ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-TUMR004

HDB++, A RETROSPECTIVE ON 5+ YEARS USING TIMESCALEDB

D. Lacoste, R. Bourtembourg, Grenoble, France
S. Rubio-Manrique, J. Ramos, ALBA-CELLS, Cerdanyola del Vallès, Spain
L. Pivetta, G. Scalamera, Elettra-Sincrotrone Trieste S.C.p.A., Basovizza, Italy
J. Forsberg, D. Egorov, MAX IV Laboratory, Lund, Sweden
T. Juerges, SKA Observatory, Jodrell Bank, United Kingdom
G. Jourjon, CSIRO, Canberra, Australia

Abstract

The Tango HDB++ project is a high-performance, event-driven archiving system that stores data with microsecond resolution timestamps. HDB++ supports various backend databases to accommodate any infrastructure choice, with TimescaleDB as the default option. TimescaleDB, an extension of PostgreSQL, is selected for its exceptional performance, reliability, and open-source license. After more than five years of using the system in production at major facilities such as the ESRF, MAX IV and SKAO, this paper presents the insights gained from operating HDB++ with TimescaleDB in a large research facility. Results are presented considering various perspectives. From a performance standpoint, the paper examines how the scalability features have maintained low query response times despite the continuous growth in data volume over the years. From the system administration perspective, findings show that standardized and proven technologies have consistently supported high-quality service delivery. Lastly, from the user perspective, we analyze how users can query data stored from the inception of the project up to last week within seconds, either from the Python API or from clients like Grafana. This capability is also enabled by the successful migration and integration of archived data from older or different systems into the database in full compliance with HDB++ standards.

INTRODUCTION

The Tango Historical DataBase++ (HDB++) archiving system [1] is developed through a collaborative effort among various facilities (Alba, Astron, Byte Physics, DESY, Elettra, ESRF, INAF, MAX IV, Observatory Sciences Limited, S2Innovation, SKAO) utilizing Tango Controls [2]. Since its inception [3], HDB++'s purpose has been to store Tango attribute values, i.e. process variables, in a backend of choice with microsecond resolution timestamps. The HDB++ community meets regularly and maintains a centralized repository with all HDB++ related developments and tools [4].

HOW DOES IT WORK?

Tango is an object-oriented control system built around the concept of devices. Devices hold attributes and can run commands. These can be executed or queried via a standard request and reply mechanism. On top of that, devices can emit data in the form of events that follow a standard publisher/subscriber scheme. HDB++ is fully event-based, and devices push the data they want to store. A special device, the EventSubscriber, is in charge of subscribing to these events and inserting them into a backend of choice.

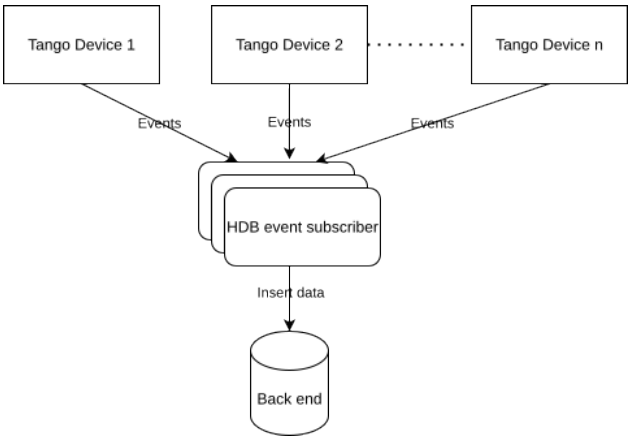

Figure 1: HDB++ Overview.

The design in Fig. 1 is quite efficient as data is pushed asynchronously by the devices, their normal operation is not hindered by the need to read data in order to archive it. It offers a simple way to scale as one can have as many EventSubscribers as needed. If there is a need to absorb more signals, simply instantiate a dedicated EventSubscriber. And finally an abstraction was introduced at the insertion level so that it is possible to insert data into different backends. The same abstraction level is used by the PyHdbppPeriodicArchiver process, to archive periodic data from those systems that for any reason (e.g. firewall rules) are not able to send events.

Extraction and Displaying of Data

HDB++ provides two graphical applications for archiving configuration and data extraction, HDBConfigurator [5] and HDBViewer [6]. In addition, there are several tools for CLI, API, and web extraction. HDB++ can be configured using scripts and yaml files [7], and a unified Python API has been developed for data extraction [8].

