# An Integrated Architecture for Real-Time and Historical Analytics in Financial Services

Lyublena Antova, Rhonda Baldwin, Zhongxian Gu, and F. Michael Waas

Datometry Inc.
795 Folsom St Ste 100
San Francisco, CA 94107, USA

{lyublena,rhonda,zgu,mike}@datometry.com

**Abstract.** The integration of historical data has become one of the most pressing issues for the financial services industry: trading floors rely on real-time analytics of ticker data with very strong emphasis on speed, not scale, yet, a large number of critical tasks, including daily reporting and back-checking of models, simultaneously put emphasis on scale. As a result, implementers continuously face the challenge of having to meet contradicting requirements and either scale real-time analytics technology at considerable cost, or deploy separate stacks for different tasks and keep them synchronized—a solution that is no less costly.

In this paper, we propose *Adaptive Data Virtualization* as an alternative approach to overcome this problem. ADV lets applications use different data management technologies without the need for database migrations or re-configuration of applications. We review the incumbent technology and compare it with the recent crop of MPP databases and draw up a strategy that, using ADV, lets enterprises use the right tool for the right job flexibly. We conclude the paper summarizing our initial experience working with customers in the field and outline an agenda for future research.

## 1 Introduction

The value of Big Data has been recognized universally in recent years but truly integrating Big Data technology with existing IT infrastructure and business processes has often remained an elusive goal. In particular, in the financial services industry, the combination of real-time data processing and historical data analysis poses a number of unsolved challenges, see e.g., [1].

Financial institutions have developed sophisticated systems that consume massive amounts of trade-and-quote (TAQ) data at a rapid pace and make split-second decisions in what is called real-time or high frequency trading. In-memory technology, often proprietary and highly specialized, has proven its mettle in this space and is widely accepted as the standard methodology, see e.g., [6, 7, 3].

However, technology developed for low latency data management cannot be extended easily to tackle Big Data challenges and institutions found themselves

facing the uneasy choice between one of the following approaches (1) use a single architecture, built on in-memory principles and attempt to scale it, or, (2) use a Big Data stack, e.g., MPP databases, in addition to in-memory systems and duplicate all application logic to work with both architectures. The former entails significant hardware expenses and complexity, the latter poses all sorts of deployment and data management issues and, on top of this, is highly error-prone. In short, both approaches are highly impractical, resource intensive, and financially unattractive. What institutions are really looking for is the ability to access both, real-time and historical data with exactly the same tools and exactly the same applications. And not only exactly the same tools but the same tools they have developed over the past decades and are using now.

Even though the urgency of the problem has accelerated recently, not last because of regulatory and security concerns, vendors have been trying for quite some time already to address the chasm between low-latency and scalable processing. Unfortunately, these approaches focused either on making low-latency in-memory systems scale, or, making scalable, distributed systems respond faster [4, 2]. And while the results are respectable in their own right, they continue to fall short of a workable solution, and reconciling the different processing paradigms has long been considered the *Holy Grail* of data processing [1].

In this paper, we devise a research agenda for a radically different approach which we call *Adaptive Data Virtualization (ADV)*. ADV abstracts and impersonates systems: instead of pursuing a "one size fits all approach", ADV enables users to access different data management technologies using the same query language and the same APIs. This way, users can leverage any data management technology—be it in-memory or MPP database systems—within the same application and without interruption of business processes. ADV eliminates the need for costly database migrations and lets business users tap into a variety of data sources, e.g., in-memory, MPP databases, NoSQL systems etc., *without* having to change or re-configure their applications.

Naturally, a complete ADV solution takes considerable effort to build and has yet to reach commercial viability. In this paper, we describe our experience in investigating the space and present learnings regarding use cases, technical difficulties, and desiderata for such a platform. We describe our first experiences with ADV, the challenges we anticipate, and opportunities for future research.

*Roadmap.* The remainder of this paper is organized as follows. In Section 2, we briefly survey the relevant basics of data processing in financial services and present current alternatives on how to address the historical data challenge in Section 3. In Section 4, we present the concept and benefits of Adaptive Data Virtualization including a catalog of desiderata for successful implementation and adoption.

