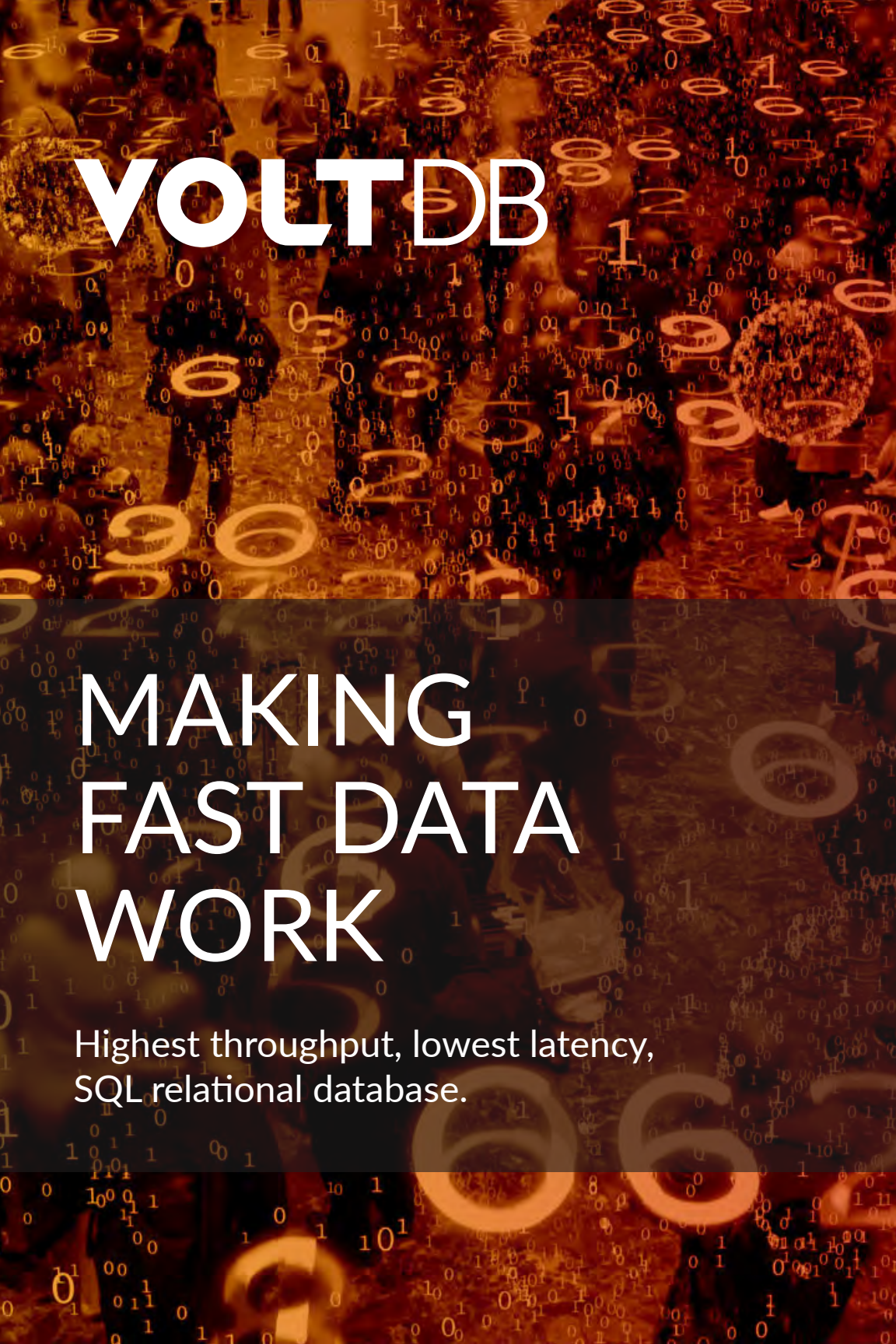
O'REILLY®

Compliments of VOLTDB

Architecting for the Internet of Things

Ryan Betts


VOLTDB
MAKING
FAST DATA
WORK
Highest throughput, lowest latency,
SQL relational database.

Architecting for the Internet of Things

Making the Most of the Convergence of Big Data, Fast Data, and Cloud

Ryan Betts

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Architecting for the Internet of Things
by Ryan Betts

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editor: Tim McGovern
Production Editor: Melanie Yarbrough
Copyeditor: Colleen Toporek
Proofreader: Marta Justak

Interior Designer: David Futato
Cover Designer: Randy Comer
Illustrator: Rebecca Demarest

June 2016: First Edition

Revision History for the First Edition
2016-06-16: First Release

978-1-491-96541-2

[LSI]

Table of Contents

CHAPTER 1

Introduction

Technologies evolve and connect through cycles of innovation, followed by cycles of convergence. We build towers of large vertical capabilities; eventually, these towers begin to sway, as they move beyond their original footprints. Finally, they come together and form unexpected—and strong—new structures. Before we dive into the Internet of Things, let's look at a few other technological histories that followed this pattern.

What Is the IoT?

It took more than 40 years to electrify the US, beginning in 1882 with Thomas Edison's Pearl Street generating station. American rural electrification lagged behind Europe's until spurred in 1935 by Franklin Roosevelt's New Deal. Much time was spent getting the technology to work, understanding the legal and operational frameworks, training people to use it, training the public to use it, and working through the politics. It took decades to build an industry capable of mass deployment to consumers.

Building telephone networks to serve consumers' homes took another 30 to 40 years. From the 1945 introduction of ENIAC, the first electronic computer, until the widespread availability of desktop computers took 40 years. Building the modern Internet took approximately 30 years.

In each case, adoption was slowed by the need to redesign existing processes. Steam-powered factories converted to electricity through

the awkward and slow process of gradual replacement; when steam-powered machinery failed, electric machines were brought in, but the factory footprint remained the same. Henry Ford was the first to realize that development, engineering, and production should revolve around the product, not the power source. This insight forced the convergence of many process-bound systems: plant design, power source, supply chain, labor, and distribution, among others.

