Automatically Make Dashboards 100x Faster

Shreyas Srivatsan, Chronosphere
November 20, 2020

Who am I?

Shreyas Srivatsan

Software Engineer @Chronosphere

- Hosted metrics & monitoring platform
- Large scale, high throughput use cases
- Built on M3

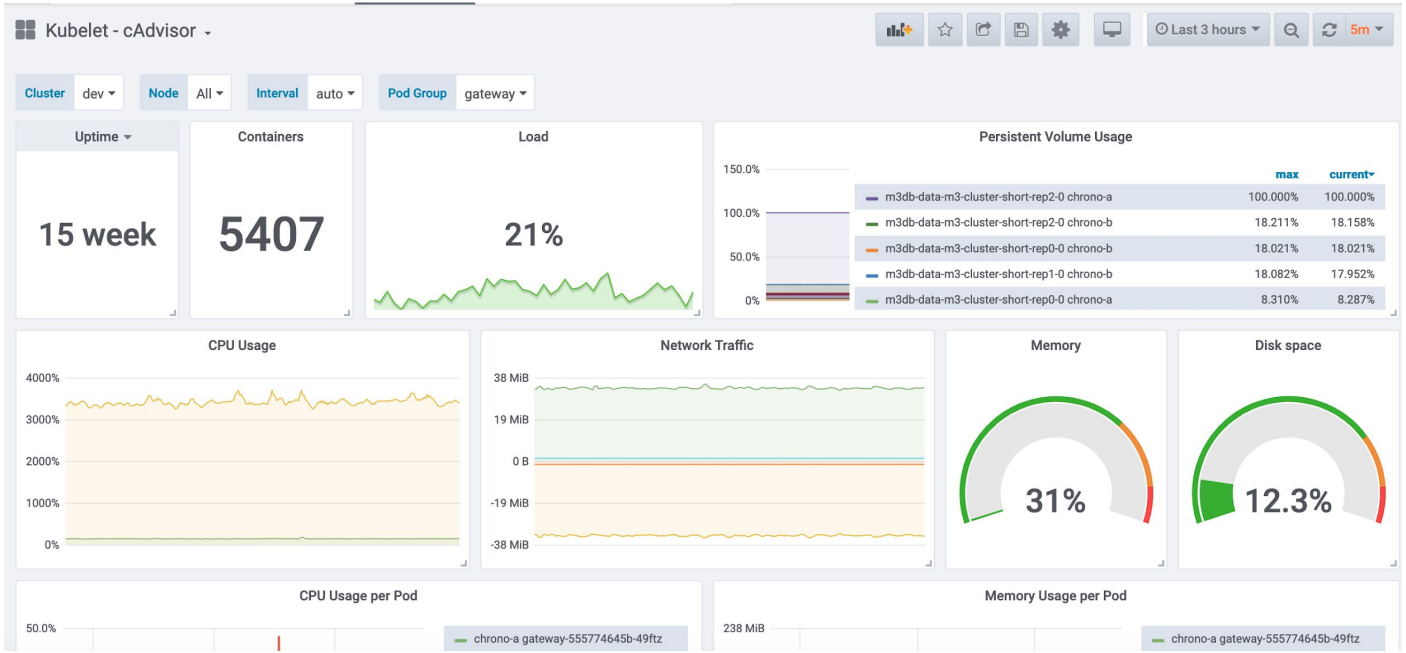
Previously Observability @Uber

Agenda

- The Problem
- Aggregating Metrics - Recording Rules and M3 Aggregation Tier
- Making Things Easy to Use
- Demo
- Q&A

