HDB++ at ALBA: Changes in MariaDB and PyTangoArchiving API

Sergi Rubio Manrique, ALBA Synchrotron, Barcelona

Hamburg, June 2019

HDB/TDB Databases at ALBA Accelerators

HDB/TDB is the "old" Tango Archiving system developed by Soleil, based on MySQL and Java archiver device servers

HDB database (with resolution > 10 seconds) storing 15993 attributes was decommissioned in 2018 and replaced by HDB++ and longterm TDB.

TDB database (with resolution> 1 second) has been extended to store 6 months of data. It currently stores 9956 attributes.

HDB and TDB data older than 6 months is moved to a backup server, where it is kept available for clients to do queries on older data.

HDB++ at ALBA

HDB++ was first introduced in MIRAS beamline in 2017, and in ALBA accelerators on winter of 2017.

Since 2018 it is used for Accelerators, currently storing 10371 attributes.

Data tables are partitioned each month and removed from the active server after three months. Older data is available on backup servers and can be queried any time via PyTangoArchiving (data origins are hidden to the user).

We still keep the TDB database active storing 9956 attributes. These attributes are not the same stored by HDB++ (for reasons explained later).

TDB and HDB++ data can be queried/plotted together transparently.

PyTangoArchiving

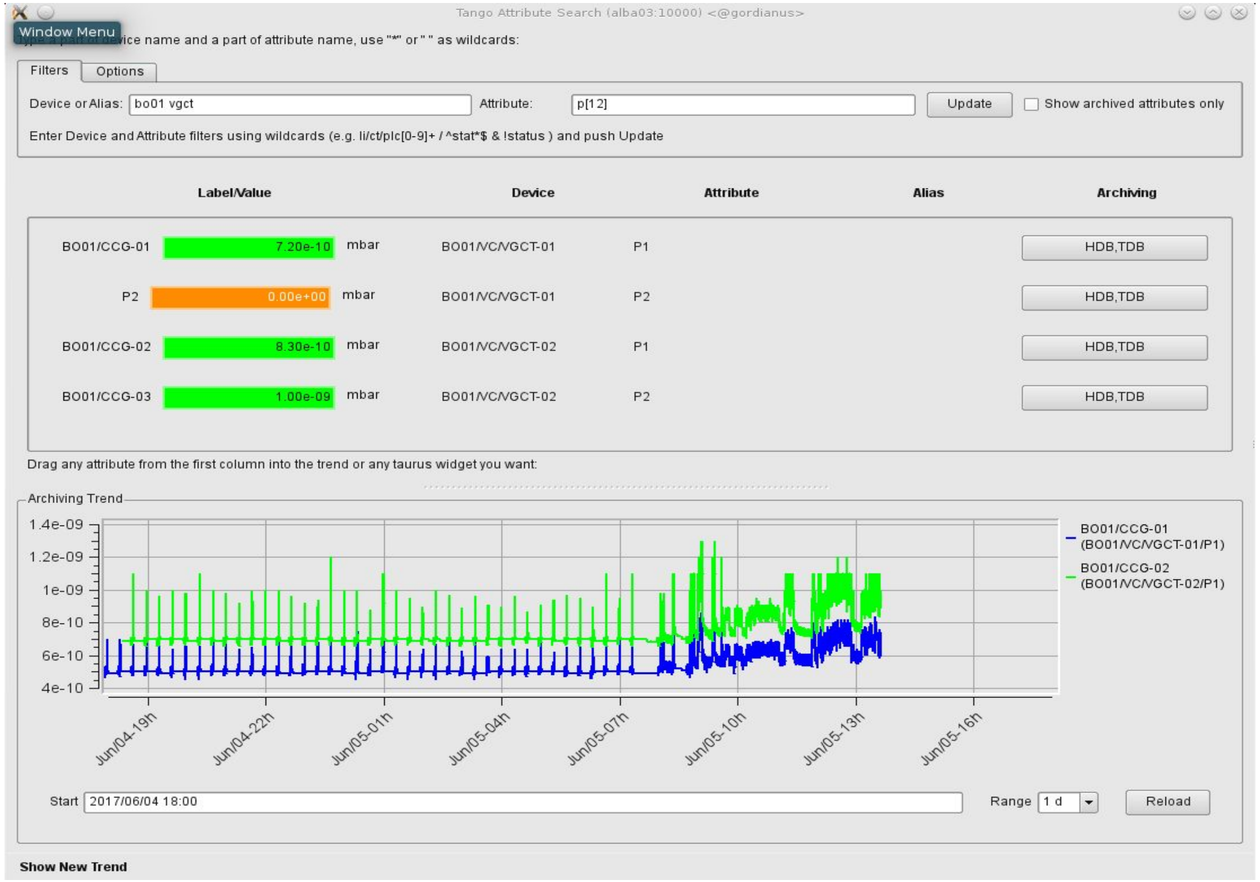
DB-agnostic data extraction using PyTangoArchiving.Reader() object

```
rd, A = PyTangoArchiving.Reader(),  'sr/di/dcct/averagecurrent'

rd.is_attribute_archived(A) : [ hdbdi, tdb, hdbacc ]

rd.get_attribute_values(A, date1, date2)

PyTangoArchiving.Reader('hdbdi').get_attribute_values(A, -3600)
```

API for HDB/TDB, HDB++ and Snap; but supports any python-based reader

Allows configuration of archiving via scripts, partition-based backup of MariaDB databases and provides scripts and Qt Widgets for extraction and plots.