In all these cases, towers of capability were built, and over decades of adoption, the towers swayed slightly and eventually converged. We can predict that convergence will occur between some technologies, but it can be difficult to understand the timing or shape of the result as different vertical towers begin to connect with one another.

So it is with the Internet of Things. Many towers of technology are beginning to lean together toward an IoT reference architecture—machine-to-machine communications, Big Data, cloud computing, vast distributed systems, networking, mobile and telco, apps, smart devices, and security—but it's not predictable what the results might be.

Precursors and Leading Indicators

Business computing and industrial process control are the main ancestors of the emerging IoT. The overall theme has been decentralization of hardware: the delivery of "'big iron'" computing systems built for insurance companies, banks, the telephone company, and the government has given way to servers, desktop computers, and laptops; as shipments of computers direct to end users have dropped, adoption of mobile devices and cloud computing have accelerated. Similarly, analog process control systems built to control factories and power plants have moved through phases of evolution, but here the trend has been in the other direction—centralization of information: from dial gauges, manually-operated valves, and pneumatic switches to automated systems connected to embedded sensors. These trends play a role in IoT but are at the same time independent. The role of IoT is connecting these different technologies and trends as towers of technology begin to converge.

What are some of the specific technologies that underlie the IoT space? Telecommunications and networks; mobile devices and their many applications; embedded devices; sensors; and the cloud com-

pute resources to process data at IoT scale. Surrounding this complicated environment are sophisticated—yet sometimes conflicting—identity and security mechanisms that enable applications to speak with each other authoritatively and privately. These millions of connected devices and billions of sensors need to connect in ways that are reliable and secure.

The industries behind each of these technologies have both a point of view and a role to play in IoT. As the world's network, mobile device, cloud, data, and identity companies jostle for position, each is trying to shape the market to solidify where they can compete, where they have power, and where they need to collaborate.

Why? In addition to connecting technologies, IoT connects disparate industries. Smart initiatives are underway in almost every sector of our economy, from healthcare to automotive, smart cities to smart transportation, smart energy to smart farms. Each of these separate industries relies on the entire stack of technology. Thus, IoT applications are going to cross over through mobile communication, cloud, data, security, telecommunications, and networking, with few exceptions.

IoT is fundamentally the connection of our devices to our context, a convergence—impossible before—enabled now by a combination of edge computing, pervasive networking, centralized cloud computing, fog computing, and very large database technologies. Security and identity contribute. Each of these industries has a complex set of participants and business models—from massive ecosystem players (Apple, Google) to product vendors (like VoltDB) to Amazon. IoT is the ultimate *coopetition* between these players. IoT is not about adding Internet connectivity to existing processes—it's about enabling innovative business models that were impossible before. IoT is a very deep stack, as shown in Figure 1-1.

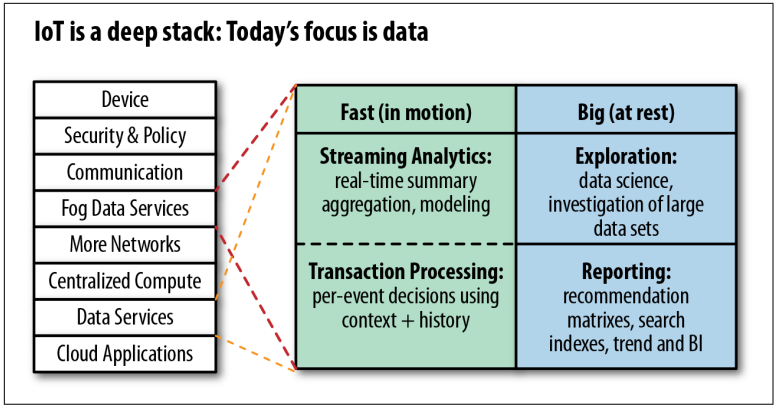


Figure 1-1. IoT is a very deep stack

As this battle continues, an architectural consolidation is emerging: a *reference architecture* for data management in the IoT. This book presents the critical role of the operational database in that convergence.

Analytics and Operational Transactions

Big Data and the IoT are closely related; later in the book, we'll discuss the similarities between the technology stacks used to solve Big Data and IoT problems.

The similarity is important because many organizations saw an opportunity to solve business challenges with Big Data as recently as 10 years ago. These enterprises went through a cycle of trying to solve big data problems. First they collected a series of events or log data, assembling it into a repository that allowed them to begin to explore the collected data. Exploration was the second part of the cycle. The exploration process looked for business insight, for example, segmenting customers to discover predictive trends or models that could be used to improve profitability or user interaction—what we now term *data science*. Once enterprises found insight from exploration, the next step of the cycle was to formalize this exploration into a repeatable analytic process, which often involved some kind of *reporting*, such as generating a large search index or building a statistical predictive model.

As industries worked through the first parts of the cycles—collect, explore, analyze—they deployed and used different technologies, so

on top of the analytic cycle there's a virtuous circle of technological and organizational innovation. New technologies lead to organizational innovations, as better insights into data enable industry leaders to adopt a data-driven operational model. The cycle is depicted in Figure 1-2.

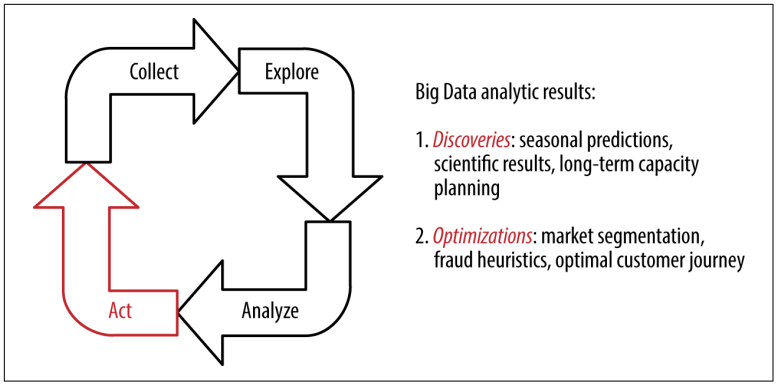


Figure 1-2. The Big Data cycle

In the nascent IoT, the *collection* phase of the analytical cycle likely deployed systems such as Flume or Kafka or other ingest-oriented tools. The *exploration* phase involved statistical tools, as well as data exploration tools, graphing tools, and visualization tools. Once valuable reporting and analytics were identified and formalized, architects turned to fast, efficient reporting tools such as fast-relational OLAP systems. However, up to this point, none of the data, insights, optimizations, or models collected, discovered, and then reported on were put to use. So far, through this cycle, companies did a lot of learning, but didn't necessarily build an application that used that knowledge to improve revenue, customer experience, or resource efficiency. Realizing operational improvement often required an application, and that application commonly required an operational database that operated at streaming velocities.

