HDB@ELK: another noSql customization for the HDB++ archiving system

Di Carlo, M., Canzari, M., Dolci, M., Smareglia, R.

M. Di Carlo, M. Canzari, M. Dolci, R. Smareglia, "HDB@ELK: another noSql customization for the HDB++ archiving system," Proc. SPIE 10707, Software and Cyberinfrastructure for Astronomy V, 107072D (6 July 2018); doi: 10.1117/12.2312464

SPIE.

Event: SPIE Astronomical Telescopes + Instrumentation, 2018, Austin, Texas, United States

HDB@ELK: another noSql customization for the HDB++ archiving system

M. Di Carlo*[a], M. Canzari[a], M. Dolci[a], R. Smareglia[b]
[a]INAF Osservatorio Astronomico d'Abruzzo, Teramo, Italy; [b]INAF Osservatorio Astronomico di Trieste, Trieste, Italy

ABSTRACT

The TANGO controls framework community has put a lot of effort in creating the HDB++ software system that is an high performance, event-driven archiving system. Its design allows storing data into traditional database management systems such as MySQL as well as NoSQL database such as Apache Cassandra. The architecture allow also to easily extend it to other noSql database like, for instance, ELK. This paper describes the step needed to extend the HDB++ and explore the possibilities to use the ELK technology in term of analysis being enabled and tools provided.

Keywords: TANGO, HDB++, Elasticsearch, ELK, noSql DB

1. INTRODUCTION TO TANGO AND HDB++

The tango controls system is built on top of the CORBA (Common Object Request Broker Architecture, [3]) standard, limiting the possibility to introduce new objects with the IDL (interface definition language) to the "device" concept, in order to build distributed control system. Therefore, a device is mainly an object with attributes within a process called "device server" [4].

One of its core feature is the event model, which allows the communication between devices according to a predefined event. One of the event typology available is the archive event, that can be configured to be fired with an absolute or relative change in value (for a particular attribute of a device) or it can be configured to archive in a polling fashion. [5]

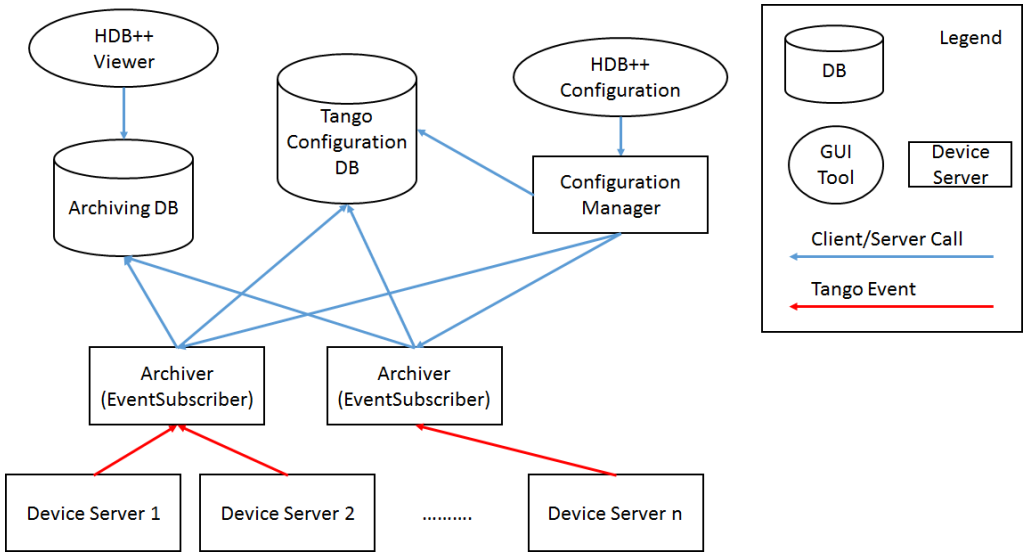


Figure 1. HDB++ Runtime View.

* matteo.dicarlo@inaf.it; phone +39 0861 439711; fax +39 0861 439740; www.oa-abruzzo.inaf.it

Software and Cyberinfrastructure for Astronomy V, edited by Juan C. Guzman, Jorge Ibsen, Proc. of SPIE
Vol. 10707, 107072D · © 2018 SPIE · CCC code: 0277-786X/18/$18 · doi: 10.1117/12.2312464

Figure 1 (available on the online documentation, [5]) shows the main runtime components of the HDB+ system that are the Configuration Manager and the Event Subscriber (aka Archiver). The first one is a device that assists in adding, modifying, moving, deleting an attribute to/from the archiving system while the second one is the receiver of the event, the one that has to store the value of the configured attribute.

