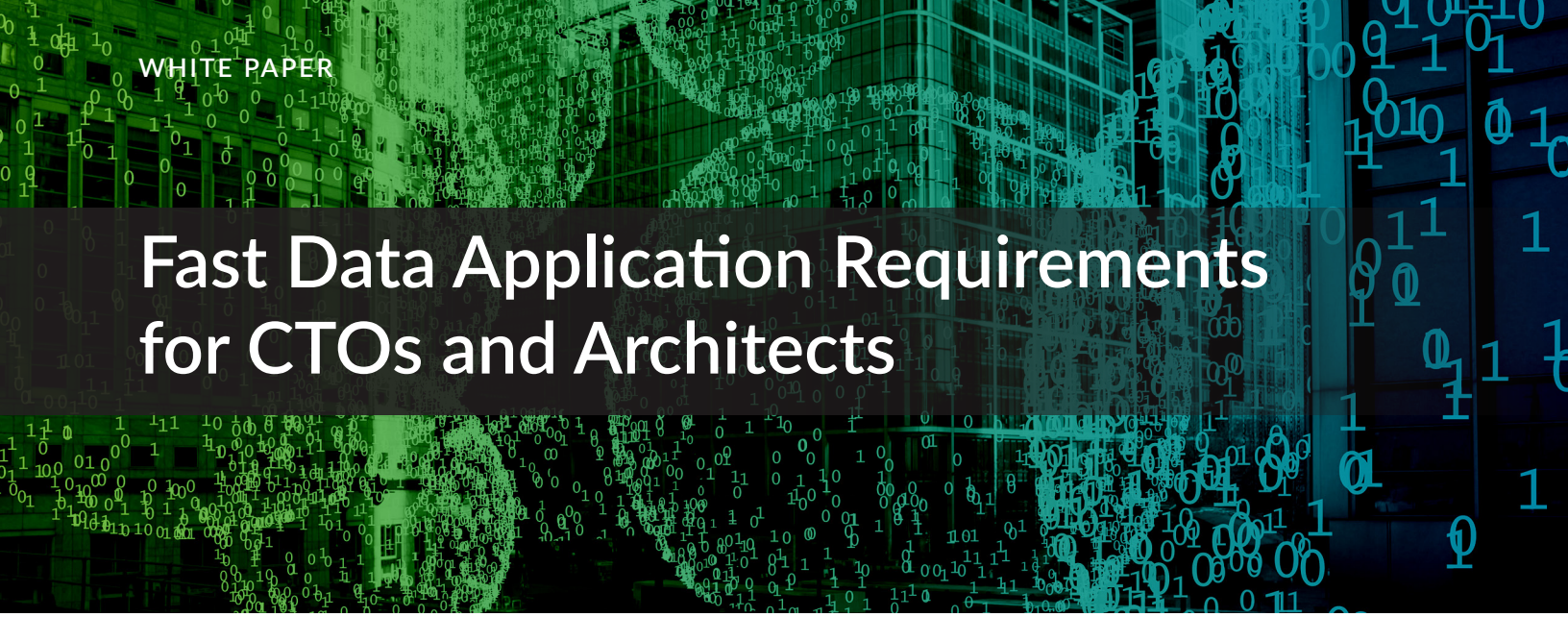

WHITE PAPER

Fast Data Application Requirements for CTOs and Architects

CTOs and Enterprise Architects recognize that the consumerization of IT is changing how software is developed, requiring an evolution in the way applications are designed and developed. The world's data is doubling in size every two years, driven by the forces of mobile adoption, connected devices, sensors, social media, and the Internet of Things (IoT). These sources generate vast amounts of fast-moving data. Successfully handling this 'Fast Data' is business's next big challenge.

The availability and abundance of fast, new data presents an enormous opportunity to extract intelligence, gain insight, and personalize interactions of all types. Companies of all sizes are examining how they build new applications and new analytics capabilities; in many cases, what were two separate functions — the application and the analytics — are beginning to merge. This evolution is leading enterprises to realize that they need a unifying architecture to support the development of dataheavy applications that rely on a fast-data, big-data workflow.

The emergence of this unifying architecture — the new enterprise data architecture — will result in aplatformcapableofprocessingstreamsofdata at new levels of complexity, scale and speed. This white paper looks at the requirements of the Fast Data workflow and proposes solution patterns for the most common problems software development organizations must resolve to build applications — and apps — capable of managing fast and big data.

