A Demonstration of the BigDAWG Polystore System

A. Elmore	J. Duggan	M. Stonebraker	M. Balazinska	U. Cetintemel	V. Gadepally	J. Heer	B. Howe	J. Kepner
Univ. of Chicago	Northwestern	MIT	Univ. of Wash.	Brown	MIT-LL	Univ. of Wash.	Univ. of Wash.	MIT-LL
T. Kraska	S. Madden	D. Maier	T. Mattson	S.Papadopoulos	J. Parkhurst	N. Tatbul	M. Vartak	S. Zdonik
Brown	MIT	Portland St U.	Intel	Intel / MIT	Intel	Intel / MIT	MIT	Brown

ABSTRACT

This paper presents BigDAWG, a reference implementation of a new architecture for "Big Data" applications. Such applications not only call for large-scale analytics, but also for real-time streaming support, smaller analytics at interactive speeds, data visualization, and cross-storage-system queries. Guided by the principle that "one size does not fit all", we build on top of a variety of storage engines, each designed for a specialized use case. To illustrate the promise of this approach, we demonstrate its effectiveness on a hospital application using data from an intensive care unit (ICU). This complex application serves the needs of doctors and researchers and provides real-time support for streams of patient data. It showcases novel approaches for querying across multiple storage engines, data visualization, and scalable real-time analytics.

1. INTRODUCTION

The Intel Science and Technology Center (ISTC) for Big Data was founded in 2012. This center, with an open IP model, has facilitated building a community of researchers with a focus on Big Data storage architectures, analytics, and visualizations while considering streaming and future disruptive technologies. The center is now entering a "capstone" phase where it is implementing a federated architecture to enable query processing over multiple databases, where each of the underlying storage engines may have a distinct data model. To tackle this challenge, the Big Data Analytics Working Group (BigDAWG) project is exploring challenges associated with building federated databases over multiple data models [6, 11], specialized storage engines [14], and visualizations for Big Data [12]. We call our architecture a *polystore* to distinguish it from previous federation efforts that used only the relational model.

1.1 MIMIC II Application

The working group first converged on a representative use case to demonstrate the challenges inherent in applications that bring together the needs of many users and data sources. This use case is based on the real intensive-care unit (ICU) dataset, MIMIC II [13] (or "Multiparameter Intelligent Monitoring in Intensive Care II").

MIMIC II is a publicly accessible dataset covering about 26,000 ICU admissions at Boston's Beth Israel Deaconess Hospital. It contains waveform data (up to 125 Hz measurements from bedside devices), patient metadata (name, age, etc.), doctor's and nurse's notes (text), lab results, and prescriptions filled (both semi-structured data). In practice, a hospital would store all of the historical data, augmented by real-time feeds from current patients. Hence, this system must support a variety of data types, standard SQL analytics (e.g., how many patients were given a particular drug), complex analytics (e.g., compute the FFT of a patient's waveform data and then compare it to "normal"), text search (e.g., find patients who responded well to a particular drug or treatment), and real-time monitoring (e.g., detect abnormal heart rhythms).

BigDAWG stores MIMIC II in a mixture of backends, including Postgres, which stores the patient metadata, SciDB [5], which stores the historical waveform data in a time-series (array) database, S-Store [2], which stores a stream of device information, and Apache Accumulo, which stores the associated text data in a key-value store. In addition, we also include several novel storage engines, as outlined in Section 2.5. Each of these databases is good at part of the MIMIC II workload, but none perform well for all of it. Hence, this application is a good example of "one size does not fit all". To demonstrate this multi-faceted use case, we have built the following interfaces for user interaction with BigDAWG:

Browsing: This is a pan/zoom interface whereby a user may browse through the entire MIMIC II dataset, drilling down on demand to access more detailed information. This interface will efficiently display a top-level view (an icon for each group of the 26,000 patients) and flexibly enable users to probe the data at different levels of granularity. To provide interactive response times, this component, ScalaR, prefetches data in anticipation of user movements.

Exploratory Analysis Users can interactively explore interesting relationships in medical data using unguided data mining. Figure 2 is an example of running one of our two implementations. Here, the system draws the user's attention to an unusual relationship in their selected patient population between race and hospital stay duration. This population reverses the trend seen in the rest of the data. From this graph, the user has the opportunity to drill down into the source data to determine what prompts this correlation. This interface demonstrates the features discussed in Section 2.2.

Complex Analytics: This screen enables a non-programmer to run a variety of complex analytics – such as linear regression, fast fourier transforms, and principal component analysis – on patient data. It highlights the research challenges explored in Section 2.4.

