OpenTSDB 2.0

Distributed, Scalable Time Series Database

Benoit Sigoure tsunanet@gmail.com
Chris Larsen clarsen@llnw.com

Who We Are

Benoit Sigoure

- Created OpenTSDB at StumbleUpon
- Software Engineer @ Arista Networks

ARISTA

Chris Larsen

- Release manager for OpenTSDB 2.0
- Operations Engineer @ Limelight Networks

What Is OpenTSDB?

- Open Source Time Series Database
- Store trillions of data points
- Sucks up ***all*** data and keeps going
- Never lose precision
- Scales using HBase

What good is it?

- Systems Monitoring & Measurement
  - Servers
  - Networks
- Sensor Data
  - The Internet of Things
  - SCADA
- Financial Data
- Scientific Experiment Results

Use Cases

OVH: #3 largest cloud/hosting provider

Monitor everything: networking, temperature, voltage, application performance, resource utilization, customer-facing metrics, etc.

- 35 servers, 100k writes/s, 25TB raw data
- 5-day moving window of HBase snapshots
- Redis cache on top for customer-facing data

Use Cases

Yahoo

Monitoring application performance and statistics

- 15 servers, 280k writes/s
- Increased UID size to 4 bytes instead of 3, allowing for over 4 billion values
- Looking at using HBase Append requests to avoid TSD compactions

Use Cases

Arista Networks: High performance networking

- Single-node HBase (no HDFS) + 2 TSDs (one for writing, one for reading)
- 5K writes per second, 500G of data, piece of cake to deploy/maintain
- Varnish for caching

Some Other Users

- **Box:** 23 servers, 90K wps, System, app network, business metrics
- **Limelight Networks:** 8 servers, 30k wps, 24TB of data
- **Ticketmaster:** 13 servers, 90K wps, ~40GB a day

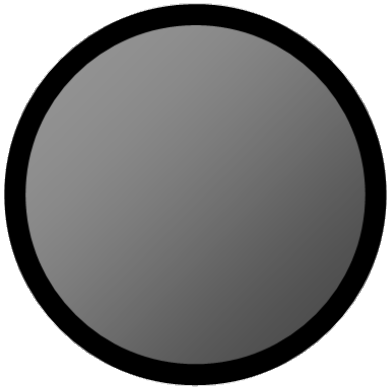
What Are Time Series?

- **Time Series**: data points for an identity over time
- **Typical Identity:**
  - Dotted string: web01.sys.cpu.user.0
- **OpenTSDB Identity:**
  - Metric: sys.cpu.user
  - Tags (name/value pairs): host=web01 cpu=0

What Are Time Series?

Data Point:

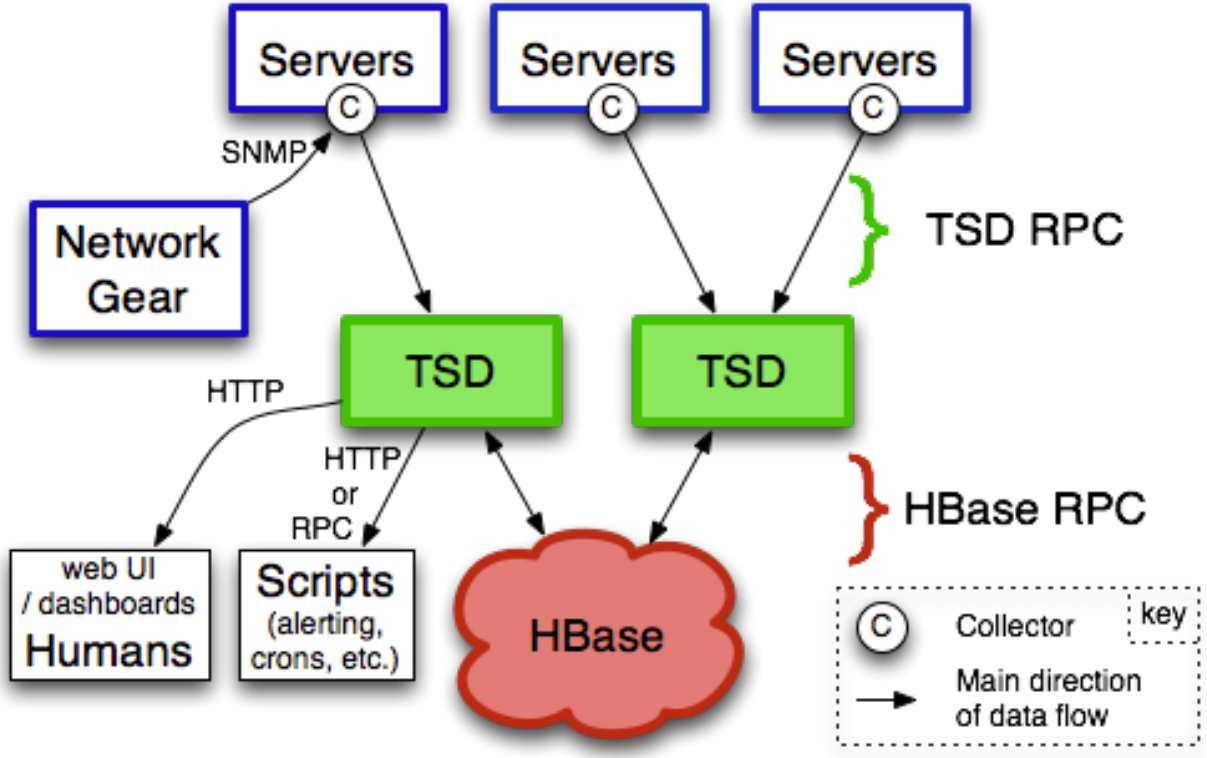
- Metric + Tags
- + Value: 42
- + Timestamp: 1234567890



