Company Information: Osprey Distributed Control Systems
304 Blue Heron Court
Ocean City, Maryland 21842-2452

Principal Investigator: Leo R. Dalesio
bdalesio@ospreydcs.com
(443) 834-3775

Project Title: A Data Science and Machine Learning Platform Supporting Large Particle Accelerator Control and Diagnostics Applications

Topic: 29 (d): Data Infrastructure for the Next Generation of Adaptive Real-Time Controls for Large-Scale Facilities

Phase I Award: DE-SC0022583 (DOE-FOA-0002555)

Phase II FOA: Initial Phase II, DE-FOA-0002991

Proprietary Data: There is no proprietary data. All work performed in this SBIR proposal is open source. It will be developed in a GIT repository and provided to the EPICS collaboration for use throughout that community.

1 PHASE I PROJECT

1.1 PROBLEM SIGNIFICANCE

The accelerator community has identified a wide variety of data science, machine learning, and general artificial intelligence solutions well suited to accelerator physics applications and facility control, including the areas of general modeling, fault detection, machine tuning, and beam quality (see whitepaper [1]). Appropriately, for the last several years the accelerator physics community been exploring applications of machine learning to the control and optimization of accelerator systems [2] [3] [4]. Application in accelerator physics include standard control room applications used to establish and tune machine set-point parameters such as magnet strengths, RF phase and amplitude, cooling temperatures, etc. [5]. Real-time online control topics include machine parameter uncertainty, beam state prediction (beam size, emittance, etc.), and set-point stabilization (e.g., temperature effects, beam loading, phase drift, etc.). Traditionally the latter applications fall under the purview of formal control theory, however, machine learning techniques offer alternative solutions [6]. Additionally, hybrid techniques combining machine learning and control theory have been investigated for machine tuning and beam state estimation [7].

One immediate difficulty in applying machine learning techniques to particle accelerators is the sheer scope of the data to be collected, processed, and managed. Particle accelerator systems are typically large, sophisticated facilities containing potentially thousands of control and diagnostic points producing data with widely differing formats (e.g., scalars, arrays, tables, images, etc.). Machine learning and general data science applications specific to accelerator systems thus require enormous data collection, correlation, processing, and archiving capabilities for the heterogeneous data available from the facility control system. These data must then be broadly accessible through a query service offering a simple, data-centric interface suitable for data mining, machine learning, artificial intelligence, and general data science applications familiar to accelerator physicists and data scientists. Moreover, for online control applications, data availability must be fast enough for data-science based control algorithms to recognize and correct for changing machine conditions during facility operations. Thus, for the successful implementation of data-science techniques applied to an operating accelerator facility, data access must be comprehensive, timely, and extensively catalogued. Furthermore, there is no standardized platform supporting such applications, one that offers application developers a consistent, data-centric interface to archived data.

1.2 NEED AND OPPORTUNITY

A dedicated platform supporting development and deployment of machine-learning and data science algorithms for accelerator diagnostics and control would be a substantial asset to the particle accelerator community. A standardized interface generalizing implementation and deployment of machine learning and data science algorithms to different operating configurations for the same beamline, or between facilities offers further benefits. Considering the nature of machine learning and the scope and scale of typical accelerator facilities, the success of such a platform would also constitute a next generation of data acquisition, archiving, and data management technologies for particle accelerator systems. It would represent the full-stack integration of machine learning capabilities into a single platform.

Osprey DCS initiated development of the *Machine Learning Data Platform* (MLDP) in direct support of machine learning and data science applications for large particle accelerator facilities and other large experimental physics facilities. It is presented as a "data-science ready" host platform for building artificial intelligence, machine learning, and general data science applications for the diagnosis, modeling, control, and optimization of these facilities. There are three primary functions of the platform, 1) high-speed data acquisition, 2) archiving and management of time-correlated, heterogeneous data, and 3) broad search and query capabilities of the archived data. Additionally, a standalone *Web Application* was developed providing remote access and interaction with the MLDP using only a standard internet web browser, requiring no programming or scripting.

Funding for MLDP development was obtained through a Small Business Innovative Research (SBIR) grant sponsored by the United States Department of Energy (DOE), Office of High Energy Physics (HEP) [8].

Leveraging off our experience and expertise in control systems, synchronous acquisition, data systems, and accelerator systems in general, Osprey DCS *developed a working prototype* of for MLDP during the Phase I effort. Exhaustive evaluations of the prototype were subsequently performed and published in the Phase I Final Report [9]. The report also contains a general overview of the MLDP, detailed descriptions of its architecture and operations, and specifics on internal technologies.

1.3 PROJECT DESCRIPTION

A high-level representation of the MLDP and its operation within an accelerator facility is shown in Figure 1. Explicitly indicated are data acquisition, real-time monitoring and processing capabilities, a data archive with full provenance, and archive search and query via a standardized Applications Programming Interface (API). Since full data provenance is maintained, any data event can be reconstructed from origin and correlated to other events of interest. This capability allows data scientists to mine archived data for anomalies, correlations, and sources. Moreover, post-acquisition user annotations and calculations can be archived along with the facility data allowing data scientists to concurrently search and query stored data and analysis results. Additionally, the full platform supports user-defined data events and conditions that can be monitored in real time as well as annotated during acquisition. This capability is essential for fast ingestion-stream processing and is a focus of the Phase II effort. Seen complementing the MLDP are common off-the-shelf tools for building data science, artificial intelligence, and machine learning applications, such as Keras, TensorFlow, PyTorch, and scikit-learn (e.g., see [10, 11, 12, 13]).

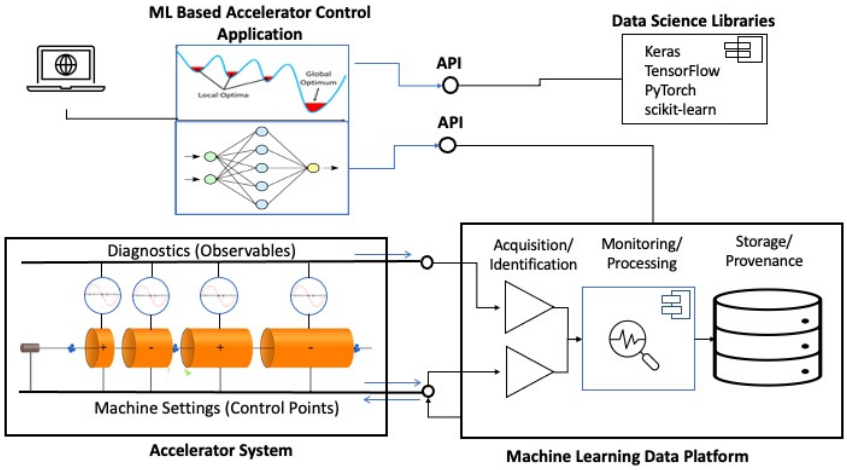

Figure 1: conceptual diagram of Machine Learning Data Platform

1.4 PROJECT OBJECTIVES

Several objectives were stated at project initiation, for both performance and operation.

1. Acquire and archive 4,000 signals at a sample rate of 1 kHz: For 8-byte scalar float values, signals must be acquired and archived minimally at 32 Kbytes/millisecond (i.e., 32 Mbytes/second). For Java implementations with 24-byte float objects the minimum ingestion data rate becomes 96 Mbytes/second.
2. Archive and query time-correlated, heterogeneous data: The archive is to contain heterogeneous data, including both scalar data (Booleans, integers, floats, character strings, etc.) and complex data types (numeric arrays, tables, statistical data samples, complex data structures, images, etc.). These differing data types must be correlated and available within a single query request.
3. Annotation of archived data with user metadata: Users must be able to annotate archived data with notes, data associations, and other artifacts both during acquisition and post-acquisition. Such information is then available to other data science applications through the archive query service.
4. Support the storage and management of data calculations produced post ingestion: Users must be able to annotate the archive with data calculations obtained from the archive itself. The post-ingestion calculations and archive data are then concurrently available to other MLDP users.
5. Wide query capabilities of the data archive directly supporting data science: The MLDP must provide heterogeneous, correlated data sets within a single query request. Data requests may contain time-correlations, metadata, relationships, and other data properties. Additionally, queries are to support

common properties, such as hardware system, equipment, alarm status, or user attributes. The data rates for query operations should be on par with that of data ingestion.

In addition to the original objectives stated at project inception, the following supplemental achievements were realized during Phase I:

6. The Web Application: An independent application was developed for remotely accessing the MLDP data archive using a standard internet web browser. The Web Application is a standalone tool allowing inspection, interaction, and potential annotation of the data archive without programming or scripting.
7. The MLDP archiving and query component is fully standalone: The MLDP archive can be deployed at facilities that are not EPICS-based. Thus, the archive has broad applicability, any facility conforming to an ingestion interface may populate the MLDP archive. Any accelerator or large experimental physics facility can utilize the archiving, management, and search capabilities of the MLDP system.

