Tango Polling Loop SIG - minutes

14 April 2026

Where we are

- Only 1 thread per device, but can have many devices per thread
- Could we make a telemetry dashboard that showed polling time stats, since Tango 9 algorithm doesn't allow splitting of individual reads with same period? Like Astor Polling Manager.
- System properties - concept in Jive, but not in general
- Could we disable default polling from code with non_auto_polled_cmd / attr? RB says it doesn't seem to work properly at the moment.
- DevAttrHistory can be useful for GUI that connects to device. Of course, this will use as much memory as the depth of the buffers require.
- Manually writing history - could be useful for hardware with precise timing. E.g. RGA, timing device.

Deficiencies

- MAX IV
  - Sardana controllers with DeviceProxy default to cache_device, so problem if polling is enabled, but controllers need to detect state changes immediately.
  - started archiving states, which required polling, but that broke controllers.
  - want to use single read_attr_hardware to combine state and status requests, issue 1582
  - if periodic event not interval of polling period. E.g., 3 sec polling, 10 sec periodic. causes errors in archiver. Can fix with configuration, but not obvious.
  - intermittent problems hard to catch. polling status not convenient to query fast. Can we keep some history, attribute in DServer?
    - Could do something like event system performance interface
  - Issue 1600. Slowing down attributes. crash if poll_attr has value lower than attr_min_poll_period
- SKAO

    - difficult to tune - different people commisioning and authoring (often in different time zones/continents)
    - getting it wrong, means items discarded
    - event on discard isn't much use to clients
    - in k8s can't do anything near to real-time. CPU resources very limited, which affects polling loop.
    - Would like to just pass the data to Tango and have it handle events nicely (want access to a buffer)
    - Could we separate caching, acquiring data, performing periodic updates?
    - mockable/controllable time for polling to aid testing (cppTango + PyTango)
- ALBA
    - no archive events, SubscribeChangeAsFallback
    - do periodic polling externally (PeriodicArchiver using Taurus polling)
    - device <10 attributes typically use internal polling
    - solutions
        - dedicated thread to read hardware, update own cache. It also pushes events.
        - still issues receiving events from outside normal Tango methods (not monitor lock), get locking issues, and sometimes segfault. E.g., Composer device (client + server). Reading, and pushing while client request comes in. PyTango 9.3.4.
        - device with > 1k events/sec too much for clients. So need some custom solution. Put events in a queue, and use a polled command to set them in batches. E.g., RF plant with 3k attributes trips, it sends events for all of them in a burst.

Scheduling theory

- optimal allocation of scarce resources over time.
- nice definition of the terms, and examples of optimisation criteria

Example implementation

- Current polling systems has 3 responsibilities: getting data, putting it in the cache, periodic updates
- current polling system: puts data into the cache
- poll model has algorithm
- polling loop gets next work item from queue
- caching/internal event system: signal bus
    - gets lock (with retry), pushes change event, archive event, fills polling buffer
- should push event put data into polling buffer?
- should Tango keep the latest value for each attribute automatically?
- currently:
    - clients can decide when attributes are read
    - polling: polling loop decides when attributes are read

    - event driven (new): device developer decides.
- Would like polling buffers to be "first class", and the only place where values are stored. Reads get data from there, and pushing events come from there. Clients don't need to choose cache/device when reading anymore.

Elettra proposal

https://gitlab.elettra.eu/lorenzo.pivetta/polling-sig/-/blob/main/proposal-class-level-async-poller.md?ref_type=heads

- Polling becomes background process directly accessing the hardware and deciding how often to read each attribute (or event driven, if possible)
- Do we keep errors as well? Yes, we should
- Why one poller per class and not per device? Not so critical.
- The existing polling is useful in many cases, so would like to keep this ease of use. However, recognise that sometimes we will need custom implementations.

