Torrus 2.0 Design Concept

Stanislav Sinyagin <ssinyagin@k-open.com>
July 24, 2008

1. Introduction

The Torrus software project has started back in 2002, mainly as an alternative to Cricket. It took the best of Cricket's design ideas, yet rendering into a completely new software. As of today, Cricket hasn't been updated for years. At the same time, Torrus has grown into a powerful and flexible system, and it is used in many medium or large-scale installations. However, some strong limitations in Torrus architecture and in user-friendliness have drawn many users towards such software packages as Cacti.

This document describes the thoughts and concepts behind a future, new-generation software package. It's called Torrus 2.0, but there's no goal to extend today's Torrus, and no intent for any backwards compatibility. If this project turns out good, this will be a completely new software. It will incorporate the best experiences gained with Torrus 1.0, and it will try to mitigate the problems and limitations of the older Torrus.

2. Limitations of Torrus 1.0

2.1. Database reliability

Because of the nature of Berkeley DB, all the database logic is done in the calling process' context. This includes the shared memory regions used by cache and locking subsystems. If a process crashes or terminates abnormally, it leaves the locks unreleased, which may lead to a completely unusable database.

In case of any abnormal process termination, all the Torrus related processes have to be stopped, and the database needs to be recovered. This does not happen too often in stable environment, but it's quite annoying in experimental set-ups and also for the new users.

2.2. Data hierarchy restrictions

The Torrus' database structure is a N-way tree. This brings some nice features, such as parameter inheritance, quick recursive walking, and it reflects the object in a hierarchical way. However, there are several problems with it, as shown below.

It is very difficult to implement incremental updates in the current hierarchical data structure. If you update some parameters at the parent level, all child nodes are affected. Thus the memory cache in all program elements should be updated. Also it is unclear how to process deleted nodes. So, currently the only efficient way is to recompile the whole tree, once some changes need to be done in parts of it. In some large installation, the whole compilation process may take up to an hour.

Also some network topologies are not convenient to model in the Torrus data model. For example, if we want to monitor several dozens of thousands cable modems, they would all lay in a single level of the tree, while they are not easy to group together. This would render in the difficulty to navigate such a tree in the Web UI.

2.3. Single-server architecture

Berkeley DB is not a networked DBMS, and thus it would require an additional layer if we would need to build a multi-server Torrus installation. This might be required, for example, in a large low-bandwidth network, where the remote collectors would need to take data from the central database.

Currently all the Torrus processes have to stay within a single OS instance in order to have access to the same database. It's still possible to build a distributed system, but the data would have to be replicated manually between the systems.

2.4. Rigid and non-intuitive Web UI

The current Web UI is quite old-fashioned, if comparing to other systems. For example, the user cannot have his or her own customization settings. There's no simple way to organize favorite data in a way that is convenient for the user. Any custom graphs, such as traffic summaries, need some sophisticated XML editing in UNIX command-line interface.

Also the access control is now implemented on a tree level. There are lots of occasions when the administrators want to provide access to a limited subset of graphs only. The implementation of limited control also requires some sophisticated tuning of XML files.

2.5. Status monitoring is not supported

Torrus is very robust and flexible in performance monitoring tasks, but there's no room for such tasks as status or failure monitoring. Many users want some status reports alongside with performance graphs, and Torrus is not able to offer such functionality, because the current software architecture is oriented to periodic collecting of numbers and counters only.

3. Torrus 2.0 design principles

3.1. RDBMS back end

The power of Berkeley DB is that every lookup operation is fast and inexpensive. Another big advantage of Berkeley DB is the availability of cursors for walking through large amounts of data.

Still, a standard SQL engine as a back end would bring lots of interesting benefits: ability to separate the DB maintenance from application maintenance; networked and distributed setup becomes easy to implement; system administrators' experience with database maintenance tasks; and more.

At the same time, the data lookups are not any more as cheap as with Berkeley DB: every query should be encoded in an SQL statement, transmitted, interpreted, then the selected data should be accumulated and sent back to the client. Also it is inefficient, and sometimes impossible, to build SELECT queries which would match dozens of thousands of rows: this would render in huge delays when retrieving the data.

Thus, the new Torrus data structure should meet several important criteria, as follows:

- Any data object and its properties should be retrieved with a limited number of SELECT queries, ideally with one query.
- The data architecture should allow to walk through the whole data set, without having to retrieve huge arrays as query results.
- As the cursors are not part of the standard SQL syntax, and every RDBMS vendor implements its own cursor operations, it is not acceptable to use cursors.
- The data structure and the whole database access layer should be vendor-agnostic, and be compatible with as many RDBMS systems as possible. However, it will most probably make use of transactions, therefore the choice of the RDBMS systems is limited to those that support them (MySQL 5.x, Postgres, Oracle, ...).