^ a data point ^

sys.cpu.user 1234567890 42 host=web01 cpu=0

How it Works


Servers
C
Servers
C
Servers
C
SNMP
Network Gear
TSD RPC
TSD
TSD
HTTP
HTTP or RPC
HBase RPC
web UI / dashboards
Humans
Scripts (alerting, crons, etc.)
HBase
key
C
Collector
Main direction of data flow

Writing Data

1) Open Telnet style socket, write:

   put sys.cpu.user 1234567890 42 host=web01 cpu=0

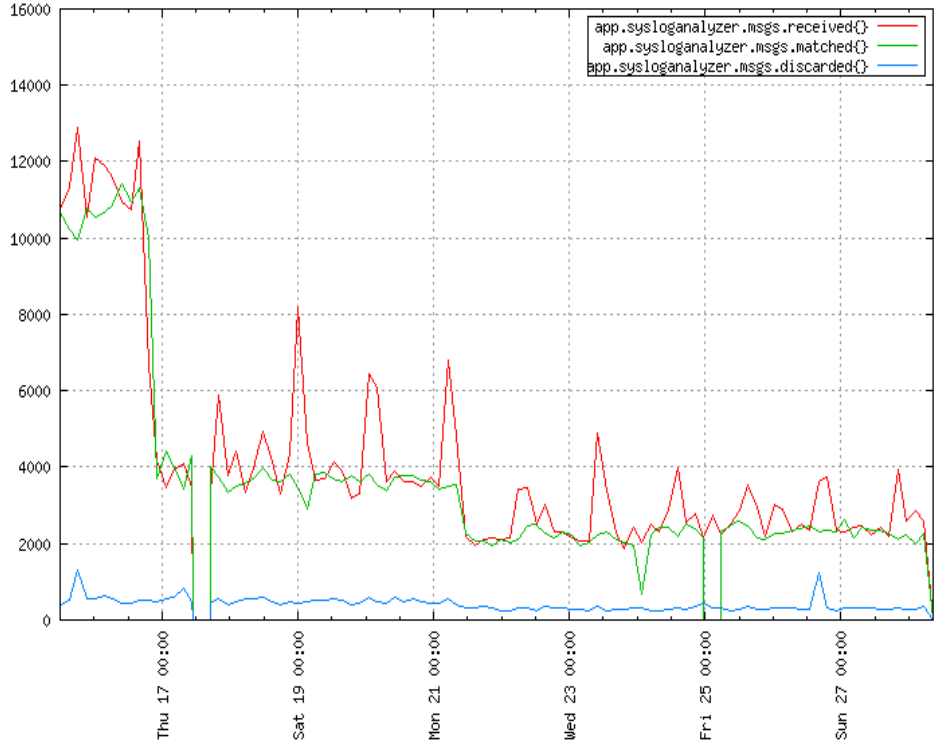
2) ..or with 2.0, post JSON to:

   http://<host>:<port>/api/put

3) .. or import big files with CLI

- No schema definition
- No RRD file creation
- Just write!

