OpenTSDB

The Distributed, Scalable, Time Series Database
For your modern monitoring needs

Collect, store and serve billions of data points
with no loss of precision

Benoît “tsuna” Sigoure
tsuna@stumbleupon.com
Tired of 10+ year old monitoring systems?

Common problems include:

- Centralized data storage (SPoF)
- Limited storage space
- Data deteriorates over time
- Plotting a custom graph is hard
- Doesn’t scale to:
  - >>10s of billions of data points
  - >1000s of metrics
  - New data every few seconds
OpenTSDB

- First open-source monitoring system built on an open-source distributed database
- Collect **all** the metrics you can imagine every few seconds
- Store them forever
- Retain granular data
- Make custom graphs on the fly
- Plug it into your alerting system
- Do capacity planning

Let's take a deep dive inside
HBase

Distributed

Scalable

Reliable

Efficient
Key concepts

- Data Points
  (time, value)

- Metrics
  proc.loadavg.1m

- Tags
  host=web42  pool=static

- Metric + Tags = Time Series

\[
\text{put proc.loadavg.1m 1234567890 0.42 host=web42 pool=static}
\]
OpenTSDB’s push model

The Big Picture™

- Deploy tcollector (agent) on all your servers
- Run as many TSDs as you need
- Write custom collectors for your custom applications
12 Bytes Per Datapoint

4TB per year for 1000 machines
12 Bytes Per Datapoint

What’s new?
- Faster write path
- Two fsck-type tools (because sh*t happens)
- Wider rows
- More memory efficient

What’s hot (just in for OSCON)
- Compacted rows / improved schema (reduces data size by 6x, allows reading >6M points/s)

Misc:
- More unit tests
- Forward compatibility with future variable length encoding
- Improved build system
OpenTSDB @

- 150 Million Datapoints/Day in a typical datacenter
- Over 70 billion data points stored (only 720GB on disk)
- 1 year anniversary as the main production monitoring system
- Completely replaced Ganglia + Munin + Cacti mix

(4x growth in 6 months) (after 5x LZO compression)
Demo Time!
Recipe For Good Performance

- #1 rule: keep good data locality
- Know your access pattern
- Use a key structure that yields good locality for your access pattern
- Avoid wide rows with big keys and many small cells
- OpenTSDB’s secret ingredient: asynchbase
  - Fully asynchronous, non-blocking HBase client
  - Written from the ground up to be thread-safe for server apps
  - Far fewer threads, far less lock contention, uses less memory
  - Provides more throughput, especially for write-heavy workloads
### Inside HBase

#### Table: tsdb-uid

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Column Family: name</th>
<th>Column Family: id</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>metrics</td>
<td>tagk</td>
</tr>
<tr>
<td>0 0 1</td>
<td></td>
<td>host</td>
</tr>
<tr>
<td>0 5 2</td>
<td>proc.loadavg.1m</td>
<td></td>
</tr>
<tr>
<td>host</td>
<td></td>
<td></td>
</tr>
<tr>
<td>proc.loadavg.1m</td>
<td></td>
<td>0 0 1</td>
</tr>
</tbody>
</table>

**Put Example:**
```
put proc.loadavg.1m 1234567890 0.42
```

**Host and Pool Example:**
```
host=web42
pool=static
```
### Inside HBase

**Table: tsdb**

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Column Family: t</th>
</tr>
</thead>
<tbody>
<tr>
<td>+0</td>
<td>+15</td>
</tr>
<tr>
<td>0.69</td>
<td>0.51</td>
</tr>
<tr>
<td>0.99</td>
<td>0.72</td>
</tr>
</tbody>
</table>

- **Row Key**: `0 5 2`
- **Column Family**: `t`
- **Values**: `0.69`, `0.51`, `0.42`, `0.99`, `0.72`
- **Command**: `put proc.loadavg.1m 1234567890 0.42`
- **Host**: `web42`
- **Pool**: `static`
## Implications of the Schema

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Column Family: t</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+0</td>
</tr>
<tr>
<td></td>
<td>0.69</td>
</tr>
<tr>
<td></td>
<td>0.99</td>
</tr>
</tbody>
</table>

- Queries always need data points for a metric and time range
- All data points for a given metric next to each other
- All data points for a time range next to each other
- Compact data + data locality = efficient range scans
- Tag filtering is pushed down to the HBase server
### TSDB Compactions

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Column Family: t</th>
</tr>
</thead>
<tbody>
<tr>
<td>+0</td>
<td>0.69</td>
</tr>
<tr>
<td>...</td>
<td>0.51</td>
</tr>
<tr>
<td>+10</td>
<td>0.42</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>+25</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

#### Table Notes:
- The table represents compactions in a TSDB (Time Series Database).
- The column `Column Family: t` contains values for different row keys.
- The values indicate the compaction level or score for each row key.
### Step 1: Concatenate all columns and values

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Column Family: t</th>
</tr>
</thead>
<tbody>
<tr>
<td>+0</td>
<td>...</td>
</tr>
<tr>
<td>+10</td>
<td>...</td>
</tr>
<tr>
<td>+25</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0.69</th>
<th>0.51</th>
<th>0.42</th>
</tr>
</thead>
</table>

Step 1: Concatenate all columns and values
## TSDB Compactions

### Step 2: Delete individual values

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Column Family: t</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+0</td>
</tr>
<tr>
<td>+0</td>
<td>0.69</td>
</tr>
</tbody>
</table>

### Table:

<table>
<thead>
<tr>
<th>Row Key</th>
<th>+0</th>
<th>+10</th>
<th>+25</th>
</tr>
</thead>
<tbody>
<tr>
<td>+0</td>
<td>0.69</td>
<td>0.69</td>
<td>0.69</td>
</tr>
<tr>
<td>+10</td>
<td>0.51</td>
<td>0.51</td>
<td>0.51</td>
</tr>
<tr>
<td>+25</td>
<td>0.42</td>
<td>0.42</td>
<td>0.42</td>
</tr>
</tbody>
</table>
100% Natural, Organic Free & Open-Source

Danger in the Corn
by Roger Smith

Fork me on GitHub
liked what you saw?
set it up in 15 minutes

• JDK + Gnuplot 1 minute (1 command)
• Single-node HBase 4 minutes (3 commands)
• OpenTSDB 5 minutes (5 commands)
• Deploy tcollector 5 minutes
Under the Hood

- TSD
- Netty
- core
- async hbase
- su async
- Local Disk (cache)
Under the Hood

put proc.loadavg.1m 1234567890 0.42 host=web42 pool=static

TSD → core
    async hbase

Netty → su async

Local Disk (cache)

HBase

>2000 data points / sec / core

Write Path

Is delay max.
Under the Hood

```
put proc.loadavg.1m 1234567890 0.42 host=web42 pool=static
```

Write Path

- TSD
- async hbase
- core
- Netty
- Local Disk (cache)

HBase

> 2000 data points / sec / core
Under the Hood

GET /q...

TSD → core → async hbase → HBase

Netty → su async

Local Disk (cache)

Read Path
Under the Hood

GET /q...

TSD -> core

async hbase

su async

Netty

Local Disk (cache)

Gnuplot

write to cache

scan

HBase

Read Path