WHITE PAPER

Streaming Data and the Fast Data Stack

Software designers and architects build software stack reference architectures to solve common, repeating problems. A great example is the LAMP stack, which provides a framework for building web applications. Each tier of the LAMP stack has viable substitutes that might be chosen simply by preference or perhaps for technical reasons. Additional layers or complementing tools can be mixed in to address specific needs. Much information can be conveyed succinctly to others familiar with the pattern by explaining the resulting architecture in the context of the generalized LAMP pattern.

Big Data is data at rest; Fast Data is streaming data, data in motion. A stack is emerging across both verticals and industries alike for building applications that process these high velocity streams of data that quickly accumulate into the 'Big Data lake.' This new stack, the Fast Data Stack, has a unique purpose: to grab real-time data and output recommendations, decisions and analyses in milliseconds. Over the next several years this emerging Fast Data Stack will gain prominence and serve as a starting point for developers writing applications for streaming data.

As with any emerging reference software stack, it's important to step up a layer and consider the workflow that motivates the Fast Data Stack's requirements. The requirements for a desktop application stack (a graphical UI, likely an MVC-style layering, a configuration database, undo capability…) are well understood, as are the generally accepted user activities that guide the design of a web application. In fact these are so well understood that developers have internalized the knowledge and forgotten that it was once new!

VOLTDB

When considering a Fast Data stack, we have to understand that few developers have created multiple large-scale data applications. Certainly very few have developed more than a handful. The developer community has not yet internalized the usage patterns and general requirements for applications that process high speed inputs from sensors, M2M devices, or IoT platforms. With massive streams of data coming into the enterprise

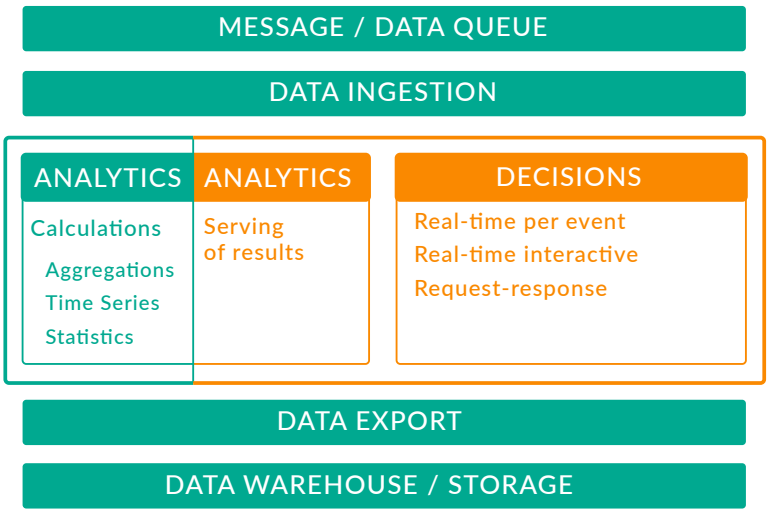
and into applications – from hundreds to millions of sources, in different formats (e.g., structured and unstructured), agreement on definition of the Fast Data Stack will benefit all. This paper will look at the emerging Fast Data Stack through the lens of streaming data to provide architects, CTOs and developers with fundamental architectural elements of the new

Fast Data Stack: a LAMP stack for streaming data applications.

A close look at the Fast Data Stack

A Fast Data stack should Ingest, Analyze, Decide, and Store. This emerging stack, shown below, addresses the core data (message) requirements of streaming apps: ingestion, analysis, decisions, and storage. Variations on the basic stack can be used for different use cases.

Fast Data Stack



Ingestion

Ingestion is the first stage in the Fast Data Stack (N.B.: some would argue the ingestion stage is similar to the Lambda Architecture). The job of ingestion is to interface to the streaming data sources, and to accept and transform or normalize incoming data. Ingestion marks the first point at which data can be transacted against, applying key functions and processes to produce value from the data – value that includes insight, intelligence, and action.

Developers have two choices for ingestion. The first is to use "direct ingestion," where a straight-through code module can hook directly into the data-generating API, capturing the entire stream at the speed that the API and the network will run, e.g., at "wire speed". In this case the analytic/ decision engines will have a direct ingestion "adapter". With some amount of coding the analytic/ decision engines can handle streams of data from an API pipeline without the need to stage or cache any of the data on disk.

If access to the data generating API is not available, usually the alternative is that the data is being served up in a Message Queue. In this case, an ingestion system is needed to process incoming data off the queue. Modern queuing systems handle partitioning, replication, and ordering of data, and can manage backpressure from slower downstream components.

Steaming analytics and real time decisions

As data is ingested, it is used by one or more analytic and decision engines to accomplish specific tasks on streaming data. At this point the data is 'Fast Data,' and the challenge for the decision engines is to keep pace with the velocity of the data stream.

Real-time decisions are used to influence the next step of processing. Real-time decision engines are doing a lot of work, in that they consume the velocity of the data stream and at the same time are processing complex logic, all in time to complete the real-time decision feedback loop. Real-time decisions are made possible by a Fast Data Stack that features:

- Continuous query mode
- OLTP anayltics mode
- Programmatic Request-Response
- Stored procedures
- Export to longer-term analytics/storage

Streaming analytics need to consume high-velocity data while maintaining real-time analytics, in the form of counters, aggregations and leaderboards. Streaming analytics require a Fast Data Stack with the following attributes:

Data export

Once Fast Data analytics are completed, the data moves through the pipeline for later processing; data ingestion and export flow at the same rate. Usually streaming analytics rely on an ingestion queue. Similar queues are used for the export stage. For real-time decisions, which process Fast Data in a continuous query mode, an Export function is needed to transform and distribute data to the Big Data warehouse/storage (OLAP) engine(s) of choice.

