Aug 1, 2023

Blurring the Lines Between Event-Driven and Data-Driven Architectures

Fusing the dynamic nature of stream processing with the analytical depth of data-driven decision making.

Posted by

Andy Pernsteiner

Consider this part 2 in our series where we explore the possibilities that open up to us when we have a fully integrated store for not only unstructured file and object data, but also for structured data (rows, columns, tables, databases).

In the previous installment we talked about using the VAST Catalog to treat your filesystem like a database. That is to say, running SQL queries against a fully integrated and consistent index of all metadata across an exabyte-scale filesystem and object store.

In the following discussion, we'll specifically focus on the unique attributes of the VAST DataBase and how it interweaves the realms of event-driven and data-driven architectures. Essentially, it's about fusing the dynamic nature of stream processing with the analytical depth of data-driven decision making.

First, let's consider some challenges when building a platform just to handle streaming data:

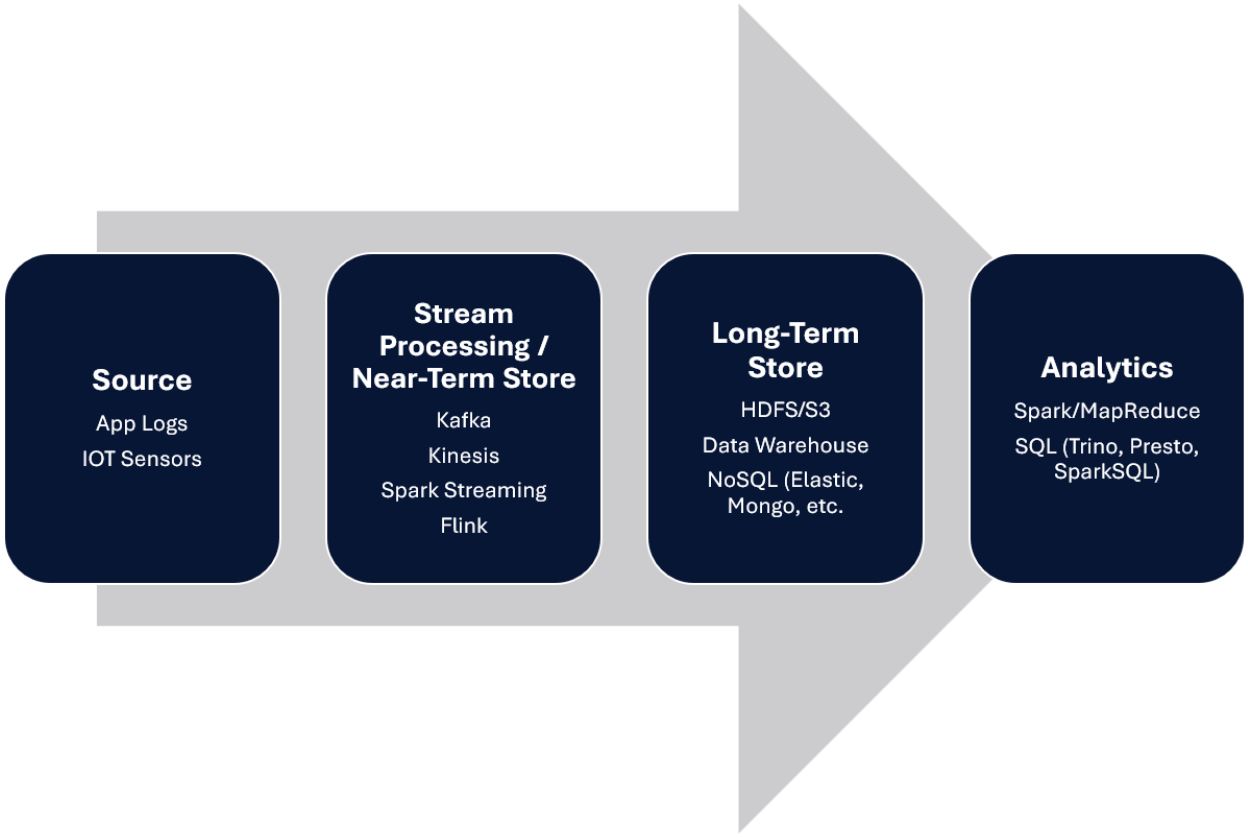
- The high volume and velocity of events, which require systems that can handle high concurrency while maintaining low latency.
- The need to analyze events in real-time without impacting ingestion or other downstream consumers.
- The need to persist data immediately to a resilient store
- The need to scale both ingestion rates and storage capacity as needs increase

Then of course, one needs to consider some things when building an analytical store (aka: a data lake):

- Consistent performance for ad-hoc queries
- Scalable performance for deep analytical queries
- Scaling both capacity and performance (ideally independently as needs change)
- Cost-effectiveness, particularly as it scales
- Support for multiple ingest and access methods and applications

The problem is... it's hard to build both sets of considerations into the same platform. It has historically necessitated a technology approach involving a number of purpose-built components that are well-suited to different parts of the streaming and analytical pipeline.