VOLTDB

Solving common, repeating problems

Software designers and architects build software stacks from reference architectures to solve common, repeating problems. For example, the LAMP stack 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. A lot of

information can be conveyed succinctly to others familiar with the pattern by explaining the resulting architecture in the context of the generalized LAMP pattern.

Across industries a software stack is emerging that enables the development of applications capable of processing high velocity feeds of data (that quickly accumulate into large volumes of data). Over the next year or two this emerging Fast and Big Data stack will gain prominence and serve as a starting point for developers writing applications against fast, largescale data.

Though reaching a consensus on a functional fast data-big data stack through trial, testing, production and extended evaluation is the goal, it's important to step up a layer and consider the workflow that motivates the stack's requirements. We know the common user activities for a desktop application stack (a graphical UI, likely an MVC-style layering, a configuration database, undo capability…). We know the common user activities that guide the design of a web application. In fact we know these so well

many of us have written and designed tens or even hundreds of applications and websites over the course of our careers that we internalize the knowledge and forget that it was once new to us.

However, few of us have developed many large-scale data applications. We haven't yet as individuals, or as a community, internalized the usage patterns and general requirements for applications that process high speed inputs from sensors, M2M devices, or IOT platforms.

The core activities of the Fast-Data, BigData workflow

Large-scale data platforms must support four distinct activities to support fast-data, big-data workflows: collecting, exploring, analyzing and acting.

The large-scale data stack must allow collecting high-speed, event-oriented data that accumulates quickly. Simply collecting incoming data can be challenging. Tools like Flume and Kafka scale the retrieval and buffering of incoming event data for further application processing, solving one piece of the puzzle. However, data often needs to be enriched — augmented with metadata or transformed to common formats — validated, or correlated (re-assembled) before it can be used in Fast Data applications. The collection portions of the Fast Data stack must enable these computations and transformations in addition to the raw bucketing of incoming feeds.

The Fast Data stack must allow exploring the resulting data for patterns, trends and correlations. We commonly refer to this process as data science. Exploration is highly interactive and is often driven by questions and intuition. The demand for qualified, experienced data scientists is evidence of the value of exploring. The popularity of tools such as the R Project and the ability of systems like Spark to run machine-learning algorithms and statistical tests against massive data sets are democratizing access to exploration.

The Fast Data stack must allow analyzing data sets to derive seasonal patterns, statistical summaries, scoring models, recommendation models, search indexes and other artifacts for use in user-facing applications. We explore to find and test hypotheses and to identify exploitable trends.

Analyzing is a distinct activity from exploring. Analyzing is formal and repeated. Analytics may run once a year or once every few seconds but are typically periodic. For example, we explore to understand if our business is affected by seasonal trends or if a regression model can predict an interesting event. If our exploration confirms our hypothesis, we analyze periodically to calculate seasonal factors or parameters to a confirmed model.

The Fast Data stack must allow acting in real time against new incoming data. Responding to users or driving downstream processes, based on the output of the analysis activity and the context and content of the present event, is a requirement of applications that improve revenue. These actions protect users from fraud, enhance customer experience, or optimize the usage of expensive resources. Analytics provides us with a path to wisdom.

Analogous to how the integrated capabilities of the LAMP stack informed the millions of web applications written over the last decade, a stack that reliably, simply and efficiently enables the core activities of the Fast DataBig Data workflow will win and become the reference architecture for large-scale data applications.

The new enterprise data architecture

It is natural that the collect, explore and analyze phases precede the act phase as we innovate applications and businesses that leverage Big Data. Consequently, the Big Data portion of the Enterprise Data Architecture emerged first and is more familiar.

Practitioners have been using OLAP systems, statistical analysis packages, and data cubes for many years. There is consensus that a complete Big Data stack requires ubiquitous storage, often leveraging distributed file systems like HDFS; columnar analytics for OLAP; and frameworks that make data science, machine learning and statistical analytics feasible.

The requirements to enable real time applications running against fast data streams the streams of data that create business value are less broadly understood. In part this is because these applications are often conceived and deployed in the later act phase of the Fast Data — Big Data workflow, and in part because the technologies that support applications at this data velocity are relatively new.

A discussion of the common requirements underlying Fast Data applications is needed.

The 5 common requirements for Fast Data solutions

Scott Jarr (VoltDB, CSO) lists five requirements of Fast Data applications (http://voltdb.com/blog/youd-better-do-fast-data-right/):

1. Ingest/interact with the data feed.
2. Make decisions on each event in the feed.
3. Provide visibility into fast-moving data with real-time analytics.
4. Integrate Fast Data systems with Big Data systems.
5. Serve analytic outputs from Big Data systems to users and applications.