Includes PyExtractor device server for extracting data without local access.



Tango Attribute Search (alba03:10000) <@gordianus>

Window Menu

...ice name and a part of attribute name, use "*" or " " as wildcards:

Filters | Options

Device or Alias: bo01 vgct Attribute: p[12] Update ☐ Show archived attributes only

Enter Device and Attribute filters using wildcards (e.g. li/ct/plc[0-9]+ / ^stat*$ & !status) and push Update

Label/Value			Device	Attribute	Alias	Archiving
BO01/CCG-01	7.20e-10	mbar	BO01/VC/VGCT-01	P1		HDB,TDB
P2	0.00e+00	mbar	BO01/VC/VGCT-01	P2		HDB,TDB
BO01/CCG-02	8.30e-10	mbar	BO01/VC/VGCT-02	P1		HDB,TDB
BO01/CCG-03	1.00e-09	mbar	BO01/VC/VGCT-02	P2		HDB,TDB

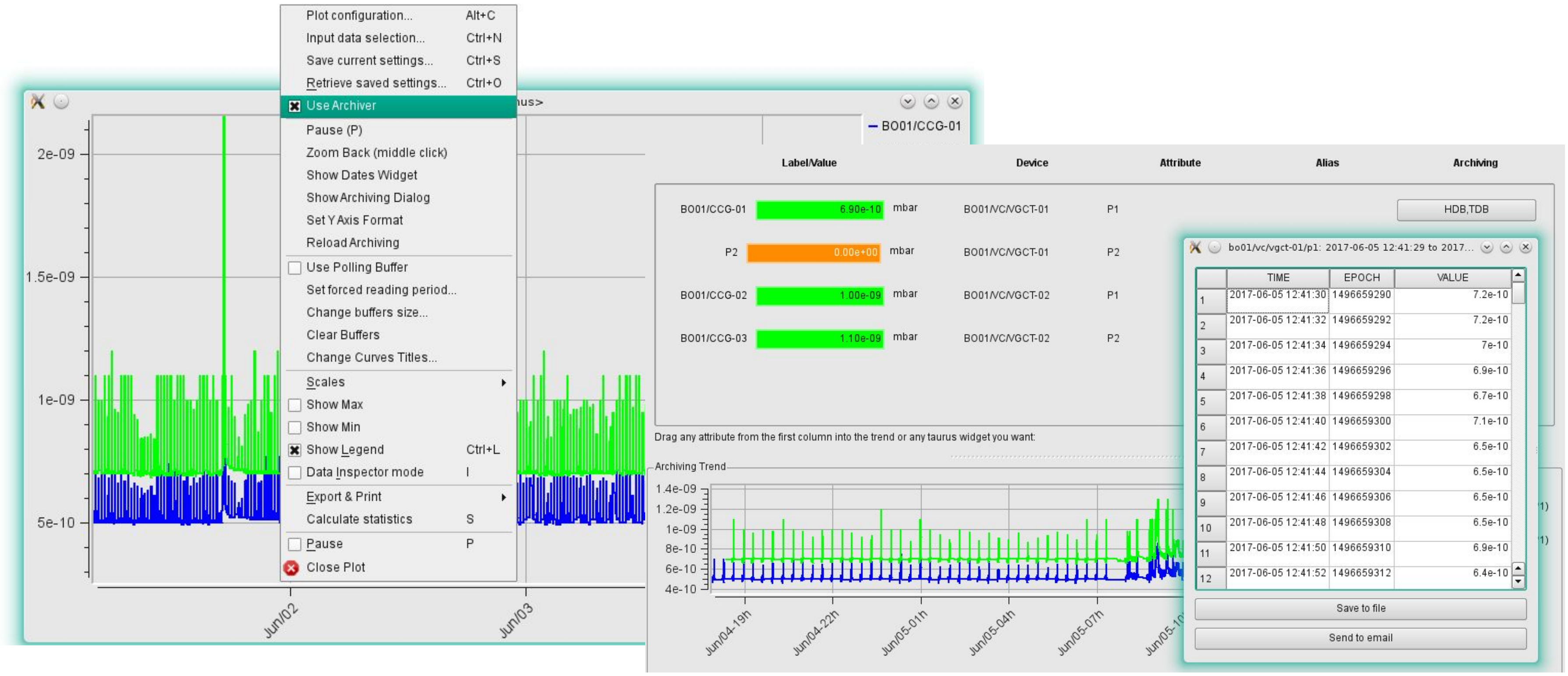
Drag any attribute from the first column into the trend or any taurus widget you want:

Archiving Trend

Start 2017/06/04 18:00 Range 1 d Reload

Show New Trend

PyTangoArchiving widgets (pre-pyqtgraph)