High Cardinality Metrics Example

cAdvisor - resource usage and performance metrics of running containers

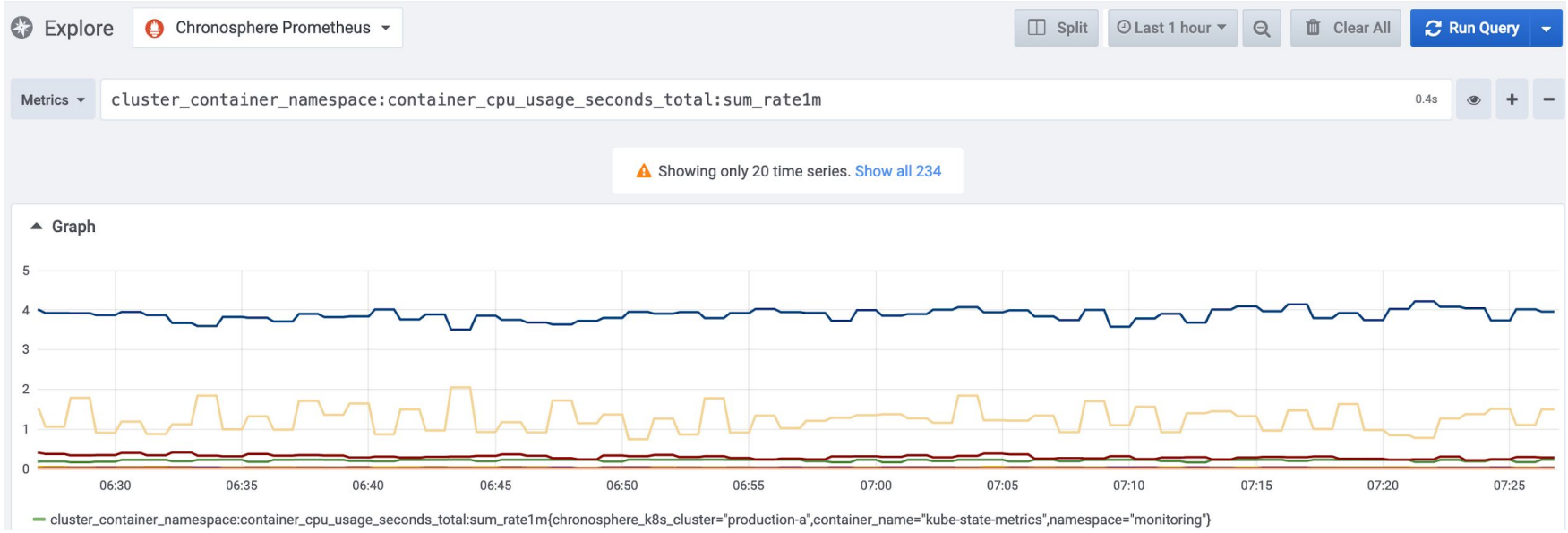


High Cardinality Metrics Example

Container CPU usage has ~16k series and takes 20s to query...

High Cardinality Metrics Example

Same metric but aggregated to just two labels - ~230 series, 0.4s to query



Slowing Dashboards

- Cardinality of dimensions keep increasing
  - Add new instances, roll out new images etc
- Slower dashboard loads and eventually the browser locking up
- Engineer notices this and needs to optimize dashboard

Debugging Slow Dashboards

First step - figure out which queries are the culprit

- Inspecting the requests from a dashboard to look for slow queries
- Can use Prometheus query log, but associating back to dashboard is difficult

Second Step - pre-aggregate metrics, to make queries faster

Recording Rules

- Prometheus provides support for recording rules
  - Allows pre-computing queries and storing back aggregate time series to the TSDB
  - Dashboard can now be pointed at pre-computed time series

```
- record: cluster_container_namespace:container_cpu_usage_seconds_total:sum_rate1m
  expr: sum(rate(container_cpu_usage_seconds_total[1m])) by (container_name, namespace)
```

Recording Rules

- Need to know what to pre-compute
  - Figure out bad queries by analyzing dashboard
  - Configure the recording rules
  - Change dashboard to query the recording rule metrics
- What happens when metric changes or a second panel becomes slow?
  - Repeat process all over again