Querying Data



- Graph with the GUI
- CLI tools
- HTTP API
- Aggregate multiple series
- Simple query language

To average all CPUs on host:

start=1h-ago
avg sys.cpu.user host=web01

HBase Data Tables

- *tsdb* - Data point table. Massive
- *tsdb-uid* - Name to UID and UID to name mappings
- *tsdb-meta* - Time series index and meta-data (new in 2.0)
- *tsdb-tree* - Config and index for heirarchical naming schema (new in 2.0)

Lets see how OpenTSDB uses HBase...

UID Table Schema

- Integer UIDs assigned to each value per type (metric, tagk, tagv) in *tsdb-uid* table
- 64 bit integers in row *\x00* reflect last used UID

CF:Qualifier	Row Key	UID
id:metric	sys.cpu.user	1
id:tagk	host	1
id:tagv	web01	1
id:tagk	cpu	2
id:tagv	0	2

Improved UID Assignment

- Pre 1.2 Assignment:
  - Client acquires lock on row *\x00*
  - uid = getRequest(*UID type*)
  - Increment uid
  - getRequest(*name*) *confirm name hasn't been assigned*
  - putRequest(*type*, *uid*)
  - putRequest(*reverse map*)
  - putRequest(*forward map*)
  - Release lock
- Lock held for a long time
- Puts overwrite data

Improved UID Assignment

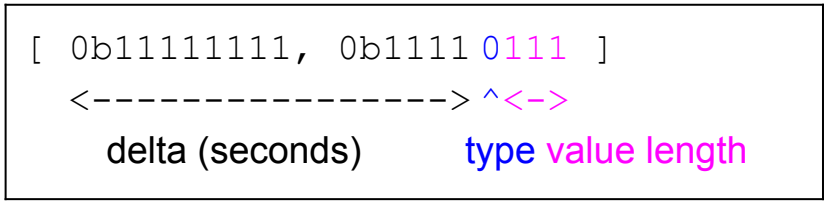
- 1.2 & 2.0 Assignment:
  - atomicIncrementRequest(*UID type*)
  - getRequest(*name*) *confirm name hasn't been assigned*
  - compareAndSet(*reverse map*)
  - compareAndSet(*forward map)*
- 3 atomic operations on different rows
- Much better concurrency with multiple TSDs assigning UIDs
- CAS calls fail on existing data, log the error

Data Table Schema

- Row key is a concatenation of UIDs and time:
  - metric + timestamp + tagk1 + tagv1… + tagkN + tagvN
- sys.cpu.user 1234567890 42 host=web01 cpu=0
  \x00\x00\x01\x49\x95\xFB\x70\x00\x00\x01\x00\x00\x01\x00\x00\x02\x00\x00\x02
- Timestamp normalized on 1 hour boundaries
- All data points for an hour are stored in one row
- Enables fast scans of all time series for a metric
- …or pass a row key regex filter for specific time series with particular tags

Data Table Schema

1.x Column Qualifiers:



- **Type**: floating point or integer value
- **Value Length**: number of bytes the value is encoded on
- **Delta**: offset from row timestamp in seconds
- Compacted columns concatenate an hour of qualifiers and values into a single column

Data Table Schema

OpenTSDB 2.x Schema Design Goals

- Must maintain backwards compatibility
- Support millisecond precision timestamps
- Support other objects, e.g. annotations, blobs
  - E.g. could store quality measurements for each data point
- Store millisecond and other objects in the same row as data points
  - Scoops up all relevant time series data in one scan

Data Table Schema

Millisecond Support:

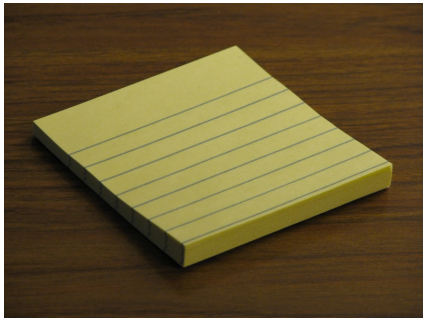
- Need 3,599,999 possible time offsets instead of 3,600
- Solution: 4 byte qualifier instead of 2
- Prefixed with *\xF0* to differentiate from a 2 value compacted column

```
[ 0b11111101, 0b10111011, 0b10011111, 0b11 000111 ]
    <--><------------------------------> ^^^<->
 msec precision              delta (milliseconds)          type value length
                                                          2 unused bits
```

