ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA166

NEW DEVELOPMENTS FOR THE HDB++ TANGO ARCHIVING SYSTEM

L. Pivetta*[1], G. Scalamera, G. Strangolino, L. Zambon, Elettra Sincrotrone Trieste, Trieste, Italy
R. Bourtembourg, S. James, J.L. Pons, P. Verdier, ESRF, Grenoble, France
S. Rubio-Manrique, ALBA-CELLS Synchrotron, Barcelona, Spain
[1]also at SKA Organisation, Macclesfield, UK

Abstract

TANGO HDB++ is a high performance event-driven archiving system which stores data with micro-second resolution timestamps, using archivers written in C++. HDB++ currently supports MySQL and Apache Cassandra back-ends but could be easily extended to support additional back-ends. Since the initial release many improvements and new features have been added to the HDB++. In addition to bug-fixes and optimizations, the support for context-based archiving allows to define an archiving strategy for each attribute, specifying when it has to be archived or not. Temporary archiving is supported by means of a time-to-live parameter, available on a per-attribute basis. The Cassandra back-end is using Cassandra TTL native feature underneath to implement the time-to-live. With dynamic loading of specific libraries switching back-ends can be done on-the-fly and is as simple as changing a property. Partition and maintenance scripts are now available for HDB++ and MySQL. The HDB++ tools, such as extraction libraries and GUIs, followed HDB++ evolution to help the user to take full advantage of the new features.

INTRODUCTION

The HDB++ TANGO archiving system [1, 2] is a tool that allows to store the attribute values in a TANGO based control system into an historical database, exploiting the publish/subscribe TANGO capabilities. The publish/subscribe paradigm is available in TANGO via the event subsystem. More in detail, the *archive* event is provided for archiving purposes and can be triggered on threshold comparison and/or periodic basis. The HDB++ architecture is fully event based; therefore, a part of HDB++ setup consists of conveniently configure TANGO devices to send events as required. The TANGO archiving system consists of two main components, namely the EventSubscriber TANGO device, or archiving engine, and the ConfigurationManager TANGO device, that simplifies archiving configuration and management. The HDB++ also provides the libraries to interface the supported back-ends, libraries for data extraction and some graphical user interfaces for configuration and data visualization. The typical HDB++ setup is shown in Fig. 1.

NEW FUNCTIONALITIES

In addition to bug-fixes and optimisations, a number of new functionalities and improvements have been developed for the HDB++ archiving system.

* lorenzo.pivetta@elettra.eu

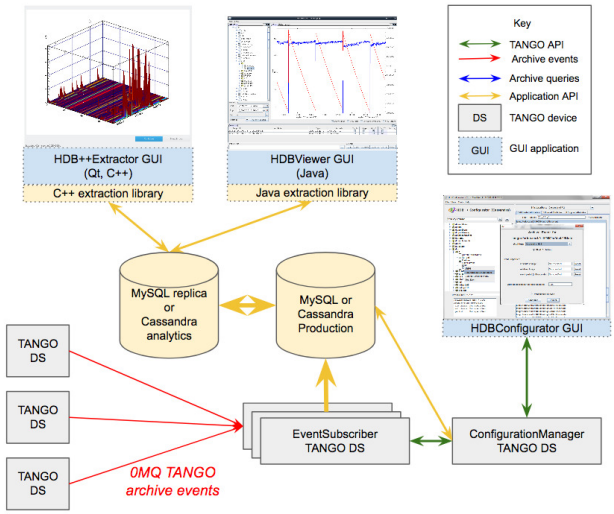

Figure 1: Typical HDB++ setup.

Context-based Archiving

The support for context-based archiving allows to define an archiving strategy for each attribute. A strategy is the list of contexts for which the attribute has to be archived. When an EventSubscriber is set to a context, only attributes that have this specific context in their strategy are archived, and all the remaining attributes are automatically stopped. The strategy configuration for each attribute is saved in an EventSubscriber property named *AttributeList*, using a "name=value" approach, where the name is "strategy" and the value is a list of context labels separated by "|". Some attribute configuration lines, taken from the *AttributeList* property, are shown in Table 1; it is worth noting the use of the Fully Qualified Device Name (FQDN) to identify each attribute, allowing an EventSubscriber to archive data coming from different TANGO facilities.