Text Analysis: From this window a user can run complex keyword searches such as "find me the patients that have at least three doctor's report saying 'very sick' and are taking a particular drug".

Proceedings of the VLDB Endowment, Vol. 8, No. 12
Copyright 2015 VLDB Endowment 2150-8097/15/08.

These non-trivial queries showcase our novel facilities for querying over multiple data models discussed in Section 2.1.

Real-Time Monitoring: The real-time waveform data of current patients is stored in S-Store, a novel transactional stream processing engine. In this screen, we display this real-time patient data. In addition, we have a workflow that compares the incoming waveforms to reference ones, raising an alert when we identify significant differences between the two. This interface demonstrates the capabilities in Sections 2.3.

1.2 Guiding Tenets

While working on this demo, we have discovered several guiding principles for building the BigDAWG reference implementation.

One size does not fit all. It is clear that SQL analytics, real-time decision support, data warehouses, and complex analytics have the highest performance when each is performed by a specialized engine. Hence, applications with complex, heterogeneous source data and query workloads will likely rely on multiple storage engines. It is clearly undesirable to require an application programmer to learn multiple query languages or to have to recode his application when data moves from one storage engine to another. By the same token, it is not reasonable to fully replicate each dataset over all engines. Therefore, lightweight multi-database support is our top priority for BigDAWG.

Our approach should be contrasted with the Berkeley/BDAS/Spark implementation where various data analytics (e.g., SQL analytics, graph operations, machine learning) can be expressed over files stored in a distributed file system. Spark is generally optimized for analytics operations rather than low-latency transaction processing or streaming applications (Spark Streaming is not designed for subsecond latencies [1]). Hence, it does not cover the full range of use-cases we envision. Furthermore, users of Spark can choose the underlying data representation (e.g., Parquet files, RCFile, HDF5, etc.). As soon as users want to query across multiple representations they will need BigDawg-like features.

Real-time decision support is crucial. The Internet of Things promises an ever-increasing reliance on *streams* of data rather than batch loading which is the norm for OLAP. To realize the potential of this emerging platform, it is crucial to act on these feeds in real time. The aforementioned anomaly detection for heart waveforms, which may be used to predict a cardiac arrest, is one such application. Decision support over streams will call for real-time ingestion and analytics of this data. Here, data rates can be quite high (hundreds of Hz), and require response times in the tens of milliseconds. In Section 2.3, we describe a stream processing system with a new approach to coupling its feeds with historical data.

Database interfaces are becoming diverse. User interactions with Big Data applications are moving away from today's form-based interactions to a visualization focus [12, 9]. Historically, visualization systems load data into main memory to provide interactive responses to user's gestures. Due to the memory limitations of a single server, such "small vis" cannot survive in a Big Data stack. Our ScalaR browser with its "detail on demand" architecture is focused on scalability. In addition, we expect a major function of Big Data systems will be to perform exploratory data mining. Rather than querying for specific items, the user will say "tell me something interesting". We have two such data exploration systems in our reference implementation.

Complex analytics are a must-have. Historically, business analysts interacted with BI tools, which are a non-programmer interface to relational analytics (e.g., COUNT, SUM, MIN, MAX, AVG with an optional GROUP BY). Increasingly analysts rely on predictive models (based on machine learning, regression, statistics, etc.) in addition to SQL analytics.

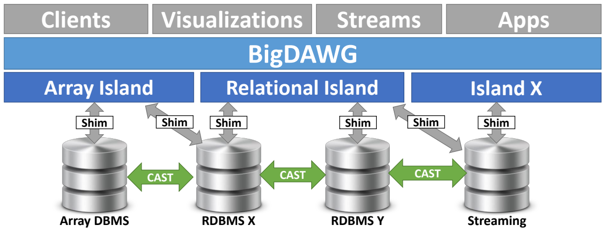

Figure 1: BigDAWG Architecture

We have addressed these tenets in the BigDAWG polystore. In Section 2, we explore our architecture. We first describe BigDAWG's primitives for federating multiple datastores including abstractions for user interactions and ones for shuffling data among its backends. After that, we outline our support for exploratory analytics. We then note how BigDAWG handles streaming data and complex analytics. Also considered are three experimental storage engines in our system with novel features. We conclude with a look at how our demo manages its workflow.

We leave for future work the extension of polystores to include data replication across systems and cross-system transactions.

2. BIGDAWG OVERVIEW

In this section, we highlight the main features of the BigDAWG demo: cross-database querying, exploratory analysis, real-time decision support for streams, and complex analytics.

