PostRR – PostgreSQL Round Robin Extension
Using PostgreSQL for storing time-series data

Sebastian ‘tokkee’ Harl <sh@tokkee.org>

teamix GmbH / collectd core team

PGConf.EU 2012
October 26, 2012
Prague
WARNING:
This talk presents work that is to be considered pre-alpha and / or random-thoughts.

Flaming, bashing, or other forms of constructive feedback are highly appreciated! :-)
Solid IT-Infrastructure
Location: Nuremberg, Munich, Frankfurt
http://teamix.net/

Open-Source
N-IX
Riverbed

Monitoring
NetApp
VMWare

Network
Juniper
Trainings

PostRR – PostgreSQL Round Robin Extension
Background: RRDtool
RRDtool Overview

- RRDtool is the de-facto standard for time-series storage and graphing
- powerful graphing features
- easy to setup up
- fixed storage size
- ⇒ also optimized for graphing
RRDtool Problems

- very inflexible backend storage (modifications of existing RRD files are a PITA)
- I/O-problems in large setups (→ RRDCacheD; but missing clustering, replication, etc.)
- missing (limited) networking support
- limited querying support (abuse of rrdgraph)

⇒ an RDBMS might help solve some of these issues
⇒ needs further features to cover base features
Writing Data to PostgreSQL
Things to consider . . .

- database layout (data identifier, timestamps, values)
- identifiers might be hierarchical
- data partitioning
- graphing
Current proposal for **collectd**:

- generic approach
- user-specified query to store one data-set
  - $\Rightarrow$ flexible database layout
  - $\Rightarrow$ may be used for experimenting
- problem: no bulk updates (e.g. `COPY`) possible
- idea: possibly build specialized plugin later
Example Configuration

collectd.conf:

```xml
<Plugin postgresql>
  <Writer sqlstore>
    Statement "SELECT collectd_insert($1, $2, \\
    $3, $4, $5, $6, $7, $8, $9);"
  </Writer>
  <Database sh>
    Writer sqlstore
    CommitInterval 10
  </Database>
</Plugin>
```

Arguments: timestamp, hostname, plugin name, plugin instance, type name, type instance, array of data-source types, array of values.
sh=> \d identifiers

Table "public.identifiers"

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>integer</td>
<td>not null default nextval(...)</td>
</tr>
<tr>
<td>host</td>
<td>character varying(64)</td>
<td>not null</td>
</tr>
<tr>
<td>plugin</td>
<td>character varying(64)</td>
<td>not null</td>
</tr>
<tr>
<td>plugin_inst</td>
<td>character varying(64)</td>
<td>default NULL::character varying</td>
</tr>
<tr>
<td>type</td>
<td>character varying(64)</td>
<td>not null</td>
</tr>
<tr>
<td>type_inst</td>
<td>character varying(64)</td>
<td>default NULL::character varying</td>
</tr>
</tbody>
</table>

Indexes:

- "identifiers_pkey" PRIMARY KEY, btree (id)
- "identifiers_host_plugin_plugin_inst_type_type_inst_key" UNIQUE CONSTRAINT, btree (host, plugin, plugin_inst, type, type_inst)
- "identifiers_host" btree (host)
- "identifiers_plugin" btree (plugin)
- "identifiers_plugin_inst" btree (plugin_inst)
- "identifiers_type" btree (type)
- "identifiers_type_inst" btree (type_inst)
\qsh \d+ values

Table "public.values"

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
<th>Storage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>integer</td>
<td>not null</td>
<td>plain</td>
<td></td>
</tr>
<tr>
<td>tstamp</td>
<td>timestamp without time zone</td>
<td>not null</td>
<td>plain</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>character varying(64)</td>
<td>not null</td>
<td>extended</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>double precision</td>
<td>not null</td>
<td>plain</td>
<td></td>
</tr>
</tbody>
</table>

Triggers:

insert_values_trigger BEFORE INSERT ON "values"
FOR EACH ROW EXECUTE PROCEDURE values_insert_trigger()