Your typical streaming design looks something like this:



Basically, as the data comes in (usually in JSON, XML, or similar semi-structured format), instant analysis is accomplished by inserting small batches into a short- term store, which may store a few hours or days (at most) of data. It then needs to be pushed into a longer-term store for analytics, in some cases requiring an ETL layer to normalize and cleanse data, create aggregations and rollups, and optimize for analytical query.

What this means is that queries against the real-time datasets go to one subset of the data, whereas analytics against the historical data are issued against another system, which is out of date. Users and applications need to look in different places to answer their questions.

So… why not just store everything historically in kafka or another streaming system?

The primary design of most real-time stream processing and storage systems is to ensure that ingestion rates can scale, messages are not lost, and messages and events can be made immediately available for real-time analysis. Accomplishing this comes at a cost… namely that these designs cannot scale to store long-term data.

Primarily, this is due to resource utilization and ultimately cost. Data is usually triplicated in these systems to deal with failure handling, which is inefficient and expensive at scale. Such implementations also typically achieve low latency by performing appends to an on-disk structure known as a Write Ahead Log, or WAL. WALs are great for a small amount of data, but must be continually compacted and merged to ensure that querying against the data stream is efficient. Such merge and compaction operations become expensive in terms of I/O and compute, and limit the scale.

The result is that streaming ingestion technologies will recommend and default to keeping no more than 1 week of data, which means you need a place to push it to long-term analytics.

Ok fine, then why not just push all the streaming data directly into a DataLake for immediate query?

In the case of a DataLake, consider that such systems are built upon HDFS and S3 or other object stores, which are effectively immutable filesystems. By immutable, it means that files or objects, once written, cannot be modified. That means you have two choices:

1) Make the batch size extremely small in an attempt to make new records available for near-real time access. This, however, results in quite small files, which are extremely inefficient on HDFS and do not lend well to large-scale analytics.
2) Make the batch size larger to aid in future analytics and get more efficiency in terms of compression and metadata. The issue here is that if the batch size is too

large, data is not made available for real-time analysis, which sort of defeats the purpose.

Note that the above is also true for data lake architectures built upon Iceberg, Hudi, and Delta. They're designed to optimize for analytics, but they cannot handle high rates of ingestion required by low latency streaming applications. This is largely due to the fact that they rely on an immutable underlying object store (S3, ADLS, HDFS, etc), and all inserts must go through a series of compactions and manifest file updates in order to be consistent, resulting in high latency for inserts and a limit to how many events per second can be added to the lake.

The fundamental challenge here is that in order to keep up with real time ingestion rates, you need something that can receive and store new records quickly, at scale, something that DataLake architectures are not designed for.

Solving the Problem

So, in an ideal world, you could have a single platform that could not only handle the ingestion and real-time analysis of streaming data, but also allow for analytical query across the entire dataset, new and old, without having to copy, transform, move, or manage data differently.