Real-time applications allow us to take insights about customer behavior or create models that describe how we can better interact in the marketplace. This enables us to use the historical wisdom—the *analytic* insights we've gained, with real-time contextual data from the live data feed—to offer a market-of-one experience to a mobile user, protect customers from fraud, make better offers via advertising technology or upselling capabilities, personalize an experience, or optimally assign resources based on real-time condi-

tions. These real-time applications, adopted first by data-driven organizations, require operational database support: a database that allows ACID transactions to support accurate authorizations, accurate policy enforcement decisions, correct allocation of constrained resources, correct evaluation of rules, and targeted personalization choices.

Streaming Analytics Meet Operational Workflows

Two needs collide in IoT applications with operational workflows that rely on streaming analytics: the high velocity, real-time data that flows through an IoT infrastructure creates the performance to handle streaming data; transactional applications that sit on top of the data feed require operational capabilities.

There are basically two categories of applications in the IoT. One type is *applications against data at rest*, streaming applications that focus on exploration, analytics, and reporting. Then there are *applications against data in motion*, the fast data, operational applications (Table 1-1). Some fast data applications combine streaming analytics and transaction processing, and require a platform with the performance to ingest real-time, high-velocity data feeds. Some fast data applications are mainly about dataflows—these may involve streaming, or collection and analysis of datasets to enable machine learning.

Table 1-1. Fast and big applications

Applications against data at rest (for people to analyze)	Applications against data in motion (automated)
Real-time summaries and aggregations	Hyper-personalization
Data modeling	Resource management
Machine learning	Real-time policy and SLA management
Historical profiling	Processing IoT sensor data

On the analytics side of IoT, applications are about the real-time summary, aggregation, and modeling of data as it arrives. As noted previously, this could be the application of a machine-learning model that was trained on a big data set, or it could be the real-time aggregation and summary of incoming data for real-time dashboarding or real-time business decision-making. Action is a critical component; however, in this Bayesian system, predictive models are

derived from the historical data (perhaps using Naive Bayes or Random Forest classifications). Action is then taken on the real-time data stream scored against those predictive models, with the real-time data being added to the data lake for further model refinement.

Fuzzy Borders, Fog Computing, and the IoT

There is a fuzzy border between the streaming and operational requirements of managing fast data in IoT. There is also an increasingly fuzzy border between where the computation and data management activities should occur. Will IoT architectures forward all streams of data to a centralized cloud, or will the scale and timeliness requirements of IoT applications require distributing storage and compute to the edges—closer to the devices? The trend seems to be the latter, especially as we consider applications that produce high-velocity data feeds that are too large to affordably transport to a centralized cloud. The Open Fog Consortium advocates for an architecture that places information processing closer to where data is produced or used, and terms this approach *fog computing*.

The industrial IoT field has been maturing slowly in its utilization of big data and edge computing. Technologies like machine learning and predictive modeling have helped industrial organizations leverage sensor data and automation technologies that have existed for years in industrial settings—and at a higher level of engagement. This has alleviated much of the inconsistency coming from a people-driven process by automating decision-making. But it also has revealed a gap in meaningful utilization of data. This solution pattern aligns with the fog computing approach and points to great potential for increasing quality control and production efficiency at the sensor level.

Fog Computing, Edge Computing, and Data in Motion

The intersection of people, data, and IoT devices is having major impacts on the productivity and efficiency of industrial manufacturing. One example of fast data in industrial IoT is the use of data in motion—with IoT gas temperature and pressure sensors—to improve semiconductor fabrication.

Operational efficiency is a primary driver of industrial IoT. Introducing advanced automation and process management techniques

with fast data enables manufacturers to implement more flexible production techniques.

Industrial organizations are increasingly employing sensors and actuators to monitor production environments in real time, initiating processes and responding to anomalies in a localized manner under the umbrella of *edge computing*. To scale this ability to a production plant level, it is important to have enabling technologies at the fog computing level. This allows lowering the overall operating costs of production environments while optimizing productivity and yields.

Advanced sensors give IoT devices greater abilities to monitor real-time temperature, pressure, voltage, and motion so that management can become more aware of factors impacting production efficiency. By incorporating fast data into production processes, manufacturers can improve production efficiencies and avoid potential fabrication delays by effectively leveraging real-time production data.

Integrating industrial IoT with fast data enables the use of real-time correlative analytics and transactions on multiple parallel data feeds from edge devices. Fast data allows developers to capture and communicate precise information on production processes to avoid manufacturing delays and transform industrial IoT using real-time, actionable decisions.

Whether we're building a fog-styled architecture with sophisticated edge storage and compute resources or a centralized, cloud-based application, the core data management requirements remain the same. Applications continue to require analytic and operational support whether they run nearer the edge or the center. In the industrial IoT, knowing what's in your data and acting on it in real time requires an operational database that can process sensor data as fast as it arrives to make decisions and notify appropriate sensors of necessary actions in a prescriptive manner.

CHAPTER 2
The Four Activities of Fast Data

When we break down the requirements of transactional or operational fast data applications, we see four different activities that need to occur in a real-time, event-oriented fashion. As data is originated, it is analyzed for context and presented to applications that have business-impacting side effects, and then captured to long-term storage. We describe this flow as *ingest*, *analyze*, *decide*, and *export*.