These requirements are broadly-shared characteristics of Fast Data applications, and appear with remarkable similarity across verticals as diverse as mobile subscriber management, smart grid, gaming, ad-tech, IoT and financial services. If you are designing an application on top of a high-speed data feed, you will eventually encounter and need to implement most (and often all) of these requirements.

Requirements are helpful; however, practitioners require meaningful guidance to structure a production solution. In order to understand how to build reliable, scalable and efficient Fast Data solutions, let's look at the problem from the perspective of the data that must be managed and the patterns of processing those data that are common in Fast Data applications.

Categorizing Data

Data management is harder to scale than computation. Computation, in the absence of data management requirements, is often easily parallelized, easily restarted in case of failure, and, consequently, easier to scale. Reading and writing state in a durable, fault tolerant environment while offering semantics, such as ACID transactions needed by developers to write reliable applications efficiently, is the more difficult problem. Scaling Fast Data applications necessitates organizing data first.

What are the data that need to be considered? There are several: the incoming data feed; metadata (dimension data) about the events in the feed; responses and outputs generated by processing the data feed; the postprocessed output data feed; and analytic outputs from the Big Data store. Some of these data are streaming in nature, e.g. the data feed. Some are state-based: the metadata. Some are the results of transactions and algorithms: responses. Fast Data solutions must be capable of organizing and managing all of these types of data.

DATA SET	TEMPORALITY	EXAMPLE
Input feed of events	Stream	Click stream, tick stream, sensor outputs, game play metrics
Event metadata	State	Version data, location, user profiles, point of interest data
Big data analytic outputs	State	Scoring models, seasonal usage, demographic trends
Event responses	Stream	Authorizations, policy decisions, threshold alerts
Output feed	Stream	Enriched, filtered, correlated transformation of input feed
Results of real-time analytics	State	Decisions, threshold alerts, authorizations

Two distinct types of data must be managed — streaming data and stateful data. Recognizing that the problem involves these different types of inputs is key to organizing a Fast Data solution.

Streaming data enters the Fast Data stack, is processed, possibly transformed, and then leaves. The objective of the Fast Data stack is not to capture and store these streaming inputs indefinitely; that's the Big Data responsibility. Rather, the Fast Data architecture must be able to ingest this stream and process discrete incoming events to allow applications that act in real time to mke fast, high-value decisions.

Stateful data is metadata, dimension data and analytic outputs that describe or enrich the input stream. Metadata might take the form of device locations, remote sensor versions, or authorization policies.

Analytic outputs are reports, scoring models, or user segmentation values: information gleaned from processing historic data that informs the real-time processing of the input feed. The Fast Data architecture must support very fast lookup against this stateful data as incoming events are processed and must support fast SQL processing to group, filter, combine and compute as part of the input feed processing.

As the Fast Data solution processes the incoming data feed, new events -event responses such as alerts, alarms, notifications, responses and decisions — are created. These events flow in two directions: responses flow back to the client or application that issued the incoming request; alerts, alarms and notifications are pushed downstream, often to a distributed queue for processing by the next pipeline stages. The Fast Data stack must support the ability to respond in milliseconds to each incoming event; integrate with downstream queuing systems to enable pipelined processing of newly created events; and provide fast export of proccessed events via an output feed to the data lake.

Categorizing Processing

Fast Data applications present three distinct workloads to the Fast Data portion of the emerging large-scale data stack. These workloads are related but require different data management capabilities and are best explained as separate usage patterns. Understanding how these patterns fit together — what they share and how they differ — is key to understanding the differences between fast and big, and key to making the management of Fast Data applications reliable, scalable and efficient.

Making Real Time Decisions

The most traditional processing requirement for Fast Data applications is simply fast responses. As high-speed events are being received, Fast Data enables the application to execute decisions: perform authorizations, policy evaluations, calculate personalized responses, refine recommendations, and offer responses at predictable millisecond-level latencies. These applications often need personalization responses in line with customer experience (driving the latency requirement). These applications are, very simply, modern OLTP. These Fast Data applications are driven by machines, middleware, networks or high concurrency interactions (e.g., ad-tech optimization or omni-channel location-based retail personalization). The data generated by these interactions and observations are often archived for subsequent data science. Otherwise, these patterns are classic transaction processing use cases.

