Volt Active Data: Why Different is Better

Real-Time Data Processing at Scale for Mission-Critical Applications

VOLT ACTIVE DATA

Let's make one thing clear from the start: Volt Active Data (Volt) is architected differently from any data processing platform you've worked with. Now let's say that another way: Volt is ***better*** than any data processing platform you've worked with or considered. It is proven, in the real world, to meet the demands of mission-critical applications (or business-critical applications) without forcing uncomfortable and/or costly compromises on things like stack complexity, TCO, scale, accuracy, or resiliency.

Volt exists because Dr. Michael Stonebraker, the creator of Postgres and Vertica, foresaw in 2008 that scaling transaction processing systems to meet the future needs of mission-critical systems, including those used for real-time analytics, stream processing, and financial transactions, required a complete architectural rethink.

In this paper, we explain how that architectural rethink eliminates the typical compromises forced by other systems and why, if you are developing applications for things like telco charging, real-time analytics, fraud prevention, industrial IoT, and other 'millisecond-sensitive' use cases, Volt should be on your must-consider list.

What Sets Volt Apart

The business case behind online transaction processing (OLTP) was (and still is) rock solid, but by 2008, the 1980s-based architectural models used to implement OLTP had become obsolete.

Volt is the OLTP platform for the 21st century.

The foundational value of Volt – what differentiates it from other data platforms and why it's so essential for mission-critical applications – is that it can process a lot of ACID transactions quickly and reliably:

- By 'a lot' we mean 'hundreds of thousands per second'.
- By 'ACID' we mean with strict serializability — the highest possible level.
- By 'quickly' we mean with an average latency of 1 to 2ms and a 99th percentile latency of less than 10ms.
- By 'reliably' we mean at least 'five nines".

This is easy to say but very hard to do. For example: high volumes and low response times can go together, but usually at the expense of ACID, which means decisions may be wrong.

The intent was to achieve all of the above at the same time, without making unacceptable compromises.

This led to a non-obvious design approach: optimizing for individual small ACID write transactions.

It turns out that by focusing on individual small ACID write transactions, Volt can do it all.

Before we discuss the architecture in detail, consider these four other things about Volt that distinguish it from other data platforms:

1. “Always process, sometimes store”

The purpose of Volt is to process data, not store data. We let you store data because you need context to make decisions, not because you might need it five years from now. When you communicate with Volt, you tell it what you want done by sending a Java class name and some parameters. The Java class lives on the server and uses the parameter list and data it finds locally to make decisions, store results, and inform downstream systems. There are many advantages to this, including the ability to solve the same business problem with far fewer calls than other systems.

2. A Single Layer

All Volt nodes are the same, and all Volt nodes are equal. Nodes cooperate behind the scenes to provide high availability. Indexing, aggregation, and everything else is done by the same nodes that do the work. There is thus only one sizing dimension for Volt: more or fewer nodes.

3. Designed for Scale, Speed, and ACID

Dr. Stonebraker created Volt as a complete OLTP solution, not a ‘minimum viable product’ that did one or two things. This means that we have never had to make any architectural compromises or kludges to retrofit speed, scale, or ACID; all three were present on day one.

