ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA020

PUSHING THE LIMITS OF TANGO ARCHIVING SYSTEM USING PostgreSQL AND TIME SERIES DATABASES

R. Bourtembourg, S. James, J.L. Pons, P. Verdier, ESRF, Grenoble, France
G. Cuni, S. Rubio-Manrique, ALBA-CELLS Synchrotron, Cerdanyola del Vallès, Spain
G.A. Fatkin, A.I. Senchenko, V. Sitnov
BINP SB RAS and NSU, Novosibirsk, Russia
L. Pivetta, C. Scafuri, G. Scalamera, G. Strangolino, L. Zambon
Elettra-Sincrotrone Trieste S.C.p.A., Basovizza, Italy
M. Di Carlo, INAF - OAAB, Teramo, Italy

Abstract

The Tango HDB++ project is a high performance event-driven archiving system which stores data with micro-second resolution timestamps, using archivers written in C++. HDB++ supports MySQL/MariaDB and Apache Cassandra back-ends and has been recently extended to support PostgreSQL and TimescaleDB [1], a time-series PostgreSQL extension. The PostgreSQL back-end has enabled efficient multi-dimensional data storage in a relational database. Time series databases are ideal for archiving and can take advantage of the fact that data inserted do not change. TimescaleDB has pushed the performance of HDB++ to new limits. The paper will present the benchmarking tools that have been developed to compare the performance of different back-ends and the extension of HDB++ to support TimescaleDB for insertion and extraction. A comparison of the different supported back-ends will be presented.

INTRODUCTION

The HDB++ Tango archiving system [1] relies on the Tango archive events feature to collect Tango attributes values coming from one or several Tango Control Systems and then store these values in the Database back-end of your choice. The following back-ends are currently supported: MySQL/MariaDB, Cassandra, PostgreSQL, TimescaleDB and Elasticsearch. This list could be easily extended, thanks to the layered design of the HDB++ architecture.

HDB++ DESIGN

The HDB++ Tango archiving system relies on two main components:

- The EventSubscriber Tango device server which subscribes to Tango archive events for a list of Tango attributes and store the received events in a database
- The ConfigurationManager Tango device server which simplifies the archiving configuration and management

An abstraction library, named *libhdb++* decouples the interface to the database back-end from the implementation. To be able to store data to a specific database back-end, the EventSubscriber and the ConfigurationManager Tango devices dynamically load a C++ library implementing the methods from the libhdb++ abstract layer. The libhdb++ back-end library is selected via a Tango device property. This allows to use the same tools to configure and manage the archiving system with all the supported Database back-end. The archiving part of the HDB++ design is presented in Fig. 1.

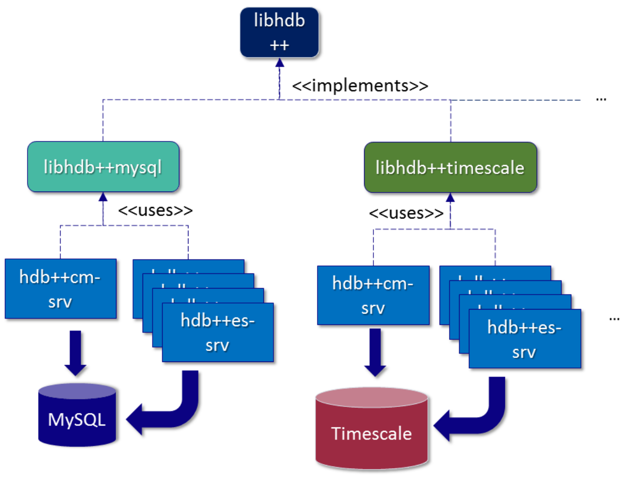


Figure 1: HDB++ Tango devices design.

Tools to help configuring the system, retrieving or viewing the archive data are also available.

SUPPORTED BACK-ENDS

MySQL/MariaDB

MySQL is a well known, widely adopted SQL database engine. After the acquisition by Oracle, in 2010, a complete open-source fork, named MariaDB, became available. MySQL and MariaDB are almost inter operable, even if some differences in the supported data types and database engines require a careful approach.