The pyhdbpp library provides a common abstract class for developing reader objects, and working implementations for most used backends, MySQL and TimescaleDB. As most of the Tango Controls tools, pyhdbpp is available for download on the Python Package Index [9] and conda-forge

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-TUMR004

[10] channels. This approach allowed to share tools between different institutes [11] and integrate different HDB++ backends on existing GUI toolkits like Taurus [12] or web visualization suites like Grafana [13] via plugins as can be seen in Fig. 2.

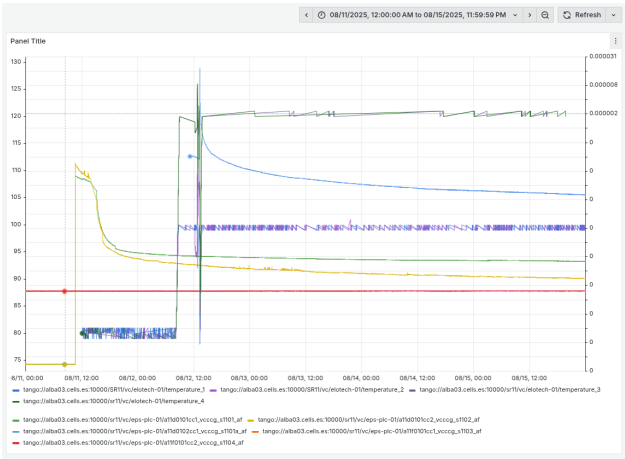

Figure 2: Visualization of bakeouts pressures and temperatures using HDB++ on a Grafana dashboard.

WHY A TIME-SERIES DATABASE? WHY TIMESCALEDB?

The initial stages of the project utilized MySQL/MariaDB or Cassandra as data storage backends. This choice was made out of pure necessity and convenience because MySQL/MariaDB was already used as the backend for the HDB++ predecessor HDB [14] and Cassandra was promoted as the promising new technology.

Despite Cassandra's strong assurances on data retention and its promise to be able to retrieve data from a large data pool within fractions of seconds, performance issues arose when retrieving data. This could be traced back to the setup of the Cassandra cluster and its misconfiguration. Due to the complexities associated with managing and configuring a Cassandra cluster, as well as the lack of clear and concise documentation for Cassandra, TimescaleDB [15] was incorporated as a backend and soon became the default choice for all facilities.

The fact that system performance remains solid despite data growth confirms the validity of the decision. In terms of both data insertion and extraction, current operations are far from reaching a performance bottleneck, as demonstrated by benchmarks. TimescaleDB, in conjunction with PostgreSQL, supported by their active communities, is considered a reliable project with extensive documentation, resources, and an ecosystem that promotes ease of cluster management.

HDB++ WITH TIMESCALEDB SAMPLE DEPLOYMENT

One can deploy a TimescaleDB cluster as they see fit, but for clarity here is a description of the deployment at the ESRF that was used as the reference for most of this paper.

This cluster consists of three nodes, a primary one that will be used for data insertion, and two live replicas that are used for data extraction. The PostgreSQL nodes are running bare metal on identical machines and connected through a 10Gb ethernet dedicated link. The three nodes are managed through Patroni [16] that handles the cluster and automatic failover. All requests are processed through an HAProxy [17] proxy that handles the load balancing between reading from the two live replicas. The system is backed up using barman [18] that allows point-in-time recovery over a certain period. The whole design can be summed up in the following Fig. 3.

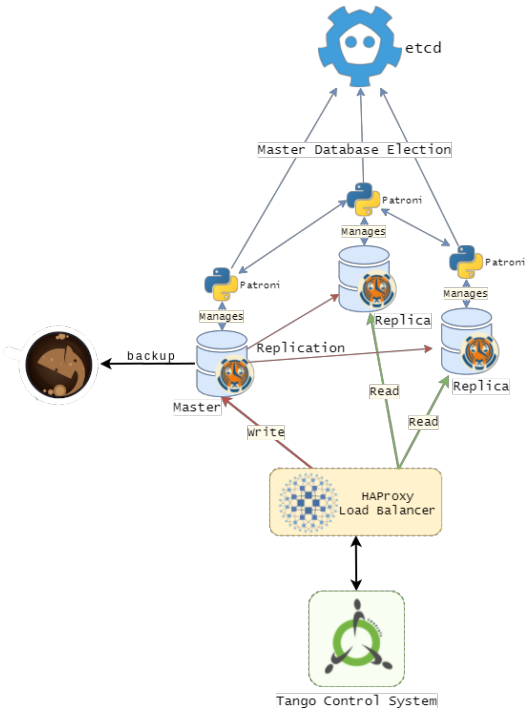

Figure 3: TimescaleDB sample cluster deployment.