4. In-memory

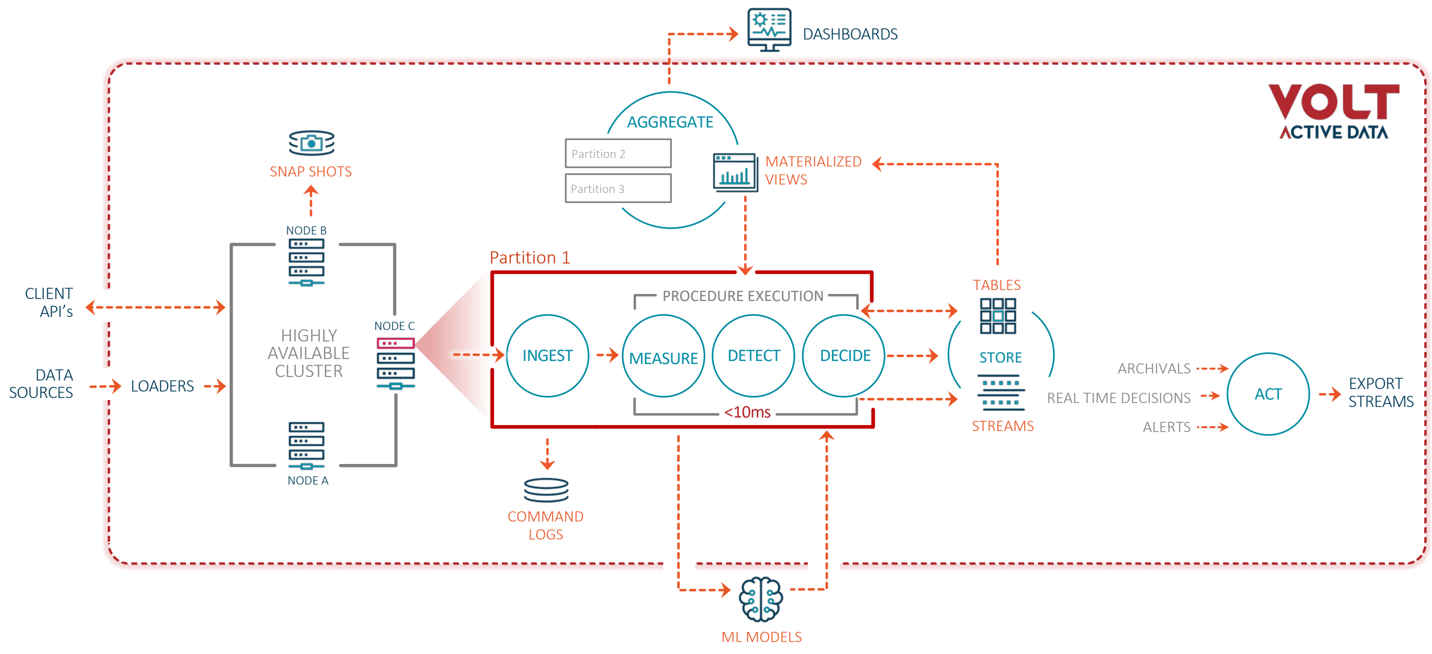
Volt keeps all needed data in RAM on the node in question. This is how it can consistently and rapidly do lots of transactions. Because Volt supports large clusters, we have users running systems involving a total of more than 5TB of RAM, while still getting predictable 1-2ms latency. Once you stop trying to store everyone’s data forever and focus on holding what you need to enable transactions that are happening today, RAM generally ceases to be a limiting factor. The benefit of doing this is that Volt never waits for IO when processing, which means it doesn’t need to spawn lots of threads to keep things moving.

The Volt Active Data Architecture

Volt uses a CPU core as a scaling unit, which we refer to as a 'partition'. Each partition is responsible for a portion of the data, which means both processing and storage. Each one runs a single thread and takes requests off a queue with a fixed order, before running them to completion. High availability comes from having the same queue run on two or more partitions on different physical servers.

Physical persistence comes from regular snapshots to local disk and writing the queue of incoming requests to a 'command log', which can then be replayed from the last snapshot.

The Volt Architecture



Volt Architecture Terminology

K Factor

The K factor is the number of extra data copies we keep. 'k=1' means 1 spare copy. Spare copies never live on the same physical server as each other or the master copy.

Partition

Volt uses a hash algorithm to divide workloads. By default, it divides each workload eight ways. Each division is handled by a 'Partition'. In practice, there is usually a 1:1 relationship between partitions and CPU cores. A partition can never use more than 1 CPU core. Volt places multiple copies of each partition on different servers to provide high availability and designates one copy as the 'Partition Leader' for clients to talk to. The other copies can become the Partition Leader if the node the Partition Leader lives on dies.

Node

A node is a physical server, virtual server or containerized pod running Volt Active Data. A node hosts one or more Partitions. As a rule of thumb, the number of partitions should be three-fourths the number of physical cores available for Volt Active Data. Under normal circumstances, some of a node's partitions will be leaders, and others will follow other partitions on other nodes.

Client

A Volt client is like a normal client but is aware of how the cluster is organized. The client sends each request to the correct Partition Leader. During cluster expansions and other events that change the topology of the system, requests could end up at the wrong node. When this happens, nodes reroute the request to the correct destination.

Snapshots

Every now and then, Volt writes the contents of RAM to a local disk using a Snapshot. Snapshots are only read when the cluster is restarted.

Command Log

The Command Log is a serialized list of requests for a Partition. When the cluster restarts, it loads the latest Snapshot and then uses the Command Log to redo all the things that happened after the snapshot.