Table 1: AttributeList Property Configuration Excerpt

tango://srv-tango-srf.fcs.elettra.trieste.it:20000/eos/climate/ ⟷ 18b20.01/state; strategy=RUN|SHUTDOWN
tango://srv-tango-srf.fcs.elettra.trieste.it:20000/pil/laser/ ⟷ evops.01/state; strategy=RUN

The list of the context labels can be defined as TANGO database free property, and/or as class property and/or as device property named *ContextsList*. Whenever defined, device property setting overrides class property which in turn overrides free property. The *ContextsList* property contains an array of strings, specifying available context names, with

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA166

optional descriptions; the default value, that can be changed by the user, is shown in Table 2. An additional class/device

Table 2: Default Value for *ContextsList* Property

ALWAYS:Always stored
RUN:Stored during run
SHUTDOWN:Stored during shutdown
SERVICE:Stored during maintenance activities

property, named *DefaultStrategy*, allows to specify the default archiving strategy for all attributes. The default applies whenever not overridden by the attribute configuration. The default value for *DefaultStrategy* is ALWAYS.

The EventSubscriber device exposes a read-write memorised attribute, named *context*, to keep trace of the current running context. The *context* attribute is also exposed in the ConfigurationManager device to allow changing the context of all the managed EventSubscriber devices.

The effectivness of contexts can be easily noted in Fig. 2, where the number of events per minute (*attributeRecordFrequency* statistic) for three archivers of the FERMI lasers are shown. These complex laser systems count a large number of equipments that are selectively used, with respect to the desired laser functionality. The contexts, mapping different laser functionalities, have been defined and deployed during the August shut-down period, together with the archiving strategies for the attributes. The adoption of context-based archiving led to a significant decrease of the archiving rates, as just the context relevant attributes are archived at any time.

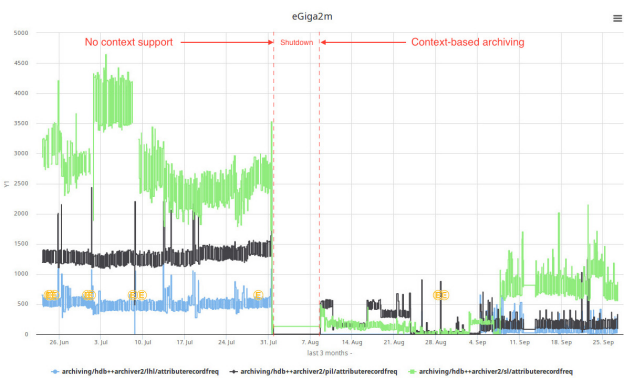

Figure 2: Archiving rates decrease with context-based archiving for FERMI laser systems. Graph plotted with eGiga2k web based tool.

Time-to-live

The support for temporary archiving has been added to the HDB++. A time-to-live parameter, expressed in hours, can be defined on per-attribute basis, and is saved as part of the attribute archiving configuration in the AttributeList device property, in the form "key=value". An excerpt of an *AttributeList* property, with the time-to-live specified, is depicted in Table 3. A number of TANGO commands and

Table 3: AttributeList Property Configuration Excerpt, Specifying Time-to-live

tango://srv-tango-srf.fcs.elettra.trieste.it:20000/eos/climate/ ↩ 18b20.01/state; strategy=RUN|SHUTDOWN; ttl=8760
tango://srv-tango-srf.fcs.elettra.trieste.it:20000/pil/laser/ ↩ evops.01/state; strategy=RUN; ttl=24

attributes have been added to the EventSubscriber Class to get/set time-to-live on each attribute configuration.

Deleting expired data is delegated to the back-end, with different implementations betweeen MySQL and Cassandra [3, 4]. For MySQL, a new column has been added to the *att_conf* table, and deletion is performed by an external program, running periodically, comparing the current time with the data timestamp plus the time-to-live. Cassandra natively allows to set a time-to-live parameter for each data and manages deletion internally.