BENCHMARKS

Compression

One of the interesting features of TimescaleDB is compression. TimescaleDB boasts great performance for this feature with a big amount of data storage saved without any penalties on speed. This feature was tested on the HDB++ schema.

The results are on par with TimescaleDB promises. Data compression is quite efficient and the decompression is fast enough that the penalty on data retrieval is quite minimal. This makes it a must-have feature for long-term storage. The

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-TUMR004

Table 1: Compression

format	double array	double scalar
raw size	1.208 GiB	351 MiB
compressed size	801 MiB	31.5 MiB
compression ratio	0.66	0.09
query time	1446 ms	4.14 ms
compressed query time	1570 ms	6.01 ms

only drawback is that compressed data is immutable (unless one is willing to decompress the data, which is a timely process) thus competing with the Time To Live (TTL) feature of HDB++ that deletes data after a configured amount of time. Setting up compression on TimescaleDB then sets up a maximum limit on the interval for TTL.

LESSONS LEARNED

Data Migration: Should One Migrate Data or Should One Not?

It is possible to operate HDB++ with different backends, be it either different instances of the same backend, or even different backends altogether. Nevertheless, with the deprecation of Cassandra as a backend, the decision was taken to investigate the possibility to migrate data from Cassandra to TimescaleDB.

This migration was carried out at the ESRF and MAX IV, all the data was extracted from the decommissioned Cassandra cluster to be imported into a dedicated TimescaleDB instance. This was a tedious process, as Cassandra suffered from issues extracting data. It was a two-step operation, the first one being data extraction to CSV files, that were then copied to the new TimescaleDB instance. Even though the two backends are different, the design was similar enough that the data could be inserted almost as-is. One of the strengths of TimescaleDB is that even though it is a time series database, data does not need to be inserted chronologically. Using PostgreSQL COPY command data insertion from a CSV file was blazing fast, and after some reordering done in the background, the TimescaleDB cluster was up with data available in a really short time.

Based on the success of this operation, it was decided to migrate data from the old HDB schema to TimescaleDB. The data was not organized the same way this time, so it needed some post-processing to match the new schema. This was done, proving that it is possible to migrate from HDB to HDB++ with TimescaleDB without too many issues.

Migrating to a new backend is a long process that takes time. If existing data can be discarded, then it is an easier task. But if data needs to be retained, then it is definitely possible, and it is worth it in the end. The reason simply is that it is the only way to have already existing data, that otherwise might have to be kept in a deprecated system, in the latest HDB++, up to date, and with all the benefits of TimescaleDB. It is also easier to maintain a couple of homogeneous clusters than to maintain different technology stacks.

Reliability and Dependability of HDB++

At the ESRF, TimescaleDB was started in production in 2019. After an overlap period in which both the Cassandra and TimescaleDB clusters were operated, the TimescaleDB cluster remained alone in operation since then, for 6 years now. During this period, the cluster underwent several maintenance operations and upgrades. The biggest one being a full replacement of the cluster, updating the hardware, the PostgreSQL major versions and most of the software stack. During these 6 years some failures happened in different parts of the cluster, it never incurred any unplanned downtime. Actually, the only downtime used for maintenance operations occurred during the control system's downtime, so it can be said that the cluster operated with 100% availability. In fact, most maintenance operations are carried out on the live system. TimescaleDB updates can be done on a live system, and since a multi-node cluster is used, restarting one of the nodes can be done without impairing the whole cluster availability.

On several occasions, one of the database instances either stopped and could not restart, or crashed, or ended up on an out of sync state with the rest of the cluster. The chosen design ensures that failure of one node is never an issue. If the primary server fails, the switchover to replicas always works, keeping the system operable with a primary node for data insertion but only one for data extraction. The failure of a replica node would end up in the same state but without switchover. In all these cases recovery is possible, in the worst case by tearing down the failing node and rebuilding it. The failure of two nodes at the same time is unlikely, and one can always add more nodes to the cluster, increasing reliability, as well as scaling up the system in one go.