You have to be able to scale to the *ingest* rates of very fast incoming feeds of data—perhaps log data or sensor data, perhaps interaction data that's being generated by a large SaaS platform or maybe real-time metering data from a smart grid network. You need to be able to process hundreds of thousands or sometimes even millions of events per second in an event-oriented streaming and operational fashion before that data is recorded forever into a big data warehouse for future exploration and analytics.

You might want to look to see if the event triggers a policy execution or perhaps qualifies a user for an up-sell or offering campaign. These are all transactions that need to occur against the event feed in realtime. In order to make these decisions, you need to be able to combine analytics derived from the big data repository with the context in the *real-time analytics* generated out of the incoming stream of data.

As this data is received, you need to be able to *make decisions* against it: to support applications that process these events in real time. You need to be able to look at the events, compare them to the events that have been seen previously, and then provide an ability to make

a decision as each event is arriving. You want to be able to decide if a particular event is in norm for a process, or if it is something that needs to generate an alert.

Once this data has been ingested and processed, perhaps transacted against and analyzed, there may be a filtering or real-time transformation process to create sessions to extract the events to be archived, or perhaps to rewrite them into a format that's optimal for historical analytics. This data is then *exported* to the big data side.

Transactions in the IoT

There's a secret that many in the IoT application space don't communicate clearly: you need transactional, operational database support to build the applications that create value from IoT data.

Streams of data have limited value until they are enriched with intelligence to make them smart. Much of the new data being produced by IoT devices comes from high-volume deployments of intelligent sensors. For example, IoT devices on the manufacturing shop floor can track production workflow and status, and smart meters in a water supply system can track usage and availability levels. Whether the data feeds come from distribution warehouse IoT devices, industrial heating and ventilation systems, municipal traffic lights, or IoT devices deployed in regional waste treatment facilities, the end customer increasingly needs IoT solutions that add intelligence to signals and patterns to make IoT device data smart.

This allows IoT solutions to generate real-time insights that can be used for actions, alerts, authorizations, and triggers. Solution developers can add tremendous value to IoT implementations by exploiting fast data to automatically implement policies. Whether it's speeding up or slowing down a production line or generating alerts to vendors to increase supplies in the distribution warehouse in response to declining inventories, end customers can make data smart by adding intelligence, context, and the ability to automate decisions in real time. And solution developers can win business by creating a compelling value proposition based on narrowing the gap from ingestion to decision from hours to milliseconds.

But current data management systems are simply too slow to ingest data, analyze it in real time, and enable real-time, automated decisions. Interacting with fast data requires a transactional database

architected to handle data's velocity and volume while delivering real-time analytics.

IoT data management platforms must manage both data in motion (fast data) and data at rest (big data). As things generate information, the data needs to be processed by applications. Those applications must combine patterns, thresholds, plans, metrics, and more from analytics run against collected (big) data with the current state and readings of the things (fast). From this combination, they need to have some side effect: they must take actions or enable decisions.

IoT Applications Are More Than Streaming Applications

In a useful application built on high velocity, real-time data requires integration of several different types of data—some in motion and some essentially at rest.

For example, IoT applications that monitor real-time analytics need to produce those analytics and make the results queryable. The analytic output itself is a piece of data that must be managed and made queryable by the application. Likewise, most events are enriched with static dimension data or metadata. Readings often need to know the current device state, the last known device location, the last valid reading, the current firmware version, installed location, and so on. This dimension data must be queryable in combination with the real-time analytics.

Overall, there are at least five types of data, some streaming (in motion), and some relatively static (at rest) that are combined by a real-time IoT application.

This combination of streaming analytics, persisted durable state, and the need to make transactional per-event decisions all lead to a high-speed, *operational database*. Transactions are important in the IoT because they allow us to process events—inputs from sensors and machine-to-machine communications—as they arrive, in combination with other collected data, to derive a meaningful side effect. We add data from sensors to their context. We use the reports that were generated from the big data side, and we enable IoT applications to authorize actions or make decisions on sensor data as it's arriving, on a per-event basis.

Functions of a Database in an IoT Infrastructure

Legacy data management systems are not designed to handle vast inflows of high-velocity data from multiple devices and sources. Thus, managing and extracting value from IoT data is a pressing challenge for enterprise architects and developers. Even highly customized, roll-your-own architectures lack the consistency, reliability, and scalability needed to extract immediate business value from IoT data.

As noted earlier, IoT applications require four data management capabilities:

Fast ingest
: Applications need high-speed ingestion, in-memory performance, and horizontal scalability to provide a single ingestion point for very high-velocity inbound data feeds. An operational database must have the performance and scalability to ingest very high-velocity inbound data feeds. These could be hundreds of thousands or even millions of events per second, billions and billions of events per day. The system needs the performance to scalably ingest these events and to be able to process these events as they arrive, discrete from one another. If the events are batched, there must be logic to that batching. Batching introduces the worries of order of event, arrival, etc. When events are processed on a per-event basis, the result is a more powerful and flexible system.

Explore and analyze
: There must be real-time access to applications and querying engines, enabling queries on the stream of inbound data that allow rules engines to process business logic. As these events are received, stored, and processed in the operational database, the system needs to allow access to events for applications or querying engines. This is a different data flow. Data events are often a one-way data flow of information into the operational system. However, using a rules engine as an example of an application accessing operational data, the data flow is a request/response data flow—a more traditional query. The database is being asked a question, and it must provide a response back to the application or the rules engine.

Act

Applications also require the ability to trigger events and make decisions based on the inbound stream: thresholds, rules, policy-processing events, and more. *Triggered events* can be updates to a simple notification service or to a simple queuing service that are pushed, based upon some business logic that's evaluated within the operational database. An operational database might store this in database logic in the form of Java-stored procedures. Other systems might use a large number of working applications, but this third requirement is the same, regardless of its implementation. You need to be able to provide business logic as events arrive to run that business logic and in many cases, push a side effect to a queue for later processing.