1.1 Customization: the "Abstract DB" abstraction

Figure 2 shows the architecture of the HDB++ in term of modules (mainly c++ library). The two main module are the "Event Subscriber" and the "Configuration Manager". Both of them use a third module, called "Database Abstraction Layer", that define two interfaces (namely DBFactory and AbstractDB) for extending the archiving system to other database system. Extending the HDB++ system, therefore, is very easy: it is only needed to implement the above two interfaces.

At the moment, there are two implementations of the abstraction layer that allows the interaction (for storage purpose) to the Mysql [6] database system and the Cassandra [7] database system.

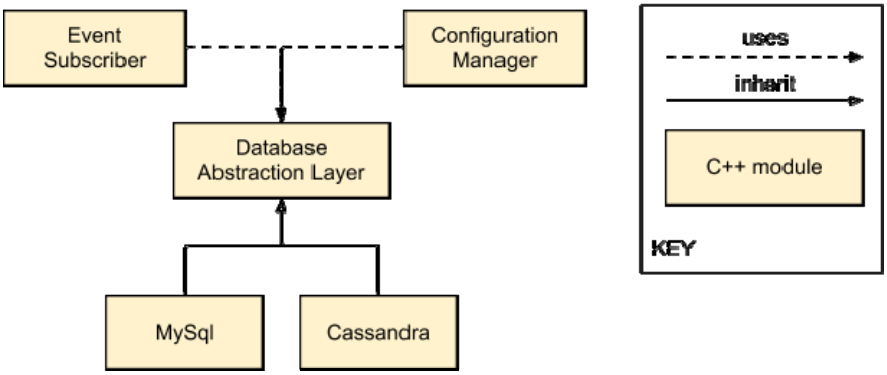


Figure 2. HDB++ Module View.

1.2 Data Model

The tradeoff of the simplicity of extending the system is a fixed data model, which is the schema of the database. This means that the archiving interfaces (both the "Event Subscriber" and the "Configuration Manager") cannot be changed and what is archived must always have the same shape. In essence, it consists of five tables, shown in Figure 3, which stores the following information:

- *AttributeConfiguration*: for each attribute (to store in DB when needed) there is an entry in this table which associate it with a specific data type;
- *AttributeConfigurationHistory*: for each operation on attributes (add/remove/start/stop), a row is inserted with the timestamp of the operation;
- *AttributeParameter*: for each attribute (to store in DB when needed) there is an entry in this table which associate it with the list of parameter already stored in the TANGO database;
- *AttributeEventData*: for each event there is a row which stores the information associated with the event (mainly timing and quality factor);
- *Value*: for each event stored in the AttributeEventData, there is an entry in a Value table (double, long, string and so on).

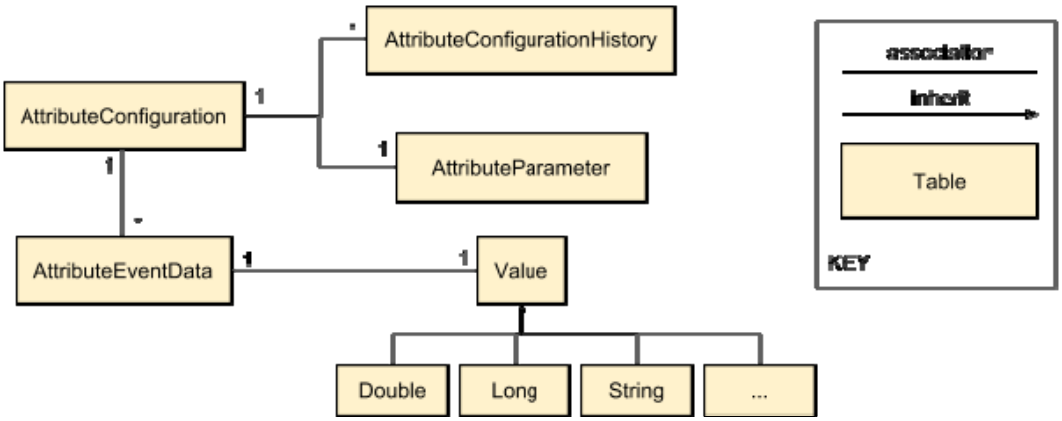


Figure 3. HDB++ Data Model.