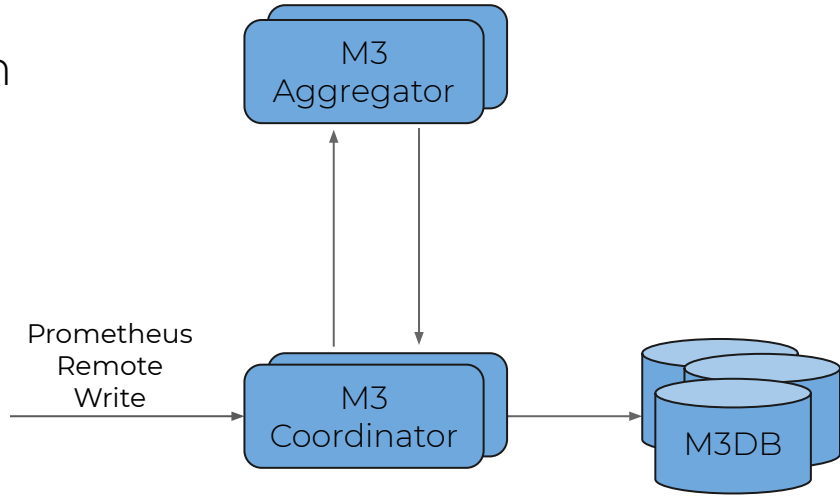
Recording Rules are Expensive

- Recording rules execute and pre-compute the query at regular intervals
- Queries accessing many time series can get expensive very quickly
- Potential to overwhelm the query engine

But, we do not always need the underlying metrics. The underlying dimensions can be dropped and not stored.

M3 Aggregation Tier

- M3 is a remote storage for Prometheus
- Move expensive recording rule computation to streaming aggregation
- Aggregator allows downsampling, dropping or aggregating metrics prior to persisting to M3DB
- Rollup rules allow aggregating metrics
- Mapping rules allow dropping metrics



Rollup Rules

Rollup rules contain a series of transforms applied in order. Metrics applied to depend on the filter match.

Step 1: Take delta

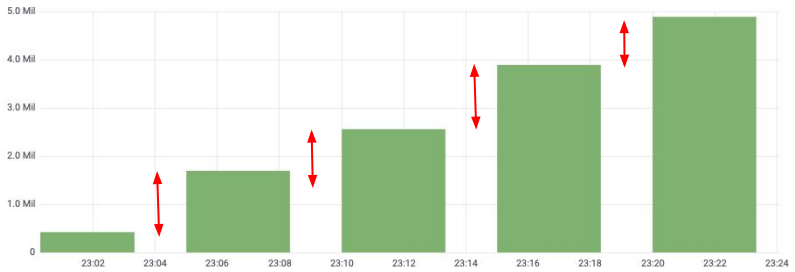
Step 2: Sum by dimension

Step 3: Create monotonic cumulative counter.

```
- name: "cAdvisor CPU usage aggregate"
  filter: "__name__:container_cpu_usage_seconds_total namespace:* le:*
name:* instance:* container_name:*"
  transforms:
  - transform:
      type: "Increase"
  - rollup:
      metricName: "container_cpu_usage_seconds_total"
      groupBy: ["container_name", "namespace"]
      aggregations: ["Sum"]
  - transform:
      type: "Add"
  storagePolicies:
  - resolution: 30s
    retention: 720h
```

Rollup Rules - Deep Dive

Take deltas of the underlying series, so that they can be used by the actual rollup




```
- name: "cAdvisor CPU usage aggregate"
  filter: "__name__:container_cpu_usage_seconds_total namespace:* le:*
name:* instance:* container_name:*"
  transforms:
  - transform:
      type: "Increase"
  - rollup:
      metricName: "container_cpu_usage_seconds_total"
      groupBy: ["container_name", "namespace"]
      aggregations: ["Sum"]
  - transform:
      type: "Add"
  storagePolicies:
  - resolution: 30s
    retention: 720h
```

Rollup Rules - Deep Dive

Sum the deltas by unique dimension specified in the group by. So a rollup for each unique container_name and namespace.

```
- name: "cAdvisor CPU usage aggregate"
  filter: "__name__:container_cpu_usage_seconds_total namespace:* le:*
name:* instance:* container_name:*"
  transforms:
  - transform:
      type: "Increase"
  - rollup:
      metricName: "container_cpu_usage_seconds_total"
      groupBy: ["container_name", "namespace"]
      aggregations: ["Sum"]
  - transform:
      type: "Add"
  storagePolicies:
  - resolution: 30s
    retention: 720h
```

Rollup Rules - Deep Dive

Perform a cumulative add for each of the metrics to get the aggregated time series.

