixed-length supported primitive types: numbers, booleans, date and times, fixed size binary, decimals, and other values that fit into a given number
- Variable-length supported primitive types: binary, string
- Nested types: list, struct, and union
- Dictionary type: An encoded categorical type

The Arrow column-oriented in-memory format provides serialization/deserialization and supports persistency to various column-oriented backend storage systems and formats. The choice for column-oriented format is based on the benefits achieved for performance reasons. Common access patterns that benefit from column-oriented data access are access elements in adjacent columns in succession and access to specific columns. Columnar data enables SIMD (Single instruction multiple data) based algorithms, vectorized algorithms, and columnar compression.

3. Artemis Data Science Framework

3.1 Overview

The Statistics Canada Artemis prototype framework leverages the Apache Arrow development platform capability, and focuses on data processing and analysis in a collaborative and reproducible manner. The front-end agnostic Arrow API allows us to define a data model to manage the sharing of tabular data across sequences of algorithms, which describe various (sometimes disparate) business processes in a single, in-memory, data processing job. The algorithms describe various business processes for the same dataset, and the algorithms can be re-used for different datasets with common pre-processing and processing requirements. Common tasks and core functionality that must be supported by the framework:

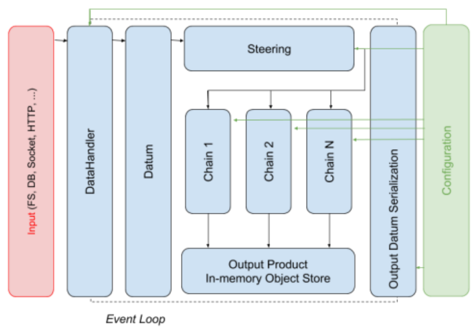
- Comprehensive, robust support for reading and writing a variety of common storage formats in cloud, on premise, local or distributed data warehousing (CSV, JSON, Parquet, legacy data, etc.).
- Ability to perform schema-on-read either through supplied schema or through type-inference and produce a well-defined schema for the dataset.
- Ability to extract meta-data and catalogue data sets in a language and application agnostic manner.
- Support for a write-once, read-many times iterative analysis with minimal data conversions, I/O and serialization overhead.
- Effective and efficient data management.
- Efficient filtering of data, e.g. selection of columns from a master dataset.
- Perform analytical operations, e.g. projections, filters, aggregations, and joins.
- Collection of dataset statistics, e.g. marginal distributions, mean, minimum, maximum.

The primary assumption for the data production is that chunks of raw data can be read into Arrow buffers and subsequently perform computation on streams of record batches. The computations may perform many kinds of filter-projection-aggregate operations on very large dataset with only a small record batch in-memory. The processing of the record batches can be parallelized across many cores or across a cluster of machines, resulting in both vertical and horizontal scaling of computing resources.

3.2 Statistics Canada Artemis Prototype Control Flow

The Statistics Canada Artemis prototype primary objective is the production of datasets which utilize memory and cpu resources efficiently to accelerate analytical processing of data on a single-node, multi-core machine. The data processing can be replicated on independent parts of the dataset in parallel across multiple cores and / or across multiple nodes of a computing cluster in a batch-oriented fashion. The resulting dataset consists of a collection of output file(s), each file is organized as a collection of record batches. Each record batch consists of the same number of records in each batch with a fixed, equivalent schema. The collection of record batches in a file can be considered a table. The columnar data structure is highly compressible and retains the schema as well as the entire payload in a given file. Arrow supports both streams and random access reads of record batches, resulting in efficient and effective data management.

Figure 3.2-1
Artemis Control flow



The control flow from the ingestion of raw data to the production of Arrow datasets proceeds as follows. The raw dataset consists of one or more datums, such as files, database tables, or any data partition. In order to organize the data into collections of a fixed number of record batches to manage the data in-memory, each datum is separated into chunks of fixed size in bytes. The datum is read into native Arrow buffers directly and all processing is performed on these buffers. The in-memory native Arrow buffers are collected and organized as collections of record batches, through data conversion algorithms, in order to build new on-disk datasets given the stream of record batches.