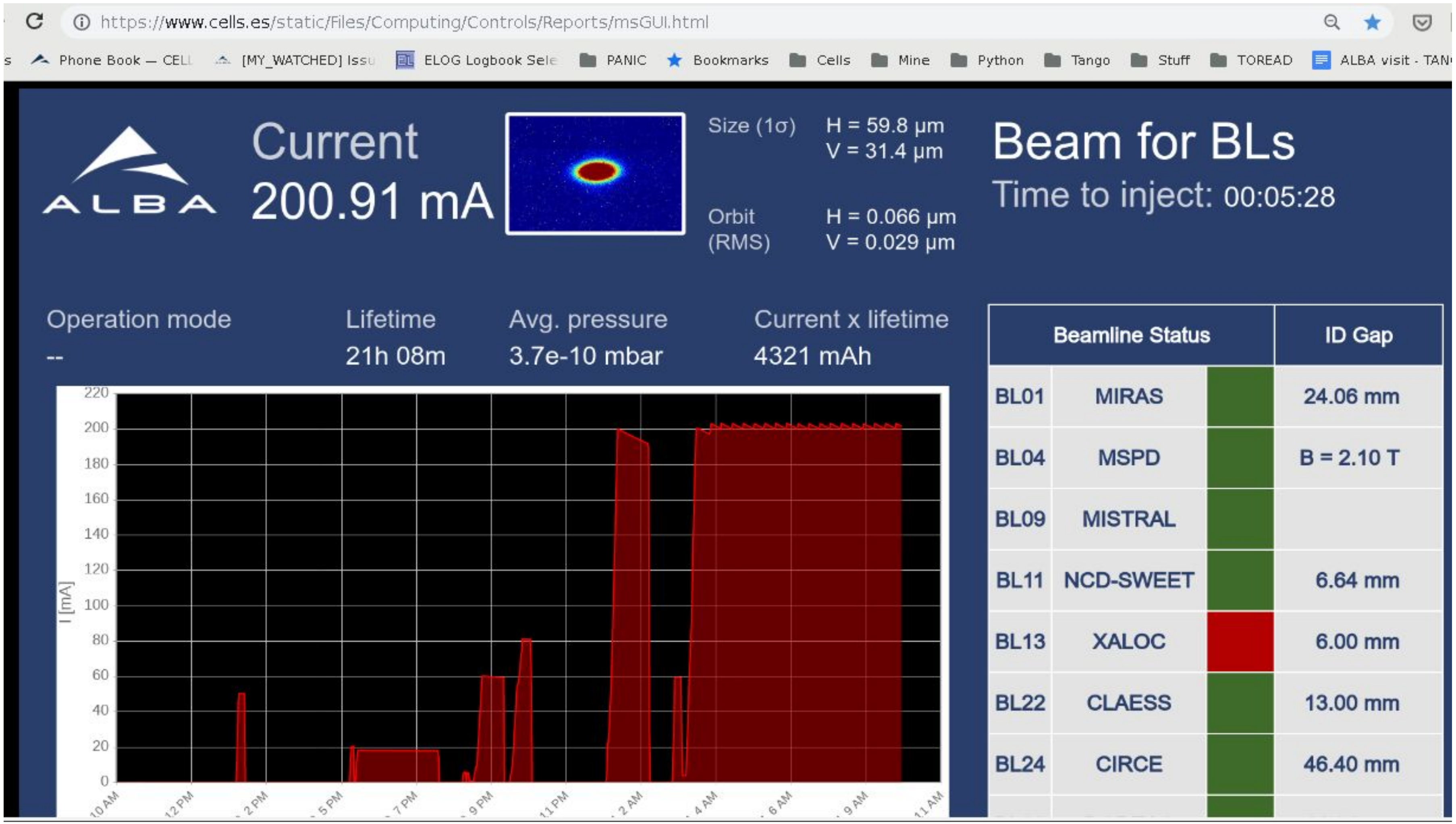


Label/Value			Device	Attribute	Alias	Archiving
BO01/CCG-01	6.90e-10	mbar	BO01/VC/VGCT-01	P1		HDB,TDB
P2	0.00e+00	mbar	BO01/VC/VGCT-01	P2		
BO01/CCG-02	1.00e-09	mbar	BO01/VC/VGCT-02	P1		
BO01/CCG-03	1.10e-09	mbar	BO01/VC/VGCT-02	P2		

Drag any attribute from the first column into the trend or any taurus widget you want:

bo01/vc/vgct-01/p1: 2017-06-05 12:41:29 to 2017...

	TIME	EPOCH	VALUE
1	2017-06-05 12:41:30	1496659290	7.2e-10
2	2017-06-05 12:41:32	1496659292	7.2e-10
3	2017-06-05 12:41:34	1496659294	7e-10
4	2017-06-05 12:41:36	1496659296	6.9e-10
5	2017-06-05 12:41:38	1496659298	6.7e-10
6	2017-06-05 12:41:40	1496659300	7.1e-10
7	2017-06-05 12:41:42	1496659302	6.5e-10
8	2017-06-05 12:41:44	1496659304	6.5e-10
9	2017-06-05 12:41:46	1496659306	6.5e-10
10	2017-06-05 12:41:48	1496659308	6.5e-10
11	2017-06-05 12:41:50	1496659310	6.9e-10
12	2017-06-05 12:41:52	1496659312	6.4e-10