This is sent to the M3DB namespaces identified by the storage policies.

```
- name: "cAdvisor CPU usage aggregate"
  filter: "__name__:container_cpu_usage_seconds_total namespace:* le:*
name:* instance:* container_name:*"
  transforms:
  - transform:
      type: "Increase"
  - rollup:
      metricName: "container_cpu_usage_seconds_total"
      groupBy: ["container_name", "namespace"]
      aggregations: ["Sum"]
  - transform:
      type: "Add"
  storagePolicies:
  - resolution: 30s
    retention: 720h
```

Mapping Rules

Mapping rules allow us to drop metrics based on the filter - that is all the original unaggregated series.

Aggregate series can take the same name as original metric.

```
- name: "cAdvisor CPU usage drop unaggregated rule"
  filter: "__name__:container_cpu_usage_seconds_total namespace:* le:* name:* instance:* container_name:*"
  drop: true
```

M3 Aggregation Tier - Summary

- Allows for ingestion time streaming aggregation
- Metrics can be aggregated or rolled up based on defined rules
- Raw metrics can be dropped based on matching filters

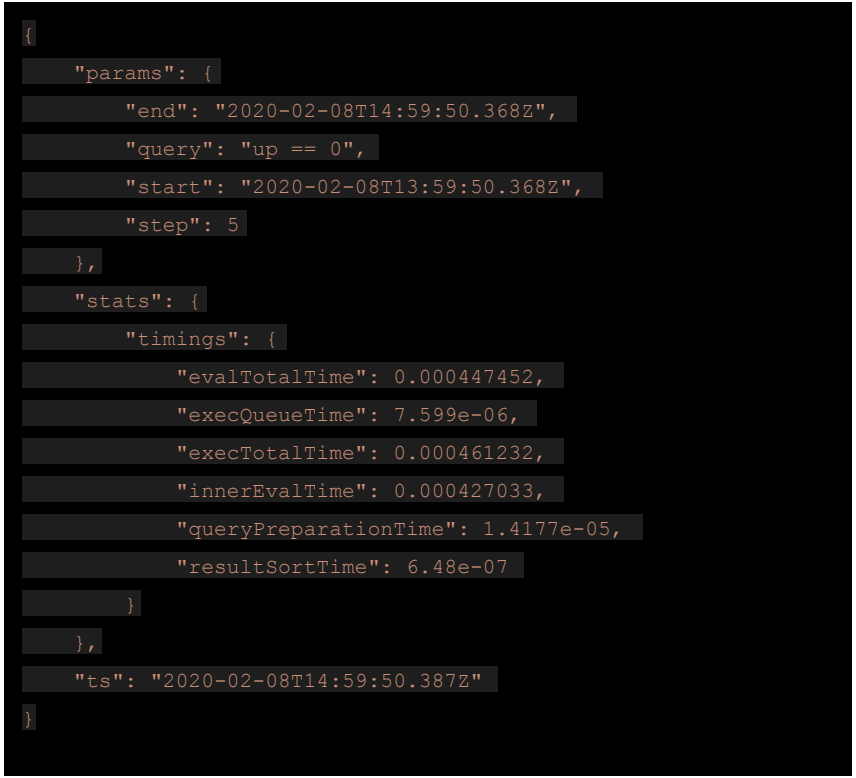
Recording Rules vs Rollup Rules

- Recording rules
  - General purpose and support full PromQL
  - Expensive, runs against the query engine so affects other queries
  - All data needs to be stored so high storage cost
- Rollup rules
  - Much more efficient to run as an ingestion time aggregation
  - Only store the aggregates we need, drop other series
  - Automatic query speedup as aggregate can have same metric name
  - Does not support full PromQL but rather specific aggregates

Demo

Prometheus Query Logs

- Logs all queries run by the engine
- Information about where time was spent in a query



High Cardinality Analyzer

- Offline process to generate recording and / or rollup rules
- Uses Prometheus query log to find candidates for aggregation
- Provides recommendations for recording rules or M3 aggregator rollup and mapping rules to create to speedup expensive queries

High Cardinality Analyzer

- Go over days of Prometheus query logs
  - Find most commonly hit expensive queries
  - Check that the cost of the query is due to number of series
- Provide proposals of recording / rollup rules to create
  - User can configure the rules as necessary
- If recording rules, dashboards and other places need to be changed
- If rollup rules, queries will speed up automatically as the query now captures the aggregate metric

Thank You
Q&A

High Cardinality Analyzer https://github.com/chronosphereio/high-cardinality-analyzer

http://bit.ly/m3slack