1.5 MLDP PROTOTYPE

Osprey DCS developed a *working prototype* of the MLDP during Phase I. It was originally intended for installation at facilities utilizing the Experimental Physics and Industrial Control System (EPICS) [14]. However, the archiving and querying component of the platform is now standalone and can be deployed anywhere (see below). In addition to the MLDP prototype a companion utility was also built, the Web Application. It was initially intended as a development tool for inspection and verification of the data archive without a formal API or other supporting services. However, the Web Application was found to have independent merit as it allows anyone, including developers and data scientists, remote access to the data archive without programming or scripting requirements.

Figure 2 depicts the basic architecture of the MLDP prototype and Web Application, also showing basic facility deployment. The MLDP prototype consists of two independent systems, the *Aggregator* and the *Datastore*. Fast real-time data acquisition is realized with the *Aggregator*, which is deployed within the EPICS control system. The remaining functionality of the platform is realized with the *Datastore*. It is responsible for archiving, data management, and query services. Note that the Datastore is the interaction point with data scientists, machine learning applications, and the Web Application; it is the data science interface to the MLDP.

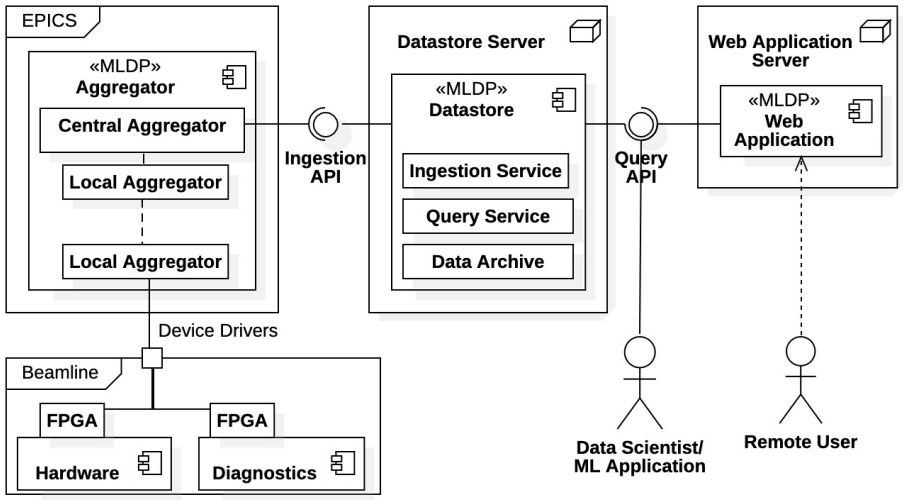

Figure 2: Machine Learning Data Platform prototype

Within Figure 2, MLDP components are identified with the `<<MLDP>>` stereotype. The Aggregator is the MLDP front end, acquiring and coalescing facility data. The Field Programmable Gate Arrays (FPGPs) are specifically identified to indicate high-speed data acquisition. The Aggregator has both distributed components, `Local Aggregators` proximal to the hardware, and a single `Central Component` where data is coalesced and staged for transport to the Datastore system. The Datastore is *standalone*, being hosted on a separate server and exposing well-defined APIs for communications. It is composed of three subcomponents, the Ingestion Service, the Query Service, and the Data Archive. Finally, the Web

Application is also standalone, running on a separate server and connecting to the Datastore via the query service API. It provides remote access and interaction with the Datastore archive.

1.6 PROJECT STATUS AND TECHNICAL FEASIBILITY

The project status is assessed from detailed evaluations of the MLDP prototype in the Phase I Final Report [9]. The following conclusions are made regarding the Project Objectives enumerated in Subsection 1.4:

1. The Aggregator system is mature, operating close to the stated performance goals. It was tested successfully to acquire 3,200 signals sampled at 1 kHz producing 1,000-row data tables every second for an overall data rate of 26 Mbytes/second (equivalent to a 78 Mbyte/second Java implementation). The Datastore still requires significant performance improvements: Maximum continuous, sustained ingestion rates ranged between 0.25 to 7 Mbytes/second, burst rates were in the 30 Mbyte/second range. Network transmission rates through gRPC communications were seen at 380 Mbytes/second.
2. The Datastore system can successfully ingest scalars of type Boolean, integer, and floating point, and complex data types of byte arrays and images. Numeric arrays and complex data structures are transmitted properly, both through ingestion and query, but there are archiving errors. Additionally, there are subtle errors with timestamp ingestion and assignment.
3. The ability to provide user attributes and other metadata during ingestion is available. However, *post-ingestion* modification of the data archive is not yet available.
4. The ability to include calculated data has not been implemented. However, the framework for post-ingestion annotation is in place; the Datastore has a separate metadata repository for storing annotations.
5. The prototype supports wide query capabilities of heterogeneous data, and the operation is robust. However, maximum data rates are seen in the 2.0 to 2.7 Mbytes/second range, below the performance goal of 100 Mbytes/second. Network transmission rates averaged 170 Mbytes/second.

Regarding the additional projection achievements, consider the following observations:

6. The web application prototype can inspect all archived data and metadata. Current search methods are via time ranges and metadata properties. It is launched simply by connecting to a designated URL.
7. Versatility of the Datastore ingestion operations was confirmed using two different data simulators, one emulating the Aggregator system and another emulating the MPEX facility at Oak Ridge National Laboratory [15].

In summary, the MLDP prototype is operational *demonstrating proof of principle*.

- Data throughput is confirmed, from the front-end hardware system, through the Aggregator system, to the Datastore archive, and available to back-end users from the Query Service.
- The Aggregator system is mature and was successfully tested on an EPICS platform simulating the LCLS Beam Position Monitoring (BPM) system at Stanford Linear Accelerator (SLAC). The test platform contained 200 nodes each supplying 16 separate time-series signals, representing the data obtained from BPMs at 200 locations for a total of 3,200 signals at 1 kHz.
- The independent gRPC communications protocol for the Datastore is fully operational and presents no bottleneck. Data transmission rates were seen between 170 and 380 Mbytes/second.
- There is still one outstanding feature of the MLDP yet to be implemented, post-ingestion annotation of the data archive. The framework is, however, available.
- The data rates for the Datastore Ingestion Service need improvement to function in conjunction with the Aggregator. Additionally, the data rates for the Datastore Query Service need improvement for practical data science applications; wait times for large data requests are too large.

- The Web Application is operational. It has a functional interface containing basic search capabilities for the MLDP data archive. The addition of standard data science features, such as visualization, statistics, fitting, etc., would greatly benefit commercialization of the Web Application.

The current technical obstacle is the Datastore system data rates: to meet specification these rates should be on the order of 100 Mbytes/second. This issue is a focus in Year 1 of the Phase II effort and the Work Plan identifies multiple solutions to the performance issue (see Subsection 2.2.1). With improved implementation and hardware, *it is expected that all performance requirements can be met*.

Future feasibility concerns the expansion of the data science use cases for the MLDP, specifically for online machine learning applications. The effort includes development of an ingestion stream processing system capable of rapid responses and identification of exceptional data conditions. Development of such capabilities is *entirely plausible*; feasibility concerns the specifics of the machine learning algorithms supported. This is an open area of investigation in Phase II which includes feedback from data scientists; details are provided in the Phase II Work Plan Subsection 2.2.6.

1.7 TECHNICAL APPROACH

Our general technical approach is research through rapid design and prototyping, then development by building out operation and performance using the knowledge gained. Additional functionality of the platform is pursued in a similar fashion: an initial "alpha" implementation, then optimize performance and functionality. Sub-optimal components can be henceforth identified, corrected and/or redesigned. When warranted, we replace proprietary systems with open-source implementations during the redesign processes, preferably with a staged replacement approach maintaining functionality. We maintain this approach through the Phase II effort, specifics are described in Technical Objectives and Phase II Work Plan.

In Phase I we exploited all available third-party systems and technologies to expedite initial prototype development of the MLDP. Doing so allowed quick evaluations of the prototype thus identifying areas requiring future attention. It also allows data scientists to evaluate a working prototype and provide feedback. Use of third-party components was almost exclusive to the Datastore system, the Aggregator was built only with available EPICS components leveraging off our experience with the EPICS control system. Identified below are some specific third-party components and technologies within the Datastore prototype, these may be targets of future replacement.

- InfluxDB [16]: The InfluxDB database is specifically intended for real-time data acquisition applications. It is designed for efficient storage and retrieval of time-series data and was used as a key component in the time-series data archive.
- MongoDB [17]: The data archive requires a non-relational, document-based (i.e., "NoSQL") database to store metadata associated with the acquisition and archiving process; MongoDB was selected.
- gRPC [18]: The Datastore network communications are realized through gRPC. This is a remote procedure call (RPC) technology based upon Google's Protocol Buffers framework [19].