HDB++ at Elettra and FERMI The HDB++ MySQL back-end has been in production at Elettra and FERMI since 2015. Both accelerators share the same architecture: two nodes, configured in high-availability, are in charge of running all the virtual machines hosting the control system

[1] https://timescale.com

servers. The deployed architecture is based on MySQL master-slave replication. The master is used as the ingest node, where only the production hosts are allowed to insert data; all the historical data queries are redirected to the replica, that is also configured to keep online historical data older than three years. ProxySQL has been used to hide the two databases hosting current and old data sets. The production setup is depicted in Fig. 2.

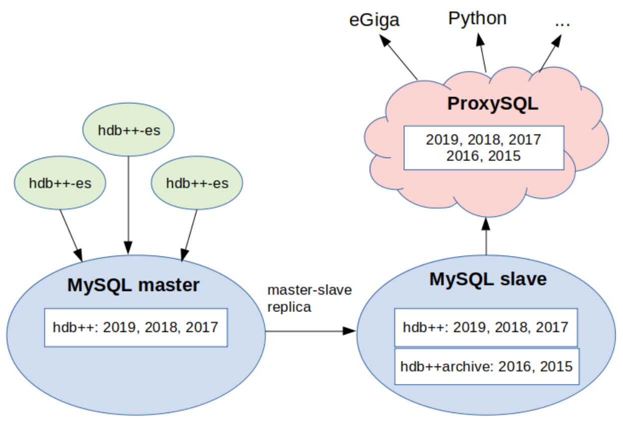


Figure 2: MySQL master-slave replica with ProxySQL gateway for historical data queries.

Due to the relatively small size of the archives, in FERMI the master is currently 350 GB on disk, one virtual machine, featuring 8 CPUs and 32 GB of RAM, can easily run the MySQL database back-end, 4 ConfigurationManager and 48 EventSubscriber Tango devices. Figure 3 shows the total number of attributes archived, currently ~14700, and the number of inserts per minute, peaking up to 55K. Machine shut-downs, where the insert rate is quite low, are clearly visible in the time window depicted, spanning over the last 9 months; archiving errors are also highlighted.

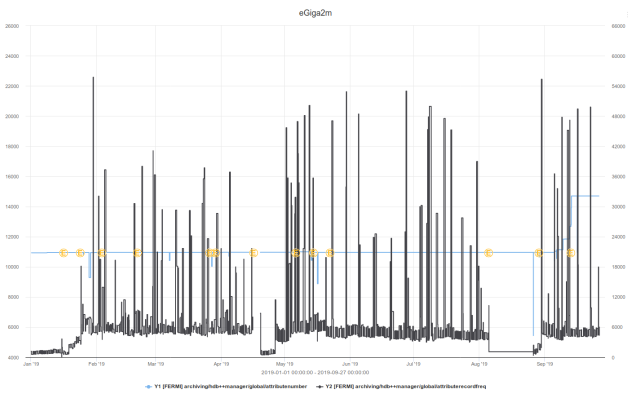


Figure 3: Total number of attributes archived (blue) and cumulative inserts per minute (black) for FERMI.

HDB++ at ALBA Synchrotron ALBA is a third generation synchrotron located in Barcelona, with 8 beamlines in operation and 4 more in construction. ALBA started deploying HDB++ Archiving System in 2016 for its Accelerators Control System and one of its beamlines. As of fall 2019, 11000 attributes are being archived in 6 different MariaDB databases (a 60% of the total archived attributes) using a total amount of 4 TB. Current databases increase at a rate of 300GB per month, but are later decimated for long term storage.

The approach of using multiple databases has been adopted to reduce the size of each of the databases, thus reducing the size of tables and indexes. It reduced the time needed for queries and any maintenance procedure being executed in the database. Distribution between databases is determined by attribute subsystem, using regular expressions. The PyTangoArchiving API [2] developed by ALBA takes care of distributing attributes amongst databases at setup time, and to extract and merge the data when queried from client applications. PyTangoArchiving allows to merge data from both HDB++ and the legacy TANGO Archiving System, as well as other sources like ALBA radiation monitors database or cooling system database.

All archived data is kept as it is inserted for a period of 3 months, being later decimated to a 1 value every 10 seconds and moved to other database servers for longer term storage. Insertion is done either by event-based subscribers (that can hit insert-rates so high as 20 Hz per attribute) or by periodic archiving collectors, polling periodically every 1 to 60 seconds. The usage of events or polling is determined by the implementation of the device servers providing the attributes.

