THE DATA PLATFORM: AN INDEPENDENT SYSTEM FOR MANAGEMENT OF HETEROGENEOUS, TIME-SERIES DATA TO ENABLE DATA SCIENCE APPLICATIONS*

C. McChesney†, C. K. Allen, M. Davidsaver, B. Dalesio, Osprey DCS, Ocean City, Maryland, USA

Abstract

The Data Platform is a fully independent system for management and retrieval of heterogeneous, time-series data required for machine learning and general data science applications deployed at large particle accelerator facilities. It is an independent subsystem within the larger Machine Learning Data Platform (MLDP) which provides full-stack support for such applications. The Data Platform maintains the heterogeneous data archive along with all associated metadata and post-acquisition user annotations. It also facilitates all interactions between data scientists and the data archive; thus it directly supports all back-end data science use cases. Accelerator facilities include thousands of data sources sampled at high frequencies, so ingestion performance is a key. We describe the operation, architecture, performance, and development status of the Data Platform.

INTRODUCTION

The Data Platform (DP) is part of the larger effort by Osprey DCS to build the Machine Learning Data Platform (MLDP) [1]. It provides tools for managing the data captured in an experimental research facility, such as a particle accelerator. The data are used within control systems and control applications to facilitate the creation of machine learning models. Note that the Data Platform component is standalone and completely independent of EPICS. The DP can be deployed locally with a formal installation system available online [2].

Key requirements for the DP include: 1) an Applications Programming Interface (API) for ingestion of heterogeneous, time-series data including scalar values, arrays, structures, and images; 2) data rates expected for an experimental research facility such as a particle accelerator (minimum performance requirement is 4,000 scalar data sources sampled at 1 KHz, i.e., 4 million samples per second); 3) provide an API for retrieval of ingested heterogeneous data; 4) provide an API for exploring metadata available in the archive; and 5) provide mechanisms for adding post-ingestion annotations to the archive and performing queries over those annotations. We consider each requirement and its solution in further detail below.

*Work performed under the auspices of the U.S. Department of Energy with funding by the Office of High Energy Physics SBIR Grant DE-SC0022583.

†Corresponding author email address: cmcchesney@ospreydcs.com

THE DATA PLATFORM

Visible to clients are 2 primary technical components of the DP, 1) an Applications Programming Interface (API) and 2) a suite of "Core Services". Additionally, a Web Application is under development which allows remote users to interact with the data archive using a standard web browser. Here we focus on the API and the Core Services of the Data Platform. Figure 1 shows the DP architecture.

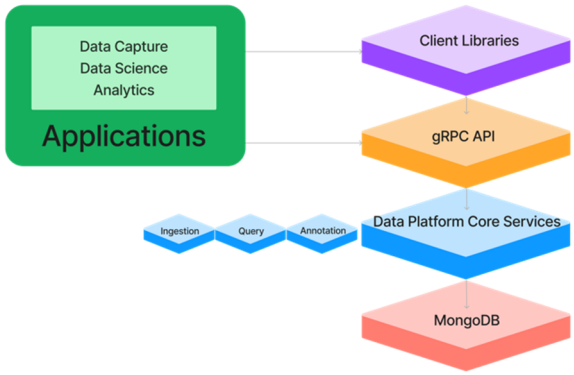

Figure 1: Data Platform architecture.

Data Platform API

The DP API is built with the gRPC high-performance remote procedure call (RPC) framework [3]. The gRPC framework was originally developed by Google but is now open source. It uses HTTP/2 for transport, Protocol Buffers as the interface description language [4] and provides features such as authentication and bidirectional streaming. It generates cross-platform client and server bindings and is essentially language independent. We chose the gRPC framework because it satisfies our performance requirements, is platform independent, and bindings are provided for most all programming languages.

Defining the API for a gRPC server application consists of first identifying the services to be provided. The operations, or Remote Procedure Calls (RPCs), supporting that service are then designed, including the data messages and types to be exchanged by client and service. The gRPC API is defined in the Protocol Buffers meta-language, also developed by Google but now open source [4]. Protocol Buffers files, or “proto” files, contain definitions for service interfaces, service procedures, and the data types exchanged by those procedures. These definitions are then compiled into a communication framework implemented by a standard programming language.

The “proto” file definitions expose the external view of the DP system. Thus, they also express the data models through data types and the service models through the remote methods. Key elements of the data model are shown in Fig. 2 and include 1) process variables, 2) data vectors, 3) handling for heterogeneous data and 4) time. The service model is elaborated in the Core Services description following.