2 PHASE II PROJECT

The Phase II effort can be broadly categorized as commercialization, deployment, and data science use case extensions. Figure 3 is an overview of the use case extensions and commercialization upgrades required for the Phase II effort. Year 1 focuses on resolving technical and performance issues with the Datastore prototype for commercialization. The effort involves modifications and upgrades of the Datastore data archive, performance upgrades, post-ingestion annotations, and advanced data science use cases. Year 2 focuses on the Advanced Data Science (ADS) capabilities of the MLDP elaborated in Subsection 2.2.6 of the Work Plan. This effort includes direct algorithm support, fast ingestion-stream processing, and MLDP algorithm plugins development. Web Application development can be addressed throughout the Phase II effort as it is an independent system; note that basic data science capabilities are included. Not explicitly seen in the diagram is MLDP installation at existing accelerator facilities.

Phase II project goals originally identified in the Phase I proposal include 1) advanced data compression techniques, 2) archive support of advanced data analysis, 3) platform deployment at existing accelerator facilities, and 4) direct support for advanced data science applications. Areas 3 and 4 remain key focuses of the Phase II project and are directedly addressed in the Work Plan. The efforts within Area 2 are incorporated within the Datastore Upgrade and the Advanced Data Sciences Applications. The need for more advanced data aggregation and compression techniques of Area 1 does not appear warranted at this time; we believe performance can be achieved without these efforts. (However, we are prepared to apply advanced aggregation if required - see Phase I Final Report Subsection 11.2 [9]).

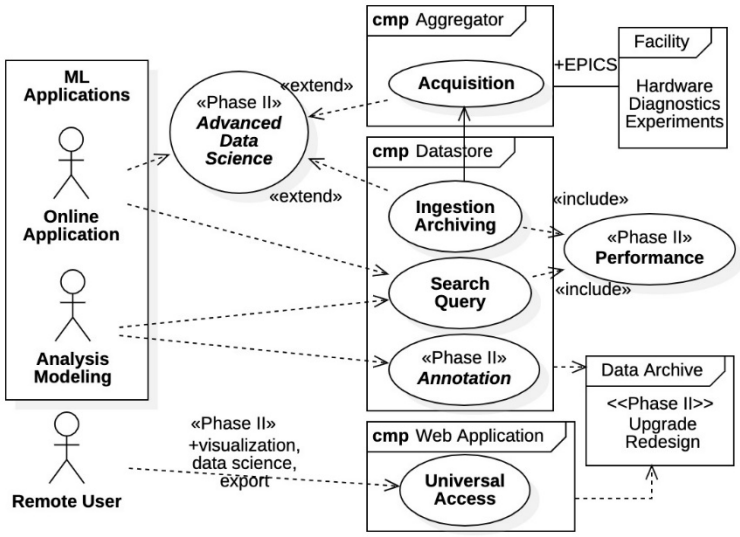

Figure 3: Phase II use case extensions and upgrades

2.1 TECHNICAL OBJECTIVES

Multiple technical and developmental objectives are required for a successful Phase II effort. Crucial issues are identified here along with project milestones. Development is addressed in the Phase II Work Plan.

Datastore Performance – Data Transport and Archiving

The Datastore requires an ingestion data rate of 100 Mbytes/second (Java implementation) to function in conjunction with the Aggregator system. It is desirable that Query Service performance is also 100 Mbytes/second. Maximum data rates observed under continuous, sustained data ingestion ranged from 0.25 to 7 Mbytes/second, while maximum query rates currently range between 2 and 2.7 Mbytes/second. Performance objectives are summarized as follows:

- Increase Ingestion Service performance by at least a factor 100, ideally a factor 200.
- Increase Query Service performance by at least a factor 50.

Meeting the performance goal of 100 Mbytes/second represents a project milestone.

Datastore Archive Annotation

A development objective is post-ingestion annotation of the Datastore archive with user notes, data associations, and data calculations. Full post-ingestion annotation capability is a task milestone.

Elimination of 3rd Party Dependencies

A commercialization objective is the staged replacement of 3rd-party systems to the extent possible.

Facility Deployment

Operational installation of the MLDP at existing accelerator facilities is a project milestone. The MLDP is a complex system containing multiple subcomponents, thus it is desirable to design a system and/or process for rapid deployment. The following are areas of investigation:

1. Automated building and installation of MLDP code bases and systems within host platform.
2. MLDP administration and configuration tools for site compatibility (e.g., an Administration API).
3. Investigate cloud-based deployment of Datastore system (for testing and demonstration).

Advanced Data Science Applications

Phase II research efforts include support for Advanced Data Science (ADS) use cases in the following areas:

1. Development and deployment of working data science applications on the MLDP to verify and validate operation. Investigate and identify class of data science algorithms with well-defined data events and conditions relevant to accelerator operation and suitable for direct ADS support.
2. Advanced tools for accessing and analyzing data sets *within the ingestion stream*. Develop a framework for fast, first-level processing of facility data directly from the ingestion stream, bypassing archiving operations. Develop fast event and condition identification capabilities within ingestion stream.
3. Realization of exceptional machine learning algorithms as packaged "plugins." Supported algorithms can be configured and deployed as modular plugins to the MLDP.

Deployment of an operational machine-learning application on the MLDP would be a project milestone. The plugin framework is an important development objective beneficial to commercialization, demonstration would be a significant project milestone encompassing most project objectives.

Web Application

A development objective is the expansion of the Web Application into a commercial-grade tool for remote data archive access capable of data visualization, basic data science operations, and archive annotations (a least a viable subset). Remote demonstration of the Web Application would be a project milestone.

2.2 WORK PLAN

The Phase II Work Plan is divided by task, further divided by subtask where appropriate. Tasks are ordered loosely by start times included in the header, extended efforts are included where appropriate. Task durations are also included in the header, level of effort is referenced in Section 2.3 Performance Schedule.

2.2.1 Datastore Upgrade (Year 1: Q1-Q4)

Datastore system upgrades are a focus of the Year 1 effort. The MLDP prototype demonstrated proof of principle but now requires significant upgrades for performance and commercialization. The goals of this task are to create a commercial-grade codebase for the Datastore core services, meet all performance and operational objectives, and to eliminate 3rd party dependencies whenever possible. This effort can be divided into the following 5 subtasks: 1) codebase upgrade, 2) post-ingestion annotations, 3) Ingestion Service performance, 4) Query Service performance, and 5) data archive redesign.

1) Codebase Upgrade (Year 1: Q1)

The basic architecture of the Datastore prototype is shown in Figure 4. The Datastore Core is composed of three primary systems, the Data Archive, the Ingestion Service, and the Query Service, the latter two running as independent services on the host platform. External API libraries expose the Ingestion Service to data providers (e.g, the Aggregator) and the Query Service to applications. The data archive is composed of three sub-components, a MongoDB NoSQL database storing metadata, an InfluxDB time-series database, and the host file system, the latter two both storing time-series data. Network communications with the core are realized with an independent remote procedure call framework based upon gRPC.

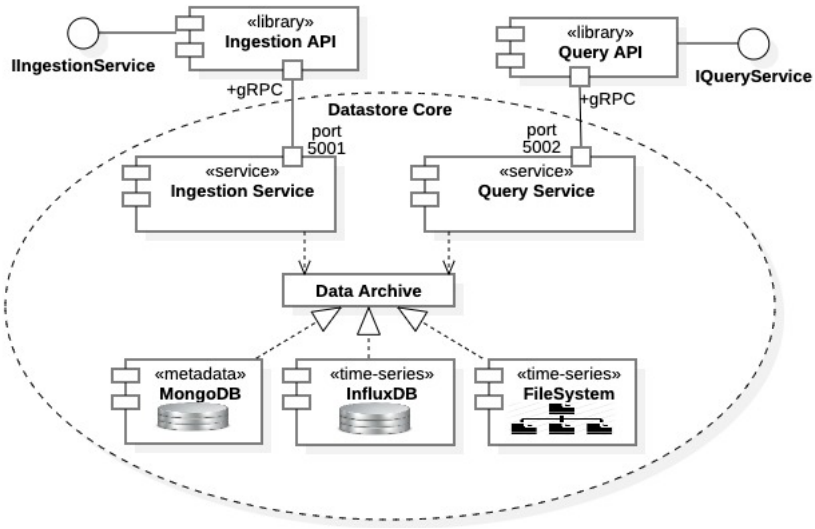

Figure 4: Datastore architecture

Initial development focuses on upgrading the Datastore Core services, specifically the Ingestion Service and the Query Service. Codebase upgrades are a crucial task and a direct reflection of the knowledge gained in Phase I. The effort entails creating a commercial-grade codebase using established software engineering principles. As well as addressing the known implementation errors and limitations identified during evaluations, upgrading the service implementations facilitate profiling, debugging, and legibility. A robust implementation allows profiling of the core services to identify bottlenecks within the data processing and archiving stack. Performance issues can then be addressed in stages where offending operations can be replaced in sequence or, if required, redesigned. A robust codebase also facilitates the Advanced Data-Science use case extensions addressed within Year 2. At the completion of this effort the Datastore will have the same basic architecture of Figure 4.