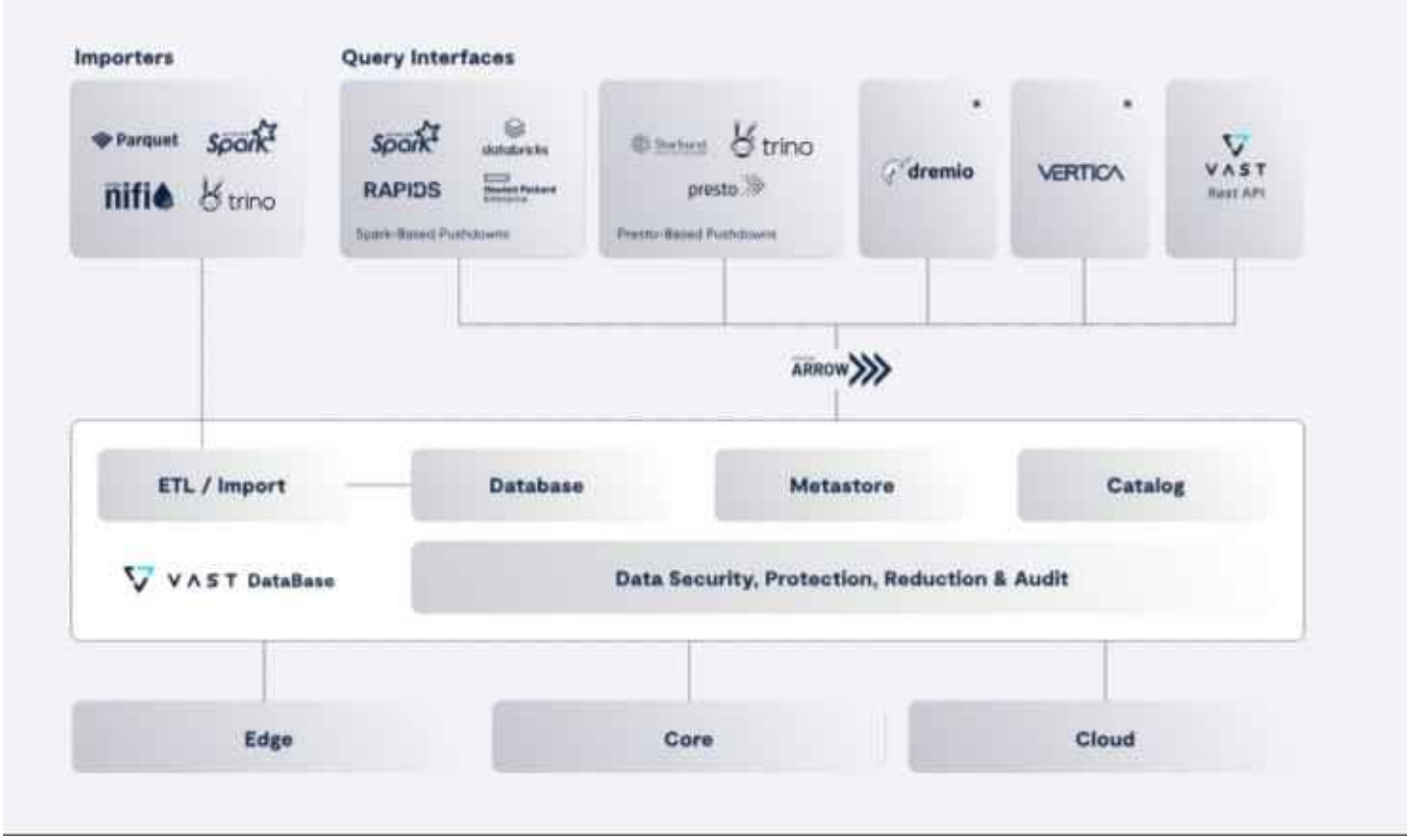
As we mentioned in our previous blog post, VAST has extended the platform to include a scalable, performant, fault-tolerant database. This means the platform allows data architects to use the same system for unstructured data, in the form of files and objects, as well as structured data, in the form of rows and columns. Previously we also discussed the VAST Catalog, which is one use case for the VAST DataBase, but that's only the beginning. When we designed the VAST DataBase, we considered all the challenges listed above and wanted to find a way to break tradeoffs and find a way to build beyond what had been done before.

I won't get into all the particulars of the VAST DataBase here, for that...head on over to our recently updated VAST Whitepaper, which goes into the gory details. That said, here are a few tidbits so you can understand what we did that is so unique.

The VAST DataBase

First of all, like all data that lands on the VAST platform, new inserts, updates, and deletes first go into a low latency layer of flash known as Storage Class Memory (SCM).

The moment it hits SCM, it is acknowledged to the user or application and is ready to be included in queries. What's more, all CRUD operations to VAST DataBase tables are ACID compliant, and can be wrapped in user-initiated transactions. These transactions can span across multiple tables, enabling sophisticated conditional updates to tables. This can greatly simplify application design, as the DataBase can be relied upon as a true system of record.



This process of handling ingestion into SCM has been used for file and object applications on VAST for years, enabling applications to perform millions of operations per second for both random and sequential workload profiles.

Ok, so: the VAST DataBase is fast for inserts, is ACID compliant, and is highly scalable.