Cassandra

Apache Cassandra [3] is an open source distributed database management system available under the Apache 2.0 license. Cassandra's master-less ring architecture where all nodes play an identical role, is capable of offering true continuous availability with no single point of failure, fast linear scale performance and native multi data center replication. When using the HDB++ Cassandra back-end [4], users can take advantage of Cassandra TTL (time-to-live) feature. If TTL is set, inserted values are automatically removed from the database after the specified time.

The HDB++ Cassandra back-end has been used at the ESRF during the last 4 years. The cluster grew up to 9 nodes, distributed in 2 data centers, one 6 nodes data center dedicated to write queries and one 3 nodes data center with SSD dedicated to read queries, with a replication factor of 3 in the 2 data centers. Cassandra appeared to be very reliable for writing data without downtime even during the Cassandra upgrades. The main issue with the HDB++ Cassandra back-end was the poor performances when querying spectrum (array) Tango attributes data. A query involving a spectrum attribute and returning a big amount of data could bring down the read data center. This is due to the fact that Cassandra is written in Java and answering to queries involving a lot of data will trigger the garbage collector at some point. The Cassandra node becomes unresponsive while the garbage collector is running and, if the garbage collector is triggered at the same time

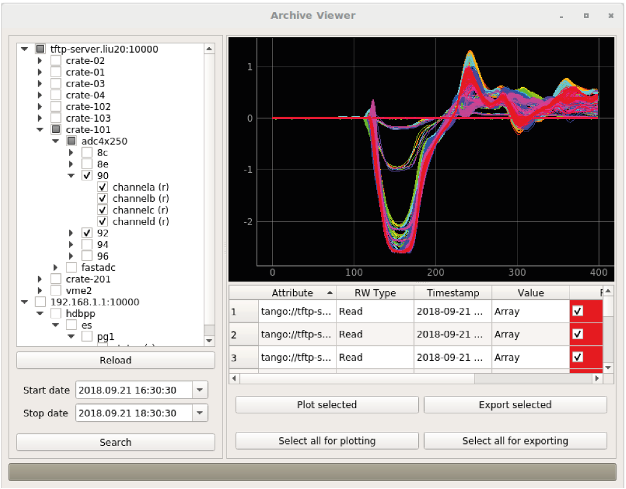
ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA020

on several nodes, the clients will get errors. This garbage collector issue could probably be avoided by using Scylla [5], a drop-in replacement of Apache Cassandra, written in C++, which should be compatible with the HDB++ Cassandra back-end. An alternative improvement could be to reduce the partitions size for the tables containing array data, to add more nodes on the read data center and to increase the replication factor on the read data center to distribute the big queries over more nodes, but this would require more resources to maintain this bigger cluster.

PostgreSQL

PostgreSQL is a well-known open-source relational DB that is widely used in the scientific community. Main motivation for the development of PostgreSQL backend for the HDB++ archiving system was the need to provide a storage for a large quantity of waveforms that are presented as spectrum (array) Tango attributes at the LIA-20 facility [6]. Deploying several Cassandra nodes was found to be impractical, and Cassandra doesn't support 64-bit unsigned types. MySQL storage of the array data type is very inefficient. PostgreSQL has an efficient way of array storage, and pguint extension was used to provide unsigned attributes storage. To increase exchange speed a binary interface for PostgreSQL was used.

Python and C++ libraries for data extraction were provided. And a new utility for viewing and extracting archived data in python based on PyQt and PyQtGraph was developed. This utility provides a similar functionality to HDB++ Viewer and also allows to export data in HDF5-format. A view of the utility is shown in Fig. 4.



Start date 2018.09.21 16:30:30
Stop date 2018.09.21 18:30:30

	Attribute	RW Type	Timestamp	Value	
1	tango://tftp-s...	Read	2018-09-21 ...	Array	✓
2	tango://tftp-s...	Read	2018-09-21 ...	Array	✓
3	tango://tftp-s...	Read	2018-09-21 ...	Array	✓

Figure 4: A view of the python archive viewer utility.