## 2 Background

Financial applications, such as *High-Frequency Trading (HFT)*, utilize a specialized environment and proprietary algorithms to profit from the trade of stocks

at scale. The HFT world is highly competitive and those that have the fastest access to information and the most sophisticated algorithms are at an advantage. In this section, we describe the general building blocks required by financial applications and introduce the problem of integrating real-time and historical data in financial services.

### 2.1 Market Data

Market data is the time-series data produced by an entity like a stock exchange. For specification and samples, we refer the reader to the New York Stock Exchange Trades and Quotes (NYSE's TAQ) dataset [5]. Market data is used in real-time to make time-sensitive decisions about buying and selling and is also used for historical data applications like predictive modeling, e.g., projecting pricing trends and calculating market risk, and backtesting of trading strategies.

### 2.2 Environment Building Blocks

A typical trade processing environment is illustrated in figure 1.

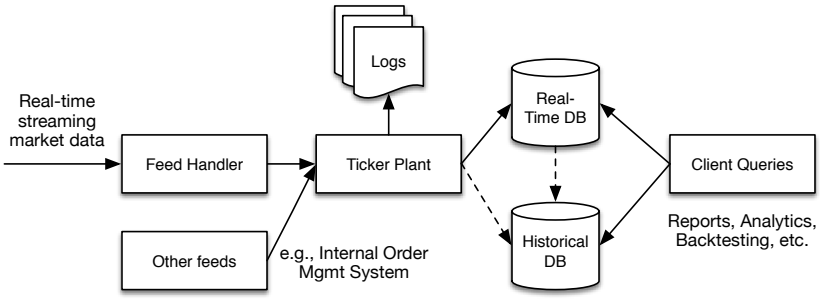


**Fig. 1.** Typical trade order processing infrastructure

The *feed handler* processes data as it streams in live from an exchange or other data provider. Processing in the feed handler generally includes simple updates to the data that are required by the ticker plant, e.g., data type conversion, formatting, removing unnecessary information, etc.

Upon receiving data from the feed handler and other internal feeds, e.g., from an Order Management System, the *ticker plant* typically writes new data to a log file and publishes relevant data to subscribed processes. While processes vary depending on environment and applications, at least one process writes to a database. Clients query the database(s) for reports, analytics, backtesting, etc.

### 2.3 Real-time and Historical Databases

The *real-time database* stores the current day's data in memory to avoid the latency of fetching data from disk. Periodically throughout the day, or in some

cases at the end of day, the data is either discarded or saved on disk, freeing up memory to store the latest data from the ticker plant.

In addition to a real-time database, newer environments also include a *historical database* to store 5-7 years worth of data for analytic workloads. In the not-so-distant past, market data was thrown away at the end of the day, after the market closed, but now market data is stored for much longer. Typically data comes directly from the ticker plant or is exported from the real-time database.

### 2.4 Integration Problem

The need to integrate real-time and historical data is prevalent in every financial services firm. For example, in an industry where a financial model can make (or cost) the firm millions of dollars, much attention is paid to backtesting: the testing of predictive models using historical data. Given a trading strategy, backtesting predicts the performance of a strategy as if it had been employed in the past. In the real world, financial models are very sophisticated, the quantitative researchers who build these models have access to multiple years worth of historical data, and those that backtest and course-correct the fastest have a huge competitive advantage. In other words, the firm that can seamlessly integrate their real-time and historical data, sets itself up for success. In the next section, we consider current approaches to the integration problem.

## 3 Current Approaches

In practice, we see several solution patterns emerging that attempt to solve the integration problem described in the previous section, including the addition of more resources to real-time databases, periodic export of data from real-time to historical databases, and migration to an entirely new technology stack. In this section, we discuss the pros and cons for each of these approaches.

### 3.1 Scaling In-memory Technology

Scaling an existing in-memory analytics engine by adding memory and CPU cores allows more than the current day's data to fit in memory for analyses, which may be sufficient for a given real-time/historical data integration scenario. This solution seems simple and straight-forward at first, however, it comes with a number of severe drawbacks:

Most importantly this approach quickly becomes prohibitively expensive, as the hardware costs associated with expanding real-time systems easily exceed more than 10x the cost of conventional RDBMS technology on commodity hardware. Another critical drawback of the scaling approach is the IT effort required to manage these systems when sharding of data across multiple instances is needed. In this case, a whole slew of highly challenging distributed systems problems need to be addressed by IT to guarantee flawless operation.