Request

This is where everything with Volt comes together. A request is a command to execute an Augmented Transaction. It consists of a Transaction Name and a set of parameters. As requests arrive at the Partition, they are written to the Command Log. But Volt does requests in a very unique way that allows us to process requests both quickly and accurately. This is an important aspect of Volt's design, which deserves further explanation.

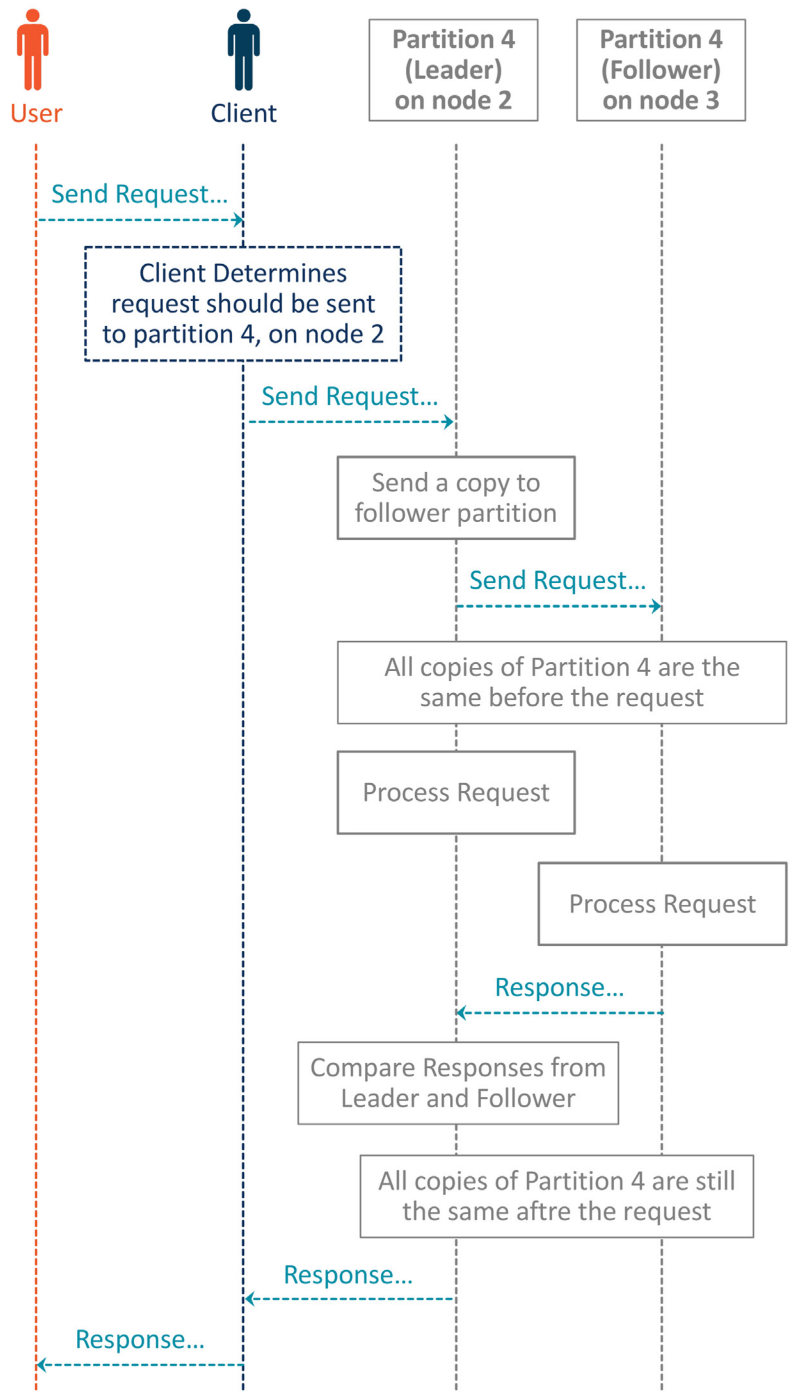
Complex Transactions, Quick Decisions: How Volt Processes Requests

To fully understand how Volt can do what it does, we need to take a closer look at the unique way it processes requests. Volt will process complex transactions that mix Java and SQL code in an atomic way, so long as the code is deterministic so all the copies come up with the same answer.

