SKA-Low HDB++ tweaks

40th Tango Community meeting

Alex Hill, SKAO

Startup speed improvements

We found the number of pending attributes building up significantly - and then event subscribers taking a long time to reach steady-state.

- ON CONFLICT DO NOTHING
  - prevents duplicates from causing a reversion to one-by-one insertion
- Per-attribute actions at startup (and shutdown)
  - Populating the att_conf_id cache - now done in one query
  - att_history and att_parameter events are still one-by-one - best fixed in libhdbpp which implies a fix in every backend
- Co-locating subscribers with the database, rather than the Tango devices

Load-bearing table_name

What is this used for? We couldn’t see anything in the libhdbpp-timescale codebase…

- So we patched the event subscriber to respect it
- We use this for archiving doubles as floats
  - DevDouble is “default” in Python, if you specify type as Python `float`
  - But none of the source data requires this
  - Would use 16-bit floats if they were available!
- Could use this for separate retention policies at timescale level
  - dropping chunks is very cheap
  - deleting individual rows is not
- Different tablespaces, for different storage backends

pguint

- gives true unsigned int types in PostgreSQL
- motivated by one frequent and large attribute - antenna bandpasses
- In StackGres, you can only use blessed extensions
  - but there are many already available
  - and this one was added this one quite quickly (weeks)

yaml2archiving

- A handful of patches, to improve speed and repeatability
- Support for multiple YAML documents in one file
- The repo on tango-controls is a mirror from an internal MaxIV repo - not all that easy to contribute changes upstream
- Configuration is now generated (on demand) by inspection of the live system, with a set of rules and exclusions/overrides
- Applied with “configmap2archiving” as part of the deployment
- The ConfigurationManager becomes a bottleneck and doesn’t add value. We know exactly the layout of event subscribers and attributes that we want ahead of time.

HDB++ schema changes

- Updated to use new TimescaleDB syntax for hypertables and aggregates
- Redundant indexes removed
- Convenience views JOIN att_conf with each data table

```
SELECT att_conf_id, data_time, value_r, ...
FROM att_array_devdouble
JOIN att_conf USING (att_conf_id);
```

- Removes a JOIN or subquery from nearly every query
- Nearly always zero-cost - VIEW is not an optimisation boundary for PostgreSQL’s planner

HDB++ roadmap features

Batched inserts with VALUES or UNNEST

Now: sequential single-row INSERTs within a transaction:

```
BEGIN;
INSERT INTO att_scalar_devdouble (data_time, att_conf_id, ...) VALUES
    (2026-06-11T09:00:00Z, 2451, ...);
INSERT INTO att_array_devstring (data_time, att_conf_id, ...) VALUES
    (2026-06-11T09:00:01Z, 1263, ...);
INSERT INTO att_scalar_devdouble (data_time, att_conf_id, ...) VALUES
    (2026-06-11T09:00:02Z, 1246, ...);
INSERT INTO att_array_devstring (data_time, att_conf_id, ...) VALUES
    (2026-06-11T09:00:03Z, 1924, ...);
...
COMMIT;
```

Batched inserts with VALUES or UNNEST

One INSERT … VALUES statement per data type:

```
INSERT INTO att_scalar_devdouble (data_time, att_conf_id, ...) VALUES
    (2026-06-11T09:00:00Z, 2451, ...),
    (2026-06-11T09:00:02Z, 1246, ...);
INSERT INTO att_array_devstring (data_time, att_conf_id, ...) VALUES
    (2026-06-11T09:00:01Z, 1263, ...),
    (2026-06-11T09:00:03Z, 1924, ...);
```

Improvement: ~3x at batch size 100, ~5x for batches of 1000.

Batched inserts with VALUES or UNNEST

One SELECT * FROM unnest … statement per data type:

```
INSERT INTO att_scalar_devdouble SELECT * FROM unnest(
    ARRAY[2026-06-11T09:00:00Z, 2026-06-11T09:00:02Z],
    ARRAY[2451, 1246],
    ARRAY[...],
    ...
);
INSERT INTO att_array_devstring SELECT * FROM unnest(
    ARRAY[2026-06-11T09:00:01Z, 2026-06-11T09:00:03Z],
    ARRAY[1263, 1924],
    ARRAY[...],
    ...
);
```

Improvement: ~1.2x over batched VALUES.