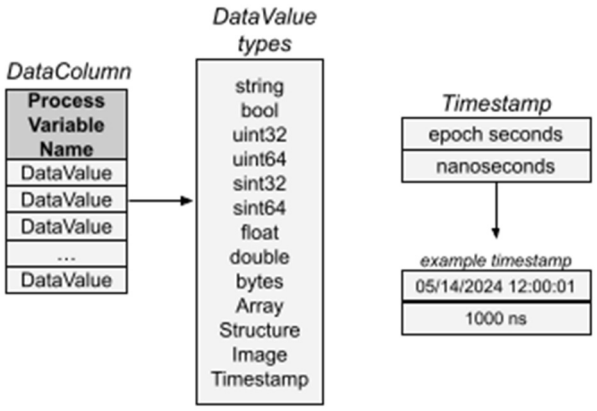

Figure 2: API data model.

Data Platform API Data Model

Process Variables

The core data element of the DP is the *process variable* (PV). These are processes within the facility being monitored or controlled. Process variables are sampled within the control system to create correlated, time-series data, which may be of differing (heterogeneous) data types. The primary purpose of the DP Ingestion Service is to archive these PV measurements, whereas the Query Service provides broad access the archived data in a fashion consistent with data science needs. Note that the DP requires that all PVs within a facility be uniquely named.

Data Vectors

The Ingestion and Query Service APIs for handling data work with vectors of PV measurements. In the APIs this aspect is reflected with data type "DataColumn", which includes a PV name and list of measurements. Note that DataColumn messages support heterogeneous data types.

Heterogeneous Data

A general mechanism for handling heterogeneous data types is required for the Data Platform API, both for ingestion and query. In the API this feature is accomplished by the "DataValue" message; it uses the gRPC "one of" mechanism to support multiple data types for the elements of a data vector. For example, PV measurements can be simple scalar values such as Booleans, integers, floating points, and character strings. However, they can also be complex data types such as multi-dimensional arrays, structures, and images. All are treated in the same fashion.

Time

Time is represented with the "Timestamp" message. It contains fields for 1) seconds since the epoch start, and 2) nanosecond offset from the epoch second. API time methods use either the data type "TimestampList" to send an explicit list of timestamps, or a "SamplingClock" message that includes the parameters of a uniform sampling clock (i.e., start time, sample period, sample count).

Data Platform Core Services

The Core Services are implemented as Java [5] server applications. There are currently 3 independent services: 1) Ingestion, 2) Query, and 3) Annotation. All services utilize the MongoDB [6] document-oriented database management system for persistence. All services share a common design model for request handling.

Ingestion Service

The Ingestion Service provides a streamlined interface for capturing data to the archive. Each ingestion request contains a set of PV data vectors supporting heterogeneous data transport via a single API operation. The API offers methods for both simple RPC request-response interaction and bi-directional streaming supported by gRPC.

Ingestion performance is a key requirement for the DP. Two notable optimizations are 1) using the "bucket pattern" for persistent time-series data and 2) storing those data in raw Protocol Buffers format, that is, the serialization format used for gRPC network communication.

The prototype DP Ingestion Service (aka the "Datastore") created a database record for each individual PV measurement in a specialized time-series database, InfluxDB. This led to serious performance issues both saving and retrieving data due to the sheer number of database records. The new implementation uses the "bucket pattern" to store time-series data in MongoDB. There each database record contains a PV measurement vector over a specified time range. This condition greatly reduces the number of individual database records, thus, greatly improving performance for both ingestion and query operations.

The initial redesign of the Ingestion Service unpacked the Protocol Buffers formatted data values and converted them to appropriate Java data types for database storage. We are currently refactoring the service to directly store the serialized Protocol Buffers data representations as MongoDB time-series data buckets. We expect significant performance gains in ingestion due to elimination of conversion time, and a smaller disk footprint due to the efficiency of Protocol Buffers data serialization.

Query Service

The Query Service facilitates access to the bucketed PV time-series data saved in MongoDB. The service API offers four different query methods meeting a variety of client performance needs, ranging from simple RPC request/response interaction to full gRPC bi-directional streaming. The time-series data query operations utilize the same data structures used in the ingestion API (i.e., "DataColumn" and "DataValue") thus supporting heterogenous data transport via a single API query response data structure. The query response contains the time-series data buckets matching the query parameters. However, high-level methods are also available for retrieving time-series data in tabular format (for example, use by the DP Web Application).

The Query Service also provides metadata query operations. Specifically, properties of the times-series data within the archive can be retrieved. This feature allows clients to explore data conditions within the archive creating more sophisticated time-series data queries.

Annotation Service

A novel DP feature is the support for post-ingestion annotation of archived data. Archive annotations can be comments, calculations, and links between related sets of data. Thus, users can interact with the data archive adding annotations which are then available to all other users.