Finally, since real-time database systems were built for a very different use case than analytics on historical data, they often lack the compliance, management utilities, and tools required by enterprises when storing massive volumes of historical data, which RDBMS and BI vendors have taken 20-30 years to perfect.

### 3.2 Multi-stack Environments

Many real-time databases provide export functionality to "integrate" with conventional or MPP database systems.

Freeing up expensive memory to focus only on real-time analytics and using less-expensive data stores for historical data is a huge cost-saver. Many financial firms are considering options like moving archival data to Hadoop and historical data that will be used heavily for analytics to a shared-nothing MPP database. MPP databases have impressive performance gains over traditional databases and continue to provide major innovations, such as Pivotal Greenplum Database's recent advances in query optimization [8].

One of the big detractors to this approach is the additional intermediate steps required to aggregate result sets from different interfaces for real-time and historical databases. Also, since the interfaces and technology stacks are different, the people and skills required to manage and use both stacks are very different. This is a limiting factor in the case of back-testing financial models, for example, where a "quant"/statistician may write the real-time model in a language that he/she is familiar and not have the experience or desire to use a separate interface for back-testing. Moreover, it may a substantial amount of non-trivial work to translate a model or query written for one database to work on another.

In financial institutions, a single source of truth is especially critical for compliance; another concern is the risk associated with exporting data. Movement of data introduces risk of loss or unavailability and the existence of multiple data copies introduces risk of mis-management.

### 3.3 Migration to Latest Technology

Starting over from scratch and building out a new environment could be the answer in some situations. Certainly the opportunity to retire old infrastructure to bring in new technology, may be advantageous for financial institutions to keep up with the pace of innovation and competitors, especially new fin-tech startups.

Unfortunately it is difficult to migrate from one environment to another. It causes huge disruptions to the business over extended periods of time. While moving data between database systems is relatively easy, migration of applications can take up to one or more years in practice, depending on the number of applications that need to be migrated.

Given institutions' investment of 20+ years in the incumbent technology, migrating is highly unattractive from a cost perspective. Not last, a migration

is a big bet and runs the risk to stunt innovation over extended periods of time with uncertain outcome. The next innovation cycle may make the newly adopted systems obsolete in only a few years.

## 4 Adaptive Data Virtualization

*Adaptive Data Virtualization (ADV)* is the concept of abstracting a data management system in a way that, simply put, lets a native application written for system X run on system Y. It eliminates the problems detailed above and enable business users to access data across the enterprise readily. Using ADV, IT gets to choose the appropriate backend database based on requirements such as scalability, performance, or cost etc., but, importantly, independent of the choice of application business users make. Figure 2 highlights the basic components and their interplay.

We propose ADV as the natural extension of *data virtualization* in general. DV is loosely defined as the concept of making different data sources accessible to a central component. However, DV does not *adapt* to the existing applications and therefore requires business users to migrate to new APIs and query language—this time the APIs and query language of the DV system, not of a specific back-end database. Commercial DV products are available and have seen uptake in the industry wherever data federation is needed, i.e., the combining of data from different data sources, see for example [9].

### 4.1 Approach

The idea to ADV arose from our work with customers and frequently recurring requests by customers wanting to access data or migrate applications to different databases to meet additional requirements such as scalability, availability or simply reduce cost. However, migrations rank among the most costly and disruptive IT operations imaginable. In particular, having witnessed countless migrations in the data warehousing space, we are keenly aware of where most of the effort in a migration needs to be placed.

We set out to accomplish the vision of ADV by reviewing and investigating the technical underpinnings needed for this undertaking. The following is a list of desiderata and an architectural blueprint for an implementation:

**Platform Architecture.** One of the most important requirements from a practitioner's point of view is that neither the application nor the backend databases need to be re-configured. This prohibits not only the recompiling of applications but also re-configurations such as loading different database connectors, including ODBC/JDBC drivers. Hence, ADV needs to be implemented as an *independent platform* completely transparent to the applications and the underlying database systems (see Figure 2).