Disk Usage

To reduce disk usage, when required, HDB++ can be configured with a reduced MySQL schema. Some of the columns of the tables used to support TANGO data types have been made optional: two of the three timestamp columns, e.g. the TANGO event timestamp and the insertion timestamp, the attribute quality and the error description carried by the event. If *lightschema=true* is specified in the *LibConfiguration* property, the archiving engine drops the support for "insert_time", "recv_time", "quality" and "att_error_desc_id" columns in all tables. Otherwise, at device startup, the library implementing the HDB++ MySQL schema checks the tables for "insert_time", "recv_time", "quality" and "att_error_desc_id" columns and builds a map with the information on which optional column is configured in which table.

Back-end Configuration

Configuration for different back-ends is supported via the LibConfiguration property; a specific set of *key=value* entries can be specified for each back-end. The libraries implementing the back-end are now dynamically loaded. The library implementing the interface, libhdbpp, parses the configuration stored in the LibConfiguration property, extracts the *libname* key and loads the shared object. Thus, to change an archiver back-end it is sufficient to change the *libname* key in the LibConfiguration property and restart the device.

Moreover, the interface in the libhdbpp library has been changed to return errors in the back-end functions as exceptions, that can be reported to the EventSubscriber and ConfigurationManager devices. The C++ libraries are now generating meaningful errors, carrying the full error stack and error description, whenever a problem on the database back-end arises.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA166

Graphical User Interfaces

Both the Java based Configurator GUI and HDB++ Viewer GUI have been improved. The Configurator GUI supports specifying per-attribute archiving strategies and time-to-live. The HDB++ Viewer can now display the attribute configuration history and attribute quality factor. Also, makes use of the "Display Unit" and "Conversion factor" as specified in the attribute configuration; a screenshot of HDB++ Viewer showing attribute configuration is depicted in Fig. 3.

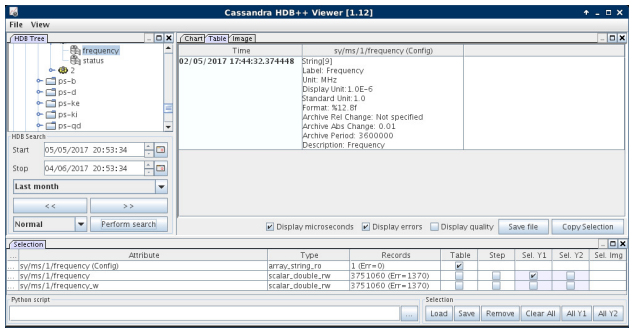

Figure 3: Java based HDB++ Viewer GUI.

A web-based viewer for HDB++, currently supporting Cassandra back-end, is under development at MaxIV. The front-end is based on node.js, Babel, React and Redux and managed by Webpack, whilst the back-end uses aiohttp and Boker/datashader. A screenshot is shown in Fig. 4.

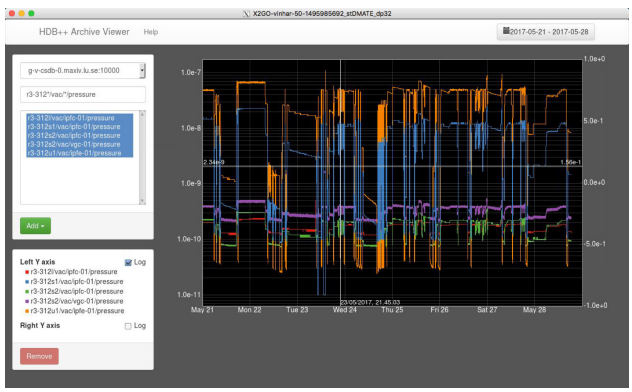

Figure 4: HDB++ web-based Viewer GUI under development at MaxIV.

CURRENT ISSUES

Cassandra being a distributed database, a special mechanism based on "tombstones" is used to delete data. When a row is deleted in Cassandra, this actually triggers the insertion of a tombstone identifying the row(s) to be deleted, with the timestamp associated to the delete. The data is not immediately deleted. When a user requests rows which have been deleted, Cassandra will compare the rows coming from different nodes and their associated timestamps and return those having the most recent timestamp. If the rows having the most recent timestamp are tombstones, no data is returned to the user for these rows. This mechanism prevents "data resurrection" whenever a node was temporarily down at the time the data have been deleted. Then, the tombstones will be deleted by the Cassandra compaction mechanism after a configurable amount of time (gc_grace parameter). Cassandra versions older than 2.2.10, 3.0.13 and 3.11.0 are impacted by a bug [5] which could lead to data resurrection after a while. The ESRF historical data archive, being based on Cassandra version 2.2.9, have been affected by this bug after the introduction of the new time-to-live feature. Upgrading Cassandra is reported to solve this bug.