Meeting the latency and throughput requirements for modern OLTP requires leveraging the performance of in-memory databases in combination with ACID transaction support to create a processing environment capable of fast per-event decisions with latency budgets that meet user experience requirements. In order to process at the speed and laten-

cies required, the database platform must support moving transaction processing closer to the data. Eliminating round trips between client and database is critical to achieving throughput and latency requirements.

Moving transaction processing in to memory and eliminating client round trips cooperatively reduce the running time of transactions in the database, further improving throughput.

Enriching Without Batch ETL

Real time data feeds often need to be filtered, correlated, or enriched before they can be "frozen" in the historical warehouse. Performing this processing in real time, in a streaming fashion against the incoming data feed, offers several benefits:

1. Unnecessary latency created by batch ETL processes is eliminated and time-to-analytics is minimized.
2. Unnecessary disk IO is eliminated from downstream Big Data systems (which are usually disk-based, not memory based when ETL is real-time and not batch-oriented).
3. Application-appropriate data reduction at the ingest point eliminates operational expense downstream — less hardware is necessary.
4. Operational transparency is improved when real time operational analytics can be run immediately without intermediate batch processing or batch ETL.

The input data feed in Fast Data applications is a stream of information. Maintaining stream semantics while processing the events in the stream discretely creates a clean, composable processing model. Accomplishing this requires the ability to act on each input event — a capability distinct from building and processing windows, as is done in traditional CEP systems.

"deltaDNA is a big data company. We're really interested in delivering fast real-time analytics on big data.

We have huge numbers of players creating billions of data points. The games are generating event data as the player plays through that allows us to understand what the players are doing in the game, how they're interacting. The objective is to collect this data to understand users. Being

able to interact, to customize the game experience to the individual player, is really important to keep those players playing.

We needed to be able to ingest data at a very high velocity. We also knew we weren't simply going to be ingesting data and pushing it into a data ware- house or data lake to do interesting stuff; we have to be able to interact with our users in real-time to make the experience personalized to the game.

VoltDB gave us the front-end technology that allows us to do this kind of interaction."

These event-wise actions need three capabilities: fast look-ups to enrich each event with metadata; contextual filtering and sessionizing (re-assembly of discrete events into meaningful logical events is very common); and a stream-oriented connection to downstream pipeline systems (e.g. distributed queues like Kafka, or OLAP storage or Hadoop/HDFS clusters). Fundamentally, this requires a stateful system that is fast enough to transact event-wise against unlimited input streams and able to connect the results of that transaction processing to downstream components.

Transitioning to Real Time

In some cases, backend systems built for batch processing are being deployed in support of sensor networks that are becoming more and more real-time. An example of this is validation, estimation and error (VEE) platforms sitting behind real-time smart grid concentrators. There are real-time use cases (real time consumption, pricing, grid management applications) that need to process incoming readings in real-time. However, traditional billing and validation systems designed to process batched data may see less benefit from being rewritten as real time applications.

A platform that offers real time capabilities to real-time applications, while supporting stateful buffering of the feed for downstream batch processing, meets both sets of requirements.

Returning to the smart grid example, utility sensor readings become more informative and valuable when one can use real-time analytics to compare a reading from a single meter to 10 others connected to the same transformer. This makes it possible to determine if there is a problem with the transformer, rather than inferring the problem lies with a single meter located at a home.

Presenting Real Time Analytics and Streaming Aggregations

Typically organizations begin by designing solutions to collect and store real-time feeds as a test bed to explore and analyze the business opportunity of Fast Data before they deploy sophisticated real-time processing applications. Consequently, the on-ramp to Fast Data processing is making real-time data feeds operationally transparent and valuable: is collected data arriving? Is it accurate and complete? Without the ability to introspect real-time feeds, this important data asset is operationally opaque pending delayed batch validation.

These real-time analytics often fall into four conceptually simple capabilities: counters, leaderboards, aggregates and time-series summaries. However, performing these at scale, in real-time, is a challenge for many organizations that lack Fast Data infrastructure.

Real-time analytics and monitoring enable organizations to capture a fast stream of events, sensor readings, market ticks, etc., and provide real-time analytic queries such as aggregated summaries by category, by specific entity, by timeframe, providing meaningful summaries of vast operational data updated second-by-second to show what is currently happening. For example, digital advertisers can launch campaigns and then tweak campaign parameters in real-time to see the immediate difference in performance.

Organizations are also deploying in-memory SQL analytics to scale highfrequency reports, or to support detailed slice, dice, sub-aggregation and groupings of reports for real time end-user BI and dashboards. This problem is sometimes termed “SQL Caching,” meaning a cache of the output of another query, often from an OLAP system, that needs to support scaleout high-frequency, high-concurrency SQL reads.