2. INTRODUCTION TO ELASTICSEARCH

Elasticsearch [1] 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 allows the discovery, interpretation, and communication of meaningful patterns in data.

It is based on Apache Lucene [8], 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").

2.1 Elasticsearch as noSQL database

Looking at the definition given in [1], it is not straightforward to assert that Elasticsearch is a noSql Database; the problem is the definition of noSql which it is not very precise: "Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontally scalable" [9].

Since that definition is not exact, the only possibility is to summarize the main features of Elasticsearch in order to understand its qualities. They are:

- *no transaction*: no support for transaction;
- *schema flexible*: there is no need to specify the schema upfront;
- *relations*: denormalization, 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 [10]
- *no security*: there is no support for authentication and authorization.

When using a software like Apache Lucene or Elasticsearch, all the features must be well considered and, also, an important consideration deserves the relations.

ELK see data as everything is flat: basically every document, stored in ELK, is independent and therefore every document should contain all of the information required to decide whether it matches a query. In particular, this helps in indexing, in searching and in scalability since documents can be spread across multiple nodes. But relations are important and there are mainly four ways to bridge the gap.

The first possibility is the application-side join, which means that there are no relations in the data and the only possibility is to make more than one query to filter and emulate a join. An example is the relation between user and posts in a blog: to make a query that join the two entities, one can make two queries.

The second technique is the data denormalization [11], which is the process of increasing read performance adding some redundant copy of the data. In the previous example, one could store, in the posts document, the information related to the user. As opposite, the write performance would decrease. Of course there are some disadvantages in denormalization, like index dimension (bigger documents, bigger index) and, overall, the concurrency (which can be solved with various lock mechanisms).

The third way of handling relations in ELK is the use of nested objects; this means that it is possible to relate a document with a nested document that is indexed together (so there is a variety of special operator in order to query these types of document).

The last possibility is the parent-child relationship which is very similar to the nested objects mechanism. Through this way it is possible to create a one-to-many relationship where a document can be a parent of another one and one child can have only one parent but the documents are completely separated. This means that one can change the parent without updating all the children or that one can add/modify/remove a child of a document.

3. IMPLEMENTATION

The selected development language was C++ mainly because HDB++ is made in that language. The new library had to be able to work with REST [12] and with Json data [13] and, for this reason, two libraries have been selected to include these functionalities: "REST client for C++" [14] and "Json for modern C++" [15].