TimescaleDB

TimescaleDB [7] is a time series extension to PostgreSQL available from Timescale as an open source, community and enterprise product.

After a comparison of TimescaleDB performances with Cassandra, the ESRF decided to use TimescaleDB as back-end for the archiving system of the upgraded synchroton, named EBS.

Deployment of the HDB++ TimescaleDB system at the ESRF is loosely in three phases:

- Phase 1: Initial deployment involving building a stable and correctly functioning HDB++ cluster.
- Phase 2: Potential optimisation and improvements to insert/extraction pipeline.
- Phase 3: Implement long term archiving strategy.

When designing a cluster around TimescaleDB, the goal was to build a fault tolerant, performant and scalable solution. The design shown in Fig. 5 is based on recommendations made by Timescale in their own evaluations [8].

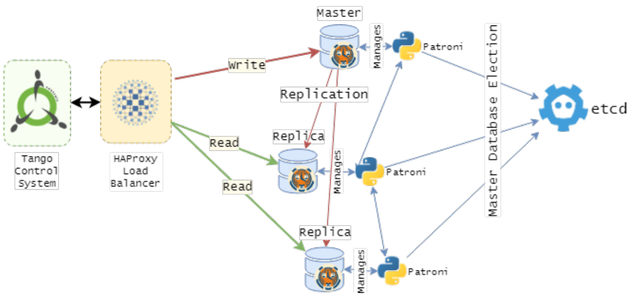


Figure 5: TimescaleDB HDB++ deployment at ESRF.

This design is built around three database nodes, one running as the Master and two Replica nodes. The cluster is fault tolerant and failover is automatic, with Patroni handling the failure event. On failover, the proxy is dynamically reconfigured with the new layout of the cluster. Performance is maximised by having the Master handle only data ingress, with all query requests load balanced to the Replicas. It is possible to scale the query performance by adding additional Replica nodes in future.

Taking advantage of libpqxx (A modern C++ layer above libpq), a new libhdbpp-timescale backend has been made available [9]. Using libpqxx provides the opportunity to quickly explore and test performance features in the future, for example batch writing data to the database.

Deployment and integration into the Tango HDB++ system is an ongoing project. The TimescaleDB database itself is rapidly evolving and adding both better performance and additional time series features that can potentially be integrated into the Tango HDB++ solution in future.

Elasticsearch

Elasticsearch [10] is a real-time distributed search and analytics engine. The term "real-time" refers to the ability to search (and sometimes create) data as soon as they are produced; traditionally, in fact, web search crawls and indexes web pages periodically, returning results based on relevance to the search query. It is distributed because its indices are divided into shards with zero or more replicas. But the main ability is the analytics engine which

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA020

allows the discovery, interpretation, and communication of meaningful patterns in data. It is based on Apache Lucene [11], a free and open-source information retrieval software library, therefore Elasticsearch is very good for full text search (like for logging data or text files in general). It is developed alongside a data-collection and log-parsing engine called Logstash, and an analytics and visualization platform called Kibana. The three products are designed for use as an integrated solution, referred to as the "Elastic Stack" (formerly the "ELK stack").

The main features of Elasticsearch are:

- no transaction: no support for transaction;
- schema flexible: there is no need to specify the schema upfront;
- relations: denormalization [12], parent-child relations and nested objects;
- robustness: to properly work, elasticsearch requires that memory is abundant;
- distributed: it is a CP-system in the CAP (Consistency-Availability-Partition tolerance) theorem [13]
- no security: there is no support for authentication and authorization.

Implementation An implementation of the HDB++ project on ELK has been done: the prototype had to be able to work with REST [14] and with Json data [15] and, for this reason, two libraries have been selected to include these functionalities: “REST client for C++” [16] and “Json for modern C++” [17]. More information about the implementation can be found here [18].

BENCHMARKING TOOLS

hdbpp-metrics Github repository [19] has been created to gather tools which can be used to help to benchmark an HDB++ archiving system.

FIRST BENCHMARK RESULTS

MySQL vs PostgreSQL

Two tests were done at BINP for PostgreSQL vs MySQL performance. First one was conducted using *wave* attribute that is a spectrum read-only of 256 DevDouble elements and was stored 250000 times. The second one used *arr* attribute that is a spectrum read-write of 4096 DevULong elements and was stored 5000 times. The results including write times and table size are summarized in Table 1. These benchmarks show that PostgreSQL significantly improves the memory and speed efficiency of the storage of array data types.