Export

Finally, the application needs the ability to export accumulated, filtered, enriched, or augmented data to downstream systems and long-term analytics stores. Often, these systems are storing data on a more permanent basis. They could be larger but less real-time operational platforms. They could be a data archive. In some situations, we see people using operational components to buffer intraday data and then to feed it at the end of the day to more traditional end-of-day billing systems.

As this data is collected into a real-time intraday repository or operational system, you can start to write real-time applications that track real-time pricing or real-time consumption, for example, and then begin to manage data or smart sensors or devices in a more efficient way than when data is only available at the end of day.

A vital function of the operational database in the IoT is to provide real-time access to queries so that rules engines can process policies that need to be executed as time passes and as events arrive.

Categorizing Data

There's a truism among programmers that elegant programs "get the data right"; in other words, beautiful programs organize data thoughtfully. 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

(like ACID transactions) needed by developers to write reliable applications efficiently is the more difficult problem. Data management is harder to scale than computation. Scaling fast data applications necessitates organizing data first.

The data that need to be considered include the incoming data feed, the metadata (dimension data) about the events in the feed, responses and outputs generated by processing the data feed, the post-processed 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, such as the metadata. Some are the results of transactions and algorithms, such as responses. Fast data solutions must be capable of organizing and managing all of these types of data (Table 2-1).

Table 2-1. Types of data

Data set	Temporality	Example
Input feed of events	Stream	Click stream, tick stream, sensor outputs, M2M, gameplay 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	Events	Authorizations, policy decisions, triggers, threshold alerts
Output feed	Stream	Enriched, filtered, correlated transformation of input feed

Three distinct types of data must be managed: *streaming*, *stateful*, and *event data*. Recognizing that the problem involves these different types of inputs is key to organizing a fast data solution for the IoT.

Streaming data enters the fast data architecture, 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's responsibility. Rather, the fast data architecture must be able to ingest this stream and process discrete incoming events.

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—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; and alerts, alarms, and notifications are pushed downstream, often to a distributed queue for processing by the next pipeline stages. The fast data architecture must support the ability to respond in milliseconds to each incoming event and must integrate with downstream queuing systems to enable pipelined processing of newly created events.

Categorizing Processing

Fast data applications present three distinct workloads to the fast data portion of the emerging IoT 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 the key to understanding the differences between fast and big, and the key to making the management of fast data applications in the IoT reliable, scalable, and efficient.

Combining the Data and Processing Discussions

Table 2-2 shows a breakdown of the different usage patterns.

Table 2-2. Differing usage patterns

Type	Real-time decisions	Real-time ETL	Real-time analytics/SQL caching
Input feed	Personalization, real-time scoring requests	Sensor data, M2M, IoT	Real-time feed being observed for operational intelligence
Event metadata	Policy parameters; POI, user profiles	Metadata about the sensors infrastructure (versions, locations, and so on)	
Big data analytic outputs	Scoring rubrics; user segmentation profile	Interpolation parameters; min/max threshold validation parameters	OLAP report results in "SQL Caching" use cases.

Type	Real-time decisions	Real-time ETL	Real-time analytics/SQL caching
Event responses and alerts	Decisions and customization results	Alerts/notifications on exceptional events (or exceptional sequences of events)	Dashboard and BI query responses. Counters, leaderboards, aggregates, and time-series groupings for operational monitoring
Output feed	Archive of transaction stream for historical analytics	Enriched, filtered, processed event feed handed downstream	

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 and 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 latencies 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 into memory and eliminating client round trips cooperatively reduce the running time of transactions in the database, further improving throughput. (Recall Little's Law.)

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:

- Unnecessary latency created by batch ETL processes is eliminated and time-to-analytics is minimized.
- Unnecessary disk IO is eliminated from downstream big data systems (which are usually disk-based, not memory-based).
- Application-appropriate data reduction at the ingest point eliminates operational expense downstream, so not as much hardware is necessary.
- 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.

These event-wise actions need three capabilities: fast lookups to enrich each event with metadata; contextual filtering and sessionizing (reassembly of discrete events into meaningful logical events is very common); and a stream-oriented connection to downstream pipeline processing components (distributed queues like Kafka, for example, 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 IoT sensor networks that are becoming more and more real time. An example of this is the validation, estimation, and error platforms sitting behind real-time smart grid concentrators. There are many 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. Recognizing when an application isn't a streaming or fast data application is important.

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.

Ingestion Is More than Kafka

Kafka is a persistent, high-performance message queue developed at LinkedIn and contributed to the Apache Foundation. Kafka is highly available, partitions (or shards) messages, and is simple and efficient to use. Great at serializing and multiplexing streams of data, Kafka provides "at least once" delivery, and gives clients (subscribers) the ability to rewind and replay streams.

Kafka is one of the most popular message queues for streaming data, in part because of its simple and efficient architecture, and also due to its LinkedIn pedigree and status as an Apache project. Because of its persistence capabilities, it is often used to front-end Hadoop data feeds.

Kafka's ability to handle high-velocity data feeds makes it extremely interesting in the big data/fast data application space. With Kafka, a database can subscribe to topics and transact on incoming messages, as fast as Kafka can deliver. This capability allows fast data applications to process and make decisions on data the moment it arrives, rather than waiting for business logic to batch-process data in the Hadoop data lake.

Kafka Use Cases

Unlike traditional message queues, Kafka can scale to handle hundreds of thousands of messages per second, thanks to the partitioning built in to a Kafka cluster. Kafka can be used in the following use cases (among many more):

- Messaging
- Log aggregation
- Stream processing

- Event sourcing
- Commit log for distributed systems

Kafka is a message queue, but it won't get you to the IoT application.

Beyond Kafka

In IoT implementations, Kafka is rarely on the front line ingesting sensor and device data. Often there is a gateway that sits in front of the queueing system and receives data from devices directly.