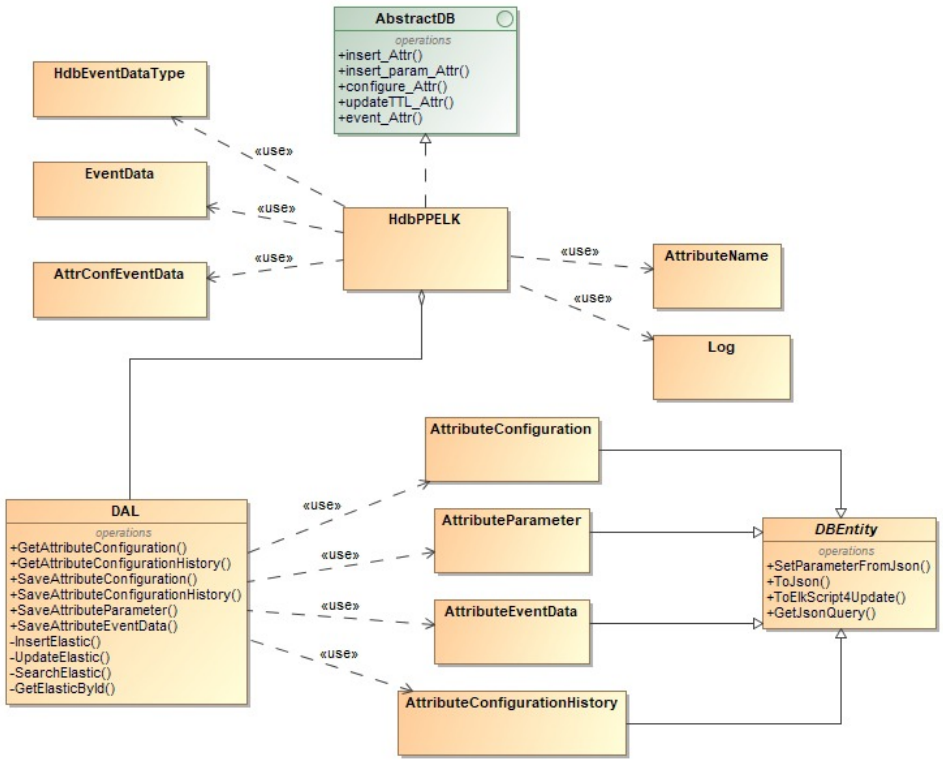


Figure 4. Class diagram.

The total amount of time needed to implement (source code development) the *AbstractDB* was around 4 weeks while the total amount of work done for testing and studying was around two months for a total amount of three months of work. Figure 4 shows the class diagram of the implementation done. The central class is the *HdbPPELK* which realizes the main methods of the interface *AbstractDB* (that uses the classes *HdbEventDataType*, *EventData* and *AttrConfEventData* belonging to the TANGO core library). The classes *AttributeName* and *Log* were already developed and are used to extract the information needed from the full attribute name and to log data messages. The class *DAL* (refer to Data Access Layer [18]) provides to the *HdbPPELK* class a simple access to the ELK functionalities. In fact, it implements

four private methods which represents the basic operation (except for the delete operation) that can be done with a database (Insert, Update, Get single entity and search for entities). Usually those methods are in another class so that more than one database can be added to the project but since the need is only for one repository, it has been decided to join the two classes. With the help of those simple methods and for each information to store, there are one or more method to save and read the data. For example, if a new archive event arrives to the system (represented with the class *AttributeEvenData*), the *HdbPPELK* will create the object (in this case an instance of *AttributeEvenData*) and simply pass it to the DAL which will store it. It is important to note that if an entity has to be stored it has to inherit from the abstract class *DBEntity* and implements four methods that will help the *DAL* to send data to the ELK repository. The other entities to consider are mainly those explained in Figure 3.

The analysis of the result obtained has been studied with two test cases: a set of monitoring data taken from the camera AMICA [16] installed in Antartide for a period of time of 116 days and the set of monitoring data taken from the Global Centroid-Moment-Tensor (CMT) Project [17] (recording earthquakes since 1976). Two TANGO devices have been also developed in order to simulate the production of monitoring data both of the camera and the CMT and stored them through the normal mechanism of the HDB++ architecture. It is worth to notice that, for each device, together with the normal attributes, there was a special attribute that allowed to archive the entire attributes in one time. This is to avoid the problem of the data aggregations explained in section "Aggregating data".

Figure 5 shows the runtime view (Figure 1) adapted for this specific test case.