Way forward

- cache separated from polling loop
- some way for device developer to say they will provide values for cache. If they are, then client can't access hardware directly. Either:
    - disallow polling, or
    - force clients to get data from cache, or
    - could raise exception
- still need to allow the old polling (it is easy, and needed for backwards compatibility)
- existing algorithm needs improvement
- two activities
    - how do we support async device authors?
    - what do we need to do to improve the current scheduler?
- New concept: "data collector". Could be existing Tango poller, or user provides their own.
    - Pushing events: handled by caching mechanism
    - Can we stay zero copy when pushing to ZMQ? Especially for large attributes (big image).
- Automatic pushing of events: opt-out.
    - could have different methods/flag on cache:
        - insert_value(attribute: str, value: Any, push: bool=True) ? Is
        - (for cppTango probably want to pass unique pointer)
    - complication: what if collector providing data faster than cache can push the event out?
    - good model: producer-consumer via FIFO (per attribute). Producer = data collector

        - is FIFO bound? I.e., do we block producer when full? Yes. So we apply back pressure.
    - default cache ring buffer size: 1
    - how many threads for consumer, how many FIFOs?
        - 1: easy
        - 1 per attribute: more configurable, better parallelisation. Multiple FIFOs/threads means clients could receive events in a different order to the order they were added to the consumer
    - For starters:
        - 1 consumer per device
        - 1 place to accept incoming data per consumer. Options - choose 1 of the following:
            - FIFO (device developer can configure the size, incl. "unbounded"), **OR**
            - 1 slot per attribute (i.e., latest data only), remove once processing starts (round robin).
        - 1 cache per attribute (configurable depth, probably ring buffer)
        - producer gets place to insert values, with something like:
            - **insert(attribute: *str*, value: *Any*)** - it blocks if FIFO is full, but it overwrites in the case of a slot
            - (attribute configuration decides if attributes are pushed by consumer)
    -
    - Need some way for producer to report some monitoring information.
        - simplest: logs
        - Producer stats (maintained by consumer?):
            - last producer insertion timestamp (liveness)
            - counters, rates
            - discards
            - extra metadata from the producer (JSON string?) via a method (producer calls a method on the consumer)
    - If you need rate limiting of the events, then can achieve this in custom producer implementation. E.g., bursty device with 4k updates in an instant once a week.

How to split Polling and Caching in cppTango?

- Rename classes, remove cross dependencies to ensure that can be used freely without polling at all:
- - PollRing -> Cache
- - PollElt -> CacheElt (each of the raw elements stored in the cache)
- - PollObj -> CachedObj (methods to insert/get per cmd/attr data/history)
- Then, rewrite PollThread and any method accessing PollRing/PollObj to use this new classes instead.

- Then, ensure that the Util object can insert data into this cache objects and retrieve it without needing to enable polling at all.
- Define proper meaning of things like "WorkItem", "Element", "Item" and "Object", because they are mixed in the code now.
-

15 April 2026

Design of collector and attribute caching

- API suggestion from TI
  - Should using this new collector, should we disable current polling configuration?
    - No. Collector gets notified when client tries to make changes - it can use them, or raise exception
    - What about existing properties? these should be passed to collector on startup? Yes, and device should fail to start if incorrectly configured.
- New design splits data collection and clients updates
  - data collection: either pull or push to get from the data source (hardware).
    - pull: typically poll the hardware. need at least 1 period, but could be more complicated
    - push: event-driven from the data source
  - client updates - how often does the client want to check for new data (in the cache)?
- Would we rename Polling loop to Scheduler?
- New suggestion for consumer:
  - no FIFO/container.
- White board
  - two designs, more favour the simpler one (righthand side)
  - how are state/status updated? Could have dev_state read from the cache.
  - client read dev source:
    - need way to indicate that direct hardware read is unavailable, and
    - need a way to indicate if data is always fresh (event driven systems)
- What happens with writing? Not just a data collector, sometimes we need to make changes, so write attribute needs to interrupt it to get hardware access.
- What happens with commands? Same things. Still have cache.
-

Scheduler

- The main issue we see with the polling loop is that it is not fair when it is mis-configured. Can we some how arrange for different work items to get dropped when it is mis-configured?
- Currently, the algorithm punishes items at the back of the queue.