2) Post-Ingestion Archive Annotation (Year 1: Q1-Q3)

The post-ingestion annotation feature can be implemented in steps, yielding staged functionality.

1. First add user notations to archive data (notes, comments, etc.).
2. Notations are broadened to include source data locations within the archive.
3. Add the ability to identify data associations within the archive.
4. Calculations obtained from source data can be added as notations to the above data associations.

The capability of archive annotation with user attributes, data associations, and post-ingestion calculations would be fully implemented in the final step. Each stage within the process would provide additional functionality toward the final goal, and each stage could be tested and verified independently before proceeding. This effort must be coordinated with the Datastore Archive Redesign.

3) Datastore Ingestion Performance (Year 1: Q1-Q2)

The Datastore Ingestion Service must be capable of accepting data at rates offered by the Aggregator system. An observation seen during evaluations is that maximum data rates vary greatly depending upon the ingested data format; data sets produced by the Aggregator performed worst. Additionally, data rates were initially fast but slowed substantially with continuous, sustained ingestion. Both observations indicate that data transmission is correct, but data processing is not. Thus, data processing within the Ingestion Service or the InfluxDB database is under performing, or both. Consider first the InfluxDB system.

The InfluxDB database system is specifically designed for efficient real-time archiving of time-series data. It is a promising technology for archiving operations and, in principle, should work. Under evaluations it was determined that InfluxDB utilization is inefficient. InfluxDB supports simultaneous, multiple write head archiving and multiple read head retrieval. This feature was not exploited in the evaluations, immediate performance improvement is expected simply with InfluxDB repository partitioning.

An immediate increase in Ingestion Service performance is seen with hardware solutions, specifically, additional server nodes are added to the host platform as shown in Figure 5. Combined with the multi-partitioning of the InfluxDB database, a multi-server host platform provides an increase in write performance essentially proportional to the number of servers. Additional sub-partitioning within each server would likely further increase performance, depending upon the hardware and the data format being ingested. The performance objective may be realized by combining improved Ingestion Service implementation along with hardware solutions. For example, to increase performance by factor 100, a factor 10 is immediately available by using 10 server

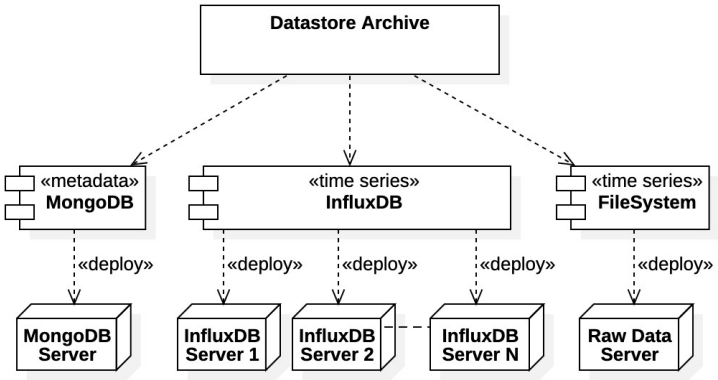

Figure 5: Datastore multi-server deployment

nodes, reducing required implementation performance factor to 10. This fact suggests that *it is the most promising immediate approach*.

Concerning Datastore Ingestion Service performance, specific implementation issues were identified in the evaluations. These issues are technical but essentially described as inefficient processing of the incoming data stream, including the excessive data stream monitoring and underutilization of InfluxDB database capabilities. The situation becomes especially weighty for wide data frames produced by the Aggregator. In summary, immediate performance enhancements should be seen simply with better application, better hardware, and a more efficient data processing implementation.

4) Datastore Query Performance (Year 1: Q2-Q3)

The Query Service performance objective is to make data available fast enough for online control applications, with target performance being equal to the Ingestion Service. The query performance can be addressed later in development, performance issues corrected with the Ingestion Service can then applied.

Concerning data processing, several implementation issues specifically affecting query performance were identified in evaluations and can be immediately addressed. It was found that the Influx language provides fine-grain control over data requests not exploited by the Datastore Query Service. Fine-grained query operations can be pushed down to the InfluxDB database where they should be efficient; query times, and thus data rates, can be improved simply with better utilization.

Note that hardware solutions also apply to the query service. Referencing Figure 5, adding multiple InfluxDB servers to the Datastore host platform will increase query performance, scaling proportionally to the number of read head and partitioned servers. Thus, the hardware and multi-partitioning solution is available for both ingestion and query performance improvements. Consequently, *if ingestion performance requirements are met, we expect the query service performance to follow*.

5) Data Archive Redesign (Year 1: Q2-Q4)

Although the intent is to initially proceed with the InfluxDB and MongoDB database systems, the eventual replacement of the Datastore data archive with an in-house implementation is anticipated. This action provides complete control over all data processing and archiving, directly addressing all performance concerns. It also reduces 3rd party dependencies, desirable for commercialization. We investigate alternate archive designs early then apply them as necessary. The preferred development approach would be a staged, component-wise replacement of archive functionality.

The most radical approach would be complete replacement of the InfluxDB database with an archive of data files on the host platform. It was found during evaluations that direct archiving to system files is fastest for raw data and images. More complex data types (vectors, arrays, data structures, etc.) can also be stored within system files using an HDF5 format [20]. In fact, an archive structure based upon the HDF5 self-describing file format is the most promising approach; Osprey DCS has expertise here, as the Aggregator system already stores output in this format.

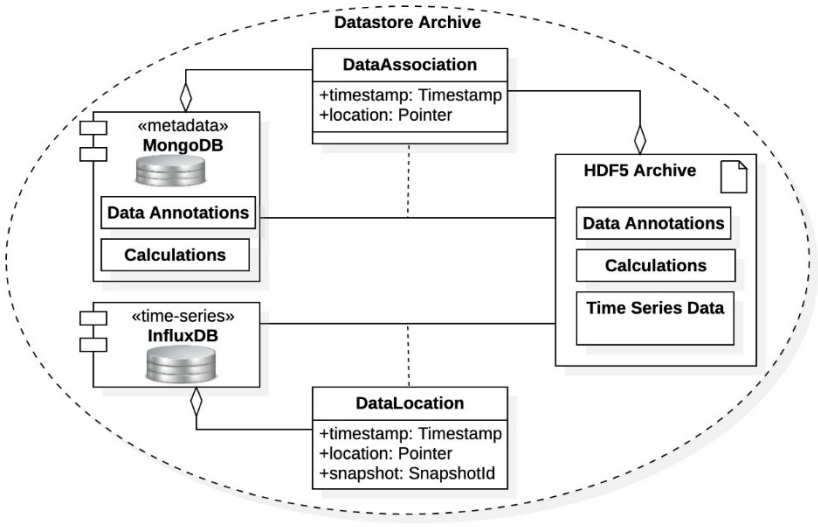

Figure 6: alternate archive design

As an alternative to abandonment of the InfluxDB model, consider the compromise shown in Figure 6. There we see a hybrid system where time-series data is stored in an HDF5 file archive on the host file system, but the InfluxDB database is still used for partial storage. Along with scalar data, data sources and timestamps would be stored in InfluxDB, along with location pointers to the HDF5 archive. The InfluxDB database is efficient at storing scalar time-series data but storing more complex data types requires more sophisticated archiving techniques. The above design attempts to combine the efficiency of InfluxDB time-series storage with the broad applicability of HDF5. The configuration also allows for quick lookups of data locations by the Query Service. As shown in the diagram metadata can still be stored in the MongoDB database as before, or it could be moved to the HDF5 archive. Data associations between time-series data can be created and stored within the metadata archive as before, or with the HDF5 archive.

The hybrid design can also be used as an intermediary toward the staged replacement of all the 3rd party database systems. Once the basic HDF5 archive structure is implemented, functionality can be added in stages. Scalar time-series data types can be first moved from InfluxDB into the HDF5 archive, followed by more complex data types as functionality becomes available. Eventually all metadata can be moved from the MongoDB system into the HDF5 archive, eliminating all 3rd party databases. Development in this fashion ensures the availability of an operational archive while gradually moving toward a final design.

A cursory design for the fully independent data archive is shown in Figure 7. The HDF5 data archive has a flexible format facilitating both comprehensive data access and post-ingestion modifications. Additionally, access to the archive file is strictly controlled; additions and modifications must be managed to maintain integrity and provenance. The latter requirement is met through an independent transaction management system, `TransactionMgr`, which monitors and records all operations within the archive. The flexibility requirement is realized with the high-level abstraction for an archive data file, type `ArchiveFile`, where all entries are maintained. Note that the archive contains both heterogeneous time-series data (type `DataSet`) and metadata (types `Annotation`, `Association`, `DataEvent` and `DataCondition`). Metadata is generated as part of the data ingestion process or added post ingestion.