All of the steps below happen in 1-2 milliseconds, with the actual processing taking *microseconds.*

This is how Volt processes a request:

1. The user sends the name of a required class that implements VoltProcedure.run, along with its parameters, to a client object.
2. The client figures out where the partition leader for the request lives, and routes the request to it. In this case, it's the leader of partition 4, on Node 2.
3. The partition leader copies the request to its follower(s) on other nodes. In this case, it gets copied to Node 3. Note that we are copying the request, not the work or the results of the work.
4. Both copies of the partition execute the request. Since a partition only does one request at a time, and the followers only do requests sent by the leader, the partitions remain in sync.
5. As soon as the leader finishes the request, it waits for a response from its follower(s), then compares the answers, and, if they agree, sends the results back to the client.
6. The client returns the result to the user.


User
Client
Partition 4 (Leader) on node 2
Partition 4 (Follower) on node 3
Send Request...
Client Determines request should be sent to partition 4, on node 2
Send Request...
Send a copy to follower partition
Send Request...
All copies of Partition 4 are the same before the request
Process Request
Process Request
Response...
Compare Responses from Leader and Follower
All copies of Partition 4 are still the same aftre the request
Response...
Response...

Volt Request Process FAQs

1. Can an awkwardly timed failure result in lost data?

If the Partition Leader's node dies, then any changes the request made will live on in the follower node, which automatically gets promoted to leader. This means it's impossible for the client to think a transaction has finished even though the changes made have been lost because a single node crashed.

2. Can the leader and follower get out of sync?

Because the leader forces all requests into a shared deterministic queue, both the leader and follower do the exact same work in the same sequence. The results are then compared to make sure they are the same before being returned to the client.

3. How fast is this in real-world conditions?

Assuming one partition with one leader and one follower on separate nodes, we'd expect to see a minimum of 5,000 transactions per second per core, with an average latency of around 1-2ms and a 99th percentile latency under 10ms. Each leader/follower pair processes dozens or hundreds of requests at any given moment, but all the requests are in the same queue.

4. Isn't doing all the work more than once inefficient?

Not really. The alternative — doing all the work in one place and sending the changes to the follower— wouldn't save any hardware; the follower's CPU needs to be available instantly in case the leader dies.

5. What if I want to access more than one partition?

Volt can do this. It uses a special mechanism to coordinate access to all the partitions. It's slightly slower (it has to access all the partitions) but works the same way.

How Volt Meets Mission-Critical Application Requirements

Now that we understand how Volt works at the request level, let's zoom out a bit to understand how this process translates into meeting the requirements of mission-critical applications.

Scale

Because Volt only has one layer and has a 'shared nothing' architecture, scale is only limited by the hardware you can afford. As a general rule, the first challenging constraint will be when people saturate the network. For example: a nine-node Volt cluster sending 70,000 transactions a second, each 20 KB in size, was able to saturate a 10 Gbit network before any other bottlenecks emerged. The largest production cluster we're aware of is larger than 30 nodes. We also have production systems that can handle 750,000 transactions per second.

Low Latency

Volt uses a blazingly fast C++ core for data management. Volt's design provides the base functionality of an OLTP RDBMS minus many of the overheads. By having a single, dedicated thread for each partition, Volt eliminates the overhead of context-switching and deciding which transaction to work on next, and, in doing so, also eliminates 'long-tail' latency. Volt also keeps all the data in RAM, which removes the need for latency-causing I/O operations. Finally, Volt replicates data as it updates the leading copy, eliminating the extra time required to create backup copies after an event has finished.

ACID Transactions

Volt achieves ACID transactions by only running one transaction at a time, to completion, on each partition. Users extend a Java class that lives on the server. This Java is used to combine individual SQL statements with server-side business logic. Clients send a request with the name of the desired class and its required parameters, which is invisibly routed to all the nodes that have a copy of the data. Dedicated threads on each server process all the work for the partitions. All transactions begin and either succeed or fail during a single trip to the server, and all copies of the data are updated at the same time. This prevents issues with eventual consistency and also enables fast recovery from node outages, as the surviving nodes have up-to-date copies of the data they need.

Cloud Native

Cloud native orchestration tools like Kubernetes originate from the stateless web application world, but many database technologies struggle to handle the complexity of expected automation and state.

Fortunately, Volt is a shared-nothing platform that's well-suited to being cloud native. Volt is available as Docker container images that can be fully orchestrated by Kubernetes and configured and deployed via Helm and the Volt Helm charts.

Volt also has a full-featured Kubernetes operator that manages the Volt specifics in starting/stopping Volt clusters, provides auto healing when the Volt pod fails, and supports autoscaling (up and down) with elastic add and shrink.