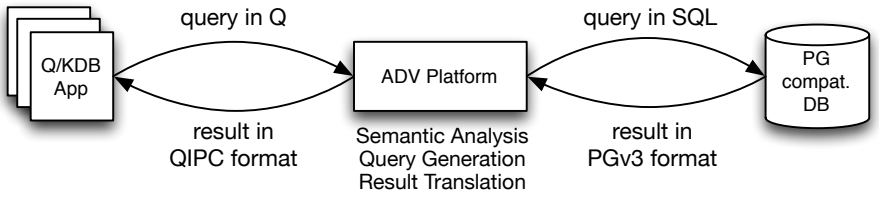


**Fig. 2.** Architecture Overview (example Q/KDB+ to PostgreSQL compatible DBs)

**Performance.** To be successful, an ADV platform must provide acceptable performance. We deliberately use the term *acceptable* instead of, say, comparable, since being a proxy, ADV naturally adds latency. However, overall performance is always the result of a complex interplay of several factors: for example, being able to use an MPP database instead of an SMP implementation did actually reduce query time for our customers for a significant number of queries in their workloads.

**Expressivity.** Enabling an application to run on any other database than the one it was written for requires a sufficiently complete implementation of all critical query and command language elements. In preparations for our undertaking we classified workloads that we obtained from customers by the feature surface they are using. Not surprisingly a non-trivial number of features in the source system are virtually unused. In contrast, the lion share of queries in a workload use only a astonishingly small portion of the available features.

**Security.** For practical viability, ADV has to be able to integrate with security and other IT services. We found this usually requires a deep integration with the wire protocols of source and target systems in addition to a language integration.

The above is by no means a complete list, however, in our research we found these to be the most pressing requirements as emphasized by prospects and customers.

### 4.2 State of Implementation

In this paper we report on ongoing research. A first implementation of a complete ADV system is currently underway at Datometry. In its present state, the system supports applications written using Q and KDB+, the market leader and most prevalent real-time analytics engine in the financial services industry (see e.g., [3]) and backend databases of the PostgreSQL family, including PostgreSQL 9.x [12], AWS Redshift [10], IBM Netezza [11], Pivotal Greenplum Database and HAWQ [8], etc.

Our system accepts connection request in Q's wire-protocol QIPC, parses query and command language requests, represents them in a more powerful

extensible algebra before converting them into one or more request in SQL for the target system and encodes it using the PGv3 wire protocol. Results retrieved from the database are translated back into the QIPC data format.

The product is currently in private beta with well-known customers in the financial services industry and initial feedback has been overwhelmingly positive. We expect to be able to report on initial field experience with the system in production soon.

## 5 Summary

In this paper we presented the concept of *Adaptive Data Virtualization*, a novel type of data virtualization that enables applications to run on alternative database systems. ADV is the result of our experience of over 15 years in implementing commercial database systems and the realization that breaking open data silos requires an approach that enables applications to move freely between different data management systems without the painful burden of re-compiling or re-configuring them.

We reported on ongoing work, including the primary requirements for the success of this type of platform. However, the initial success of the platform we created, indicates a strong interest in the market for this type of technology—not only in the financial services industry.

We firmly believe ADV is the starting point of fundamental transformation of the database industry, moving away from an market place in which databases dictate one of the strongest vendor lock-ins in the IT industry, to an open environment where databases become an interchangeable commodity.

## References

[1] Garland, S.: Big Data Analytics: Tackling the Historical Data Challenge. Wired Magazine, Innovation Insights, October 2014
[2] SAP HANA. May 2015. http://hana.sap.com/abouthana.html
[3] Kx Systems. May 2015. http://kx.com
[4] MemSQL. May 2015. http://www.memsql.com
[5] New York Stock Exchange: Market Data, Data Products—Daily TAQ. May 2015. http://www.nyxdata.com/Data-Products/Daily-TAQ
[6] Security Technology Analysis Center: STAC Benchmark Council—Various Benchmark Results. May 2015. http://stacresearch.com/research
[7] Shasha, D.: KDB+ Database and Language Primer. Kx Systems, May 2005. http://kx.com/q/d/primer.htm
[8] Soliman, M. et al.: Orca: A Modular Query Optimizer Architecture for Big Data. ACM SIGMOD Conference, May 2014
[9] Informatica. May 2015. http://www.informatica.com
[10] Amazon Redshift. May 2015. http://aws.amazon.com/redshift
[11] IBM Netezza. May 2015. www.ibm.com/software/data/netezza
[12] PostgreSQL. May 2015. http://www.postgresql.org