For example, Message Queue Telemetry Transport (MQTT) is a lightweight publish/subscribe protocol designed to support the transfer of data between low-power devices in limited-bandwidth networks. MQTT supports the efficient connection of devices to a server (broker) in these constrained environments.

Applications using MQTT can retrieve device and sensor data and coordinate activities performed on devices by sending command messages.

MQTT is used for connections in remote locations where a small code footprint is required or network bandwidth is limited. Running on top of TCP/IP, MQTT requires a message broker, in many cases, Kafka. The broker is responsible for distributing messages to clients based on the message topic.

Several other lightweight brokers are available, including RabbitMQ (based on the AMQP protocol), XMPP, or the IETF's Constrained Messaging Protocol.

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.

Organizations are also deploying in-memory SQL analytics to scale high-frequency reports—or commonly to support detailed slice, dice, subaggregation, 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 scale-out, high-frequency, high-concurrency SQL reads.

Shortcomings of Streaming Analytics in the IoT

Streaming analytics applications are centered on providing real-time summaries, aggregation, and modeling of data, for example, a machine-learning model trained on a big data set or a real-time aggregation and summary of incoming data for real-time dashboarding. These "passive" applications analyze data and derive observations for human analysts, but don't support automated decisions or actions.

Transactional applications, on the other hand, take data events as they arrive, add context from big data or analytics—reports generated from the big data side—and enable IoT applications to personalize, authorize, or take action on data on a per-event basis as it arrives in real time. These applications require an operational component—a fast, in-memory operational database.

Why Integrate Streaming Analytics and Transactions?

IoT platforms deliver business value by their ability to process data to make decisions in real time, to archive that data, and to enable analytics that can then be turned back into actions that have impact on people, systems, and efficiency. This requires the ability to combine real-time streaming analytics and transactions on live data feeds.

The data management required for the centralized computing functions of an IoT platform is essentially the same stack of data management tools that has evolved for traditional big data applications. One could argue that IoT applications are an important type of big data application.

Managing Multiple Streams of High-Velocity Inbound Data

Data flows from sensors embedded in electric meters that monitor and conserve energy; from sensors in warehouse lighting systems; from sensors embedded in manufacturing systems and assembly-line robots; and from sensors in smart home systems. Each of these sensors generates a fast stream of data. This high-velocity data flows from all over the world, from billions of endpoints, into edge computing, fog computing, and the cloud, and thence to data-processing systems where it is analyzed and acted on before being passed to longer-term analytic stores.

As we look forward through the rest of the IoT architecture stack and explain some of the different examples of what companies and entities have built in terms of IoT platforms, we provide examples of more traditional big data platforms; this will illustrate that the reference architectures built for IoT infrastructures closely resemble the architectures assembled for other non-IoT big data applications. This emerging pattern is a good sign: it means we have begun to find a series of tools or a stack of tools that has applicability to a breadth of problems, signaling stability in the data management space.

At the End of Every Analytics Rainbow Is a Decision

People are seldom the direct users of operational systems in the IoT—the users are applications. Application requirements in the faster IoT infrastructure happen on a vastly different scale at a vastly different velocity than they do in other systems. The role of managing business data in operational platforms is changing from one in which humans manage data directly by typing queries into a database to one in which machine-to-machine communications and sensors rely on automated responses and actions to meet the scale and velocity challenges of the IoT. IoT apps require an operational database component to provide value by automating actions at machine speed.

The following architectural observations implicitly state the requirements for an operational database in an IoT platform. Use of an operational component is the only way to move beyond the insights

gleaned from analytics to make a decision or take an action in the IoT.

First, an IoT architecture needs a *rules engine* to enable the augmentation or filtering of data received from a device, write data received from a device to a database, save a file to another resource, send a push notification, publish data to a queue for downstream processing, or invoke a snippet of code against data as it's arriving to perform some kind of business processing or transformation.

Almost all of these functions require access to operational data. If you're going to enrich data as it arrives, you need to have access to the dimension data to use to enrich incoming data streams, or you need access to the real-time analytics that have been aggregated to enrich the incoming sensor message. An operational component can filter information from a device. Filtering is rarely done in a stateless environment. Filters are rules applications that require state to know when to trigger an action. Rules engines know the first time a system sees a message, the most recent time it's seen a message, if the message indicates that a device has moved, and more.

These filtering applications require access to operational data. Notifications are typically sent as the result of a policy trigger. The system needs to understand whether a notification is of interest to a downstream consumer, whether you're notifying that a threshold has been crossed, and so on.

Rule application components require tight *integration with operational data*. This is fundamentally the role of an operational database in an IoT platform: to provide real-time interactive access to intraday data or to recent data needed to evaluate rules to manage the routing of data to downstream applications or to process real-time business logic as these events are arriving. What makes these operational versus pure analytical functions is that they happen in line with the event arriving and being first processed, and often they happen in line with a user experience or a decision that needs to be propagated back to a device.

Take note: *very fast decisions are the lingua franca of the IoT*. They take the analytics and data generated by sensors and connected devices, add context, and provide necessary actions back to devices, also pushing that data via export upstream to longer-term analytics stores. Without decisions and actions, the IoT would simply be the sound of one hand clapping.

CHAPTER 3

Writing Real-Time Applications for the IoT

Let's look at examples of real-time applications for the IoT.

Case Study: Electronics Manufacturing in the Age of the IoT

A global electronics manufacturer of IoT-enabled devices was dealing with multiple streams of high-velocity inbound data. Figure 3-1 shows its architecture.

Event data from thousands to millions of devices—some mobile, some "smart," some consumer appliances—arrived via the cloud to be processed by numerous apps, depending on the device type. Sitting between the data sources and the database tier (Cassandra, PostgreSQL, and Hadoop) was a rules engine that needed high-speed access to daily event data (e.g., a mobile device subscriber using a smart home app), which it held in an in-memory data grid used as an intraday cache.

As the rules engine ingested updates from the smart home application, it used the intraday data (such as location data on the device) from the in-memory grid to take actions (such as turn on lights when the mobile device wasn't in the home).