- Are there different use cases?
  - Late-but-fair: Read everything, it is ok if we are late
  - On-time-but-unfair: Timing is important, drop things (fairly) to achieve the deadlines
- Ideas for failure situation:
  - N attempts, then go to fault state (in DServer)
  - Randomly reshuffle if we aren't meeting the deadlines
  - **Round-robin, ignoring lateness**
    - but still report it to the operator, more detailed info, something like event system performance system: e.g., SchedulerStatus() command on DServer
    - Use DServer State, because it owns PollingThread anyway
  - ~~Round-robin, starting at the next item after the one that was late. (Punishing the item that was late)~~ Discarded, people do not like this idea.
  - Credit-based system. If you take too long, you lose credit. At zero you lose a turn.
  - Dropping items "fairly", optimise to meet deadline
    - keep track of time per work item (from previous attempts)
    - keep track of number of times discarded (this could be priority)
    -
  - Move items to the front of the queue if they are dropped

Should we have multiple scheduler threads per device? No, complicated and not useful in most cases. Would need a NO_SYNC device.

If all attribute data comes from cache, then could use NO_SYNC serialisation. Should it be the default? Probably not. What about only attributes are NO_SYNC, but commands SYNC?

Collector ideas

Do we need a way to allow multiple cache updates in one go? E.g., got data for many attributes, set it all at once. Yes, would be nice. Provide a way for collector to take cache lock, or just add ***bulk_insert()*** method.

Should ***bulk_insert()*** also allow multiple values for the *same* attribute? No. Collector can just call multiple times.

16 April

Scheduler

- Look at some examples of non-pre-emptive round robin.
- Will we lose some accuracy in the happy case? E.g., extra jitter if one work item take variable time to read, does that affect the other work items. Could we offset them a bit, to get better performance.
- Aim: new method should work as well for existing "good" configurations.
- Will help to have **better diagnostic tools.**
  - We should focus on these tools first to help us work out a better algorithm

What problems are we trying to solve?

- What to do when late? No longer try tuning, just accept lateness
- What do we do with discarding? No longer discard.
- Not able to acquire serialisation lock when device is spending all its time polling, how to we send a command (e.g., Status, or Stop, or disable polling). This will be better with data collector being the only one doing hardware access and not acquiring the tango monitor lock. Not solved in the scheduler algorithm changes.
  - make an issue: why does disabling polling time out? [Dmitry]

How will the user select the new algorithm?

- Currently, you can set the "polling_before_9" device server property (boolean).
- Do we want a "group readings with same polling period" device property? I.e., use one read_attributes call, or use many read_attribute calls. Yes.
  - Name: "polling_group_reads".
  - Type: bool.
  - Device property
  - Only applies to algorithms after v10.
- Device server property to select the global algorithm:
  - name: scheduler_algorithm
  - type: str (pre-defined options)
    - round_robin
    - pre9?
    - 9? decide later...

- Default: ?

Polling period configured in code

- Device code defines a default.
- Start device, value written to database
  - This is confusing. Value from first time device is started "wins". Changing the period in the code later has no effect (since database value has priority).
  - Can we stop writing this value to DB, but still see the correct thing clients like Jive?
- User disables polling
- If restart via RestartDevice, it does not restart polling (so stays off)
- If restart process, the polling starts again.
- Do we want the change to this, so user change takes priority? Yes!
  - Could write to non_auto_polled_attr property. Does it work?
- Jive's "reset" button removes all the polling attributes from the device, including non_auto_polled_attr.
- To do:
  - document non_auto_polled_attr and non_auto_polled_cmd
  - stop writing the code-defined default polling value to DB at startup [TI: Can we do this for all properties?]
  - Write non_auto_polled_attr/non_auto_polled_cmd when polling is disabled
  - Verify that disabling polling, and also prevents polling next time device server is started. And that polling can be enabled again by the user.