The biggest task at hand was the cluster update. The strategy was to perform the update as follows:

1. Prepare the new cluster
2. Stop the old cluster
3. Perform an in-place update on one of the machines from the old cluster
4. Start this machine as the primary node on the new cluster
5. Build the new nodes
6. Stop the primary node that was part of the old cluster
7. Decommission the old cluster

This approach was chosen because it is possible to stop and roll back at any point, making it safe. Furthermore, PostgreSQL offers several options for upgrading between major versions. Logical replication can be used between nodes running different versions; it does not require any

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-TUMR004

Table 2: HDB++ Deployment for SKA Low with Four Stations

SPC cluster	MCCS per station	MCCS cluster total
60 devices	79 devices	399 devices
57 device servers	20 device servers	227 device servers
220 attributes archived	1499 attributes archived	8149 attributes archived

downtime then as the new node can be populated and then a switchover can happen. Currently, Patroni does not support logical replication; however, once supported, it could be the preferred method for this task. The other options are to perform an in-place update or dump the database to a compliant file format, such as CSV, and reimport it. The last approach, while guaranteed to work, requires much more time and disk space, and thus was discarded.

PostgreSQL is battle-tested in high-demand environments, one can build a high availability service on top of it without worries. In the facilities where it is deployed, it has again been proven that it performs very well.

Scalability When Needed

At SKAO [19] HDB++ is deployed in Kubernetes [20] as part of the Engineering Data Archive (EDA) [22]. The entire EDA software package consists of a web-based configuration and monitoring front-end, an HDB++ configuration manager, and an HDB++ event subscriber. Additionally, a TimescaleDB server is required as HDB++ storage backend.

At the SKA-Low telescope [21], which is still under construction, HDB++ archives monitoring data of four SKA Low stations (about 1300 of 130k dipoles), of the general station cluster's monitor and control software, and of an SPC cluster. Table 2 shows how many Tango devices with their attributes are publishing Tango archive events in the current deployment.

The first test results clearly show that a single HDB++ event subscriber is insufficient. This outcome matches the prediction of the HDB++ community when SKAO consulted it before the EDA deployment was planned in its current form. The team on site has modified the deployment and manually added additional HDB++ event subscribers to distribute the load of event subscription and archiving of the event data. Improvements in the EDA source code and Helm charts have recently been released and allow the EDA to support multiple event subscribers in the same deployment. The new version has not been deployed yet.

The first results also indicated that the combination of Grafana, its PostgreSQL connector and TimescaleDB can lead to apparent dead locks either in TimescaleDB or in the HDB++ event subscriber that prevent it from storing any further attribute values in the TimescaleDB backend. The root cause investigation is ongoing.

The first deployments also clearly indicate that always deploying a TimescaleDB server together with the EDA software is impractical. The simple reason being that the TimescaleDB server is used for multiple EDA deployments with their own tables. An update of one deployment would re-deploy the TimescaleDB server and with it make it unavailable during the update for the other deployments. Therefore have the software packages been split up into the TimescaleDB server deployment and the EDA deployment where it will create the needed tables in the already available TimescaleDB server.

Backend-Agnostic Visualization Tools

Several clients have been developed to access either MySQL or TimescaleDB HDB++ engines using SQL language from different types of clients (Python, PHP, Java, C++, SQLAlchemy). Web-based tools have been used for decades to visualize Tango Control Systems archiving data [23] [24], for both old legacy system and HDB++ in its different flavors (Cassandra, MySQL/MariaDB, TimescaleDB). Both performant and shared between some institutes, adapting them was sometimes challenging due to the particularities of each of the HDB++ installations.

The pyhdbpp library was developed to be backend-agnostic, serving as an intermediate layer to facilitate reuse of visualization and extraction tools. Its reader abstract class allows to extract data in the same format for any backend and to mix different data sources into a single client, enabling easy integration of HDB++ data into any visualization tool and overcoming the previous limitations by providing a single API. Following the same approach, connectors between pyhdbpp and non-HDB++ data sources have also been developed.