2.1 Cross-DB Querying

With BigDAWG, it is our goal to enable users to enjoy the performance advantages of multiple vertically-integrated systems (such as column stores, NewSQL engines, and array stores) without sacrificing the expressiveness of their queries nor burdening the user with learning multiple front-end languages.

BigDAWG offers users *location transparency*, so that application programmers do not need to understand the details about the underlying database(s) that will execute their queries. We have implemented this construct using *islands of information*. Each island is a front-facing abstraction for the user, and it includes a query language, data model, and a set of connectors or *shims* for interacting with the underlying storage engines that it is federating.

Since it is unlikely that a single island will offer the full functionality of all of the federation's database engines, our framework is designed for multi-island operations. Likewise, a storage engine is not restricted to a single island. For example, in Figure 1 we show BigDAWG supporting multiple islands, with several underlying engines that each are accessible via one or more islands.

Each island provides the intersection of the capabilities provided by its underlying storage engines. Hence, it is easy for users to express their queries in a single query language (for example, SQL) over multiple data stores. On the other hand, users do not want to lose any of the functionality their databases provided before federation. To provide the union of the capabilities of the federator's underlying storage engines, BigDAWG offers *degenerate islands*. These islands have the full functionality of a single storage engine.

When a query cannot be supported by a single island, the user will express his specification using any number of island languages and BigDAWG will move data and/or intermediate results from one storage engine to another as needed. To specify the island for which a subquery is intended, the user indicates a *SCOPE* specification. A cross-island query will have multiple scopes to indicate the desired semantics of its parts. BigDAWG also relies on a *CAST* operator to move data between engines. For example a user may issue a relational query on an array A via the query:

```
RELATIONAL(SELECT * FROM CAST(A, relation)
  WHERE v > 5)
```

In our reference implementation, we have a total of eight islands, as will be noted below. Obviously, we should make island query languages as expressive as possible. Hence, we are investigating higher-level declarative systems that do not require users to manually write plans with SCOPE and CAST operations. Specifically, when multiple islands implement common functionality with the same semantics, then BigDAWG can decide which island will do the processing automatically. To identify such common subislands, we are constructing a testing system that will probe islands looking for areas of common semantics.

Moreover, we are investigating techniques to make cross-database CASTS more efficient than file-based import/export. For maximum performance, each system needs an access method that knows how to read binary data in parallel directly from another engine.

Lastly, we are investigating cross-system monitoring that will migrate data objects between storage engines as query workloads change. We are building a monitoring system that will re-execute portions of a query workload on multiple engines, learning which engines excel at which types of queries. For example, if the majority of the queries accessing MIMIC II's waveforms use linear algebra, this data would naturally be migrated to an array store.

2.1.1 Multi-system Islands

In BigDAWG, we implemented two cross-system islands: D4M [10] and Myria [7]. Each offers a different interface to overlapping set of backend engines. Myria has adopted a programming model of relational algebra extended with iteration. Among other engines, it includes shims to SciDB and Postgres. Myria includes a sophisticated optimizer to efficiently process its query language.

On the other hand, D4M uses a new data model, associative arrays, as an access mechanism for existing data stores. This data model unifies multiple storage abstractions, including spreadsheets, matrices, and graphs. D4M has a query language that includes filtering, subsetting, and linear algebra operations, and it contains shims to Accumulo, SciDB and Postgres.

BigDAWG currently consists of the above scope-cast facility with island implementations from Myria and D4M, along with degenerate islands for three production databases (Accumulo, SciDB, and Postgres). In addition, we also incorporate access to the experimental database systems noted in Section 2.5.

2.2 Exploratory Analysis

A trend we have observed is that users are shifting away from SQL-based querying and toward exploratory analysis. Here, a data scientist looks for interesting distributions or relationships in the data using visual analytics. Often this analysis occurs in a very high-dimensional space where a manual and exhaustive exploration is not feasible. Instead, one needs an automated system to mine for interesting patterns, and we have built two of them in the BigDAWG project. Both sit at the top of the BigDAWG stack as visualizations, as shown in Figure 1.

The first system, **SeeDB**, computes SQL aggregates with a GROUP BY clause over the search space of all possible combinations of attributes. To provide reasonable response times over massive datasets, SeeDB uses sampling and pruning to identify a candidate set of visualizations that are then computed over the full dataset. This visualization generator identifies interesting graphs using a variety of metrics, the foremost of which is a deviation-based utility, i.e., it selects visualizations that show users unusual or interesting aspects of their query results. The user can further explore these visualizations through an interactive tool or ask the system to "find me more like this one." In our demo, we use SeeDB to explore patient attribute data. Figure 2 shows a sample SeeDB-generated graph.