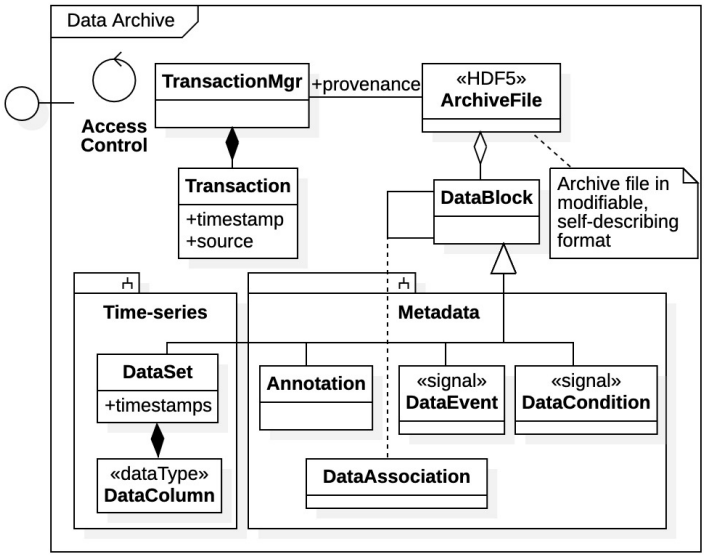

Figure 7: independent archive design

An archive data file is essentially a composition of data blocks (type `DataBlock`) of one form or another. The archive file may also include additional information concerning provenance, archive structure, or other internal requirements. Data blocks are also composite structures, fundamentally generic containers of related information. They can be time-series data sets, annotations produced by the users or by the MLDP system, data events, data conditions, associations between related data blocks, or other information required for use-case extensions. Data annotations are realized as special data blocks within the archive rather than external NoSQL documents. Also, data associations are themselves specialized data blocks that reference other data blocks. The associations support post-ingestion archive annotation and the advanced data science capabilities. Note that the design contains explicit support for *data events* and *data conditions* of the Advanced Data Sciences in Subsection 2.2.6. A practical implementation would likely separate time-series data and metadata to adhere to the staged replacement strategy described above (as indicated in the diagram).

Component-wise separation of the archive by data block type yields a more modular design. Such a design also accommodates a multi-server host platform for increased performance.

2.2.2 Datastore API Libraries (Year 1: Q2 – Q3, Year 2: Q3)

To accommodate wide applicability of the Datastore system, two ingestion API libraries are available. One contains a narrow API specifically for ingesting data provided by the Aggregator, the other contains a wide, general-purpose interface accepting data in native Java formats. The Datastore Query Service also has two query API libraries. One API is designed for data science applications with simple request formats and offering results as heterogeneous data tables. The alternate query API library supports Datastore developers; it is a wide interface offering responses as gRPC messages native to Datastore communications, data requests are made using an SQL-like query language particular to the Datastore. A Datastore administration API library is also available providing tools for data management, data integrity testing, and performance testing. No performance differences were seen between API libraries.

The Phase II effort contains the following development tasks:

1. Develop an additional query API library for the Python programming language producing results in the form of Pandas `DataFrame` objects [21]. The Pandas library is popular in data science applications.
2. Expansion and formalization of the Administration API to support facility deployment, configuration, and verification.
3. The operation and use of all API libraries should be documented. The APIs are the aspects of the MLDP seen by the largest user group and should be well documented in the public domain.

2.2.3 Web Application (Year 1 – Year 2: Ongoing)

For commercialization of the Web Application, additional features are recommended, as well as building out existing ones. Also, stylistically the interface should be improved with better "look and feel" supporting simpler user interaction. The following is a list of project development tasks:

1. Data exporting of archived time-series data and metadata: Isolate and export archive data in common formats (CSV, Excel, NumPy, etc.). Data sets of interest are identified remotely then downloaded for local analysis and processing.
2. Addition of common data science features: Provide data visualization in the form of graphs and charts. The ability to analyze time-series data using standard data science techniques such as averages, fitting, and correlations. Extensive development in this area may be unwarranted as specialized analysis is available through data exporting and common spreadsheet applications, or by Advanced Data Science and machine learning applications supported by the MLDP.
3. Post-ingestion archive annotations (i.e., a viable subset): Features of interest are the ability to annotate archived data with user notes and comments, and to provide data associations within the archive.

The Web Application is an independent system, development can be performed at any point in Phase II.

2.2.4 MLDP Evaluations (Year 1: Q4, Year 2: Q4)

The Aggregator and the Datastore systems of the MLD) were tested on separate platforms. Eventually a test platform capable of simulating full data throughput should be implemented before facility deployment. Such testing is only possible when the Datastore Ingestion Service is performing on par with the Aggregator service at the end of Year 1.

At the end of the Phase II the MLDP will again be evaluated for performance and operations including all Advanced Data Science and use-case extensions. Evaluations will be included in the Phase II Final Report.

2.2.5 MLDP Deployment (Year 1: Q4, Year 2: Q4)

A typical MLDP deployment scenario is shown in Figure 8. Beamline instruments and diagnostic hardware connect to EPICS Input/Output Controllers (IOCs) in the top right-hand side of the diagram. Asterisk decorations (*) indicate multiple hardware and IOCs distributed along the beamline. The diagram includes FPGA controllers to indicate high-speed data acquisition and control. The distributed Aggregator

components are hosted within IOC servers proximal to the hardware (not shown). The Central Aggregator is hosted on an IOC of singular importance, it connects with the Datastore Ingestion Service. The Datastore is hosted on a separate server platform. It ingests data from the Central Aggregator through the interface exposed by an API library. The InfluxDB and MongoDB database systems are also hosted on the Datastore platform. The Datastore hosting platform may contain multiple servers (e.g., for performance) and, consequently, these databases can also be deployed on independent platforms.

The left-hand side of Figure 8 shows a set of facility control room clients: the Operations Console, the Physics Console, and the Web Application. The Operations Console is a platform for hosting Datastore administration utilities. The Physics Console hosts the machine-learning applications utilizing the MLDP. The Web Application is both a client to the MLDP and a service to the Physics Console. The `IAdministration` interface offers methods to manage the Datastore operations and the Datastore archive, while the `IQueryService` interface provides the archive search and query operations provided by the MLDP. Clearly other deployment configurations are possible and would depend upon the facility. The important point to note is that the Aggregator system is deployed within EPICS control system while the Datastore has a separate host platform with well-defined communications interfaces. The Web Application is also deployed on a separate host server; remote users outside the control room can access the MLDP archive via the Web Application server.

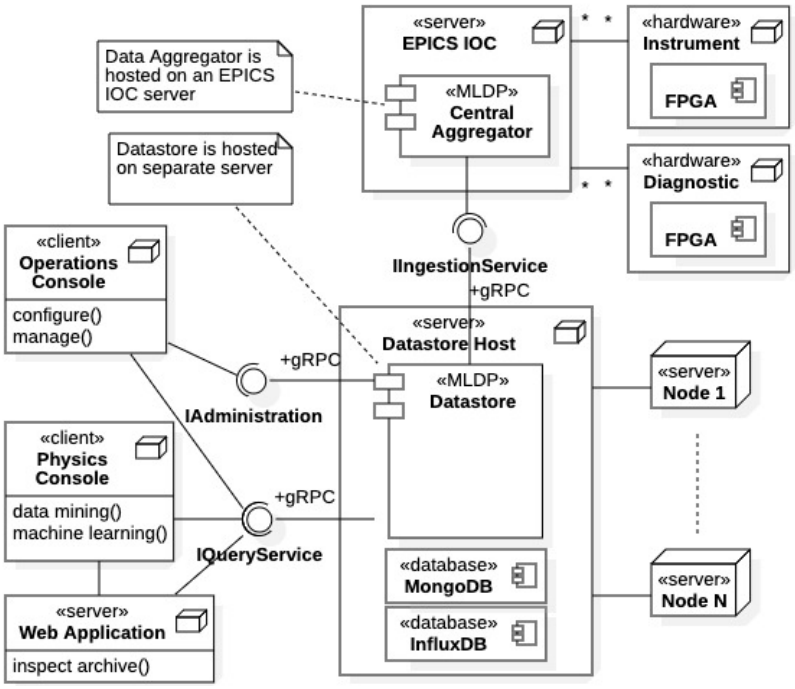

Figure 8: typical MLDP deployment

The Aggregator is mature enough for immediate deployment. Thus, MLDP installation can begin early in the Phase II and concurrently with other efforts. The MLDP can be deployed in the following stages:

1. Install the Aggregator system within the facility EPICS control system.
2. Install the Datastore core services when performance is met; add ADS extensions when available.
3. Develop and install cloud deployment for the Datastore system.
4. The Web Application can be demonstrated whenever a Datastore archive is available.

A project objective is to automate the above process as much as possible; thus, the Phase II effort includes the development of a commercial installation package. An additional objective is cloud-based deployment of the Datastore system; an available cloud service would allow and rapid facility testing and demonstration. Both objectives can be rapidly prototyped using 3rd party deployment solutions such as Kubernetes [22].

