Processing 10M samples/second to drive smart maintenance in complex IIoT systems

Geir Engdahl - CTO, Cognite
Daniel Berqvist - Developer Advocate, Google
DEMO
Charting library you just saw is open-sourced

https://github.com/cognitedata/griff-react

- High performance charting of large time series
- Dynamic data loading
- No tight coupling to Cognite TSDB
- Uses React and d3

yarn add @cognite/griff-react

Or

npm i @cognite/griff-react
IoT & the data explosion

50 billion devices connected to internet by 2023 according to Statista (2018) [1]. Cognite currently covers 500,000 sensors, each producing one GB every two years.

Time series requirements

- Robustness
- High volume of reads and writes
- Low latency
- Arbitrary granularity aggregates
- Efficient backfill
- Efficient sequential reads

Surely there must be an off-the-shelf solution that satisfies this!
Databases for IoT - two approaches

Single node*

Horizontally scaling

* Often does master - slave, or other read-only replication, but not partitioning
OpenTSDB experiments

- No limit parameter on queries
- No batch inserts, so slow backfills
- Can lose incoming data points
- Aggregates not pre-computed on write

Disclaimer: OpenTSDB experiments from summer 2017 on version 2.3.0
The case for Cloud Bigtable

- Fully managed
- 10k writes/s per node (SSD)
- Scalable to 100s of PBs
- Can scan forward efficiently
- Column families and versioning
A brief introduction to Google Cloud Bigtable

Achieve your performance goals

Serve global audiences

From DevOps to NoOps

Supercharge your applications

- Single digit ms write latency for performance-critical apps
- 99.99% availability across Google’s dedicated network
- Reduce management effort from weeks to minutes
- Stream, secure, analyze and drive ML/AI
**NoSQL** (no-join) distributed key-value store, designed to scale-out

Has only **one index** (the row-key)

Supports atomic **single-row transactions**

**Sparse**: Unwritten cells do not take up any space

### Wide-columnar data model

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Column-Family-1</th>
<th>Column-Family-2</th>
<th>Column-Family-1</th>
<th>Column-Family-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>r1, cf1:cq1</td>
<td>r1, cf1:cq2</td>
<td>r1, cf2:cq1</td>
<td>r1, cf2:cq2</td>
</tr>
<tr>
<td>r2</td>
<td>r2, cf1:cq1</td>
<td>r2, cf1:cq2</td>
<td>r2, cf2:cq1</td>
<td>r2, cf2:cq2</td>
</tr>
</tbody>
</table>
Every cell is **versioned** (default is timestamp on server)

Configurable **garbage collection** retains latest N versions (or after TTL)

Expiration can be set at **column-family level**

<table>
<thead>
<tr>
<th>Row Key</th>
<th>CF:CQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>“r1”</td>
<td></td>
</tr>
</tbody>
</table>

```
value @ time(latest)
value @ time(previous)
value @ time(earliest available)
```
Cloud Bigtable separates processing from storage through use of nodes, each of which provides access to a group of database rows.

Rebalancing automatically reduces the load on highly active nodes (in this case there is a lot of activity for data group A).

User-driven resizing as needed to match data throughput targets, with no downtime.

#StrataData
Cloud Bigtable replication

Regional replication
- SLA increased to 99.99%
- Isolate serving and analytics
- Independently scale clusters
- Automatic failover in case of a zonal failure

Global replication
- Increases durability/availability beyond one region
- Fastest region-specific access
- Option for DR replica for regulated customers

Current regions and number of zones:
- Seoul
- Salt Lake City
- Tokyo
- Osaka
- Taiwan
- Hong Kong
- Mumbai
- Singapore
- Sydney
- Mumbai
- Oregon
- Salt Lake City
- Iowa
- Montréal
- London
- Netherlands
- S. Carolina

Future regions and number of zones:
- São Paulo
- Finland
- Belgium
- Zürich
- Helsinki
- Belgium
- Netherlands
- StrataData
### Recommendations for row key design

<table>
<thead>
<tr>
<th>Recommendations for row key design</th>
<th>Recommendations for data column design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use tall and narrow tables</td>
<td>Rows can be big but are not infinite (1000 timestamp/value pairs per row is a good rule of thumb)</td>
</tr>
<tr>
<td>Prefer rows to column versions</td>
<td>Keep related data in the same table; keep unrelated data in different tables</td>
</tr>
<tr>
<td>Design your row key with your queries in mind</td>
<td>Store data you will access in a single query in a single column family</td>
</tr>
<tr>
<td>Ensure that your row key avoids hotspotting</td>
<td>Don’t exploit atomicity of single rows</td>
</tr>
<tr>
<td>Reverse timestamps only when necessary</td>
<td></td>
</tr>
</tbody>
</table>
How Cognite stores data in Cloud Bigtable

Row key

"Customer1-Sensor1-2018-07-24-01"

"Customer1-Sensor1-2018-07-24-02"

"Customer1-Sensor2-2018-01-01-01"

"Customer1-Sensor2-2018-01-01-02"

This is the only thing you can lookup, but can also scan **forward**

Group by customer ID, sensor ID first

Then chronologically

#StrataData
Improved key schema

Row key

Group by sensor ID first

<hash of sensor id><customer id><sensor id><time bucket>

Then chronologically
How Cognite stores data in Cloud Bigtable

Row key

Column family: qualifier

“Sensor1-2018072412”

1000, 2000, 3000, ...

“ts:pressure”

“val:pressure”

“Sensor2-2018072412”

“ts:flowrate”

“val:flowrate”

“Sensor3-2018072412”

“val:flowrate”

“val:flowrate”
How Cognite stores data in Cloud Bigtable

<table>
<thead>
<tr>
<th>Row key</th>
<th>Column family: qualifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Sensor1-2018072412”</td>
<td>“ts:pressure”</td>
</tr>
<tr>
<td></td>
<td>27.5, 27.8, 28.3...</td>
</tr>
<tr>
<td>“Sensor2-2018072412”</td>
<td>“ts:flowrate”</td>
</tr>
<tr>
<td></td>
<td>“val:flowrate”</td>
</tr>
<tr>
<td>“Sensor3-2018072412”</td>
<td>“val:flowrate”</td>
</tr>
<tr>
<td></td>
<td>“val:flowrate”</td>
</tr>
</tbody>
</table>
System performance

Performance:

- **Throughput**: Handles up to 10M data points per second
- **Latency**: Data queryable after 200ms (99th percentile)
Protobuf vs JSON

Size of 100k data points

Time to de-serialize 100k data points
Machine learning
Unsupervised anomaly detection

Forecasting

Clustering
Unsupervised detection with AutoEncoders

Architecture search....

... to learn a parameterization of normality
Machine learning architecture

Process

- API gateway
  - Kubernetes Engine
  - Multiple Instances

- Aggregates
  - Cloud
  - Pub/Sub

- TSDB aggregator
  - Kubernetes Engine
  - Multiple Instances

- TSDB writer
  - Kubernetes Engine
  - Multiple Instances

Analyze

- Raw queue
  - Cloud
  - Pub/Sub

- Periodic run
  - Cloud scheduler

- Make predictions
  - ML Engine

- TSDB
  - Cloud Bigtable
Future improvements

- Ability to query consistent snapshots back in time
- High frequency time series
- Efficient latest data point query
Next steps

Cloud Bigtable
- cloud.google.com/bigtable
- cloud.google.com/bigtable/docs/schema-design-time-series

Machine learning
- cloud.google.com/products/ai
Q&A
Rate today’s session

Session page on conference website

O'Reilly Events App