- Can mix second and millisecond precision in one row
- Still concatenates into one compacted column

Data Table Schema

Annotations and Other Objects



- Odd number of bytes in qualifier (3 or 5)
- Prefix IDs: *\x01* = annotation, *\x02* = blob
- Remainder is offset in seconds or milliseconds
- Unbounded length, not meant for compaction

```
{
   "tsuid": "00000100000100000 1",
   "description": "Server Maintenance",
   "notes": "Upgrading the server, ignore these values, winter is coming",
   "endTime": 1369159800,
   "startTime": 1369159200
}
```

TSUID Index and Meta Table

- Time Series UID = data table row key without timestamp
  E.g. sys.cpu.user host=web01 cpu=0
  \x00\x00\x01\x00\x00\x01\x00\x00\x01\x00\x00\x02\x00\x00\x02
- Use TSUID as the row key
  - Can use a row key regex filter to scan for metrics with a tag pair or the tags associated with a metric
- Atomic Increment for each data point
  - ***ts_counter*** column: Track number of data points written
  - ***ts_meta*** column: Async callback chain to create meta object if increment returns a *1*
  - Potentially doubles RPC count

OpenTSDB Trees

- Provide a hierarchical representation of time series
  - Useful for Graphite or browsing the data store
- Flexible rule system processes metrics and tags
- Out of band creation or process in real-time with TSUID increment callback chaining

OpenTSDB Trees

Example Time Series:

myapp.bytes_sent dc=dal host=web01
myapp.bytes_received dc=dal host=web01

Example Ruleset:

Level	Order	Rule Type	Field	Regex
0	0	tagk	dc	
1	0	tagk	host	
2	0	metric		(.*)\..*
3	0	metric		.*\.(*.)

OpenTSDB Trees

Example Results:

Flattened Names:
dal.web01.myapp.bytes_sent
dal.web01.myapp.bytes_received

Tree View :

- dal
  - web01
    - myapp
      - bytes_sent
      - bytes_received

^ *leaves* ^

Tree Table Schema

Column Qualifiers:

- ***tree***: JSON object with tree description
- ***rule:<level>:<order>***: JSON object definition of a rule belonging to a tree
- ***branch***: JSON object linking to child branches and/or leaves
- ***leaf:<tsuid>***: JSON object linking to a specific TSUID
- ***tree_collision:<tsuid>***: Time series was already included in tree
- ***tree_not_matched:<tsuid>***: Time series did not appear in tree

Tree Table Schema

- Row Keys = Tree ID + Branch ID
- Branch ID = 4 byte hashes of branch hierarchy

E.g. **dal.web01.myapp.bytes_sent**

dal = 0001838F (hex)

web01 = 06BC4C55

myapp = 06387CF5

bytes_sent = leaf pointing to a TSUID

Key for branch on Tree #1 =
00010001838F06BC4C5506387CF5

Tree Table Schema

Row Key	**CF: “t” Keys truncated, 1B tree ID, 2 byte hashes**		
\x01	“tree”: {“name”:”Test Tree”}	“rule0:0”: {<rule>}	“rule1:0”: {<rule>}
\x01\x83\x8F	“branch”: {“name”:”dal”, “branches”:[{ “name”:”web01”},{“name”:”web01”}]}		
\x01\x83\x8F\x4C\x55	“branch”: {“name”:”web01”, “branches”:[{ “name”:”myapp”}]}		
\x01\x83\x8F\x4C\x55\x7C\x5F	“branch”: {“name”:”myapp”, “branches”:null}	“leaf:<tsuid1>”: {“name”:” bytes_sent”}	“leaf:<tsuid2>”: {“name”:” bytes_received”}
\x01\x83\x8F\x4D\x00	“branch”: {“name”:”web02”, “branches”:[{ “name”:”myapp”}]}		
\x01\x83\x8F\x4D\x00\x7C\x5F	“branch”: {“name”:”myapp”, “branches”:null}	“leaf:<tsuid3>”: {“name”:” bytes_sent”}	“leaf:<tsuid4>”: {“name”:” bytes_sent”}

Row Key Regex for branch “**dal**”: ^\Q\x01\x83\x8F\E(?:.{2})$
Matches branch “**web01**” and “**web02**” only.