Deployment of the MLDP system is planned at two separate accelerator facilities within the United States, the Stanford Linear Accelerator (SLAC) National Accelerator Laboratory in Menlo Park, California [23], and at the Brookhaven National Laboratory (BNL) in Upton, New York [24]. Details are described in Section 2.5 Research Institutions.

2.2.6 Advanced Data Science Applications (Year 1 - Ongoing, Focus Year 2: Q1-Q4)

Crucial to Phase II is the development of Advanced Data Science (ADS) features for the MLDP; this is a broad effort that includes machine-learning application development, use-case extensions, and direct algorithmic support. The development and deployment of machine learning applications within the MLDP is required to verify and validate operation and representing a significant project milestone. Another area of investigation is the inclusion of data science algorithms directly within the ingestion stream; the effort requires investigation of exceptional data science algorithms that are particularly well-suited for ingestion stream processing. This approach provides for the fastest real-time availability of algorithmic data in support of online control and optimization applications. A culmination of the ADS work is the realization of exception algorithms as modular *plugins* to the MLDP system. Successful demonstration of an ADS plugin is a significant project milestone, an operational plugin would encapsulate most advanced features of the MLDP. The ADS effort has three primary areas: 1) development of MLDP applications for accelerator operations, 2) direct ingestion stream algorithm processing, and 3) direct support for exceptional data science algorithms through plugins.

1) Data Science Algorithms for Accelerator Applications (Year 1: Q1 - Ongoing)

An area of investigation continuing throughout Phase II is the application of machine learning and data science techniques presented in the Problem Significance 1.1. The objective is to develop and deploy machine learning and general data science algorithms on the MLDP for real-world accelerator applications. A further objective is to identify exceptional algorithms for direct support with the MLDP ADS system, ultimately as plugins. Exceptional algorithms are identified by certain criteria: real-time processing requirements, rapid identification of triggering data conditions, representability within a standardized framework, and importance to the accelerator community. More general data science algorithms are supported by the advanced data management and search capabilities of the MLDP Query Service.

These investigations will be initiated immediately in Phase II as the MLDP prototype is operational. However, the ingestion stream processing development would be initiated in Year 2, as data archive support must be first addressed. Osprey staff will work with data scientists at SLAC to identify and develop machine learning, artificial intelligence, and general data science applications for deployment on the MLDP (see attached Letter of Commitment). The SLAC facility has a dedicated group focusing on machine learning and artificial intelligence applied to accelerator sciences [2]. Additionally, they will help identify exceptional algorithms suitable for direct ADS support.

2) Ingestion Stream Processing (Year 1: Q3, Year 2: Q1 - Q4)

Ingestion stream processing is the inline, real-time processing of algorithm data within the MLDP data stream (from Aggregator to Datastore). It provides the most rapid data availability for online machine learning applications. Rather than querying for data sets from the archive, applications identify data sets and conditions of interest *a priori*. The intent is to then provide value-added features through fast first-level processing (applications not requiring fast processing could simply monitor EPICS process variables).

Figure 9 shows the basic data flow supporting ingestion stream processing for an online application. The incoming facility data is twinned from the Datastore Ingestion Service and sent to an independent Datastream Processor component. There, fast first-level processing is performed then pre-processed data sets are delivered to the online application whenever special triggering events are recognized. Indicated in the diagram, the Datastream Processor requires instructions for both data pre-processing *and* identification of triggering conditions. Instructions will likely

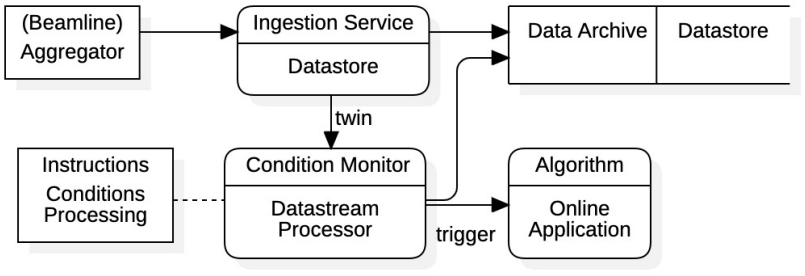

Figure 9: ingestion stream monitoring and processing

be strongly coupled to the specific algorithm; however, we discuss generalizations below. Support for ingestion stream processing requires both the development of a data processing infrastructure and a monitoring system capable of quickly identifying triggering conditions. Additionally, to be practical it must also be configurable, at least for some class of algorithms. Task feasibility concerns the extent to which we can formalize datastream conditions, and the speed of processing.

Before presenting an architectural framework consider some typical applications. Many machine learning algorithms utilize internal dynamics parameters, say *state variables*, that can be supported with ingestion stream processing. For example, the decision policy in reinforcement learning or, more generally, random variables from any Markov process. Support vectors representing data clusters can also be considered state variables. Recursive estimation techniques such as Kalman filters or Bayes tracking also use internal state variables, as do many other state-estimation techniques. Non-linear programming techniques (e.g., in optimization) also produce dynamic state vectors potentially requiring only local information from the ingestion stream. All such algorithms are possible candidates for ingestion stream processing.

Another area of direct support is the generation of training data sets for neural networks. In this case training data could be taken directly from the ingestion stream. For supervised learning, however, classifier determination would typically be a computational matter dependent upon machine characteristics derived from facility data. Here programmatic support for classifier calculations is of concern.

From the above example, ingestion stream processing requires the algorithmic identification of datastream conditions. Some generalization can be made. For example, computation of the training set classifier for supervised learning is a datastream condition. Likewise, the computation of the reward function in reinforcement learning is also a datastream condition, as is any merit functional. Additionally, standard statistical quantities represent data conditions. All have a similar computation framework: quantities are computed from machine conditions or state variables determined by facility data.

To clarify the ingestion stream processing requirements, we present a general mathematical framework. First, let the vector $\mathbf{d}_i$ represent the ingestion stream data at discrete time i. Note the decomposition

$$\mathbf{d}_i \triangleq \left(u_i^{(1)}, \ldots, u_i^{(M)}; y_i^{(1)}, \ldots, y_i^{(N)}\right) \tag{1}$$

where the $u_i^{(m)}$ are control variables (i.e., adjustable machine parameters) and the $y_i^{(n)}$ are observables (i.e., diagnostic equipment outputs). That is, the ingestion stream data contains both machine parameters that algorithms may adjust and the diagnostic outputs responding to the machine variations. Now assume some internal state variable $\mathbf{x}_i$ associated with the algorithm. For example, this may be a decision policy, a statical sample (e.g., correlation matrix), or other machine state estimate. The algorithm's state transition is given by $\mathbf{x}_i = \mathbf{F}_i(\mathbf{x}_{i-1}; \mathbf{d}_i)$ where $\mathbf{F}_i$ is a function representing the state dynamics and depends upon the previous state $\mathbf{x}_{i\text{-}1}$, the current facility data $\mathbf{d}_i$, and the time i. Let J_i represent the value of the algorithm's datastream condition at time i. This value may be discrete in the case of a classifier, or it may be floating point if representing a reward or merit, it may even be a vector in the case of deep learning. In any event, there must be a deterministic formula J involving the facility data and the internal states, say $J_i = J(\mathbf{d}_i, \mathbf{x}_{i-1}, \mathbf{x}_i)$. We include the possibility that J depends on the transition between states $\mathbf{x}_{i-1} \to \mathbf{x}_i$ (e.g., for reinforcement learning). Let the output of the algorithm at time i be the vector $\mathbf{o}_i$. This output would involve a calculation $\mathbf{K}$ potentially including the facility data $\mathbf{d}_i$, the current state $\mathbf{x}_i$, and the condition value J_i, that is $\mathbf{o}_i = \mathbf{K}(\mathbf{d}_i, \mathbf{x}_i, J_i)$. For example, in supervised learning $\mathbf{K}$ may simply select a subset of the facility data $\mathbf{d}_i$ and include the classifier value J_i. Finally, we provide for a Boolean triggering condition t_i used to signal a response. The trigger condition could be a data event, data condition, or state condition represented generally by the functional $t_i = T(\mathbf{d}_{i-1}, \mathbf{d}_i, \mathbf{x}_{i-1}, \mathbf{x}_i)$. For example, a shift in parameter space, state transition, or even a simple alarm condition can all be represented by T.

Collecting all the above yields the following mathematical framework for ingestion stream processing:

$$\begin{aligned} \mathbf{x}_i &= \mathbf{F}_i(\mathbf{x}_{i-1}, \mathbf{d}_i)\,, \\ J_i &= J(\mathbf{d}_i, \mathbf{x}_{i-1}, \mathbf{x}_i)\,, \\ \mathbf{o}_i &= \mathbf{K}(\mathbf{d}_i, \mathbf{x}_i, J_i)\,, \\ t_i &= T(\mathbf{d}_{i-1}, \mathbf{d}_i, \mathbf{x}_{i-1}, \mathbf{x}_i)\,. \end{aligned} \tag{2}$$

The first equation represents the algorithm dynamics, and the third equations is the algorithm output to the application. The second equation represents the datastream condition while the fourth equation identifies the trigger condition. Thus, outputs may be sent to the application at every *i,* or when the trigger condition t_i is met. Datastream processing instructions are represented by the function **K**. Datastream conditions are represented by the functionals *J* and *T*. The function $\mathbf{F}_i$ is internal to the algorithm and is the only (potentially) time-dependent operator. We present a design framework for (2) in the next subsection.