TimescaleDB vs MySQL vs PostgreSQL

Preliminary tests have been run to compare the performance of the different supported back-ends on the same hardware. The relevant system and application software versions used are: Ubuntu 16.04.3 LTS, MySQL 5.7.18, PostgreSQL 11.3 and TimescaleDB 1.3.0. A dump of the actual FERMI historical database has been done, taking care of addressing the differences in the database schema used by the different back-ends, focusing on the largest table, which currently counts more than 4 billion rows. *pt-fifo-split* has been used to split the table into chunks before loading into PostgreSQL; loading the 4 billion row table took ~30 hours using the standard SQL *COPY* and ~20 hours with the *timescaledb-parallel-copy* command, at the expense of increasing disk usage by ~5%.

A typical query, extracting a time series for a Tango double scalar attribute, spanning over two different time windows, has been run for the different back-ends. The first time window results in ~1 M rows query, the second in ~10 M rows. Moreover, the time necessary to execute the queries has been measured in two conditions: after a database cold restart, guaranteeing all the caches are flushed, and just after the first query. For TimescaleDB the measurements have also been taken before and after the *CLUSTER* command. Results are presented in Table 2.

Table 1: A Comparison of PostgreSQL vs MySQL Waveform and Arr Storage. Times are in μs, table size is in Mbyte

Attribute	Min	Average	Max	Size
MySQL wave	2551	4669	253042	4440
PostgreSQL wave	243	358	23087	768
MySQL arr	50718	61145	198584	5511
PostgreSQL arr	1641	2082	14173	251

Table 2: HDB++ Supported Database Engine Query Benchmark

Engine	Cache	1 M	10 M
MySQL (InnoDB)	Cold	2.3 s	22.7 s
	Hot	2.0 s	21.0 s
TimescaleDB	Cold	5.3 s	36.5 s
	Hot	1.2 s	12.2 s
TimescaleDB+CLUSTER	Cold	1.9 s	15.0 s
	Hot	1.2 s	11.9 s
PostgreSQL	Cold	1.8 s	15.9 s
	Hot	1.2 s	12.6 s

It is worth noting the good performance numbers of plain PostgreSQL with respect to TimescaleDB. Likely, the explanation can be found in the simple queries used for this benchmark, that, anyhow, resemble most of the queries run. An effective advantage using TimescaleDB should arise whenever complex queries are made.

TimescaleDB vs Cassandra Evaluation

Here the community edition is evaluated as a replacement for Cassandra in the ESRF HDB++ archiving system.

Test Setup Tests were run on a production grade borrowed Cassandra server, with 32 Cores, 128GB RAM and a RAID 0 SSD array. TimescaleDB was tuned using the parameters offered from the online tuning tool PGTune. 1 year of data was extracted from the live ESRF Cassandra cluster. For the best performance, data was ingested in time order, thus all test data files were sorted into time order for bulk loading. Achieving the best extraction performance means the data tables have to be clustered on a composite attribute id + data time index for the HDB++ schema. Finally, the TimescaleDB hyper table chunk sizes tested were all below 7 days. All results presented are "cold", i.e. not using database cache capabilities to improve performance.

Insert Performance TimescaleDB is built with very high insert performance as a feature [20]. Testing was with a Timescale provided bulk loading tool. Insert rates achieved were above 440,000 rows per second for a scalar double, and 45,000 rows per second for an array double.

Query performance Testing with a simple query to extract data between two time points for a scalar archived once per second, the results show a remarkable improvement. The improvement is over 1000% against Cassandra for each time period as shown in Table 3.

Table 3: TimescaleDB vs. Cassandra Scalar Attribute Query Benchmark

Query	Rows return	Timescale	Cassandra
1 Hour	3 298	8 ms	111 ms
1 Day	84 238	127 ms	~2.6 s
1 Week	505 858	748 ms	~11.9 s
1 Month	2 498 762	~3.8 s	~1 min
3 Months	7 204 234	~10 s	~2 min 37 s
12 Months	28 699 357	~43 s	~10 min

Choosing an extreme scenario (an array attribute archived multiple times per second) with the same simple query, select data between two time points, the results shown in Table 4 are again extremely good.