Current
200.91 mA

Size (1σ)	H = 59.8 μm
	V = 31.4 μm

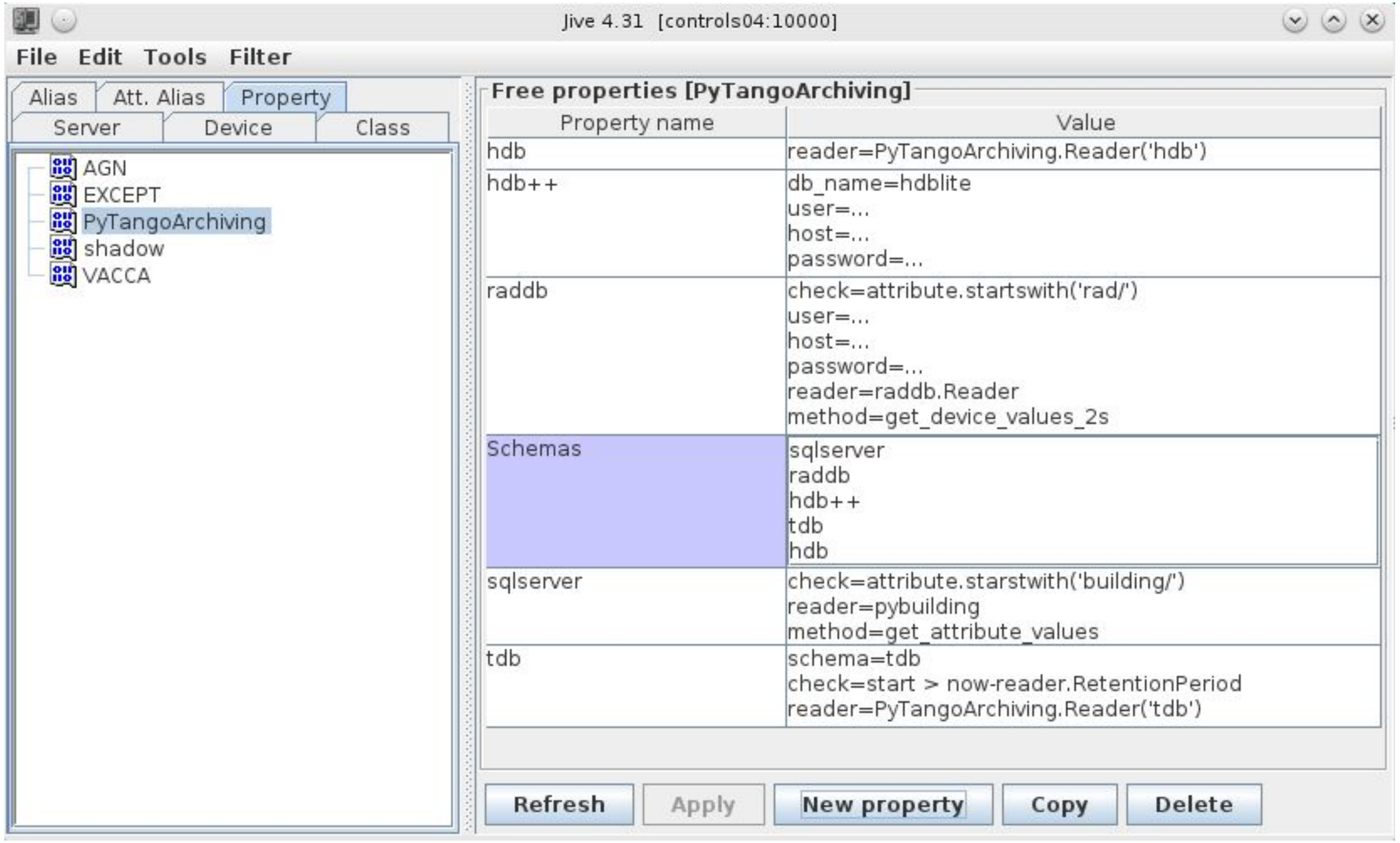
Orbit (RMS)	H = 0.066 μm
	V = 0.029 μm

Beam for BLs
Time to inject: 00:05:28

Operation mode	Lifetime	Avg. pressure	Current x lifetime
--	21h 08m	3.7e-10 mbar	4321 mAh

Beamline Status			ID Gap
BL01	MIRAS	🟩	24.06 mm
BL04	MSPD	🟩	B = 2.10 T
BL09	MISTRAL	🟩	
BL11	NCD-SWEET	🟩	6.64 mm
BL13	XALOC	🟥	6.00 mm
BL22	CLAESS	🟩	13.00 mm
BL24	CIRCE	🟩	46.40 mm

PyTangoArchiving: Multiple Schemas



Free properties [PyTangoArchiving]

Property name	Value
hdb	reader=PyTangoArchiving.Reader('hdb')

hdb++	db_name=hdblite
	user=...
	host=...
	password=...

raddb	check=attribute.startswith('rad/')
	user=...
	host=...
	password=...
	reader=raddb.Reader
	method=get_device_values_2s

Schemas	sqlserver raddb hdb++ tdb hdb

sqlserver	check=attribute.starstwith('building/')
	reader=pybuilding
	method=get_attribute_values

tdb	schema=tdb
	check=start > now-reader.RetentionPeriod
	reader=PyTangoArchiving.Reader('tdb')

HDB++ at ALBA Accelerators, introduction

Some particularities of ALBA:

- old hardware (~10 years old, 32 bit cpu's, suse 11, Tango 7)
- DB's migrated from MySQL5.5 to Debian 9 and MariaDB 10
- Python 2.6, thus, cpu/memory consumption is usually higher than on C++ based control systems (and one affects each other).
- We don't use Java clients for configuration nor data retrieval. Instead we rely on python API's (PyTangoArchiving, fandango) and clients (Taurus).
- By design, we avoid using "jive-based" polling to generate events. Instead, devices push events from code on hardware conditions change.
- For those devices not triggering events, we rely on client-based polling.