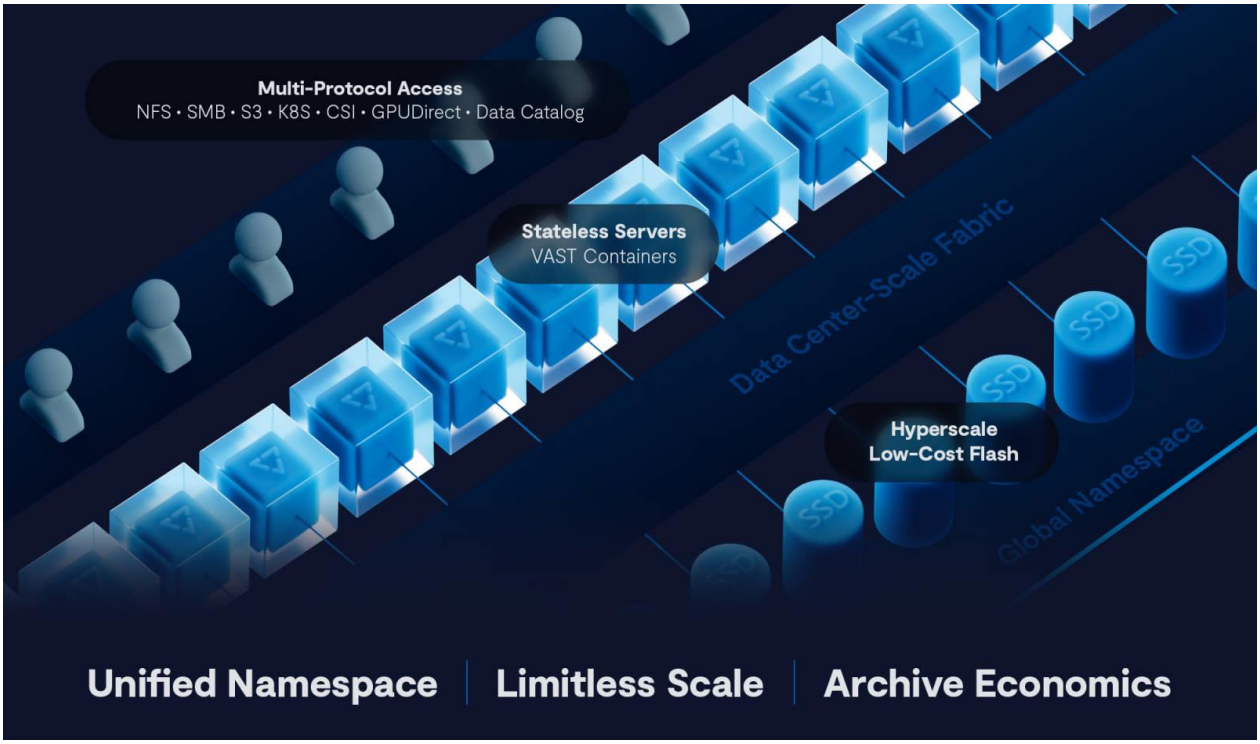
But how do we handle queries against all the historical data? Well, as data is written into the SCM buffers, those buffers fill. When they reach a point of fullness, the platform automatically performs a migration process to lay it out in efficient stripes in lower cost,

scalable flash. For structured or columnar data, this process also optimizes how the data is laid out to be extremely efficient for queries. In essence, statistics and other metadata are compiled about what is stored in the actual records, and this metadata is written to SCM while the actual columnar data is migrated down to flash.

As the data is migrated, pointers within SCM are updated to reflect the current location, to ensure that any queries are served based on the most up-to-date information, regardless of where it happens to live underneath the covers.

The metadata in SCM allows the system to quickly find data requested by the query, and significantly reduces the amount of I/O and processing required to retrieve the result set. Queries against large-scale tables (billions or trillions of records) are parallelized within the platform automatically, allowing analysis to scale along with the dataset, regardless of size.

And don't forget, this is all based upon the Disaggregated Shared Everything (DASE) architecture VAST invented. This means that performance and capacity can scale independently and enable just-in-time adjustments should performance or capacity needs change over time. Every piece of data can be read in parallel from many machines simultaneously, which lends itself well to highly clustered applications such as Spark and Trino.



In essence:

- Ingest is fast
- Queries are fast
- Tables can scale (trillions of records, thousands of columns)
- Performance scales

Recall also that this is all housed within the VAST Data Platform, alongside the VAST DataStore, where files and objects live. To take advantage of that, we've included a Parquet file importer into the DataBase. This means that existing ETL pipelines that output to Parquet format can simply point their output at a bucket or export on a VAST cluster, and that data can be instantly imported into the VAST DataBase's optimized table format, enabling all the benefits outlined above.

The DataBase for ML/DL

What we've done here is created a system of record that can not only be used to handle high rates of streaming events as they come in, but also give users the ability to analyze all data together. This greatly simplifies not only the design but also the deployment and maintenance of a combined streaming and analytical platform.

More importantly though, it enables the next generation of machine learning and deep learning applications to be able to not only perform deep analysis, but also have a constantly fresh set of data to work with. Models can be re-trained and tested more quickly against incoming data.

Have we truly merged the worlds of event-driven and data-driven architectures? We think so…but tell us what you think.

As I've said before…we're just getting started, and can't wait to see what our customers build!