Locking

Volt uses deterministic queues of requests in RAM to manage inbound and outbound messages. Multiple requests for the same resource will always be forced to form an orderly queue. It's not possible for two requests to try and access the same data at the same time, so locking issues don't happen.

Global High Availability with Geo-replication

Via Volt Active(N), Volt has production systems running double-, triple-, and even quadruple-active geo-replication across multiple physical sites in production. The two primary drivers for this level of geo-replication are survivability and ensuring low latency for continent-size deployments. Volt automatically resolves conflicts using a 'last write wins' algorithm and tells you what was lost as part of the resolution so that the application can mitigate the situation, if needed.

Volt's implementation streams row changes, and conflicts are resolved on a 'last write wins' basis, with site ID as a tiebreaker. Where other data platforms make the 'losing' row — and with it, a completed transaction — vanish, resulting in lost data, which can lead to serious customer experience issues, Volt streams details of the conflict to Kafka. This leaves an auditable trail of what happened and what was lost, which means you can fix scenarios where, for example, a high-value transaction at time 'x' was erased by a low-value transaction that took place 500 miles away at time 'x plus 1 millisecond'.

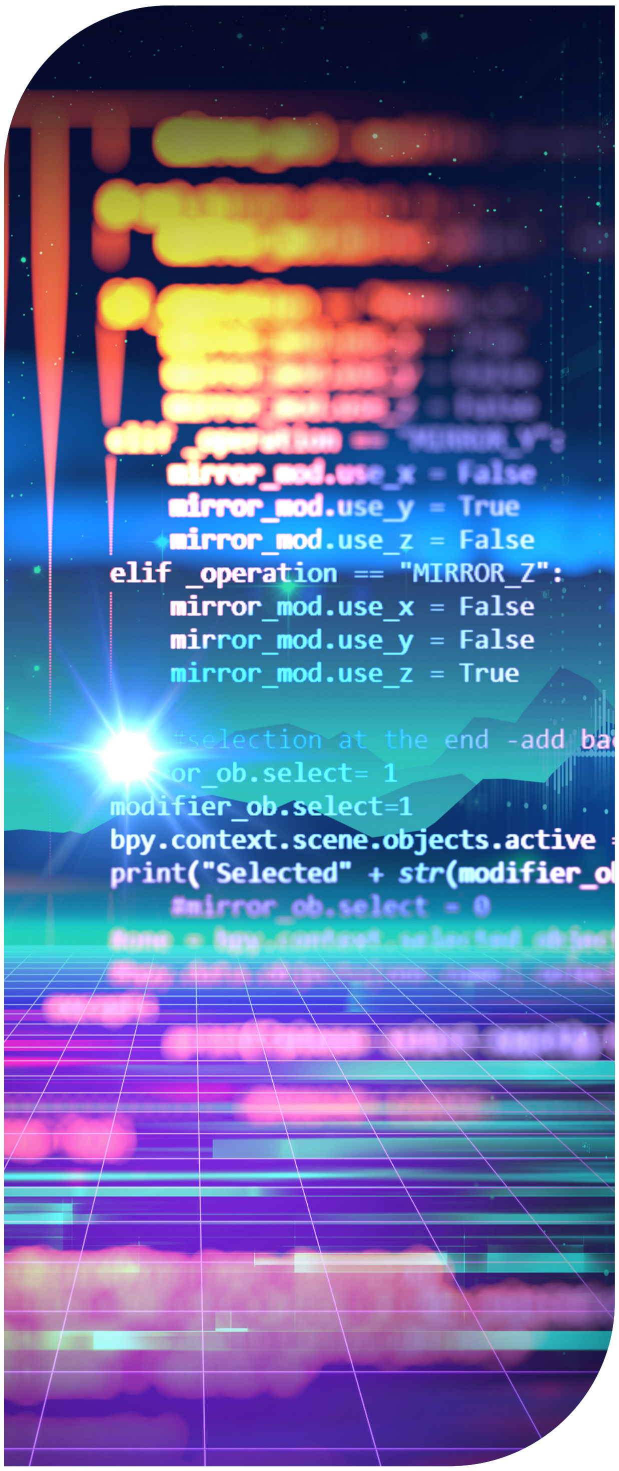
Low Total Cost of Ownership

Volt simplifies your stack, enabling much faster and more efficient data processing.