HDB++ at ALBA, device-related problems

DAQ/RF/Diagnostic/Orbit devices at ALBA update their attribute values at frequencies that may reach up to 10 Hz.

Pushing Archiving events from Tango devices required polling from Jive every 100 ms or to modify the device code to push Change and Archive events.

But, as discovered by Graziano Scalamera (Elettra), pushing both events caused delays and high cpu consumption of devices, generating timeouts on devices that used to work smoothly.

High cpu triggers mem leaks in python (known issue in python 2.6); and we cannot assume a PyTango 9.3 upgrade in machines using DAQ hardware

HDB++ at ALBA, server-side problems

As we were running TDB and HDB++ in parallel, we have made some comparison between them.

For the same amount of data (Beam Loss monitors, read every second), the amount of disk needed by HDB++ is 4 times bigger (300GB/year against 70GB/year).

The size of indexes for each partition is as big (or even bigger) than the data itself (20-50GB per partition/month depending on the data type). This is caused by indexes combininig both key and timestamp, instead of only time as in the old DB.

Apart of that, sometimes events are not stored without any error detected by subscriber. We blame on ZMQ and T8 devices, but it affected users confidence.

HDB++ at ALBA, extraction-related problems

HDB/TDB stored each attribute on a different table. The amount of data generated by one device didn't affect others. But only 1 DB was allowed.

In HDB++ all attributes of the same type (e.g. devdouble read-only scalars) are stored together in the same table. And literally "compete" when multiple attributes are queried.

This creates huge index files and very slow queries. As comparison, requesting 4-5 days of machine current data from HDB++ (2e6 points) may take 5 minutes, longer than requesting a full year of data from TDB (31e6 points).

This is why we keep TDB online, as HDB++ is being avoided by users and used only when high-resolution data is needed.

HDB++ at ALBA, device-related solutions

We modified **HdbEventSubscriber** (via PR) to be able to subscribe to CHANGE event instead of ARCHIVE as a Fallback.

This behaviour it's not the default, it's introduced via property (SubscribeChangeEventAsFallback); and only applies whenever an attribute is added and not providing archive events.

It allowed to archive data from devices with huge number of attributes that do not support jive-based polling (e.g. Sardana).

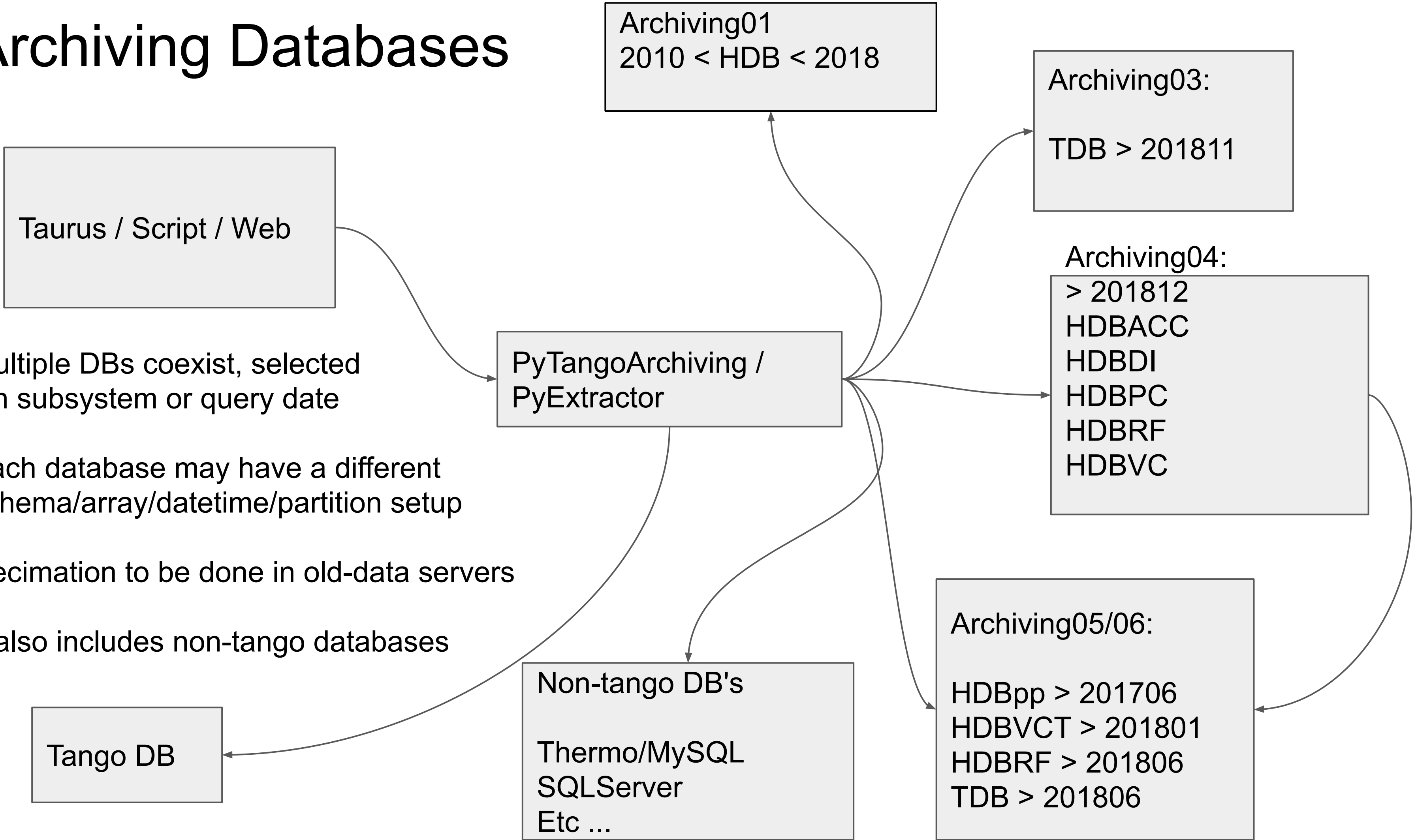
In addition, we developed **PyHdbppPeriodicArchiver** collector. It is a polling based collector (as the one used by TDB) that allows to store data periodically in HDB++ tables without the need of doing any change on the device side.