Child tables: "values$2012-10-24",
"values$2012-10-25",
"values$2012-10-26",
"values$2012-10-27"
collectd_insert() (1)

```
SELECT id INTO ds_id
FROM identifiers
WHERE host = p_host
  AND plugin = p_plugin
  AND COALESCE(plugin_inst, '') = COALESCE(p_plugin_instance, '')
  AND type = p_type
  AND COALESCE(type_inst, '') = COALESCE(p_type_instance, '');

IF NOT FOUND THEN
  INSERT INTO identifiers (host, plugin, plugin_inst, type, type_inst)
  VALUES (p_host, p_plugin, p_plugin_instance, p_type, p_type_instance)
  RETURNING id INTO ds_id;
END IF;
```
i := 1;
LOOP
    EXIT WHEN i > array_upper(p_value_names, 1);
    INSERT INTO values (id, tstamp, name, value)
    VALUES (ds_id, p_time, 
            p_value_names[i], p_values[i]);
    i := i + 1;
END LOOP;
• constant flow of data
• rather small amounts of data (MB - GB range)
• lots of single queries

ideas:
• commit interval (or insert count per transaction)
• use `COPY` → harder to implement in a generic way
• use temporary tables and sync to regularly
• use multi-threading / multiple connections?
• note: benchmarking should be done in an application specific manner (e.g. traffic generator `collectd-tg`)
• limit generic approach (a bit)
• use function to lookup (database) id for collectd identifier
• → more generic: generic lookup functions to define arbitrary columns
• cache id in collectd (limit overhead to startup)
• let user specify data format specification
The PostgreSQL Round-Robin Extension (PostRR)
• https://github.com/tokkee/postrr
• BSD license
• adds new data-types and helper functions to PostgreSQL
• similar to the general storage idea implemented in RRDtool
• may be used in addition to storing the actual values
• early pre-alpha state for now ;-)
Why?

- because I can (or rather couldn’t)
- flexible queries
- link to other (meta-)data (e.g. spacial data)
- after having RRDCacheD available the next logical step was to extend that to behave similar to a database system
- … now the next step after (for) that is to extend an existing database system
• Round-Robin Timeslice
• based on timestamp
• uses ‘length’ settings to define a time-slice
• uses ‘count’ settings to build a round-robin storage
• → stored in a “private” table
• splits the time-line into non-overlapping slices with unique id (sequence number; less than ‘count’)
• index on sequence number (TODO: and timestamp)
• casts to and from timestamp
• Usage: RRTimeslice(<len>, <count>)
- Consolidated Data
- based on double precision
- uses a built-in consolidation function (AVG, MIN, MAX)
- stores multiple consolidated data points (+ meta information like number of values and number of undefined values)
- casts from and to numeric types
- Usage: CData(<AVG>|<MIN>|<MAX>)
- CData_update(cdata, cdata) to merge two values
Round Robin Archives

- assigns a name / identifier to multiple “archives” storing values with different resolution, consolidation functions and value counts (using RRTimeslice and CData columns)
- “multiplexer” to insert a value into multiple tables and / or columns
- `PostRR_update(<rra_name>, <timestamp>, <value>)` used to insert a new value into an archive
- overwrites values based on their sequence number
sh=# \d postrr.rrarchives
Table "postrr.rrarchives"
Column  | Type     | Modifiers
---------+----------|-----------
rraname  | text     | not null
.tbl     | name     | not null
tscol    | name     | not null
vcol     | name     | not null
sh=# select postrr_update('name', 'now', 10);
postrr_update
-------------------
 10 (AVG U:0/1)
 10 (MIN U:0/1)
 10 (MAX U:0/1)

sh=# select postrr_update('name', 'now', 100);
...
sh=# select postrr_update('name', 'now', 1);
postrr_update
-------------------
 37 (AVG U:0/3)
 1 (MIN U:0/3)
100 (MAX U:0/3)
Now, we’re missing some way to actually create graphs of the data.

- `rrdgraph_libdbi` might be extended to be able to run more complex queries
- PostgreSQL CSV output could be used with Gnuplot; would need some wrapper scripts (benefit: lots of backends available)
- Graphite DB backend?
- ...
Thanks for your attention!

Any questions, thoughts, comments?
Contact:
Sebastian “tokkee” Harl
teamix GmbH, Nürnberg
<sh@tokkee.org>

<collectd@verplant.org> — irc.freenode.net/#collectd — http://identi.ca/collectd

https://github.com/tokkee/postrr