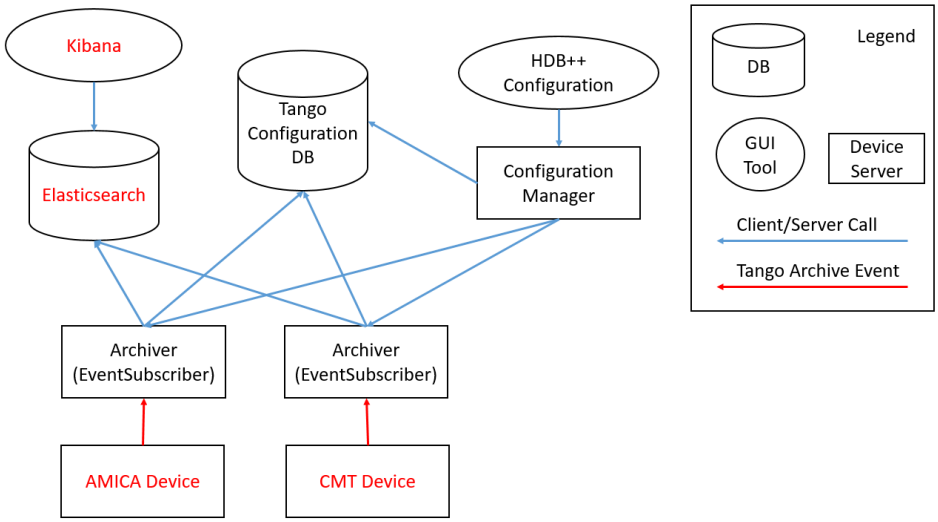


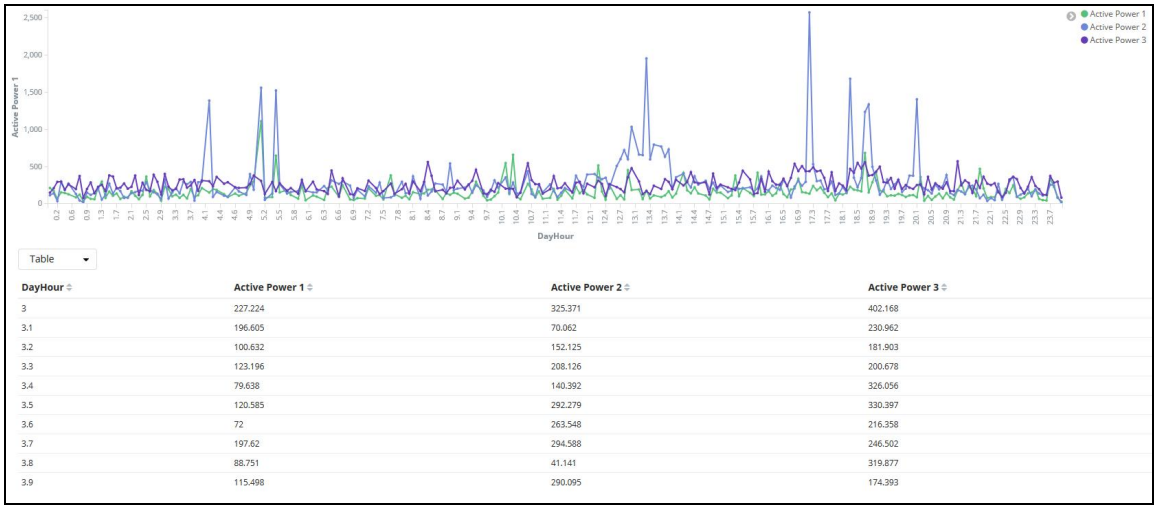
Figure 5. HDB++ Runtime View - Test case.

3.1 Archiving result

It has been simulated more than forty thousand events for the CMT project and more the two hundred thousand events for the AMICA project. The goal was to find out the possibilities in using the ELK tools such as Kibana.

There are also many possibilities in term of standard plots like histograms, line graphs, pie charts, sunbursts and so on. For example, Figure 6 shows the trend of the three lines of a three-phase expressed in Volt.

Figure 7 shows the plot of the geolocation for the CMT project grouping them by position (the bigger the point, the more events there are).



Table

DayHour	Active Power 1	Active Power 2	Active Power 3
3	227.224	325.371	402.168
3.1	196.605	70.062	230.962
3.2	100.632	152.125	181.903
3.3	123.196	208.126	200.678
3.4	79.638	140.392	326.056
3.5	120.585	292.279	330.397
3.6	72	263.548	216.358
3.7	197.62	294.588	246.502
3.8	88.751	41.141	319.877
3.9	115.498	290.095	174.393

Figure 6. AMICA project Active Power trend.