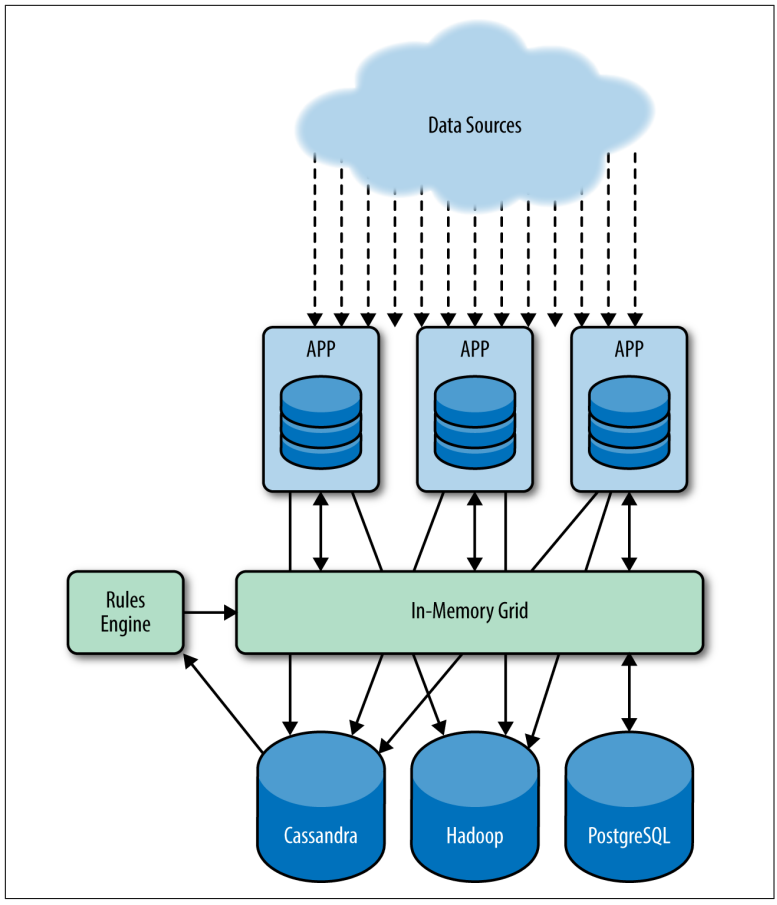


Figure 3-1. The architecture of a manufacturer of IoT-enabled devices

The rules engine queried Cassandra directly, creating latency and consistency issues. In some cases, the rules engine required stricter consistency than was guaranteed by Cassandra, for example, to ensure that a rule's execution was idempotent. Scalability issues added to the problem—the rules engine couldn't push more sophisticated product kits to Cassandra fast enough.

The architecture included PostgreSQL for slow-changing dimension data. The in-memory grid cached data from PostgreSQL for use by the rules engine, but the rules engine needed faster access to Cassandra. In addition, each app needed to replicate the incoming event stream to Cassandra and Hadoop.

Further, the scale-out in-memory grid was not capable of functions such as triggering, alerting, or forwarding data to downstream systems. This meant the rules engine and the applications that sat on top of the grid were each responsible for ETL and downstream data push, creating a many-to-many relationship between the ETL or ingest process from the incoming data stream to downstream systems. Each application was responsible for managing its own fault-tolerant ingestion to the long-term repository.

The platform was strangled by the lack of a consolidated ingest strategy, painful many-to-many communications, and performance bottlenecks at the rules engine, which couldn't get data from Cassandra quickly enough to automate actions. Grid caching was insufficiently fast to process stateful data that required complex logic to execute transactions in the grid, for example, the instruction previously mentioned—turn on the lights in the smart home whenever the mobile device is outside the home.

Fast Data as a Solution

A new, simplified architecture was implemented that replaced the in-memory grid with a SQL operational database. With this fast operational database, the rules engine was able to use SQL for more specific, faster queries. Because it is a completely in-memory database, it met the latency and scalability requirements of the rules engine. The database also enabled in-memory aggregations that previously were difficult due to Cassandra's engineering (e.g., consistency) trade-offs.

Because the database is relational and operational, it became the authoritative owner of many of the manufacturer's master detail records. Master detail records could be associated with intraday events, easing operations, and maintaining consistency between inbound data and dimension data (e.g., device id data). Finally, using the database's export capability created a unified platform to take the enriched, augmented, filtered and processed intraday event data and push it or replicate it consistently to Cassandra and Hadoop while replicating dimension data changes to the PostgreSQL master detail record system.

Note the simplified architecture shown in Figure 3-2.

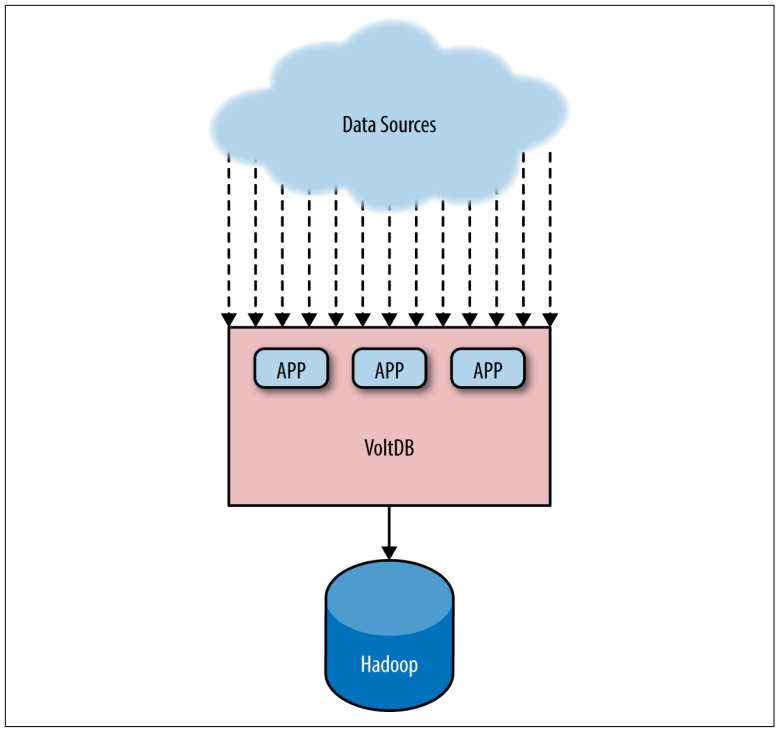


Figure 3-2. Simplified architecture for manufacturer of IoT devices

Adding an operational database to this architecture solved three pain points: it provided an extremely fast consolidated ingest point for high-velocity feeds of inbound IoT data; it provided processing on inbound data requests that required state, history, or analytic output; and it provided real-time processing of live data, enabling automated actions in response to inbound requests, all with speed that matched the velocity of the inbound data feeds.