The plan going forward: roadmap

- Changes for 10.4
  - Release date set to 2026-10-30 in cppTango meeting 2026-02-05.
  - Issues we tagged with 10.4 milestone
    - https://gitlab.com/tango-controls/cppTango/-/work_items/1654
    - https://gitlab.com/tango-controls/cppTango/-/work_items/1552
    - https://gitlab.com/tango-controls/cppTango/-/work_items/1600
    - non_auto_polled_attr thing...: https://gitlab.com/tango-controls/cppTango/-/work_items/1655
    - https://gitlab.com/tango-controls/cppTango/-/work_items/1550
    - https://gitlab.com/tango-controls/cppTango/-/work_items/1540
  - Scheduler stats command [champion: DE]
- Undecided version:

    - https://gitlab.com/tango-controls/cppTango/-/merge_requests/1627
- Changes for a future release after 10.4.x
    - New polling algorithm [champion: TB]
    - Data collector [champion: TJ, potential worker: SKAO]
        - API changes
        - cache changes
    - RFC changes [champion: TB/TJ - i.e., the changes related to your feature]:
        - List existing and new properties: https://tango-controls.readthedocs.io/projects/rfc/en/latest/8/Server.html
        - New DServer command for scheduler status
        - Mention data collectors. E.g., in request/reply RFC, https://tango-controls.readthedocs.io/projects/rfc/en/latest/10/RequestReply.html, which talks about the cache.
        - Do we want to tag the RFC? v1, v2? IDL version? date version? No tag? Decision: don't tag, but you can reference the RFC via:
            - git log --pretty=reference -n1
            - 7f3b07b (Merge branch 'add-free-object-naming-convention' into 'main', 2026-03-12)
        - We want a draft RFC MR before we start implementing (this helps the implementer and reviewer).

Additional topics

Starter

- Option for parallel startup: https://gitlab.com/tango-controls/starter/-/merge_requests/47
-

Elettra AlarmHandler

- Can/should it be moved to gitlab.com/tango-controls? LP to check with Graziano.

Polling loop

- Event system heartbeats are sent via polling loop code:
    - https://gitlab.com/tango-controls/cppTango/-/blob/807febab00f3246a59308730b459d2e0e617c5dc/src/server/PollThread.cpp#L670
    - However, this is a separate instance, just for heartbeats.
        - https://gitlab.com/tango-controls/cppTango/-/blob/main/src/server/Util.cpp#L642
    - Code could be seperated

State/status calls use read_attr_hardware?

- Would be useful to read data from hardware once to determine state and status
- PyTango: DeviceProxy.state() is different to DeviceProxy.State() ! Lowercase version gets wrapped with green mode. This is bad. They must be the same. Which one is right?
  - state is actually a special attribute in the IDL: https://gitlab.com/tango-controls/tango-idl/-/blob/main/include/tango.idl?ref_type=heads#L907
  - There are 3 different ways of getting state/status from cppTango:
    - CORBA state: https://gitlab.com/tango-controls/cppTango/-/blob/9a4a7ab194275478ca34ec346e5ac9433dc28d48/src/server/DeviceImpl.cpp#L1994
    - Command: https://gitlab.com/tango-controls/cppTango/-/blob/9a4a7ab194275478ca34ec346e5ac9433dc28d48/src/server/basiccommand.cpp#L107
    - Attribute: https://gitlab.com/tango-controls/cppTango/-/blob/main/src/server/Device_3Impl.cpp#L737
  - How to fix:
    - cppTango DeviceImpl could ensure the read_attribute version is always called, not matter what the client asked for (state(), command_inout, or read_attribute).
    - Should read_attribute version trigger read_attr_hardware for state/status?
      - could make it an option. Is that complexity worth it? No.
      - Decision: Yes, call it.
    - PyTango, still need consistency with wrapping/green mode.
    - When to fix in cppTango? version 11.

TangoTickets

https://gitlab.com/tango-controls/TangoTickets/-/work_items?sort=created_date&state=opened&search=poll&first_page_size=100

- https://gitlab.com/tango-controls/TangoTickets/-/work_items/74 we could remove the command cache, if state and status requests always come from the attribute cache (which would be the case after we make the DeviceImpl changes above).
- https://gitlab.com/tango-controls/TangoTickets/-/work_items/119
  - Still useful for client to know this, even with new collector.
  - If **using new collector (opt-in)**, device developer shouldn't be allowed to manually call push_XXX_event.

All done!

Ended with a tour of MAX IV.