3) Algorithm Plugins (Year 2: Q1 - Q4)

The objective here is to provide a software framework directly supporting a class of machine-learning and data science algorithms, with focus towards online applications. Direct support yields fastest availability of algorithmic calculations, enhanced by ingestion stream processing. Note that a machine learning algorithm could be packaged fully, or in parts, depending upon its complexity. For example, algorithms that identify clusters or statistics in data would likely be fully realized plugins whereas neural network training algorithms would likely be partially realized as plugins that produce training data sets. Additionally, plugins could produce metadata and other annotations that can be monitored and searched during ingestion.

A design for the ingestion stream processing and plugin framework based upon the above considerations and the mathematical framework of Eqs. (2) is shown in Figure 10. Here we again see the basic component for ADS operation, the Datastream Processor. It is an independent subsystem that couples to the Datastore Ingestion Service through an additional service labeled `Twinning`. The Ingestion Service essentially operates as before, archiving facility data and metadata produced the Aggregator, however, it now provides access to the data stream through the twinning service. The Datastream Processor component supports ADS Plugins (labeled `Plugin`) through a well-defined software port `plugin framework`. An online application selects plugins by algorithm type, parameter configurations are made via the exposed API `IPlugin`. Algorithm output is also available through the API but can also be saved to the data archive for historical reconstruction. Note the inclusion of a staging area within the Datastream Processor to maintain intermediate data during calculations $\mathbf{F}_i$, **K**, *J*, and *T*.

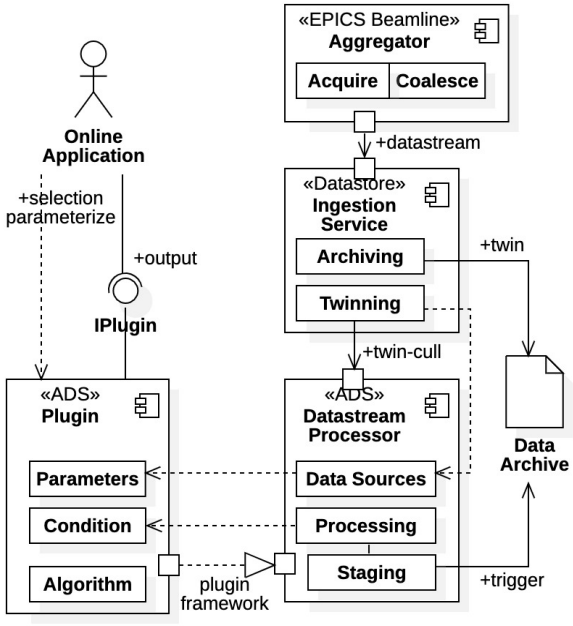

Figure 10: ingestion stream processing and plugin

The algorithm dynamics $\mathbf{F}_i$ are realized within the `Algorithm` subcomponent of the Plugin while datastream conditions *J* and *T* are defined in the `Condition` component. For fastest responses, it is preferably that condition values J_i and t_i, as well as output $\mathbf{o}_i$, are computed within the Datastream Processor (i.e., the `Processing` component) and stored within the staging area. That is, we develop a programmatic framework for *defining* calculations *J*, *T*, and **K** within the plugin, but *performing and storing* the calculations within the Datastream Processor. Once triggering conditions are met, the pre-processed data set $\mathbf{o}_i$ would be transmitted to the supported application (and potentially the data archive as an annotation). However, calculations may require implementation within the plugin if too complex or impractical (for example, if

calculations require state vectors $\mathbf{x}_i$). Distribution of the processing capabilities between the Plugin and Datastream Processor components will be an area of investigation. Finally, explicitly indicated is that for added efficiency the ingestion stream can be culled during twining (i.e., by the Ingestion Service) using the Plugin `Parameters` to identify specific data sources.

Now consider Figure 11 depicting a more detailed data flow diagram with a more advanced plugin framework design; access to the data stream being dictated by algorithmic requirements. Pushing data access farther down the data stream reduces plugin architecture requirements, however, the cost is reduced response times. Seen in the diagram are three types of plugins, one monitoring *data events*, one monitoring *data conditions*, and one that performs *data mining*. The data event plugin, `Event Monitor`, is situated as before and provides the fastest response to observed data events (i.e., calculation *T*). The plugin monitoring data conditions *J*, the `Condition Monitor`, pulls directly from the archive; thus, the plugin sees the entire archive but need only concern itself with desired data sets required by *J*. However, responses are now limited by archive times. The simplest plugin is the data mining plugin, `Data Mining`. Having full access to the Query Service all broad query capabilities are available. However, the response times are the slowest; this type of plugin essentially represents a "packaged application." Included in Figure 11 is the common staging area; in addition to buffering intermediate results, it can also store metadata for fast monitoring by other applications.

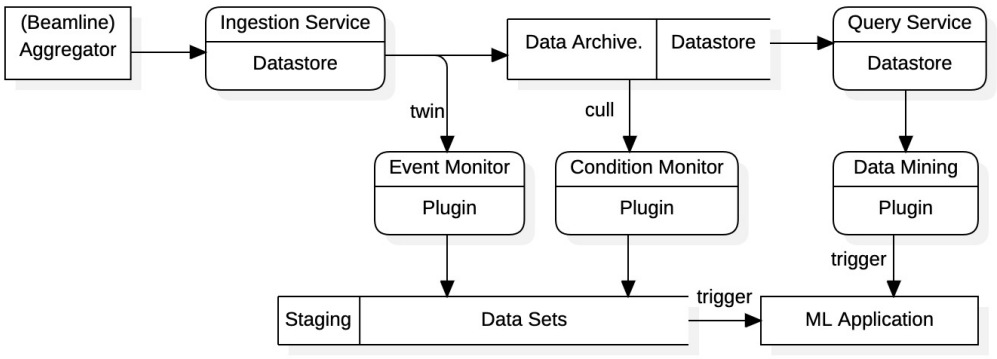

Figure 11: advanced ingestion stream support for data sciences applications

Advanced Data Sciences Development Strategy

The diagram of Figure 11 suggests a reasonable development strategy:

1. Identify and develop a suitable machine learning or data science algorithm using only the MLDP Query Service (confirm on test data).
2. Develop the data mining plugin architecture. The design here would be independent and provide basic requirements of a plugin framework, and a test algorithm.
3. Concurrently develop the ingestion stream processing infrastructure. Provide basic ingestion stream access and first-level processing framework.
4. Develop data conditions architecture. This "meet in the middle" approach yields design requirements at both ends. Decide what additional data conditions/events can be supported in new framework.
5. Formalize the plugin framework and interface. Document the programming requirements and communication protocols for any MLDP plugin component.

FPGA Datastream Processing

For completeness we mention another avenue in direct support of data science applications: data stream processing could be deployed at the FPGA level. The approach provides the fastest possible response times for real-time applications. Algorithms deployed on FPGAs could sample at megahertz rates while producing results available in the kilohertz range. This configuration would allow scientists to identify collective data patterns in fine-grained machine response. Exceptional algorithms identified in high-level applications that can be appropriately implemented would be moved to the FPGA level. Thus, the approach essentially constitutes "one-off" development efforts.

2.3 PERFORMANCE SCHEDULE

Performance schedules for each project year are described in the tables below. The schedules are divided by tasks described in the Phase II Work Plan. Start times are given by yearly quarters (i.e., Q1 through Q4), level of effort is in Man Months (MM), and yearly staffing is provided as Full Time Employee (FTE). Significant milestones for tasking are also included.

2.3.1 Year 1

Task	Items	Start	End	Effort	Yearly	Milestones
Datastore Upgrade	Upgrade/Redesign	Q1	Q3	6 MM		100 Mbyte/sec Full Annotations
	Performance	Q1	Q3	3 MM		
	Annotations	Q2	Q3	1 MM		
			Senior Staff	10 MM	0.83 FTE	
APIs	Administration API	Q2	Q3	2 MM		
	Python Query API	Q3	Q3	1 MM		
			Junior/Senior Staff	3 MM	0.25 FTE	
Adv Data Science	Applications	Q2	Cont.	1 MM		
	Archive Support	Q4	Q4	2 MM		
			Senior Staff	3 MM	0.25 FTE	
Deployment	MLDP Testing	Q4	Q4	1 MM		Data Throughput Aggregator Install.
	Aggregator Install.	Q4	Cont.	1 MM		
			Senior Staff	2 MM	0.16 FTE	
Web Application	Upgrades	Q1	Cont.	2 MM		
	Visualization	Q1	Cont.	1 MM		
	Data Science	Q1	Cont.	1 MM		
			Junior/Senior Staff	4 MM	0.33 FTE	

Year 1 total support is for 1.85 full-time employees for a total of 22 man-months effort. The majority effort is focused on archive performance and commercialization upgrades. Project milestones are full ingestion and query performance rates of 100 Mbytes/second, full data throughput through the MLDP, and Aggregator facility installation. Approximately 3 to 4 man-months will be provided by junior-level staff.