HDB++ AT ELETTRA/FERMI

HDB++ is used at Elettra synchrotron and FERMI free electron laser (FEL). A virtual host for each particle accelerator is dedicated to running the MySQL back-end, the EventSubscriber and the ConfigurationManager TANGO devices. The Elettra setup consists of ten EventSubscribers and one ConfigurationManager, archiving a total of 2200 attributes. The average archive event rate is 1200 events/minute. For the FERMI FEL, three ConfigurationManager devices have been deployed, dedicated to the accelerator, the lasers and the photon optical sampling systems. A total of 39 archiver instances take care of more than 8100 attributes. The numbers for Elettra and FERMI are summarised in Table 4. The current size of the historical archive on disk is about 400 GB for FERMI.

Table 4: HDB++ Deployments at Elettra/FERMI

Elettra	EventSubscriber instances	10
	ConfigurationManager instances	1
	Nr. of attributes archived	~2200
	Average events/minute rate	~1200
FERMI	EventSubscriber instances	39
	ConfigurationManager instances	3
	Nr. of attributes archived	~8100
	Average events/minute rate	~7300

HDB++ AT ESRF

HDB++ deployment at ESRF is based on 36 EventSubscriber instances, archiving more than 8200 attributes into a Cassandra back-end, featuring six Cassandra nodes, release 2.2.9, in two different data centers. The first data center is dedicated to operations and accessible for writing by the archivers; the second is devoted to analytics. Moreover, the same Cassandra back-end is used for both the accelerator and the beamlines. The current setup is depicted in Fig. 5. The number of attributes and average number of archive events/s in the last three years are shown in Fig. 6. Apache Spark has been installed on the analytics data center; a simple Spark cron job, written in SCALA, computing simple statistics and storing the results in a special Cassandra table is running once a day. These results are not yet exploited but could

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA166

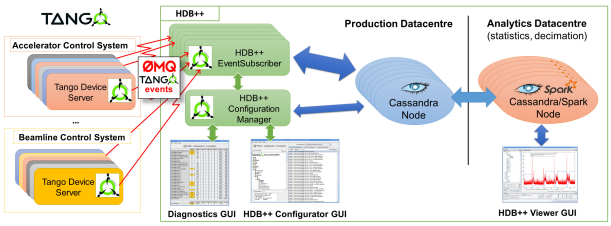

Figure 5: Current HDB++ deployment at ESRF.

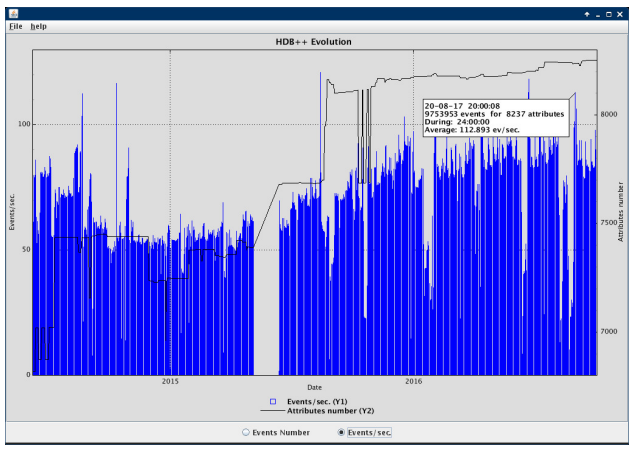

Figure 6: Number of archived attributes and events/s trends.

be used in the future by the HBDViewer GUI to prevent huge queries from the users and display only statistics in the appropriate case. For instance, if a user requests one year of master source frequency attribute, which is stored 3 times per second in average, the GUI will display only the 365 statistics computed for each day (min/max/average/standard deviation values, number of records for the day, number of errors stored for the day). The user will then hopefully be able to refine the query to get a more reasonable amount of data. Currently, the size of the historical database is about 850 GB.