HDB++ at ALBA, server-side solutions

Solutions appeared due to difficulties in HDB++ maintenance:

- we cannot stop vacuum/temperatures archiving during shutdowns, so we had to stored them separately.
- a separate database improved both insertion and extraction performance, so we did the same for the rest of subsystems.
- Now we have 6 HDB++ databases (RF, DI, VC, CT, ACC, PC); storing 1K-2K attributes each and with sizes between 100 and 300GB for 6 months data.
- Splitting the database in subsystems removed all the previous problems, finally allowing to set HDB++ as the default source of data for clients
- In addition, we implemented aggregation of data on queries (on demand by user), reducing the amount of data to be sent/converted by clients.

Archiving Databases



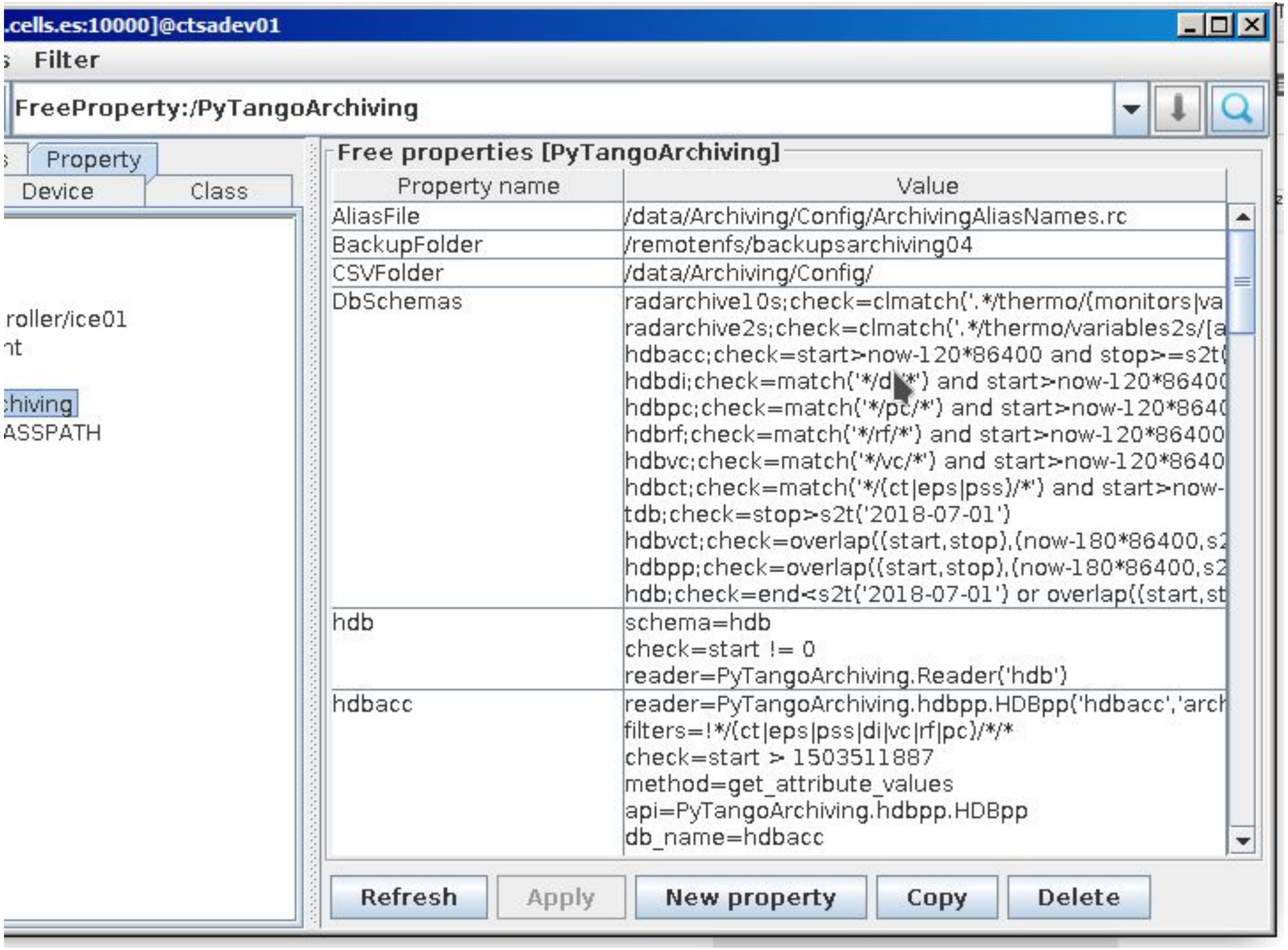
Multiple DBs coexist, selected
On subsystem or query date