2.3.2 Year 2

Task	Items	Start	End	Effort	Yearly	Milestones
Adv Data Science	Applications	Q1	Q4.	2 MM		MLDP Appl. Plugin Operation
	Datastream Process.	Q1	Q3	4 MM		
	Plugin Framework	Q1	Q3	4 MM		
	Integrat./Performance	Q2	Q4	4 MM		
			Senior Staff	14 MM	1.16 FTE	
APIs	ADS Support	Q2	Q3	1 MM		Facility Install.
	Documentation	Q3	Q3	1 MM		
			Senior Staff	2 MM	0.16 FTE	
Deployment	Deployment Systems	Q1	Q4	1 MM		Cloud Deploy. Facility Install.
	Cloud Deployment	Q1	Q4	1 MM		
	Facility Installation	Q1	Q4	1 MM		
	Senior Staff			3 MM	0.25 FTE	
Web Application	Commercialization	Q1	Q4	2 MM		Remote Demon.
	Data Science	Q1	Q4	1 MM		
			Junior/Senior Staff	3 MM	0.25 FTE	

Year 2 total support is for 1.85 full-time employees for a total of 22 man-months effort. The majority effort is focused on Advanced Data Science use-case extensions and their application. Project milestones include

cloud deployment of the Datastore, full MLDP deployment and operation at an existing accelerator facility, MLDP application demonstration, and demonstration of an operational MLDP plugin. Remote demonstration of the Web Application in conjunction with MLDP facility operation is also a project milestone. Approximately 3 man-months of effort are provided by junior-level staff.

2.4 FACILITIES AND EQUIPMENT

Specialized hardware is required in Year 1, the Work Plan includes a dedicated multi-node server platform for performance improvements. Costs are not expected to exceed $5,000.

No facility costs are expected, specialized facility access is granted by our partner institutions under LOCs.

2.5 RESEARCH INSTITUTIONS

Two particle accelerator facilities have agreed to participate in the Phase II project; included in the Phase II submissions are their Letters of Commitment (LOC). Work is to be formed on site by Osprey DCS under LOC and no additional funding or contractual obligations are required. Institutions have agreed to provide facility access, data availability, and staff support for MLDP deployment and testing.

SLAC National Accelerator Laboratory

2575 Sand Hill Road	**Contact: Greg White**
Menlo Park, CA 94025-7015	Phone: (650)926-4277
	Email: greg@slac.stanford.edu

The Stanford Linear Accelerator (SLAC) facility in at SLAC National Accelerator Laboratory is an ideal candidate for initial MLDP deployment and testing. SLAC is an established facility having a two-mile long linear accelerator producing electrons with energies in the GeV range [22]. The facility employs the EPICS control system and has thousands of control and observation points necessary to confirm the full performance of the data platform. Osprey DCS has extensive experience working with the SLAC facility and maintains a continuing interaction. Additionally, SLAC has a team devoted to the application of machine learning techniques to accelerator applications which can provide feedback on Advanced Data Science platform upgrades in Phase II [2].

The deployment will begin in Year 1 by installing the Aggregator component of the MLDP onto the Low-Level Radio Frequency (LLRF) systems used for controlling high-power RF acceleration. The LLRF system contains feedback and feedforward control systems for stabilization and is well characterized, data throughput can be readily evaluated. The Aggregator system itself would also provide enhancement in facility data acquisition by producing correlated data sets. Full MLDP installation proceeds later in Phase II with the Datastore system. Performance of search and retrieval will be evaluated using all accelerator hardware signals. Data scientists at SLAC then would provide guidance in developing machine learning algorithms hosted on the MLDP where application can be tested on facility data. The Advanced Data Science capabilities can then be installed when available during Year 2.

Brookhaven National Laboratory

P.O. Box 5000	**Contact: Yuke Tian**
Upton, New York 11973-5000	Phone: (631) 344-2872
	Email: ytian@bnl.gov

Brookhaven National Laboratory (BNL) houses the National Synchrotron Light Source II (NCLS-II) facility, maintaining a 3 GeV electron ring with 28 experimental beamlines [23], as well as the proposed Electron-Ion Collider (EIC) [24]. The facility is EPICS based supporting Aggregator deployment. Installation at either of these facilities would proceed in a similar fashion as for the SLAC facility described above.

3 REFERENCES

[1] A. Edelen, C. Mayes, D. Bowring, D. Ratner, A. Adelmann, R. Ischebeck, J. Snuverink, I. Agapov, R. Kammering, J. Edelen, I. Bazarov, G. Valentino and J. Wenninger, "Opportunities in Machine Learning for Particle Accelerators," November 2018. [Online]. Available: https://www.osti.gov/biblio/1529359.

[2] "Machine Learning at SLAC," SLAC National Accelerator Laboratory, [Online]. Available: https://ml.slac.stanford.edu/.

[3] Brookhaven National Laboratory, "Artificial Intelligence," [Online]. Available: https://www.bnl.gov/science/artificial-intelligence.php.

[4] W. Ferguson, "Machine Learning Paves Way for Smarter Particle Accelerators," Lawrence Berkeley National Laboratory, 19 July 2022. [Online]. Available: https://newscenter.lbl.gov/2022/07/19/ml-particle-accelerators/.

[5] A. Scheinker, D. Rees, B. Garnett, S. Milton, A. L. Edelsen and D. Bohler, "Applying Artificial Intelligence to Accelerators," in *9th Intern. Particle Accel. Conf. (JACoW-IPAC2018-THYGBE1)*, Vancouver, British Colombia, Canada, 2018.

[6] M. Gnida, "AI learns physics to optimize particle accelerator performance," Stanford Linear Accelerator National Laboratory, 29 July 2021. [Online]. Available: https://www6.slac.stanford.edu/news/2021-07-29-ai-learns-physics-optimize-particle-accelerator-performance.aspx.

[7] A. Scheinker, "Adaptive Control and Machine Learnning for Particle Accelerator Beam Control and Diagnostics," in *10th Int. Beam Instrum. Conf.*, Pohang, Korea, 2021.

[8] Osprey DCS, "A Data Science and Machine Learning Platform Supporting Large Particle Accelerator Control and Diagnostics Applications," United States Department of Energy, Grant #DE-SC0022583, 2022.

[9] C. K. Allen, B. Dalesio, G. McIntyre, C. McChesney and M. Davidsaver, "Machine Learning Data Platform, TM-01-2032," Osprey DCS, Ocean City, MD, 2023.

[10] "Keras," [Online]. Available: https://keras.io/.

[11] "TensorFlow," [Online]. Available: https://www.tensorflow.org/.

[12] "PyTorch," [Online]. Available: https://pytorch.org/.

[13] "scikit-learn," [Online]. Available: https://scikit-learn.org/stable/index.html.

[14] "EPICS - Experimental Physics and Industrial Control System," 2022. [Online]. Available: https://epics-controls.org/.

[15] J. Rapp and et. al., "The Material Plasma Exposure eXperiment: Mission and conceptual design," vol. 156, no. 111586, July 2020.

[16] "InfluxDB," InfluxData Inc., 2022. [Online]. Available: https://www.influxdata.com/.

[17] "MongoDB," MongoDB Inc., 2022. [Online]. Available: https://www.mongodb.com/home.

[18] "gRPC," gRPC Authors, 2022. [Online]. Available: https://grpc.io/.

[19] "Protocol Buffers," Google, 2022. [Online]. Available: https://developers.google.com/protocol-buffers/.

[20] The HDF Group, "The HDF5® Library & File Format," The HDF Group, 2008. [Online]. Available: https://www.hdfgroup.org/solutions/hdf5/. [Accessed 2022].

[21] "pandas.DataFrame," numFocus, Inc., 2022. [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html.

[22] The Linux Foundation, "Kubernetes," Cloud Native Computing Foundation, 2023. [Online]. Available: https://kubernetes.io.

[23] "SLAC National Accelerator Laboratory," [Online]. Available: https://www6.slac.stanford.edu/.

[24] Brookhaven National Laboratory, "Brookhaven National Laboratory," [Online]. Available: https://www.bnl.gov/world.

[25] Brookhaven National Laboratory, "National Synchrotron Light Source II," [Online]. Available: https://www.bnl.gov/nsls2/.

[26] Brookhaven National Laboratory, "The Electron-Ion Collider," [Online]. Available:https://www.bnl.gov/eic/.