Specifically, Volt offers:

- Shared-nothing hardware
- No requirement for fancy hardware
- Very efficient use of CPU that's 9x faster than legacy RDBMS and 4x faster than NoSQL

This means Volt has very low running costs. For YCSB on generic AWS hardware (spot instances), Volt can get 500K TPS for a hardware cost of about US$2.50 an hour.



SQL

Volt is ANSI-92 compatible.

It also has extensions for working with streams, migration of data to offline storage, and timed deletion of stale data:

- A MIGRATE DML command moves data from a table to an export stream on command. This can also be done using the TTL option of CREATE TABLE.
- STREAM objects are used to get data from Volt into Kafka in an ACID-compliant manner. CREATE STREAM functions much the same way as CREATE TABLE, except the only legal DML operation is INSERT.
- All VIEW objects in Volt are like TABLE objects in that they have real rows and can be indexed. Adding a view to a table slows down operations by around 1-3%, making them both affordable and useful. Views can also be built on STREAM objects, so you can easily track how many and what kind of records you sent to a STREAM.

High Availability Within a Cluster

For high availability within a cluster, all state lives on a partition that exists on at least two nodes in a cluster (what we call the 'Leader' and 'Follower' partitions). Because of how we process requests, all 'follower' partitions are as close to the leader as they can be. There is no extra latency to copy changes and no eventual consistency.

Streaming Data

Volt is 100% compatible with Kafka. It can be configured to continuously read Kafka and turn each message into a Volt request. This will happen at the speed of Kafka and the scale of Volt.

Volt can also create SQL “TABLES” which are in fact Kafka streams. These can be written to using the SQL INSERT command and follow normal transactional semantics. Volt can do all this while also doing conventional client-server requests at the same time. So with Volt, you can have a request that takes a decision, sends a message back to a client, and at the same time writes a record describing the decision to Kafka. This also means there is no need to do bulk extraction of data from Volt, as it can be streamed out as generated.

Volt can also emulate Kafka at the wire protocol level, which means that a Volt cluster can look and act like a Kafka cluster on the network, despite having a completely different level of capability.

White-label Ready for OEM Partners

For the reasons explained in this document, Volt offers an unparalleled foundation for software vendors and systems integrators to build their own mission-critical applications as an OEM partner. In fact, the majority of Volt’s customers follow this approach. Completely horizontal in scope, Volt allows OEMs to take advantage of its reliability and performance to reduce time to market and deliver robust, high-quality applications to their end users. With years of real-world, field-tested performance, comprehensive ‘telco-grade’ support, rigorous testing, and continuous updates, Volt is the ideal real-time data processing platform for businesses looking to innovate and stay ahead in a competitive landscape.

Proof Points

Scale – Volt has a Telco charging system in APAC that does up to 750,000 TPS per second.

Speed – A major Indian fantasy sports company uses Volt to register 1,000,000 contestants for a single game in 14.5 seconds.

Accuracy – A major UK bank uses Volt to report trades to multiple regulators that have extreme expectations around accuracy and timeliness.

Customer satisfaction – >98% of our customers renew their contracts.

True geo-replication – We have live active-active-active deployments.

Enterprise upgrades – You can upgrade all the nodes of a cluster without any downtime or dropped transactions.

Geo-replication and upgrades – You can independently upgrade schema and code for different sites in a geo-replicated deployment, without downtime.

Consistently low TCO – AWS operating costs for an OLTP application are around US$2.50/hr for each 100,000 TPS supported.

Conclusion: Why You Need Volt

In the end it all comes back to data — processing it quickly, without downtime or errors.

If you can do this in the age of 5G, Edge, and IoT, you unquestionably have a leg up on your competition.

If you're running mission-critical applications that can't afford to go down or lose data, you need Volt.

As we said at the start, Volt's ''raison d'etre' is to support the seamless functioning of mission-critical applications by quickly and reliably processing data and performing many ACID transactions at once, without forcing costly or uncomfortable compromises.

Now more than a decade old, Volt has proven itself in industries from telco to finance to manufacturing to supply chain. All of these industries have one thing in common: the use of mission-critical applications that can't afford to be wrong or go down.

That's why one customer called us "the Fedex of the data world" — because Volt just works. What they meant was they are confident deploying their own mission-critical applications to serve their customers, knowing that Volt can efficiently run the same applications at workloads from a few hundred TPS to hundreds of thousands of TPS, with no code, minimal configuration changes, and no post-deployment behavior changes. This is one of the main reasons our customer renewal rate has been more than 98% for years.

Try Volt Active Data Today

© Volt Active Data, Inc