Scaling Automated Database Monitoring at Uber

... with M3 and Prometheus

Richard Artoul
01 Automated database monitoring
02 Why scaling automated monitoring is hard
03 M3 architecture and why it scales so well
04 How you can use M3
Uber’s “Architecture”

2015

2019

4000+ microservices - Most of which directly or indirectly depend on storage

22+ storage technologies - Ranging from C* to MySQL

1000’s of dedicated servers running databases - Monitoring all of these is hard
Monitoring Databases
Hardware Level Metrics
Technology Level Metrics

- Total Reads/s (includes replication)
- Total Writes/s (includes replication)
- Total RangeQueries/s (includes replication)
- Reads/s per Host (includes replication)
- Writes/s per Host (includes replication)
- RangeRequests/s per host (includes replication)
- Reads/s per Keyspace (includes replication)
- Writes/s per Keyspace (includes replication)
- RangeRequests/s per Keyspace (includes replication)
Application Level Metrics

- Number of successes, errors, and latency broken down by
  - All queries against a given database
  - All queries issued by a specific service
  - A specific query
    - `SELECT * FROM TABLE WHERE USER_ID = ?`
What’s so hard about that?
Monitoring Applications at Scale

1200  Microservices w/ dedicated storage
100   Instances per service
20    Instances per DB cluster
20    Queries per service
10    Metrics per query

480 million dollars worth of time series
Workload

800M Datapoints emitted pre-aggregation
110M Writes per second/s (post replication)

200B Datapoints read per second
9B Unique Metric IDs
How do we do it?
A Brief History of M3

- 2014-2015: Graphite + WhisperDB
  - No replication, operations were ‘cumbersome’
- 2015-2016: Cassandra
  - Solved operational issues
  - 16x YoY Growth
  - Expensive (> 1500 Cassandra Hosts)
  - Compactions => R.F=2
- 2016-Today: M3DB
M3DB

An open source distributed time series database

- Store arbitrary timestamp precision data points at any **resolution** for any retention
- Tag (key/value pair) based **inverted index**
- Optimized file-system storage with **minimal need for compactions of time series data for real-time workloads**
High-Level Architecture

Like a Log Structured Merge Tree (LSM)

Except a typical LSM has levelled or size based compaction

M3DB has time window compaction
Topology and Consistency

- Strong consistent topology (using etcd)
  - No gossip
  - Replicated with zone/rack aware layout and configurable replication factor
- Consistency managed via synchronous quorum writes and reads
  - Configurable consistency level for both read and write
Cost Savings and Performance

● Disk Usage in 2017
  ○ ~1.4PB for Cassandra at R.F=2
  ○ ~200TB for M3DB at R.F=3

● Much higher throughput per node with M3DB
  ○ Hundreds of thousands of writes/s on commodity hardware
But what about the index?
Centralized Elasticsearch Index

- Actual time series data was stored in Cassandra and the M3DB
- But indexing of data (for querying) was handled by Elasticsearch
- Worked for us for a long time
- … but scaling writes and reads required a lot of engineering
Elasticsearch Index - Write Path

Influx of new metrics:
1. Large Service Deployment
2. Datacenter Failover

"Don’t write cache"
Elasticsearch Index - Read Path

1. Query to E.S
2. Query to M3DB
Elasticsearch Index - Read Path

Query → Redis Query Cache → E.S

Need high T.T.L to prevent overwhelming E.S

... but high T.T.L means long delay for new time series to become queryable
Elasticsearch Index - Read Path

Query → Merge on read → Redis Query Cache

Short TTL

Redis Query Cache → E.S Short

Long TTL

Redis Query Cache → E.S Long
Elasticsearch Index - Final Breakdown

- Two Elasticsearch clusters with separate configuration
- Two query caches
- Two don’t-write caches
- A stateful indexing tier that requires consistent hashing, an in-memory cache, and breaks everything if you deploy it too quickly
- Another service just for automatically rotating the short-term Elasticsearch cluster indexes
- A background process that’s always running and trying to delete stale documents from the long term index
M3 Inverted Index

- Not nearly as expressive or feature-rich as Elasticsearch
- … but like M3DB, it’s designed for high throughput
- Temporal by nature (T.T.Ls are cheap)
- Fewer moving parts
M3 Inverted Index

- Primary use-case, support queries in the form:
  - service = “foo” AND
  - endpoint = “bar” AND
  - client_version regexp matches “3.*”
M3 Inverted Index - F.S.Ts

- The inverted index is similar to Lucene in that it uses F.S.T segments to build an efficient and compressed structure for fast regexp.
- Each time series’ label has its own F.S.T that can be searched to find the offset to unpack another data structure that contains the set of metrics IDs associated with a particular label value (postings list).

**Encoded Relationships**

- are --> 4
- ate --> 2
- see --> 3

Compressed + fast regexp!
M3 Inverted Index - Postings List

- For every term combination in the form of service="foo" need to store a set of metric IDs (integers) that match, this is called a **postings list**.
- Index is broken into **blocks** and **segments**, so need to be able to calculate **intersection** (AND - across terms) and **union** (OR - within a term).

![Diagram](image)

Primary data structure for the postings list in M3DB is the roaring bitmap
M3 Inverted Index - File Structure
M3 Summary

- Time series compression + temporal data and file structures
- Distributed and horizontally scalable by default
- **Complexity pushed to the storage layer**
How you can use M3
M3 and Prometheus

Scalable monitoring
Directly query M3 using coordinator for single Grafana datasource
Roadmap
Roadmap

- Bring the Kubernetes operator out of Alpha with further lifecycle management
- Arbitrary out of order writes for writing data into the past and backfilling
- Asynchronous cross region replication (for disaster recovery)
- Evolving M3DB into a generic time series database (think event store)
  - Efficient compression of events in the form of Protobuf messages
We’re Hiring!

- Want to work on M3DB? We’re hiring!
  - Reach out to me at rartoul@uber.com