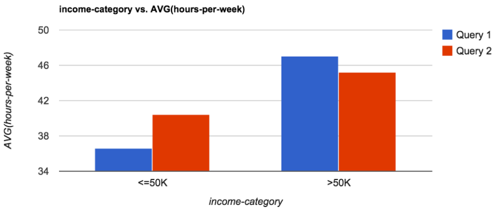

Figure 2: Sample visualization from SeeDB

Searchlight is BigDAWG's second data exploration system; it looks for interesting results using constraint programming (CP). Searchlight enables data- and search-intensive applications by uniquely integrating the ability of DBMSs to store and query data at scale paired with the rich expressiveness and efficiency of modern CP solvers. It is built on the BigDAWG API and can invoke third-party solver logic as User-Defined-Operators that directly participate in query execution plans. Searchlight first speculatively searches for solutions in main-memory over synopsis structures and then validates the candidate results efficiently on the actual data.

2.3 Real-Time Operations

An increasing amount of data will be generated by the Internet of Things. These streams of information come from smart devices, sensors, wearable computing devices, and other similar components. Applications, like MIMIC II's use case, will rely on information from these devices to make real-time decisions. Queries rely on well-known streaming primitives, including filter, join, aggregate, and window. These primitives process records as they stream through an engine at an extremely high rate. Traditional database systems lack the ability to handle the high insert rates intrinsic to streams. As a result, data of this kind is often processed by a stream processing engine, such as NiagaraCQ [4] or TelegraphCQ [3].

In our opinion a streaming engines must be tightly coupled with other backends to support real-time decision making that utilizes both real-time and historical data. For example, a doctor monitoring high-risk patients may set a trigger on a windowed aggregate from a heart monitor for patients who are classified as high-risk based on active prescriptions and a calculated FFT on historical heart rate activity. We have built a novel stream processing engine that is integrated into the BigDAWG system that does the real time component of this task, as will be noted below.

2.4 Complex Analytics

As noted in the introduction, we expect simple (SQL) analytics to give way to more complex predictive models. Such models, for example, might predict whether a patient is likely to suffer a cardiac arrest, so that corrective measures can be taken while their heart is still beating. Many different algorithms are used for predictive modeling. The vast majority, however, are based on linear algebra and often use recursion. These include regression analysis, singular value decomposition, eigenanalysis (e.g, power iterations), k-means clustering, and graph analytics. Most of these algorithms have an inner loop containing a small number of matrix operations (e.g. multiply). Moreover, linear algebra packages, such as BLAS and ScaLAPACK, have been tuned over the years by software and hardware experts and offer very high performance. As such, the prevailing wisdom is to support complex analytics using an array

DBMS, such as SciDB, coupled to a linear algebra package.

Unfortunately, this approach has a big problem. The DBMS and the linear algebra package differ on the size of computation tiles, the choice of networks, compression techniques, etc. Hence, the two systems must be loosely coupled and it is expensive to convert data back and forth between their respective formats. In addition, ScaLAPACK is optimized for dense matrices and the majority of the use cases we see require sparse techniques. As a result we have embarked on a research project to tightly couple a next generation sparse linear algebra package to TileDB, described below.

2.5 Prototype Engines

As BigDAWG adheres to the "one size does not fit all" ethos, exploring the development and integration of new specialized engines is an important component of the BigDAWG project. In this section we discuss cutting-edge engines that have been integrated into the BigDAWG framework.

Tupleware offers a Map-Reduce style interface to applications. It compiles functions aggressively, using advanced compiler techniques, into distributed programs to avoid any unnecessary runtime overhead. Furthermore, by taking statistics about UDFs (e.g., predicted number of CPU cycles), the hardware and the data into account, Tupleware offers low-level optimizations that neither a traditional query optimizer nor a compiler can perform alone. As a result, this system is nearly two orders of magnitude faster than the standard Hadoop codeline, and dramatically outperforms Spark.

Our second engine is **TileDB**, a prototype implementation of an array store. TileDB defines a fundamental unit of storage and computation, intuitively called a *tile*. A tile is an irregular subarray that can be optimized for dense or sparse objects. TileDB was built from scratch using simple, modular C++ infrastructure, and is designed to facilitate the exploration of novel ideas in array databases.

Our final engine is **S-Store** [2], a system for processing high-velocity streams and transactions with scalable performance and well-defined correctness guarantees. Built on top of H-Store [8], it inherits the NewSQL engine's scalable infrastructure for ACID transactions and high-throughput processing, but extends it with: (i) streams and sliding windows represented as time-varying tables, (ii) an ingestion module for absorbing data feeds directly from a TCP/IP connection, and (iii) a lightweight recovery scheme.