The Annotation Service extends the core data model with the notion of "datasets", which are used to identify the relevant data for a particular annotation. A "dataset" consists of a collection of "data blocks" where each "data block" specifies a list of PV names and a time range. Consider the entire data archive as a giant spreadsheet, with a column for each PV name and a row for each measurement timestamp, a "data block" specifies some region within that spreadsheet, and a "dataset" contains a collection of those regions. The idea is illustrated in Fig. 3.

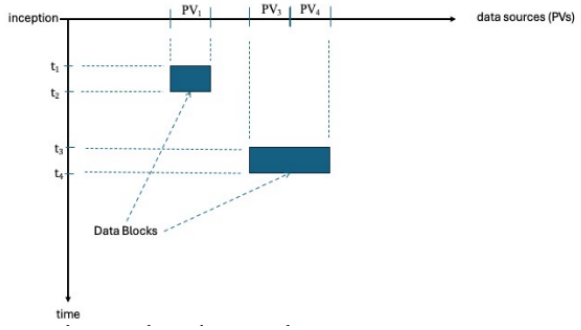

Figure 3: Annotation datasets and data blocks.

The Annotation Service provides a method for retrieving annotations matching specified query parameters. The query result includes the dataset that is the subject for each annotation. The data blocks in those datasets can be used directly as parameters for time-series data queries since they are of the same domain (i.e., a data block is equivalent to a query). In addition to their use in annotations we will be building additional features utilizing datasets, such as a data export mechanism supporting a variety of formats.

Service Request Handling

All DP Core Service implementations share a common framework for handling incoming requests. All requests are processed asynchronously on separate worker threads where results are then dispatched back to clients. The framework provides a generic handler that adds incoming requests to a task queue, in the form of request jobs. A pool of workers is then available to process the pending jobs within the queue. Each job also includes the code for processing a request, such as code for writing data to MongoDB, or executing a database query. The job uses a dispatcher to send the request results back to the client in the API response stream.

The framework also includes a mechanism to support bi-directional-streaming gRPC operations. There, successive requests within in a single client stream are dispatched to the job controller created for that request.

STATUS AND FUTURE DIRECTIONS

The DP Ingestion and Query Services are fully functional for scalar PV measurement data with uniform sampling. The current service implementations exceed the stated project performance goals. Large scale load testing is planned following completion of the current development cycle to measure sustained ingestion data rates, associated database growth, and query performance. A mechanism for handling complex data types (i.e,, arrays, tables, structures, and images) is under development. Additionally, support for data with explicit measurement timestamps is ongoing (i.e., non-uniform sampling).

The initial implementation of the Annotation Service focused on developing the data model for targeting annotations. It laid the foundation for handling a variety of different types of annotations, while directly supporting comment annotations to archive datasets. Support for additional annotation types, such as calculations and linked datasets, will be added in future releases. A data export mechanism based on the Annotations API dataset concept is also planned.

Finally, we mention a short list of longer-term development plans. Most immediate is a token-based authentication and authorization for the Query API. Performance is a continuing focus which includes parameter tuning, horizontal scaling of the Core Services, and employing techniques such as connection pooling and sharing to make more efficient use of MongoDB. We also intended to investigate alternative data streaming architectures (such as Apache Kafka) and exploring long-term data archiving to a structured file format (such as HDF5).

CONCLUSIONS

Initial progress in Data Platform development has been promising. Performance exceeded stated project goals early in project timeline, and will continue to be a focus. We expect further improvement by archiving data in serialized Protocol Buffers format within MongoDB (as well as a reduction in code complexity). The DP Annotation Service provides a novel “value-added” feature to the data archive through user interaction; DP users can annotate the data archive which is then available to all other users. The DP is fully independent and can be deployed locally with a formal installation system available online [2]. Beta releases are soon to be publicly available, or current development releases can be downloaded upon request.

REFERENCES

[1] C. Allen, C. McChesney, M. Davidsaver, and L. Dalesio, “A data science and machine learning platform supporting large particle accelerator control and diagnostics applications”, presented at the IPAC'24, Nashville, TN, USA, May 2024, paper TUPS71, this conference.

[2] Data Platform, Osprey DCS, `https://github.com/osprey-dcs/data-platform`

[3] gRPC, gRPC Authors, https://grpc.io/ [Online documentation], https://github.com/grpc [code repository] (2024).

[4] Protocol Buffers, Google LLC, https://protobuf.dev/ (2024).

[5] Java, Oracle, https://www.oracle.com/java/ (2024).

[6] MongoDB Community Edition, MongoDB Inc., https://www.mongodb.com/products/self-managed/community-edition (2024).