Evolution of the Prometheus TSDB
Who am I?

Engineer passionate about running software reliably in production.

- Core developer of Prometheus
- Studied Computer Science in Trinity College Dublin.
- Google SRE for 7 years, working on high-scale reliable systems.
- Contributor to many open source projects, including Ansible, Python, Aurora and Zookeeper.
- Founder of Robust Perception, provider of commercial support and consulting for Prometheus.
Prometheus

Inspired by Google’s Borgmon monitoring system.

Started in 2012 by ex-Googlers working in Soundcloud as an open source project, mainly written in Go. Publically launched in early 2015, and continues to be independent of any one company. Incubating with the CNCF.

Over 500 companies have started relying on it since then.
Prometheus does Metrics

Prometheus is a metrics-based monitoring system.

It tracks overall statistics over time, not individual events.

It has a Time Series DataBase (TSDB) at its core.
Powerful Data Model and Query Language

All metrics have arbitrary multi-dimensional labels.

Supports any double value with millisecond resolution timestamps.

Can multiply, add, aggregate, join, predict, take quantiles across many metrics in the same query. Can evaluate right now, and graph back in time.

Can alert on any query.
Reliability is Key

Core Prometheus server is a single binary.

Each Prometheus server is independent, it only relies on local SSD.

No clustering or attempts to backfill "missing" data when scrapes fail. Such approaches are difficult/impossible to get right, and often cause the type of outages you're trying to prevent.

Option for remote storage for long term storage.
Growing the Database

The Prometheus codebase is 5 years old.

The core ideas have remained largely the same.

What has changed is the scale and usage patterns.

Let's look at how the storage evolved over the years.
v1: The Beginning

For the first 2 years of its life, Prometheus had a basic implementation. All time series data and label metadata was stored in LevelDB.

The label metadata was a map of all (name, value) pairs of labels to the fingerprints of the metrics. This allows for looking up metrics by label. And a map of (name,) to fingerprints to allow for non-equality matchers.
v1: Outcome

Each scrape had its data buffered in memory.

Data was flushed to LevelDB every 15 minutes.

If Prometheus was shutdown, data was lost.

Ingestion topped out around 50k samples/s.

Enough for 500 machines with 10s scrape interval and 1k metrics each.
Why Metrics TSDBs are Hard

Writes are vertical, reads are horizontal.

Write buffering is essential to getting good performance.
v2: Improvements

v2 was written by Beorn, and addressed some of the shortcomings of v1. It was released in Prometheus 0.9.0 in January 2015.

Time series data moved to a file per time series. Writes spread out over ~6 hours. Double-delta compression, 3.3B/sample.

Regular checkpoints of in-memory state.
v2: Additional Improvements

Over time, various other aspects were improved:

Basic heuristics were added to pick the most useful index.

Compression based on Facebook Gorilla, 1.3B/sample.

Memory optimisations cut down on resource usage.

Easier to configure memory usage.
v2: Outcome

Much more performant, the record is ingestion of 800k sample/s.

Not perfect though. That big a Prometheus takes 40-50m to checkpoint.

Doesn't deal well with churn, such as in highly dynamic environments. Limit of on the order of 10M time series across the retention period.

Write amplification is an issue due to GC of time series files.

LevelDB has corruption and crash issues now and then.
Where to go?

We need something that:

- Can deal with high churn
- Is more efficient at label indexing
- Avoids write amplification

Supporting backups would be nice too.
v3: The New Kid on the Block

Prometheus 2.0 will have a new TSDB, written by Fabian.

Data is split into blocks, each of which is 2 hours. Blocks are built up in memory, and then written out. Compacted later on into larger blocks.

Each block has an inverted index implemented using posting lists.

Data accessed via mmap.

A Write Ahead Log (WAL) handles crashes and restarts.
v3: Outcome

It is early days yet, but millions of samples ingested per second is certainly possible.

Read performance is also improved due to the inverted indexes.

Memory and CPU usage is already down ~3X, due to heavy micro-optimisation.

Disk writes down by ~10X.
Review

Prometheus storage has gone through 3 revisions:

- LevelDB with some in memory buffering
- Custom file per time series, LevelDB for indexing
- Custom time-block design, with posting lists for indexing

Each improved upon the previous, allowing performance to go from 50k samples/s to millions of samples/s and working better in more dynamic environments.
Resources

Official Project Website: prometheus.io

User Mailing List: prometheus-users@googlegroups.com

v3 Storage Talk: https://www.youtube.com/watch?v=b_pEevMAC3I

Training: training.robustperception.io

Blog: www.robustperception.io/blog

Queries: prometheus@robustperception.io