HDB++ AT ALBA

HDB++ has been deployed at the MIRAS beamline (2016). It currently consists of one EventSubscriber instance with two devices archiving 160 attributes, 20 of them archived every 200 ms. Beamlines archiving is not centralized at ALBA, running instead small virtual database servers for each of them.

A "lite" version of the MySQL schema has been applied to reduce the disk footprint and memory requirements of an HDB++ database. The timestamp precission has been limited to milliseconds and modified to use native MySQL types (DATETIME). It allowed to halve the size of table indexes and apply monthly table partition. This setup allowed to load table indexes on RAM, improved overall performance and enabled online decimation of older data. The yearly disk footprint has been reduced from 70GB to 37GB.

A new HDB++ archive (MySQL) has been recently deployed for the Radiofrequency and Diagnostics archiving system. Attributes are being migrated gradually from polling-based archiving to HDB++, but the legacy archiving system may be still running for some years. To allow both systems to coexist, ALBA added methods to configure HDB++ and extract data to the PyTangoArchiving library and adapted the GUI, based on Taurus [6], to be able to plot data from both archiving systems. A screenshot of the Taurus based GUI is depicted in Fig. 7.

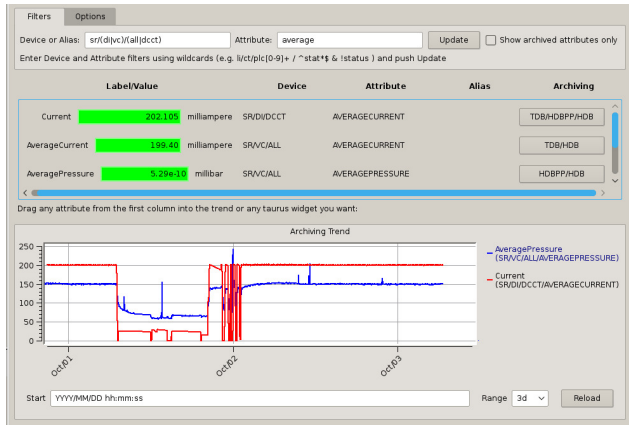

Figure 7: Taurus based GUI developed at ALBA.

REPOSITORY AND DOCUMENTATION

The HDB++ source code repository has been migrated from Sourceforge to Github. The current release, as well as the development branch, are available at: `https://github.com/tango-controls` and will in short be available as a dedicated organisation at Github [7]. Improved README and INSTALL instructions are available for each HDB++ component, as well as the Debian package build scripts. Moreover, building the HDB++ has been moved from simple Makefile to cmake. The project documentation has aligned to the TANGO Collaboration standard, and is now available on readthedocs [8].

Finally, pre-release Dedian packages are available at bintray [9].

CONCLUSION

At present, the HDB++ archiving system is in production at both Elettra/FERMI and ESRF sites since more than four years, and recently at the ALBA Synchrotron. The latest developments brought in some new functionalities to increase efficiency and optimise historical data management, improved flexibility for back-end configuration and more effective graphical user interfaces. Building scripts for Debian packages are available for HDB++, managing all dependancies and simplifying installation.

ACKNOWLEDGEMENT

Thank to Johan Forsber from MaxIV for his work on an alternative web-based historical data browser/viewer.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA166

REFERENCES

[1] TANGO Controls, http://www.tango-controls.org

[2] L. Pivetta *et al.*, "HDB++: a new archiving system for TANGO", in *Proc. ICALEPCS'15*, Melbourne, Australia, Oct. 2015, paper WED3O04, pp. 652–655.

[3] Apache Cassandra, http://cassandra.apache.org

[4] R. Bourtembourg *et al.*, "How Cassandra improves performances and availability of HDB++ TANGO archiving system", in *Proc. ICALEPCS'15*, Melbourne, Australia, Oct. 2015, paper WEM310, pp. 686–688.

[5] Cassandra bug, https://issues.apache.org/jira/browse/CASSANDRA-13153

[6] Taurus Scada, http://www.taurus-scada.org

[7] HDB++ repository, https://github.com/tango-controls-hdbpp

[8] HDB++ project documentation, http://tango-controls.readthedocs.io/en/latest/tools-and-extensions/archiving/HDB++.html

[9] Dedian pre-release packages, https://bintray.com/tango-controls/debian