3.2. Move most of site configuration to the database

The current Torrus implementation keeps most of site-specific options in a Perl configuration file. Every time when such options are changed, the Torrus processes need to be restarted.

In the new Torrus, only a few parameters are allowed in configuration files, and the rest should be moved to the database. Also the database should present the configuration objects the same way as any other data objects.

The following parameters are allowed to be configurable from local files:

- Database connection parameters, such as database host, user name, and password.
- Site identification parameters. In a multi-server installation, each OS instance needs to be uniquely identified.
- Local file storage parameters, such as the directory path for RRD files.

3.3. Separate data from its visual presentation

In Torrus 1.0, the presentation of the data was bound to the data itself: the data hierarchy was identical to the visual hierarchy in the UI, and also the presentation parameters were the integral part of the data.

The new Torrus data model should set the separation between the visual presentation and the data itself. The data should describe itself in order to identify its type and its relation to other data elements, but the visual parameters should be managed by independent structures.

3.4. Update instead of “erase and re-build”

The tree structure of Torrus 1.0 has made it difficult to make the updates to the data without re-building the whole tree.

In Torrus 2.0 all data should be updatable. Existing objects should be easy to find and modify, and new objects should be easy to add to the hierarchy (for example, interfaces of a router may vary in number between the discovery runs).

Obsolete objects are not deleted, but set to inactive, and deleted later after a certain configurable timeout period.

3.5. Less dependent on RRDtool

The new Torrus will be able to collect the status and failure information from the monitored devices. This data is not suitable for RRDtool, and needs to be stored in some other way, such as in an RDBMS repository. Also the performance monitoring results, such as traditional counter values, may require other than RRDtool storage.

Currently Torrus already supports various storage types, such as RDBMS or flat files. Still, many parts of it are RRD-centric. For example, the Web UI and the monitor daemon are completely RRD-dependent.

The new Torrus core should be designed as storage-agnostic, still taking most of advantage from RRDtool as a means for storing and displaying of periodically collected data.

3.6. Web 2.0 User Interface

The UI of the new Torrus should meet a number of criteria:

- Modular structure, allowing to display various types of data: graphs, events, numerical reports, configuration options, administrative tasks, and so on.
- Customizable layout.
- Ability to tag the graphs, group them as needed, set favorites, add descriptions or notes, and so on.
- Granular access control.
- Easy navigation on the graph: selecting the time span, zooming, etc.
- Easy combining of several data sources on a single graph.
- Easy selection of aggregation functions on long-term graphs.
- Use Ajax where needed.

4. Requirements to the new data model

4.1. Data abstraction

Same way as in Torrus 1.0, the data model should be abstract from the type of the data source. It must not contain such terms as “host” or “interface”. It should be flexible enough to be able to describe hosts and interfaces, and also any other type of data. The data elements should allow some hierarchical grouping, so that they would model various complex sets of data.

Some examples of data grouping and hierarchy follow:

- SNMP Interface counters, such as in- and output octets, errors, discards, etc., are grouped into the interface counters group. Interfaces, along with other types of SNMP objects, are grouped together and model an SNMP host.

- The Netflow collector has a set of rules for aggregating and filtering the Netflow data. Each such rule generates a number of counters.
- The Apache log analyzer generates the hit statistics for a number of URI's inside a website. These URI's may form a hierarchy that reflects the logics of the website.

4.2. Fast bulky retrieval

The collector is the most time-critical component of Torrus, as its primary job is to collect the data in accordance to the schedule. In case of SNMP collection, a single instance of the collector may periodically query 100K or more SNMP objects from few thousand SNMP agents. When the collector process starts, it has to retrieve all the relevant data elements from the database without delay. The collector instance should receive only the objects that are related to this particular instance.

In case of SNMP collector, it would be advantageous to retrieve the data elements as hierarchical groups, so that the hierarchy would automatically determine the grouping of the data inside the collector memory. Also such grouping information can reduce the amount of data (for example, host+community are transmitted from the database only once for the host, and not every time for every OID), and also help in avoiding large SQL result sets.

4.3. Multi-purpose data

As the data is persistent in the database, it makes sense to keep the SNMP discovery instructions as attributes of the data objects representing the hosts for the collector.

Another practical use would be the network topology information that is stored alongside the hosts and interfaces. Such metadata would not be used by any component of Torrus, but it could be built and used by external third-party applications.

4.4. Partial Updates

The data model should group the object attributes in a way that would allow easy updates of sets of attributes. For example, the SNMP discovery engine would update the attributes that are relevant to the SNMP collector, leaving other attributes untouched.

4.5. Universal use

Ideally the data model would allow to store some data types which were not considered as data in Torrus 1.0. This includes the configuration options from torrus-config.pl; UI users, groups and permissions; presentation parameters for various types of graphs, etc.

5. The new data model

To be continued...