Each database may have a different
schema/array/datetime/partition setup

Decimation to be done in old-data servers

It also includes non-tango databases

Multiple Schemas



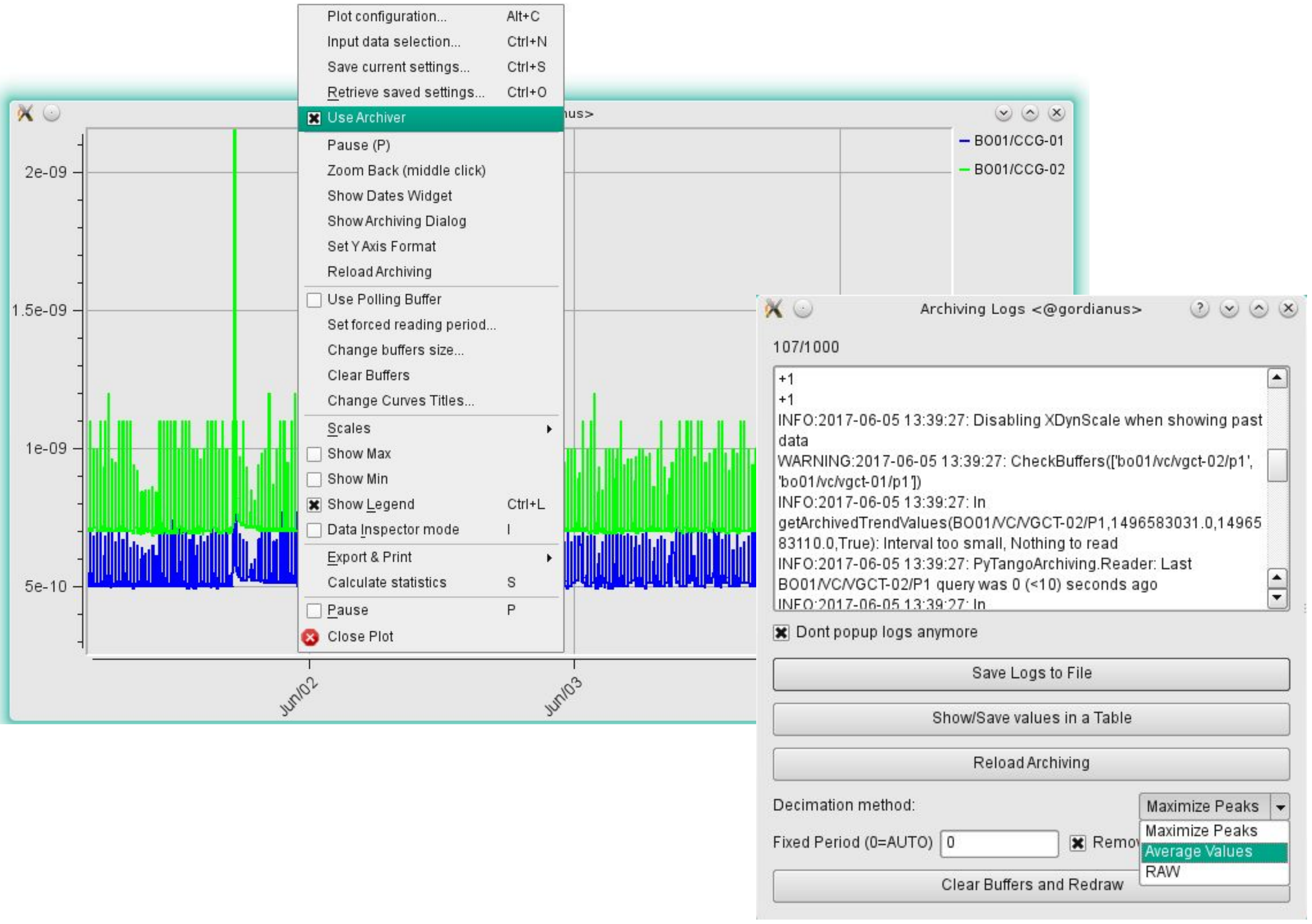
roller/ice01
ıt
:hiving
ASSPATH

Free properties [PyTangoArchiving]