In order to support any arbitrary, user-defined data transformation, the prototype framework defines a common set of base classes for user defined *Chains*, representing business processes, as an ordered collection of algorithms and tools which transform data. User-defined algorithms inherit methods which are invoked by the prototype application, such that the *Chains* are managed by a *Steering* algorithm. The prototype framework manages the data processing *Event Loop*, provides data to the algorithms, and handles data serialization and job finalization. The design is influenced by the ATLAS High-Level Trigger system, a real-time software-based event filtering system used for selecting high-energy physics collision events of interest (ATLAS, 2003).

3.3 Business Process Model

Statistics Canada Artemis prototype design decouples the definition of the business process model, *BPM*, from the execution of those business processes on the data. Business process models are defined by the user and retained in the metadata. The flexibility of defining, retaining and storing the *BPM* in the metadata enables various configurations to be used on the same data, allows for the job to be reproducible, and facilitates data validation and code testing.

The *BPM* can be expressed as a directed graph, describing the relationship between data, their dependencies, and the processes to be applied to the data. The user defines the input(s), the output, the process(es) to be applied to the input(s), and the algorithms which constitute a distinct business process. Each business process consumes data and produces new data, where the new data is the output of one or several algorithms which transform the data. Once the business processes are expressed in a series of algorithms, the processes must be transformed from a directed graph to a linear ordering of processes (where each process is a list of algorithms using a common input). The ordering of algorithms must ensure that the data dependencies are met before the execution of an algorithm.

The ordering of the algorithmic execution is solved with a topological sorting algorithm. The topological sort of a directed graph is a linear ordering of the vertices such that for every directed edge *uv* from vertex *u* to vertex *v*, *u* comes before *v* in the ordering. Users only need to ensure their pipeline defines the input(s), the sequence of algorithms to act on those inputs, and the output. The directed graph of data relationships and the execution order is defined in the Artemis metadata.

Once raw data is materialized as Arrow record batches, the framework needs to provide the correct data inputs to the algorithms so that the final record batches have the defined *BPM* applied to the data. The framework has a top-level algorithm, *Steering*, which serves as the execution engine for the user-defined algorithms. The *Steering* algorithm manages the data dependencies for the execution of the *BPM* by seeding the required data inputs to each *Node* in a *Tree* via an *Element*. The *Tree* data structure is the directed graph generated from the user-defined *BPM Menu*. *Steering* holds and manages the entire *Tree* data structure, consisting of the *Elements* of each *Node*, the relationships between *Nodes*, and all Arrow buffers attached to the *Elements* (a reference to the Arrow buffers). The *Elements* serve as a medium for inter-algorithm communication. The Arrow buffers (data tables) are attached to *Elements* and can be accessed by subsequent algorithms.

3.4 Metadata Model

The Statistics Canada Artemis prototype metadata model has three primary components:
1. The definition of the data processing job, i.e. all the required metadata to execute a business process model.
2. Job processing metadata, i.e. metadata gathered during the execution of the *BPM* and data provenance.
3. Summary metadata, i.e. statistical information gathered during the processing of the data.

Artemis metadata model is defined in a google protocol buffer messaging format. Protocol buffers (Google) are language-neutral, platform-neutral, extensible mechanism for serializing structured data – think XML, but smaller, faster, and simpler. Protocol buffers were developed by Google to support their infrastructure for service and application communication. Protocol buffer message format provides a way to define how to structure metadata once, then special source code is generated to easily write and read the structured metadata to and from a variety of data streams and using a variety of languages. Protocol buffer messages can be read with reflection without a schema, making the format extremely flexible in terms of application development, use and persistency. The messages can be persisted simply as files on a local filesystem, or for improved metadata management the messages can be cataloged in a simple key-value store. Applications can persist and retrieve configurations using a key.