Time Series Naming Optimization

- Designed for fast aggregation:
  - Average CPU usage across hosts in web pool
  - Total bytes sent from all hosts running application MySQL
- High cardinality increases query latency
- Aim to minimize rows scanned
- Determine where aggregation is useful
  - May not care about average CPU usage across entire network.
    Therefore shift web tag to the metric name

Time Series Naming Optimization

Example Time Series

- sys.cpu.user host=web01 pool=**web**
- sys.cpu.user host=db01 pool=**db**
- host = 1m total unique values, 1,000 in **web** and 50 in **db**
- pool = 20 unique values

Very high cardinality for “sys.cpu.user”

- Query 1) 30 day query for “**sys.cpu.user pool=db**” scans **720m** rows (24h * 30d * 1m hosts) but only returns **36k** (24h * 30d * 50)
- Query 2) 30 day query for “**sys.cpu.user host=db01 pool=db**” still scans **720m** rows but returns **36**

Time Series Naming Optimization

Solution: Move **pool** tag values into metric name

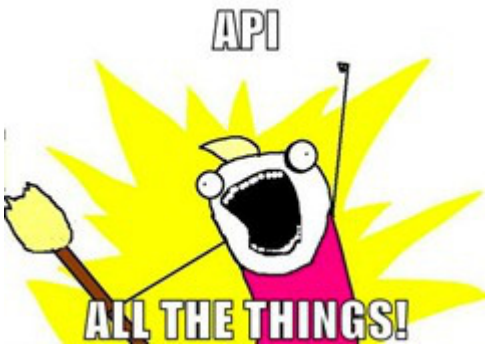
- **web**.sys.cpu.user host=web01
- **db**.sys.cpu.user host=db01

Much lower cardinality for “sys.cpu.user”

- Query 1) 30 day query for “**db.sys.cpu.user**” scans **36k** rows (24h * 30d * 50 hosts) and returns all **36k** (24h * 30d * 50)
- Query 2) 30 day query for “**db.sys.cpu.user host=db01**” still scans **36k** rows and returns **36**

New for OpenTSDB 2.0

- RESTful HTTP API
- Plugin support for de/serialization
- Emitter plugin support for publishing
  - Use for WebSockets/display updates
  - Stream Processing
- Non-interpolating aggregation functions

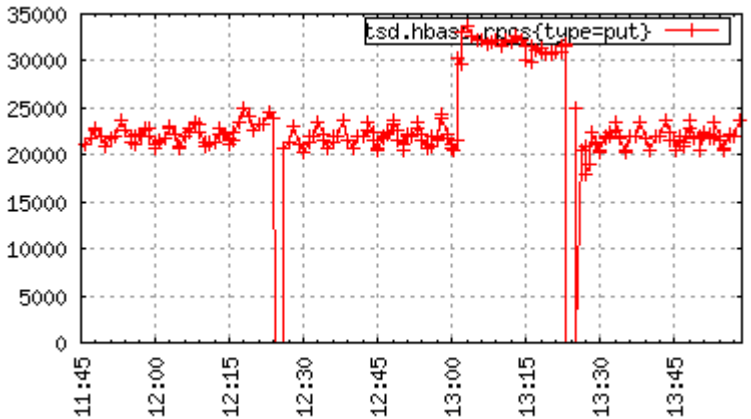


New for OpenTSDB 2.0

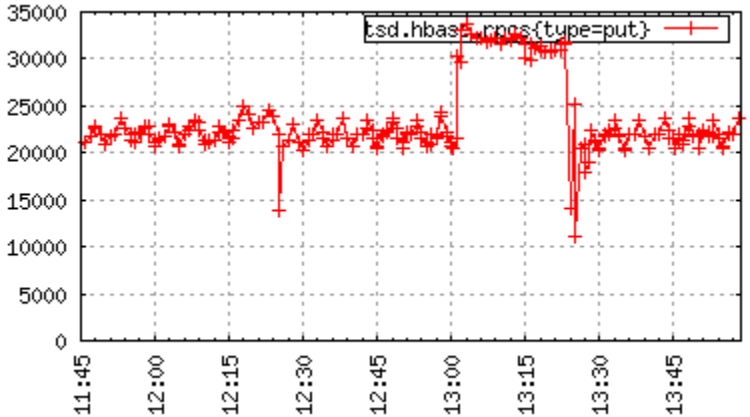
Special rate and counter functions

- Suppress spikes

Raw Rate:

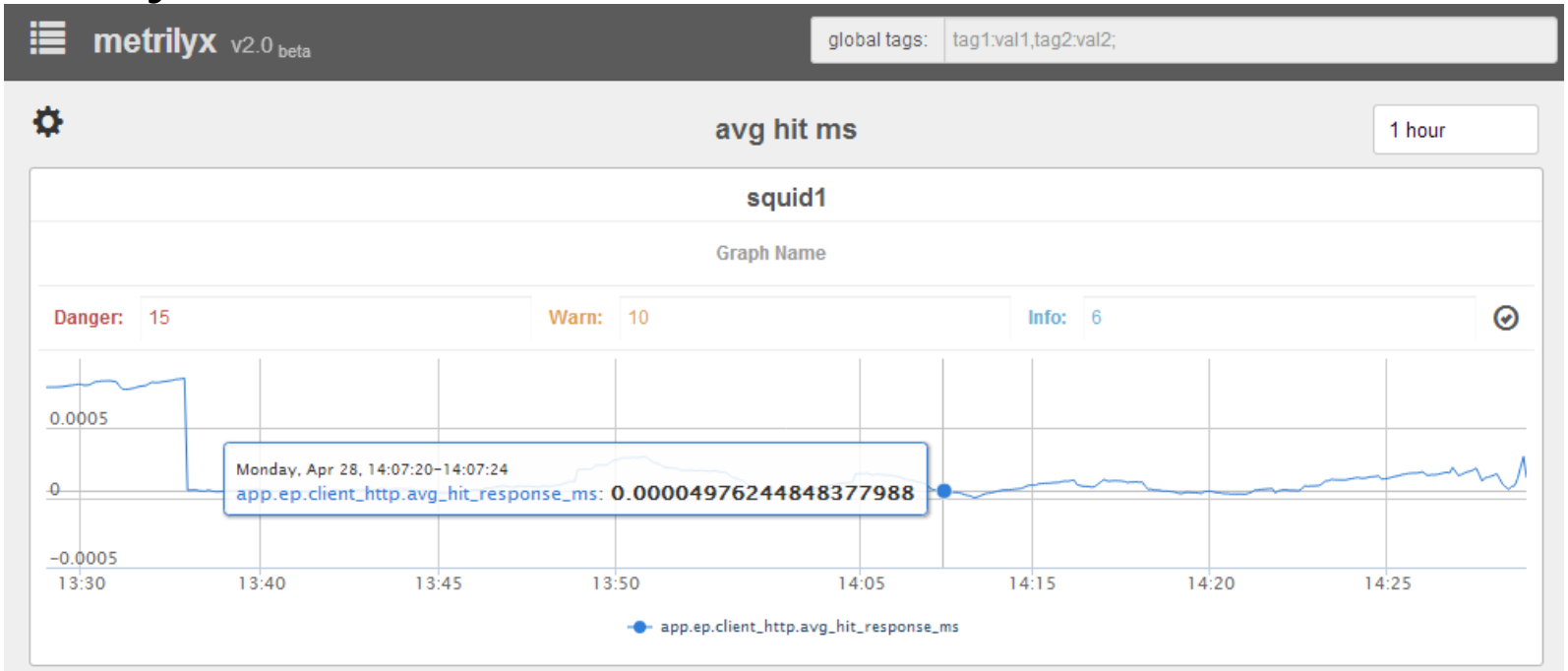


Counter Reset = 1000



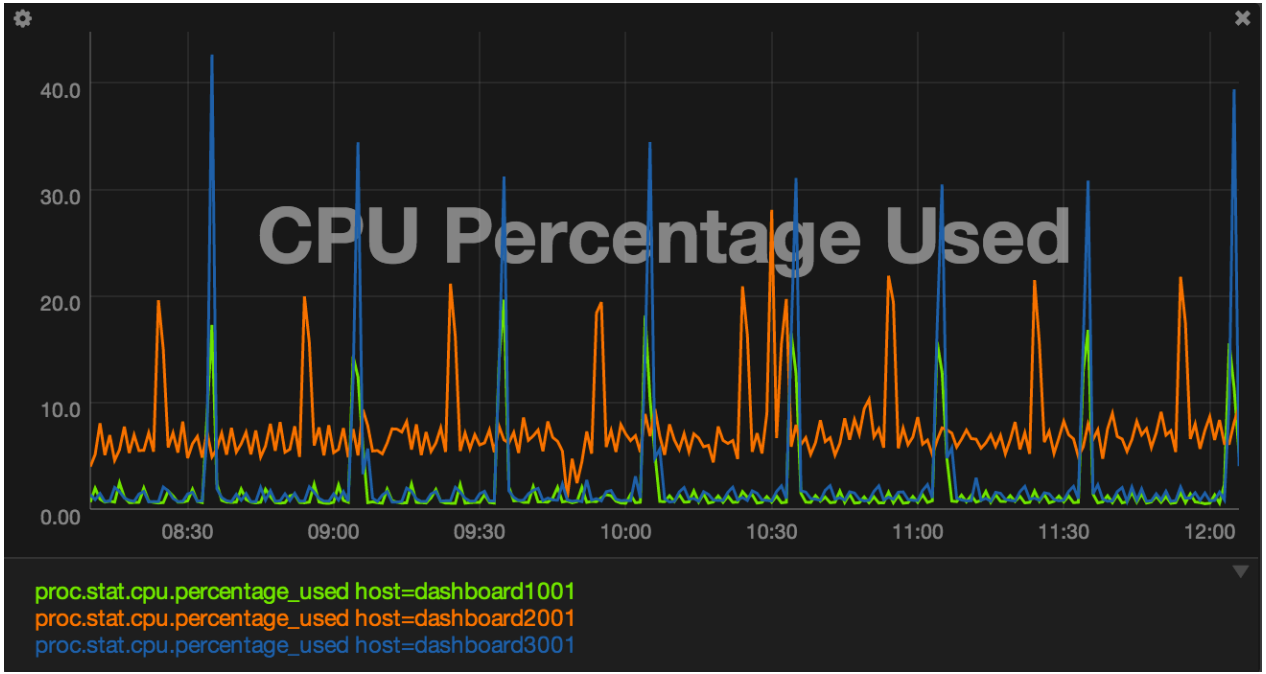
OpenTSDB Front-Ends

Metrilyx from TicketMaster https://github.com/Ticketmaster/metrilyx-2.0



OpenTSDB Front-Ends

Status Wolf from Box https://github.com/box/StatusWolf



New in AsyncHBase 1.5

- AsyncHBase is a fully asynchronous, multi-threaded HBase client
- Now supports HBase 0.96 / 0.98
- Remains 2x faster than HTable in PerformanceEvaluation
- Support for scanner filters, META prefetch, “fail-fast” RPCs

The Future of OpenTSDB

The Future

- Parallel scanners to improve queries
- Coprocessor support for query improvements similar to Salesforce's Phoenix with SkipScans
- Time series searching, lookups (which tags belong to which series?)

The Future

- Duplicate timestamp handling
- Counter increment and blob storage support
- Rollups/pre-aggregations
- Greater query flexibility:
  - Aggregate metrics
  - Regex on tags
  - Scalar calculations

More Information

- Thank you to everyone who has helped test, debug and add to OpenTSDB 2.0!
- Contribute at github.com/OpenTSDB/opentsdb
- Website: opentsdb.net
- Documentation: opentsdb.net/docs/build/html
- Mailing List: groups.google.com/group/opentsdb

Images

- http://photos.jdhancock.com/photo/2013-06-04-212438-the-lonely-vacuum-of-space.html
- http://en.wikipedia.org/wiki/File:Semi-automated-external-monitor-defibrillator.jpg
- http://upload.wikimedia.org/wikipedia/commons/1/17/Dining_table_for_two.jpg
- http://upload.wikimedia.org/wikipedia/commons/9/92/Easy_button.JPG
- http://upload.wikimedia.org/wikipedia/commons/4/42/PostItNotePad.JPG
- http://openclipart.org/image/300px/svg_to_png/68557/green_leaf_icon.png
- http://nickapedia.com/2012/06/05/api-all-the-things-razor-api-wiki/
- http://lego.cuusoo.com/ideas/view/96
- http://upload.wikimedia.org/wikipedia/commons/3/35/KL_Cyrix_FasMath_CX83D87.jpg
- http://www.flickr.com/photos/smkybear/4624182536/