Testing HDB++

A brainstorming on benchmarking, evaluating, accepting

Purpose of testing

Why do we need to test the system?

- Validate the code, its consistency, and that it does what should be done.
- Measure performance. Performance of the existing code and each supposed "improvement" on the system, to avoid regression.
- To provide a guide on how to tune the system for optimal performance.
- To give users practical information on "what to expect".

Things to test in an archiving system

- Max. number of values to inserted per attribute/second/archiver.
  - Statistics already provided by the current tools
- Compatibility with tango types (64bit, unsigned, encoded, arrays)
- Optimal attribute/archiver ratio depending on the above.
- Feasibility of backup/restore during operation.
- Usability and client satisfaction, query time and latency.
- Analysis of the errors recorded, and study how to reduce them.

Testing compatibility

- Targets of benchmarking are not only to provide an specs number of the max. performance that can be achieved.
- Benchmarking should provide a complete guide on how to tune the system for each of the different usages.
- E.g. optimal attribute/archiver ratio may depend not only on the number of attributes/events but on their type (arrays, strings).

Testing infrastructure

An initial question to be solved is: Testing with simulators or testing the archiving system on real systems?

By experience, it seems that only testing on real systems provides us enough information regarding final user experience.

Testing "offline" systems based on simulators may "mask" some flaws that affect the final performing.

HDB++ allows to run several databases in paral.lel. So we can store same attributes using two different strategies and evaluate performance separately.

But, testing/benchmarking all types can be achieved only through simulators. So both types of testing are needed.

Testing infrastructure

One of the best advantages of HDB++ is the huge amount of metrics that we are adding since the beginning. So we can evaluate many different options:

- Distributing attributes per subscriber randomly or per event load:
    - It will produce an heterogeneous distribution of attributes/types per subscriber.
    - It should be the most load-balanced distribution.
- Distributing event subscribers per attribute type:
    - it will lead to all attributes of the same archiver to be written to the same table.
    - It will give us the optimal rate of insertion per type, or at least is a way to easily measure it.
    - But, Is it optimal or the opposite? Is it dangerous to not distribute the load on a table on different processes?
- Distributing event subscribers per device:
    - It will gave us metrics to study the performance of our control system
    - Probably, it is the least useful from a benchmarking point of view, but the most useful from the control engineer point of view.

Testing on the client side

The client side should be tested in two different ways:

- On script-mode:
  - Testing the api/extractor tool to verify the time needed per query, depending on type, dates, and number of values to be expected.
  - Depending on this results, we may need an API that "tunes" the query according to the recorded statistics in order to optimize query time.
  - Measuring the latency of the data (the gap between current data availability and the last stored value). User must be aware of this gap to query the data according to it.

Testing on the client side

The client side should be tested in two different ways:

- On GUI:
  - The graphical back-end will limit the amount of data that can be plot.
  - Javascript/Web, Java, Qt (qwt, pyqtgraph); each backend has its own limitations, both in terms of memory and in the amount of cpu used by the graphical plot when displaying big numbers of items (e.g., Qwt practically limited to 65K points on display despite of the hardware used).
  - The user workflow and the resultant queries. If we decimate the data to be shown (to reduce the number of points) ... what to do when zooming? New queries to database or decimation on client?
  - At ALBA we are requested to mix live data and stored data. Does this requirement applies to the other institutes?
  - How the workflow affects our query times? Do we have access to statistics so we can tune the querys? (e.g. executing decimation on the server side prior to sending the data to the client).

Testing on the client side

Brainstorming :

- How to evaluate queries performance?
- How to get rid of caches effect?
- How to compare attributes with completely different event density?
    - Taking into account that many attributes go to the same tables, so an event-intensive attributes affects the queries to all the rest of attributes in the same database.

User lives matter? Should we involve them globally on the evaluation?

Backup / Restore

Feasibility of online backup / restore is not typical taken into account when testing.

Or, typically it involves using replication to not affect insertion of data into the database.

But, how backup on the replicated server affects the queries from users?

Do we care?

Testing on the client side

Shall we recover the Extractor tool/device concept?

- Would allow to optimize the queries from any graphical back-end, or even any database back-end if using library as plugin. And we will have a common performance measure for everyone.
- Should implement caches or will be just another memleak to take care of?
- Would be capable of managing returning arrays with time-value for any type? (e.g., using pipes or encoded as returning type)
- Are all client api's (e.g. Javascript) capable of integrating asynchronous commands and events?

Brainstorming

What do we do now?