3.5 Data Quality

The data quality is an inherent part of data analysis, and therefore needs to be part of the core design of prototype. The importance of data quality transcends domains of science, engineering, commerce, medicine, public health and policy.

Traditionally, data quality can be addressed by controlling the measurement and data collection processes and through data ownership. The increased use of administrative data sources poses a challenge on both ownership and controls of the data. Data quality tools, techniques and methodologies will need to evolve (Keller, et. Al., 2017).

In the scientific domain, data quality is generally considered an implicit part of the role of the data user. Statistical infrastructure will need support the data users ability to measure data quality throughout the data lifecycle. Statistical tools required for data quality measurements need to be developed to address data quality issues in administrative data. The prototype framework primary statistical analysis tool for data quality is the histogram.

Histograms are the bread-and-butter of statistical data analysis and data visualization, and a key tool for quality control (Freedman, 1981). They serve as an ideal statistical tool for describing data, and can be used to summarize large datasets by retaining both the frequencies as well as the errors. The histogram is an accurate representation of a distribution of numerical data (or dictionary encoded categorical data), and it represents graphically the relationship between a probability density function $f(x)$ and a set of n observations, x, x_1, x_2, ... x_n (Cowan, 1998).

The framework retains distributions related to the processing and overall cost of the processing for centralized monitoring services and also supports user-defined histograms in the algorithms. Central monitoring histograms include timing distributions for each processing stage and each algorithm in the *BPM*, payload sizes, including datums and chunks, memory consumption, distributions of batches and records processed, and statistics on processing errors.

Separate histogram-based monitoring applications can be developed as a post-processing stage and will be able to quickly iterate over large datasets since the input data are Arrow record batches. Automated profiling of the distributions of the data is foreseen as well for input to the post-processing stage data quality algorithms.

Summary

Many statistical organizations are shifting to an administrative data first approach for producing official statistics. Inherent in the production of high-quality, fit-for-use administrative data is the ability for analysts to use the data, measure data quality, and iteratively develop data applications and statistical tools. Administrative datasets are becoming increasingly larger and complex, challenging the ways we approach data analysis and computing. The Apache Arrow in-memory columnar data format provides the ability to develop statistical infrastructure to address data quality and fit-for-use challenges with administrative data; better define computing requirements with the efficient use of CPU and memory; build sustainable, secure and coherent data ecosystems with interoperability; foster a collaborative environment due to code re-use. The Statistics Canada Artemis prototype data processing framework demonstrates the ease to design, build and scale data science systems using a single, open source library to deliver high-quality, data sets designed for efficient processing on single-node, multicore environments.

References

Apache Arrow Project (2017), “Apache Arrow, a cross language development platform for in-memory data”, retrieved from http://arrow.apache.org.

Berger, et. Al. (2015), “ARTUS – A Framework for Event-based Data Analysis in High-Energy Physics”, Berger, J.; Colombo, F.; Friese, R.; Haitz, D.; Hauth, T.; Müller, T.; Quast, G.; Siber, G., arXiv:1511.00852

Atlas Collaboration, “ATLAS high-level trigger, data-acquisition and controls: Technical Design Report”, CERN-LHCC-2003-022; ATLAS-TDR-16.

Google, “Google Protocol Buffers”, retrieved from *https://developers.google.com/protocol-buffers/*

Van Der Walt, Stefan; Colbert, S. Chris; Varoquaux, Gael (2011), “The NumPy array: a structure for efficient numerical computation”, *Computing in Science and Engineering* 13, 2 pp. 22-30.

S. Keller; G. Korkmaz; M. Orr; A. Schroeder; and S. Schipp (2017), “The Evolution of Data Quality: Understanding the Transdisciplinary Origins of Data Quality Approaches” , *Annu. Rev. Stat Appl*. 4:85-108

Freedman, David; Diaconis, Persi (December 1981). "On the histogram as a density estimator: L2 theory", Probability Theory and Related Fields. HeidelbergL: Springer Berlin. 57

Glen Cowan (1998), *Statistical Data Analysis*, Oxford Press.