Property name	Value
AliasFile	/data/Archiving/Config/ArchivingAliasNames.rc
BackupFolder	/remotenfs/backupsarchiving04
CSVFolder	/data/Archiving/Config/
DbSchemas	radarchive10s;check=clmatch('.*/thermo/(monitors|va radarchive2s;check=clmatch('.*/thermo/variables2s/[a hdbacc;check=start>now-120*86400 and stop>=s2t(hdbdi;check=match('*/di/*') and start>now-120*86400 hdbpc;check=match('*/pc/*') and start>now-120*8640 hdbrf;check=match('*/rf/*') and start>now-120*86400 hdbvc;check=match('*/vc/*') and start>now-120*8640 hdbct;check=match('*/(ct|eps|pss)/*') and start>now- tdb;check=stop>s2t('2018-07-01') hdbvct;check=overlap((start,stop),(now-180*86400,s2 hdbpp;check=overlap((start,stop),(now-180*86400,s2 hdb;check=end<s2t('2018-07-01') or overlap((start,st

hdb	schema=hdb
	check=start != 0
	reader=PyTangoArchiving.Reader('hdb')

hdbacc	reader=PyTangoArchiving.hdbpp.HDBpp('hdbacc','arch
	filters=!*/(ct|eps|pss|di|vc|rf|pc)/*/*
	check=start > 1503511887
	method=get_attribute_values
	api=PyTangoArchiving.hdbpp.HDBpp
	db_name=hdbacc



107/1000

```
+1
+1
INFO:2017-06-05 13:39:27: Disabling XDynScale when showing past
data
WARNING:2017-06-05 13:39:27: CheckBuffers(['bo01/vc/vgct-02/p1',
'bo01/vc/vgct-01/p1'])
INFO:2017-06-05 13:39:27: In
getArchivedTrendValues(BO01/VC/VGCT-02/P1,1496583031.0,14965
83110.0,True): Interval too small, Nothing to read
INFO:2017-06-05 13:39:27: PyTangoArchiving.Reader: Last
BO01/VC/VGCT-02/P1 query was 0 (<10) seconds ago
INFO:2017-06-05 13:39:27: In
```

[✓] Use Archiver
[✓] Show Legend
[✓] Dont popup logs anymore

Decimation method: Maximize Peaks

Fixed Period (0=AUTO): 0 [✓] Remov

Changes in MariaDB schema

Data_time resolution reduced to milliseconds.

Indexes may be based on data_time or TO_SECONDS (integer).

This uses a virtual column (int_time), used for querying.

Partition by months, only current month is backup, loading is incremental.

Data split in databases depending on subsystem.

It provides smaller indexes, files and shorter maintenance times.

Allow to apply different schemas/db engines for different applications.

Scripts available on PyTangoArchiving/hdbpp module

MariaDb issues, and time for Timescale

Supports Aria engine, an evolution of MyISAM that outperforms InnoDB for both writing and retrieval.

But performance still drops respect to MySQL 5.5. Not as bad as Oracle's MySQL 6/7, but still a 15-20% query slowness respect to pre-fork mysql.

Does heavy usage of temporary files out of /var/lib/mysql, causing issues on partitioned filesystems.

And deprecates current mysql backup tools (mysqldump, mysqlhotcopy). The (mariadbbackup) replacement does not allow yet to do partition-based backups.

Documentation used to be very bad, but it has improved a lot in the last year.

Timescale opportunities

Issues with MySQL/MariaDB schema on HDB++ are how arrays are stored. Timescale may solve that with native array types from PostgreSQL.

We actually use key,index,time to store and retrieve data from thermocouples; that are read together and extracted separately.

This is not possible with current JSON-based archiving on MySQL (or at least, not without a huge performance drop).

But, it is still a matter of how you want to access the data (yet another argument for multiple DB's, depending on subsystem/usage).

AND! We need a common API for extraction, as we have for insertion. Actually Elettra's, ESRF and Max IV use different API's for extraction. It should be unified and transparent to users/developers.

fandango

Fandango / PyTangoArchiving / Panic and Python 3

All 3 packages are highly dependent, so it's a move-all or nothing deadlock.

Latest Fandango is still used in SuSE 11, tango 7, python 2.6 machines (in fact is the de-facto astor/jive/taurus/pytango replacement for those machines).

I plan to "froze" fandango at 14.* releases; and release Fandango 15 as python 3 / Tango 9 only, to be released in October.

Same approach will follow on PyTangoArchiving and Panic Alarm System.

In this case, community seems far more interested in Panic, but it will require an extra effort on testing (any help will be appreciated).

Questions?

https://www.tango-controls.org/community/forum/
https://www.pypi.org/project/pytangoarchiving
https://www.github.com/tango-controls/pytangoarchiving
https://www.github.com/tango-controls/fandango
https://www.github.com/tango-controls/panic
https://www.github.com/alba-synchrotron

Sergi Rubio Manrique, ALBA Synchrotron, Barcelona