#ASLI ENGINEERING

Architecture of Pinterest's Time Series DB

SWIPE

BY

ARPIT BHAYANI

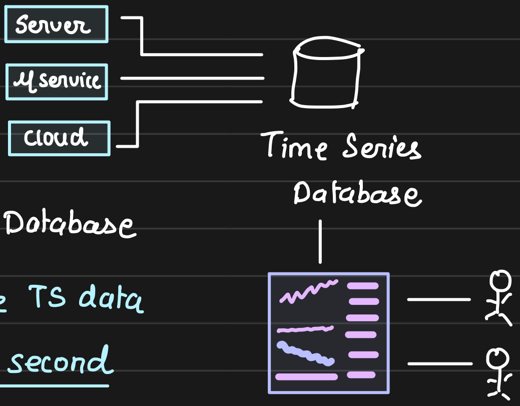
Goku - Pinterest's Time Series Database

Great companies run on Analytics

They measure everything

The data (metrics, vitals, events)

is stored in a specialized Time Series Database



Pinterest used OpenTSDB to store the TS data and they ingest million points every second

but OpenTSDB (based on HBase) has

1. GC issues
2. Crashes are common

$\xrightarrow{\text{Hence, they built}}$ Goku : A OpenTSDB compliant Time Series DB

Time Series Data Model

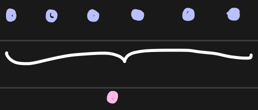
$$\underbrace{\text{tc.proc.stat.cpu.total}}_{\text{metric}} \ \underbrace{\{\text{host=ec2-1, service=auth}\}}_{\text{tags}} = (\underbrace{1527724520}_{\text{timestamp}}, \underbrace{98.6}_{\text{value}})$$

Used for filtering points

(Exact, Wildcard, Regex)

Aggregators : Sum, Max, Min, Avg, Count, Deviation

Downsampling :



One point to represent several points

Challenges and Key Decisions

1. Scan : OpenTSDB scans are inefficient → disk based, bucketed
   Goku scans are fast → in-memory, inverted indexbased

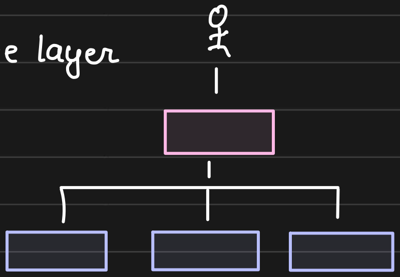
2. Data Size : Goku uses Facebook's In-memory TSDB 'Gorilla'
   which gives 12x compression out of the box

3. Compute and Aggregation :

   OpenTSDB scatters the request, gathers the data on one machine and then aggregates.

   Goku does 1st aggregation on storage layer and then on proxy and then the results are sent to the client

   * minimal data transfer over the n/w



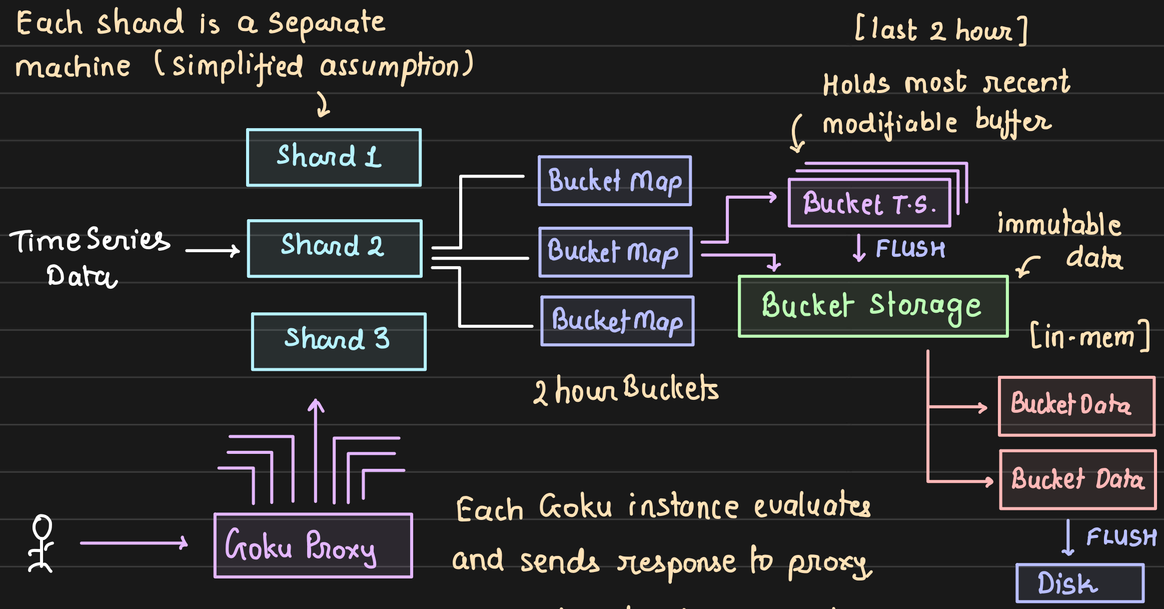
4. Serialization :

   OpenTSDB uses JSON → worst, too slow
   Goku uses Thrift Binary protocol to serialize

ARPIT BHAYANI

Architecture

Goku uses Facebook's Gorilla in-memory storage engine to store most recent data from past 24 hours.



Each Goku instance evaluates and sends response to proxy proxy does final aggregation and then responds.

Given a datapoint, find a shard where it belong

$$f(\text{metric name}) \rightarrow i \rightarrow \text{shard}_i$$

Query is also metric specific, so we know where to go to

ARPIT BHAYANI