An example of scaling end-user visibility is provided by a cross-channel marketing attribution software provider which chose a Fast Data solution to support its real-time analysis and decisioning infrastructure. The company’s Big Data systems run complex algorithms for marketing attribution and optimization. The company needed a high performance, high throughput database with real-time in-memory analytics to enable its clients to make key decisions on media spend. The selection of a scale-out Fast

Data architecture supported the company’s cloud-based delivery model, enabling fast response times for multiple concurrent users.

Combining the Data and Processing Discussions

A platform that enables Fast Data applications must account for both stateful and streaming data and provide high speed, fault tolerant connectors to up stream and downstream systems in the data pipeline. When first approaching Fast Data problems, many developers focus solely on the input event feed. This is a mistake. It is important to consider enrichment data, allowing interactive querying of real time analytics using BI and dashboard tooling, combining OLAP analytic outputs with current data to make real time decisions. All of these functional components must be present in a Fast Data architecture to support applications that act.

VoltDB's in-memory, scale-out, relational database provides all of these functions.

- VoltDB offers high speed transactional ACID performance the ability to process thousands to millions of discrete incoming events per second.
- VoltDB includes export connects that parallelize streaming to downstream queues, databases and Hadoop clusters.
- VoltDB's in-memory SQL support enables interactive queries against real time analytics, streaming aggregations and operational summary data.
- VoltDB's durable, relational data model supports hosting enrichment data, OLAP analytics directly in the Fast Data platform, obviating secondary databases, simplifying the overall architecture and reducing network latency.

Shopzilla, Inc. a leading source for connecting buyers and sellers online, partnered with VoltDB to improve its high-speed, low-latency ingestion of merchant feeds as part of an inventory platform upgrade, with the goal of delivering a better customer experience.

Shopzilla wanted to update the way it delivered cost comparisons of over 100 million products from tens of thousands of retailers a month (including constantly-updated discounts), through its unique portfolio of engaging and informative shopping brands.

Shopzilla chose VoltDB to narrow the data ingestion-to-decision gap. VoltDB's ability to run tens of thousands of writes, and hundreds of thousands of reads per second

while performing real-time tracking enabled Shopzilla to drive revenues by delivering up-to-date information to consumers, and by passing along more highly targeted leads to the thousands of retailers paying Shopzilla on a pay-per-click basis.

Conclusion

Applications built to take advantage of fast and big data have different processing and system requirements. Implementing a component on the front end of the enterprise data architecture to handle fast ingestion

of data and interact on data to perform real real-time analytics enables developers to create applications that can make data-driven decisions on each event as it enters the data pipeline. Thus applications can take action on real-time, per-event data; processed data can then be exported to the long-term data warehouse or analytics store for reporting and analysis.

A unified enterprise data architecture provides the connection between fast and big, linking high-value, historical data from the data lake to fastmoving in-bound streams of data from thousands to millions of endpoints. Providing this capability enables application developers to write code that solves business challenges, without worrying about how to persist data as it flows through the data pipeline to the data lake. An in-memory, scaleout operational system that can decide, analyze and serve results at Fast Data's speed is key to making Big Data work at enterprise scale.

A new approach to building enterprise data archicttures to handle the demands of Fast Data gives enterprises the tools to process high volume streams of data and perform millions of complex decisions in real-time. With Fast Data, things that were not possible before become achievable: instant decisions can be made on real-time data to drive sales, connect with customers, inform business processes, and create value.

About VoltDB

VoltDB is an in-memory transactional database for modern applications that require the ability to manage data at unprecedented scale and volume, with 100% accuracy.

Unlike OLTP, Big Data, and NoSQL offerings that force users to compromise, only VoltDB supports all three modern application data requirements:

Millions — VoltDB processes relentless volumes of data from users, devices and sources.

Milliseconds — VoltDB ingests, analyzes, and acts on data instantaneously.

100% — Data managed by VoltDB is always accurate, all the time, for all decisions.

Telcos, Financial Services, Ad Tech, Gaming and other companies (including iOT technologies) use VoltDB to modernize revenue-critical applications. VoltDB was founded by a team of world-class database experts, including Dr. Michael Stonebraker, winner of the coveted ACM Turing award.

VOLTDB

© **VoltDB, Inc.** 209 Burlington Road, Suite 203, Bedford, MA 01730 voltdb.com