In the last ten years, Grafana has become a common tool for visualization in all Tango collaboration institutes. A pyhdbpp-plugin has been developed [25] to enable using Grafana for any of the hdb++ backends, and to easily add any new backend (or non-Tango / non-hdb++ data source) into the same dashboards. Usage of the plugin alleviated some of the deadlocks encountered when accessing database sources directly from Grafana, and simplified applying decimation and aggregation methods on the extracted data.

CONCLUSION

Archiving poses special requirements on software as it needs not only to be efficient and reliable, but also capable of operating for decades. In the lifetime of a scientific facility, a lot of software changes, from the smallest part to the control system itself. HDB++ has proven to be an excellent archiving solution, especially with TimescaleDB as the backend. It shows great performance, top of the shelf reliability. It is feature rich and extendable. Migrating data to TimescaleDB is possible, sometimes tedious, but lets one have all historical

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-TUMR004

data in a single subsystem substantially improves the overall maintainability of an archiving system. TimescaleDB is considered the default backend today, but the extendable HDB++ design makes adding a new backend straightforward enough that the project can easily support better alternatives, may they be found. Backed by the support of the HDB++ community, this project should remain the cornerstone of archiving for the Tango community for years to come.

ACKNOWLEDGEMENTS

The authors would like to thank the Tango Controls HDB++ community members for their contributions and great ideas.

REFERENCES

[1] HDB++ documentation at readthedocs, https://tango-controls.readthedocs.io/en/latest/Explanation/archiving/hdbpp.html

[2] About Tango, https://www.tango-controls.org/about-us

[3] L. Pivetta *et al.*, "HDB++: A New Archiving System for TANGO", in *Proc. ICALEPCS'15*, Melbourne, Australia, Oct. 2015, pp. 652–655. doi:10.18429/JACoW-ICALEPCS2015-WED3O04

[4] HDB++ GitLab repository, https://gitlab.com/tango-controls/hdbpp

[5] GUI tool for HDB++ configuration, https://gitlab.com/tango-controls/hdbpp/hdbpp-configurator

[6] GUI viewer for HDB++, https://gitlab.com/tango-controls/hdbpp/hdbpp-viewer

[7] CLI tool for HDB++ bulk configuration, https://gitlab.com/tango-controls/hdbpp/yaml2archiving

[8] pyhdbpp, the HDB++ Python connector, https://gitlab.com/tango-controls/hdbpp/libhdbpp-python

[9] pyhdbpp packages at the Python Package Index, https://pypi.org/project/pyhdbpp/.

[10] List of Tango Conda Packages, https://tango-controls.readthedocs.io/en/latest/How-To/installation/conda.html

[11] D. Lacoste *et al.*, "New Developements on HDB++, the High-performance Data Archiving for Tango Controls", in *Proc. ICALEPCS'23*, Cape Town, South Africa, Oct. 2023, pp. 1190–1194. doi:10.18429/JACoW-ICALEPCS2023-THMBCM001

[12] Taurus framework for scientific GUI's development in Python, https://taurus-scada.org

[13] Grafana Dashboards, https://grafana.com/grafana/dashboards/.

[14] HDB repository, https://gitlab.synchrotron-soleil.fr/software-control-system/tango-controls-archiving

[15] TimescaleDB PostgreSQL extension, https://github.com/timescale/timescaledb

[16] Patroni orchestrator for PostgreSQL clusters, https://github.com/patroni/patroni

[17] HAProxy High Availability proxy for PostgreSQL, https://www.haproxy.org

[18] Barman, Backup and Recovery Manager for PostgreSQL, https://pgbarman.org

[19] The Square Kilometre Array Observatory, https://skao.int

[20] Kubernetes, https://kubernetes.io

[21] SKA-Low, https://www.skao.int/en/explore/telescopes/ska-low

[22] Documentation SKAO Engineering Data Archive, https://developer.skao.int/projects/ska-tango-archiver/en/latest/.

[23] eGiga2m, a web graphic data viewer and exporting tool, https://gitlab.elettra.eu/puma/client/egiga2m

[24] Webapp for viewing HDB++ archived Tango attribute data in a TimescaleDB database, https://gitlab.com/tango-controls/hdbpp/archviewer

[25] HDB++ Grafana Python plugin, https://gitlab.com/tango-controls/hdbpp/libhdbpp-grafana-connector