3. MIMIC II DEMO ARCHITECTURE

We now briefly touch on the workings of the BigDAWG demonstration. Users will start from any one of the five interfaces described in Section 1. They can pose queries over our polystore using the cross-system islands, Myria and D4M, or via SCOPE/CAST operations over combinations of degenerate islands.

The interfaces will highlight interactive data browsing, guided exploratory analysis that mines interesting patterns in the data set, real-time monitoring of streaming data, and the ability to run complex and text analytics over a collection of storage engines using BigDAWG's *SCOPE* and *CAST* multi-database operators.

To demonstrate the multi-database support in BigDAWG, our demo partitions the MIMIC II dataset across the various engines described above. For example, waveform data from MIMIC II patients will enter BigDAWG through S-Store, with real-time processing and modification provided by stored procedures. Ultimately, the data ages out of S-Store and is loaded into SciDB, for historical analysis. Since all of the streaming data persists in either S-Store or the array engine, the real-time monitoring and complex analytics on waveform data will use cross-system query support to obtain a complete picture of a patient. Other demo interfaces also highlight multi-database support, including the browsing interface querying data in Postgres, SciDB, and S-Store and the complex analytic interface that will query data stored in SciDB or TileDB.

4. CONCLUSIONS

The BigDAWG demonstration shows the need for and benefit from a tightly coupled polystore system. The interactive use cases will give conference-goers the opportunity to explore and analyze data stored across a variety of mature and prototype database systems. In addition, we expect our architecture to outperform a "one size fits all" system by one-to-two orders of magnitude.

5. REFERENCES

[1] Spark Streaming. `https://spark.apache.org/docs/0.9.0/streaming-programming-guide.html#setting-the-right-batch-size`.

[2] U. Cetintemel, J. Du, T. Kraska, S. Madden, D. Maier, J. Meehan, A. Pavlo, M. Stonebraker, E. Sutherland, and N. Tatbul. S-Store: A Streaming NewSQL System for Big Velocity Applications. *PVLDB*, 7(13), 2014.

[3] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah. TelegraphCQ: continuous dataflow processing. In *SIGMOD*, pages 668–668, 2003.

[4] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continuous query system for internet databases. In *SIGMOD Record*, volume 29, pages 379–390, 2000.

[5] P. Cudré-Mauroux, H. Kimura, K.-T. Lim, J. Rogers, R. Simakov, E. Soroush, P. Velikhov, D. Wang, M. Balazinska, J. Becla, et al. A Demonstration of SciDB: A Science-Oriented DBMS. *PVLDB*, 2(2):1534–1537, 2009.

[6] M. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *Sigmod Record*, 34(4):27–33, 2005.

[7] D. Halperin, V. Teixeira de Almeida, L. L. Choo, S. Chu, P. Koutris, D. Moritz, J. Ortiz, V. Ruamviboonsuk, J. Wang, A. Whitaker, et al. Demonstration of the Myria big data management service. In *SIGMOD*. ACM, 2014.

[8] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. B. Zdonik, E. P. C. Jones, S. Madden, M. Stonebraker, Y. Zhang, J. Hugg, and D. J. Abadi. H-store: A high-performance, distributed main memory transaction processing system. In *PVLDB*, volume 1, 2008.

[9] S. Kandel, A. Paepcke, J. M. Hellerstein, and J. Heer. Enterprise data analysis and visualization: An interview study. *Visualization and Computer Graphics, IEEE Transactions on*, 18(12):2917–2926, 2012.

[10] J. Kepner, W. Arcand, W. Bergeron, N. Bliss, R. Bond, C. Byun, G. Condon, K. Gregson, M. Hubbell, and J. Kurz. Dynamic distributed dimensional data model (d4m) database and computation system. In *ICASSP*. IEEE, 2012.

[11] H. Lim, Y. Han, and S. Babu. How to fit when no one size fits. In *CIDR*, volume 4, page 35, 2013.

[12] A. Nandi and H. Jagadish. Guided interaction: Rethinking the query-result paradigm. *PVLDB*, 4(12):1466–1469, 2011.

[13] M. Saeed, M. Villarroel, A. T. Reisner, G. Clifford, L.-W. Lehman, G. Moody, T. Heldt, T. H. Kyaw, B. Moody, and R. G. Mark. Multiparameter Intelligent Monitoring in Intensive Care II (MIMIC-II): A public-access intensive care unit database. *Critical Care Medicine*, 39:952–960, 2011.

[14] M. Stonebraker and U. Cetintemel. "One Size Fits All": An Idea Whose time has come and gone. In *ICDE*, pages 2–11, 2005.