Why do we need a time-series database?

赵书 2019141420228

These two years the Internet industry has a new wind, always listening to a variety of lofty new terms.

Previous system, do data visualization, information management, process control. Businesses are no longer satisfied with simple management and control. Data visualization analysis, big data information mining, statistical prediction, modeling and simulation, and intelligent control have become the pursuit of various businesses.

"All is lost in time like tears. Time is dying." We used to use the Internet to solve real problems. Now we are not satisfied with the reality, the data will be connected into time series, the history can be viewed in advance, the regularity can be revealed, the trend can be grasped in the future, the trend can be predicted.

We began to store a large amount of data, summarized the structural characteristics and common usage scenarios of these data, and continuously improved and optimized, creating a new database classification -- Time Series Database.

A time series database (TSDB) is a database optimized for time-stamped or time series data. Time series data are simply measurements or events that are tracked, monitored, downsampled, and aggregated over time. This could be server metrics, application performance monitoring, network data, sensor data, events, clicks, trades in a market, and many other types of analytics data.

A time series database is built specifically for handling metrics and events or measurements that are time-stamped. A TSDB is optimized for measuring change over time. Properties that make time series data very different than other data workloads are data lifecycle management, summarization, and large range scans of many records.

Time series databases are not new, but the first-generation time series databases were primarily focused on looking at financial data, the volatility of stock trading, and systems built to solve trading. But financial data is hardly the only application of time series data anymore — in fact, it’s only one among numerous applications across various industries. The fundamental conditions of computing have changed dramatically over the last decade. Everything has become compartmentalized. Monolithic mainframes have vanished, replaced by serverless servers, microservers, and containers

Today, everything that can be a component is a component. In addition, we are witnessing the instrumentation of every available surface in the material world — streets, cars, factories, power grids, ice caps, satellites, clothing, phones, microwaves, milk containers, planets, human bodies. Everything has, or will have, a sensor. So now, everything inside and outside the company is emitting a relentless stream of metrics and events or time series data.

This means that the underlying platforms need to evolve to support these new workloads — more data points, more data sources, more monitoring, more controls. What we’re witnessing, and what the times demand, is a paradigmatic shift in how we approach our data infrastructure and how we approach building, monitoring, controlling, and managing systems. What we need is a performant, scalable, purpose-built time series database.

In the middle of the big data hype cycle, before IoT was on the top of everyone’s buzz list, before Cloud Native was common lingo and before large enterprises were starting the work to de-silo their infrastructure monitoring and metrics data, Paul Dix, Founder of InfluxData, began building a purpose-built Time Series Platform. Flash forward to today, time series is now the fastest growing database segment, and the market is clearly moving beyond the re-purposed Cassandra and Hbase implementations that defined the segment at that time. The following post is a firsthand account from Paul Dix outlining the problems he witnessed and why he built a modern Time Series Platform.

I am frequently asked: "Why build a database specifically for time series?" The implication was that a general SQL database can act as a TSDB by ordering on some time column. Or you can build on top of a distributed database like Cassandra. While it’s possible to use these solutions for solving time series problems, they’re incredibly time consuming and require significant development effort.. I talked to other engineers to see what they had done and found that there was a common set of tasks that led to the need for a common Time Series Platform. Everyone seemed to be reinventing the wheel, so it looked like there was a gap in the market for something built specifically for time series.

In this post, I'll define the time series problem, lay out what differentiates time series from other use cases and database workloads and look at other approaches I've seen taken to handle the unique requirements of time series data. Finally, I'll look at the advantages of building specifically for time series.

Many of our users started off working with time series by storing their data in common SQL RDBMSes like PostgreSQL or MySQL. Generally they find this works for a time, but things start to fall apart as the scale of the data increases. If we take our server monitoring example from before, there are a few ways to structure things, but there are some challenges.

Structure OptionsChallengesConclusionCreate a single table to store everything with the series name, the value, and a time.Separate lookup index if we wanted to search on anything other than the specific name (like server, metric, service, etc.).

This naive implementation would have a table that gets 172M new records per day. This would quickly cause a problem because of the sheer size of the table.With time series, it's common to have high-precision data that is kept around only for a short period of time.

This means that soon you'll be doing just as many deletes as inserts, which isn't something a traditional DB is designed to handle well.Create a separate table per day or some other period of time.Requires the developer to create application code to tie the data from the different tables together.More code must be written to compute summary statistics for lower-precision data and to periodically drop old tables.Then there's the issue of scaling past what a single SQL server can handle. Sharding segments of the time series to different servers is a common technique but requires more application-level code to handle it.

Conclusion: Relational technologies were not designed to solve the specific time series issues, and trying to get them to solve them is impractical.

So this brings us back around to the point of this post: Why build a Time Series Data Platform?

Developer Happiness

One of our goals we envisioned when making a Time Series Platform was optimizing for a user’s or developer’s time to value. That is, the faster they get their problem solved and are up and running, the better the experience will be. That means that if we see users frequently writing code or creating projects to solve the same problems, we’ll try to pull that into our platform or database. The less code a developer has to write to solve their problem, the faster they’ll be done.

Time is Peculiar

Other than the obvious usability goals, we also saw that we could optimize the database around some of the peculiarities of time series. It’s insert only, we need to aggregate and downsample, we need to automatically evict high-precision data in the cases where users want to free up space. We could also build compression that was optimized for time series data. We also organized the data in a way that would index tag data for efficient queries. At the database level, there were many optimizations we could get.

Going Beyond a Database to Make Development Easier

The other advantage in building specifically for time series is that we could go beyond the database. We’ve found that most users run into a common set of problems they need to solve—how to collect the data, how to store it, how to process and monitor it, and how to visualize it.

We’ve also found that having a common API makes it easier for the community to build solutions around our stack. We have the line protocol to represent time series data, our HTTP API for writing and querying, and Kapacitor for processing. This means that over time, we can have pre-built components for the most common use cases.

We find that we can get better performance than more generalized databases while also reducing the developer effort to get a solution up by at least an order of magnitude. Doing something that might have taken months to get running on Cassandra or MySQL could take as little as an afternoon using our stack. And that’s exactly what we’re trying to achieve.

By focusing on time series, we can solve problems for application developers so that they can focus on the code that creates unique value inside their app.

Database architecture is about trade-offs and priorities. Do you need speed or accuracy or volume or predefined schemas? The proof is in the benchmarks. Measure everything. Don’t choose a tool or a product—choose a solution to your problem. Specialty tools are made for special problems, so time series databases are optimized for time series problems.

Once you begin to see more of the information your applications store as time-series data, you still have to pick a time-series database that best fits your data model, write/read pattern, and developer skillsets. Although NoSQL time-series database options have prevailed for the past decade as the storage medium of choice, more and more developers are seeing the downside to storing time-series data separately from business data (most time-series databases don’t provide good support for relational data). In fact, this poor developer experience was one of the driving factors in why we started Timescale. Keeping all of your data in one system can drastically reduce application development time – and the speed at which you can make key decisions.

Nowhere is this more evident than with the rise of numerous self-service business intelligence tools like Tableau, Power BI, and yes, even Excel. When precious time-series data is kept separate from business data, users struggle to make timely, business-critical observations. Instead, users find that they need to rely on these third-party tools to mash up data into something meaningful. There are many valid and good reasons to use these powerful tools, but being able to quickly query your time-series data alongside meaningful metadata information shouldn’t be one of them. SQL has been built and honed over decades to provide efficient ways of generating these valuable aggregations and analyses.

The bottom line: knowing where your time-series data is and where you store it can have a dramatic impact on your future success.