Pain points with M3 and how replication works

Rob Skillington
M3 and M3DB team

Why would you try to run anything clustered?

Most people don’t (and rightly so)

Why WOULD YOU NOT try to run anything clustered?

Running large distributed systems typically takes a lot of work and learning a concrete type of distributed stateful system well (e.g. Cassandra)

Sometimes involve several magnitudes more time than learning a programming language

Why would you try to run anything clustered?

Have so many metrics that single applications metrics need to be collected by > 1 single monitoring node

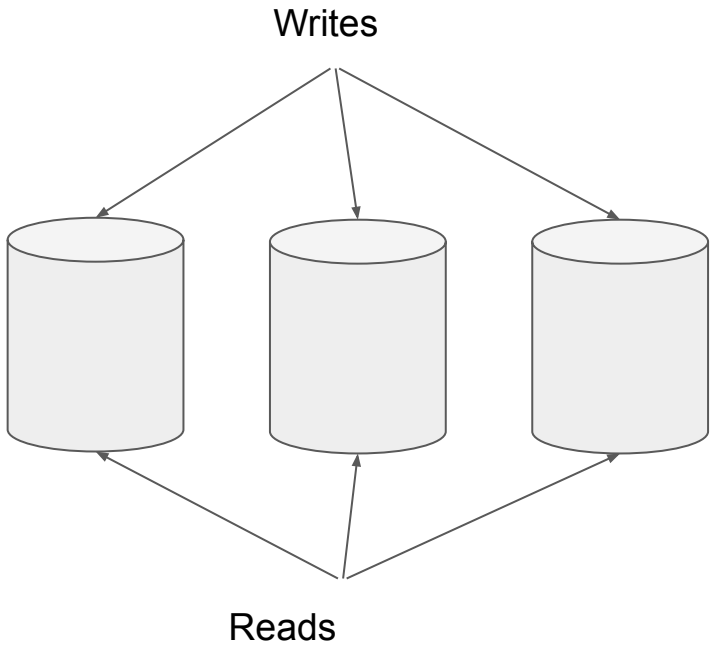
You depend heavily on metrics, especially when things are failing in your environment, so immediate replication is required (ideally RF=3 across 3 availability systems)

Why would you try to run anything clustered?

You want to be able to lose an ephemeral instance (i.e. using local SSD instances in cloud) and everything keeps working

Quick background

- Must run at least RF=3 for high availability with default quorum writes



Categories of pain

- Bootstrap times with large instances
- Memory
  - How much memory am I _actually_ using?
  - How can I control how much memory M3DB uses?
- Downsampling
  - Works well with sidecar model
  - Harder with central solution

Bootstrap times

- For DB nodes running with large heap sizes, bootstrap can take up to 30-90 minutes 😫😫
- This is “ok” (no not really but still) since if you only ever lose or deploy one node at a time, other replicas are up serving quorum writes and reads
- Even if you lose two or more replicas at a time, the node actually accepts writes while it bootstraps so you don’t lose metrics that you’re ingesting
- Now moving to deferred and asynchronous merging of datapoints https://github.com/m3db/m3/pull/1989

Memory

- How much memory am I _actually_ using?
  - Mapped mmap pages will not be released by the OS and counts towards RSS - only evicted from page cache when node is experiencing memory pressure
  - Makes it hard to monitor how much memory a node is _really_ using since is it 60gb or 20gb? How much of that is mapped vs heap used by a Go program's data structures and Go runtime for GC?
  - Now moving to calling madvise DONT_NEED after checksumming mmap'd files, removes pages from cache until used for a read https://github.com/m3db/m3/pull/2037

Memory

- How do I control my memory?
  - Need to be careful how you set the block size, this allows for fine tuning for your workload but also requires knowing your workload...
  - How do you know what's right for your workload until you've seen your workload using a configuration that is close to correct?
  - … unfortunately mainly try and try again
  - Now moving to allowing for "hybrid compaction" where time series that are very unlikely to be appended again are moved to disk before the end of a block time window, and linked list buffers are used instead of actively compacted if series already exists on disk for that block (no re-encoding)

Downsampling

- Required if you want to keep downsampled data or perform aggregations of sum(rate(foo[1m])) by (bar) across millions of time series on way into TSDB
- Sidecar model much easier, since stateful downsampling happens right next to your stateful Prometheus instance
    - Really should use memory and CPU limits either with cgroups or if running in kubernetes then with pod limits
- Central model requires running a cluster of M3 aggregators that uses leader election for failover of aggregation nodes
- If do not want to run as a sidecar - the Kubernetes M3DB operator could be extended to operate the aggregator