Table 4: TimescaleDB Arrays Query Benchmark

Query	Rows return	Data Size	Timescale
1 Hour	3 298	~3.7 MB	693 ms
1 Day	84 238	~90 MB	~15.6 s
1 Week	505 858	~633 MB	~114 s
1 Month	2 498 762	~2.62 GB	~8 min
3 Months	7 204 234	~7.38 GB	~30 min
12 Months	28 699 357	~17.77 GB	~63 min

The worst case 12 months query completed in just over 60 minutes, pulling over 17GB of data from over 28 millions rows. It should be noted Cassandra could not complete a query greater than a day for this attribute archive at high rate. As can be seen from the results above, querying large chunks of attribute array data is expensive. PostgreSQL provides an additional tool, the ability to query individual elements of an array. While this is around 80% slower than an equivalent simple scalar query, it is consistently 1500% quicker than requesting the full array.

Benchmark Conclusion TimescaleDB offers a significant performance boost over Cassandra under the evaluated conditions. The bulk of the data requested by ESRF users is currently time period based, but TimescaleDB provides support for significant gains with more complex queries, opening up new possibilities for the retrieval and visualisation of data in the future.

CONCLUSION

HDB++, thanks to its design, allows to store data into different DB backends. This encourages contributions in the HDB++ community and 5 backends are already supported. As presented in this paper, the new PostgreSQL and TimescaleDB backends improve the user experience when dealing with Tango spectrum (array) attributes. The Cassandra back-end is currently a good fit for use cases where high availability is required for writing the archiving data and where scalar attributes data has to be stored. It is not a good fit for retrieving big chunks of data, in particular for getting multi-dimensional data. The MySQL/MariaDB backend is deployed in several institutes. Some deployment ideas for this backend have been presented in this paper. The Elasticsearch backend is a good fit for users interested in benefiting from the advantages of the ELK stack. The HDB++ project is supported by an active community. The latest face-to-face collaboration meeting took place in Grenoble from 18th to 20th September 2019 [21].

ACKNOWLEDGEMENTS

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

REFERENCES

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

[2] PyTangoArchiving, https://github.com/tango-controls/PyTangoArchiving

[3] Apache Cassandra, https://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] Scylla, https://www.scylladb.com/

[6] G. A. Fatkin *et al.*, "LIA-20 Control System Project", in *Proc. ICALEPCS'17*, Barcelona, Spain, Oct. 2017, pp. 1485–1488. doi:10.18429/JACoW-ICALEPCS2017-THPHA052

[7] TimescaleDB Github repository, https://github.com/timescale/timescaledb

[8] Evaluating high availability solutions for TimescaleDB + PostgreSQL, https://blog.timescale.com/blog/high-availability-timescaledb-postgresql-patroni-a4572264a831/

[9] libhdbpp-timescale Github repository, https://github.com/tango-controls-hdbpp/libhdbpp-timescale

[10] Elasticsearch, https://www.elastic.co/

[11] Apache Lucene, https://lucene.apache.org/

[12] Denormalization, http://en.wikipedia.org/wiki/Denormalization

[13] CAP theorem, http://en.wikipedia.org/wiki/CAP_theorem

[14] REST, http://en.wikipedia.org/wiki/Representational_state_transfer

[15] JSON, http://www.json.org

[16] REST client for C++, https://github.com/mrtazz/restclient-cpp

[17] JSON for Modern C++, https://github.com/nlohmann/json

[18] M. Di Carlo *et al.*, "HDB@ELK: another noSql customization for the HDB++ archiving system", in *Proc. SPIE 10707, Software and Cyberinfrastructure for Astronomy V*, Austin, USA, Jul. 2018. doi:10.1117/12.2312464

[19] hdbpp-metrics, https://github.com/tango-controls-hdbpp/hdbpp-metrics/

[20] TimescaleDB vs. PostgreSQL for time-series, https://blog.timescale.com/blog/timescaledb-vs-6a696248104e/

[21] HDB++ September 2019 Meeting Minutes, https://github.com/tango-controls-hdbpp/meeting-minutes/blob/master/2019-09/Minutes.md