Data store and historical analytics

OLAP engines enable fast queries against raw (structured) data. They append streaming data to historical data and store it for later, further Big Data analysis on the entire data set, reducing time-to-reporting and enabling historical BI. Many OLAP systems use columnar compression to store large amounts of data in memory and to increase query speed. OLAP data stores and historical analytics have the following capabilities:

- Periodic query
- OLAP/data warehouse model
- BI, Reports
- Dashboards
- User interaction

Streaming data and its demands

Let's return to a discussion of streaming data, and review its demands on application developers.

Data arriving continuously, and in massive quantity, is called a "stream". Data in a stream may be in many data types and formats. Most often the data provides information about the process that generated it; this information may be called messages or events. This includes data from new sources, such as sensor data, as well as clickstreams from Web servers, machine data, and data from devices, events, transactions, and customer interactions.

Enterprises adopting Big Data strategies quickly realize that Big Data is created by Fast Data, e.g. high-speed streams of incoming data. High counts of small messages or events add up to massive amounts of streaming data. And the types of data streaming into the enterprise are increasing, seemingly daily. It has become a significant challenge for enterprises not only to ingest these relentless feeds of data, but also to capture value from them.

Stream processing is not new

Complex Event Processing (CEP) was an early form of stream processing. Time windows, state changes (events), and simple analysis techniques of correlation or classification were used to synthesize useful information in near real-time from the stream. By limiting the type of analysis that could be performed on a stream in real-time, the system could extract certain patterns or events. These techniques proved to be very useful (e.g., for programmatic stock trading), in financial fraud detection (money laundering or credit card fraud), manufacturing control, and military Communication, Command, Control, and Information (C3I) systems.

Domain-specific forms of stream processing also were introduced. In the network security field, stream processing of network traffic proved to be an effective way to implement Intrusion Detection Systems/Intrusion Prevention Systems (IDS/IPS). Byte-patterns of known malware could be detected in-transit. To run fast enough, these systems were implemented in specialized hardware.

Both of these examples were ultimately limited by processing speed.

Stream processing finds new life in the Cloud

Cloud-based applications began generating demand for stream processing due to scale. Large web sites serving millions of subscribers, and Cloud services for stocks, travel, marketplaces, and financial services, generated massive streams of data about user behavior, product sales, search terms, click streams, and much more. This formed the “demand side” for stream processing.

The same architecture that enabled applications to create massive streams of data also allowed designers to create high-capacity stream processing systems. The underlying architecture of computing clusters enabling parallel execution, and large pools of compute and memory, were among the capabilities needed to build these. New stream processing solutions were implemented, taking advantage of the underlying commodity scale-out architecture of the cloud. The clustered, scale-out architecture opened the door for all-software, high-performance implementation of stream processing. This formed the “supply side” for stream processing.

This newfound ability to process streaming data with commodity resources (and inexpensively) has enabled new types of applications: applications that can process both Fast and Big Data. With the arrival of this new class of applications comes the definition of a Fast Data Stack.

Fast Data (streaming) use cases

Real-time trading decisions

In this example use case, quote streams are copied to the Fast Data Stack input queue. A real-time decision engine combines client, regulatory, pricing, fees, if and where to trade, and other data sources which the business requires; calculates a decision; and reports this decision up to the execution systems. In this use case, there is no historical data processing; all effort is aimed at Fast Data.

Ad serving: real-time decisions and historical analysis

In this example use case, navigation and user information is sent to the Fast Data Stack via a direct API. The real-time decision engine combines data about recent ads served and combines this with other data. It calculates a decision on which ad to serve and reports this decision up to the ad placement systems. Additionally, the ad service result is exported to a data warehouse, where it is merged with longer-term historical ad service data and made ready for batch historical analysis. This use case includes both Fast Data processing and historical Big Data processing.

Smart meter sensing, predicting, and reporting

In this use case, meter streams are copied to the Fast Data Stack input queue and accessed by the real-time decision engine. The real time decision engine immediately applies a predictive model and alerts the supply grid of immediate demand forecasting. The streaming analytics engine also accesses the data and calculates and records values every 15 minutes from the Smart Meter, alerting on any exceptions (e.g., power failures). All data is saved for billing and historical analysis. This use case features Fast Data processing and historical Big Data processing.

Conclusion

The growth of streaming data, as well as the capability of new distributed systems architectures to process that data in real time, has spurred the development of new applications that can benefit from a new technology stack, the Fast Data Stack. These new applications process real-time data and output recommendations, decisions and analyses in milliseconds. When we look at the requirements of these applications, we find in most cases we are looking at an OLTP problem. OLTP problems are often solved with databases, but streaming problems may seem ill-suited to such an approach because of the limitations of traditional database technologies.

Because traditional databases cannot handle streaming volume, developers have resorted to coupling streaming offerings along with data stores often non-ACID, non-transactional data stores to try to capture value from Fast Data. This approach has many limitations: it requires the use of many components, as in the Lambda Architecture, introducing complexity and risk.

Unlike traditional databases, VoltDB, an in-memory scale-out relational database, can process streams of data and produce analyses and decisions in milliseconds. As a single integrated platform, VoltDB reduces the complexity of building Fast Data applications by eliminating the need to connect streaming systems and non-relational data stores, and also provides a familiar, proven interaction model (SQL), simplifying application development and capturing real-time analytics using industry standard SQL-based tools.

Using VoltDB puts decision-making as close to the head of the data pipeline as possible. Because VoltDB can handle all traffic from ingestion, there is no need for the overhead of a separate ingest module or stream analytics module. These characteristics make VoltDB an ideal match to the demands of handling stream processing, real-time analysis and decision-making, using a relatively simple to implement the Fast Data Stack at scale.