In this IoT example, the operational database served not only as a fast intraday in-memory data cache but also as a transaction processing engine and a database of record. The bottleneck of reading data from Cassandra was eliminated, as was the *n*-squared complexity of ingesting and orchestrating many-to-many communications from apps and myriad data sources. An additional benefit was access to analytics that were not previously available from Cassandra due to the sheer volume of processed data.

Case Study: Smart Meters

A number of electric utilities are using operational databases to collect real-time data from IoT smart electric and water metering systems. These different IoT platforms require all four of the capabilities of an operational database we described.

Smart metering platforms typically provide meter readings to the IoT data management infrastructure every 15 minutes. Usually meters are associated with some kind of a *concentrator*—a device in a sensor network that collects data flowing from separate sensors or meters, batches that information, and provides it to the data management infrastructure. Once that data has arrived, there are a number of different rules that need to be applied. Industry-specific validation, error checking, and estimation rules need to be applied. For example, if a meter reading is lost, the system might want to interpolate the value between the last two events. The goal is to be able to guarantee that a reading isn't obviously corrupt, and that it's a value that is valid.

With a number of other relatively straightforward validation processes, being able to supply or execute these validation processes in near real time improves operational efficiency, makes it clear when data is being corrupted or lost, and also allows interesting operational applications to be developed as a benefit of the real-time infrastructure. For example, if the system hasn't received readings from some set of meters over the last two reporting periods, it's important to understand if those meters are associated with one concentrator or are distributed across a number of concentrators. This understanding might indicate two different operational problems that need to be resolved in different ways.

At the same time, as this data is being collected into a real-time intraday repository or operational system, you can start to write real-time applications that track real-time pricing and consumption, and then begin to manage data or smart metering grids in a more efficient way than when data is only available at the end of day. However, the billing infrastructure that's calculating total utilization and generating the eventual bill to the consumer is still expecting data in a bulk fashion. This system doesn't expect data to trickle in over the course of the day; rather, it expects the traditional format of data to be provided at the end of the day or at the end of some longer period. In this case, the system needs to be able to collect that

intraday data, apply the events, rules, and triggers to it that we discussed, and then at the end of the billing period, gracefully dump that to the billing system as an input in the time period that it expects. The same process applies if the utility wants to capture all of this data to a historical system for long-term offline analytics, exploration, and reporting.

Here we see that a smart metering system uses an operational database in all four ways that were described earlier—for fast ingest of events, the application or the ability for a rules engine to access real-time data to support real-time analytics that might trigger alerting/alarming to other operational applications, to buffer data for export to an end-of-day billing system, and then finally, to become an ingest point to an offline storage system or a nearline storage system like Hadoop.

Conclusion

What's interesting about IoT is what can result from the convergence of the macro-trends discussed in the introduction, and what happens as the different industries that use this shared infrastructure begin to lean together and collide with one another. IoT is inseparable from data and cloud. The overlap between these sectors is strong, and, like cloud, IoT is an extremely data-driven effort. IoT platforms derive their value from their ability to process data to make decisions, to archive that data, and to enable analytics that can then be turned back into actions that have impact on people and efficiency.

Finally, let's talk a little bit about what might happen going forward, using an example from another industry. In the early 1900s, there were hundreds of companies making automobiles. Think of the database universe maps analysts produce with a hundred logos. In the 1900s, that's what the auto manufacturing industry looked like. Everybody was making cars, and there were many different kinds of cars: electric cars, steam cars, and cars with internal combustion engines. The manufacturers came from different disciplines: carriage-maker, wheelwright, engine-builder, metal-worker, locomotive builder. Their different focuses influenced the types of cars they built: the Stanley Steamer, the *Flocken Elektrowagen* (the first electric-powered car), and the hydrogen-gas fueled Hippomobile. Nikolaus Otto built the first viable internal combustion engine. Each

effort had its own appeal, but over time, one model claimed 50% of the market very quickly: the Ford Model T, which was cheap, reliable, and available in any color you wanted, as long as it was black. Henry Ford, and the way in which he built that car, forced hundreds of towers of vertical capabilities to converge and drove hundreds of other manufacturers out of business in the span of a decade or two.

What we're beginning to see in the world of IoT is people learning how to build scalable IoT platforms, composed of elements contributed by other industries and technologies that are converging. People in these industries are beginning to understand the roles that different data technologies have in IoT platforms. We are beginning to see consistent use of operational databases in those platforms. We're starting to see a lot of different architectures replaced and consolidated into a relatively consistent architecture that's being adopted across a number of different implementations, from IoT to mobile to manufacturing to energy to financial services. It's reasonable to expect this process of convergence within the database space to continue, and to start to see best practices emerge in the development of the IoT.

About the Author

Ryan Betts is one of the VoltDB founding developers and is presently VoltDB CTO. Ryan came to New England to attend WPI. He graduated with a B.S. in Mathematics and has been part of the Boston tech scene ever since, earning an MBA from Babson University along the way. Ryan has been designing and building distributed systems and high-performance infrastructure software for almost 20 years. Chances are, if you've used the Internet, some of your ones and zeros passed through a slice of code he wrote or tested.