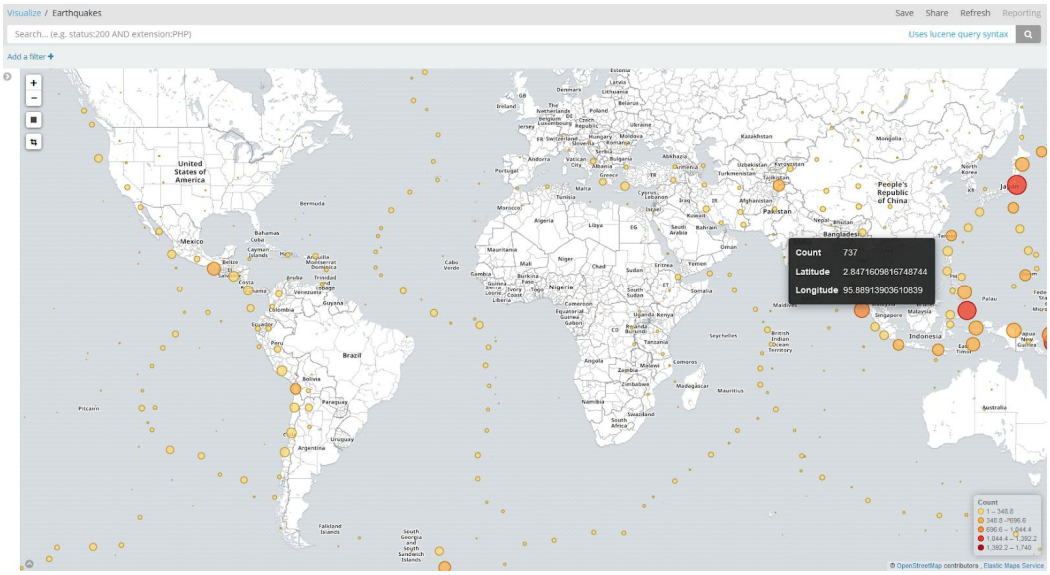


Figure 7. CMT project Aggregating GeoLocation.

3.2 Aggregating data

Those two figures are just two simple examples of what can be done with a tool like Kibana but were enough to show some controversial aspect of aggregating data. Let's consider the simplified view of the data model shown in Figure 8; it shows how the system stores a specified number of attributes M in a specific number of events N (the table shown is the AttributeEventData of the data model that can be simplified seeing it together with the value tables).

Unless the purpose of the archiving are time series, it is not easy to aggregate the data available with the above model. In fact, the only possibility (without creating a specific software for doing it) is to perform the join with the the event information (hopefully same time means same event).

Figure 9 shows an example for the CMT project. The event information is present for every attribute and for each attribute, there is one value with all the other column with null value. While for time series, this model works very well, this is not true for structured data, like the geographics location (Figure 7). This means that, with a tool like ELK, it is not possible (at query time) to transform the initial table into a usable table.

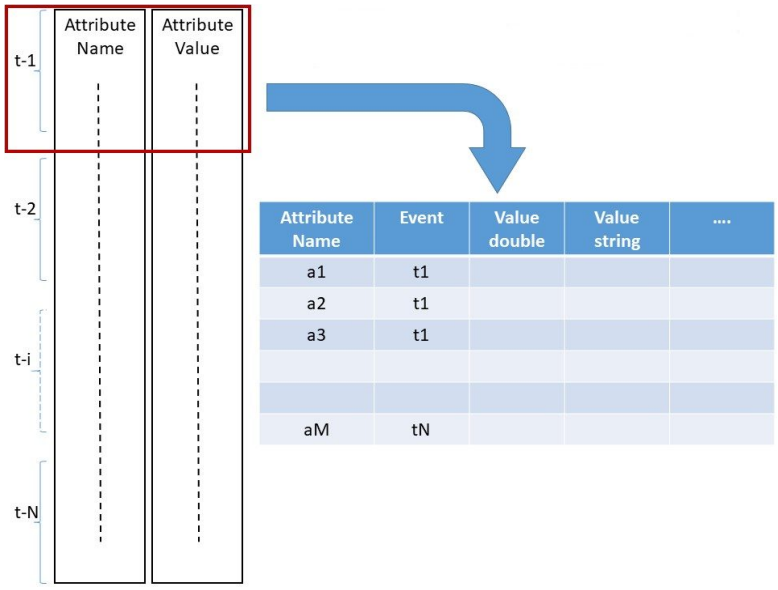


Figure 8. Archiving M attributes with N events.

The general transformation is shown in Figure 10. It is worth to notice that it depends on the grouping needed: in this case, it is a cube: event, device and attribute. However, it can be any kind of grouping.

In order to avoid the transformation problem, a json attribute (with the needed changes in the source code to be able to archive another type) was introduced whenever the aggregation was needed. In this way, it has been used the nested type relationship described above.

Event	Attribute Name	Value double	Value string	Value Date
t1	Latitude	-28.39	-	-
	Longitude	-176.79	-	-
	Location	-	"KERMADEC ISLANDS REGION"	-
	Date	-	-	1976/02/15 21:23:22.6
t2	Latitude	-14.74	-	-
	Longitude	167.10	-	-
	Location	-	"VANUATU ISLANDS"	-
	Date	-	-	1976/03/04 02:50:00.5
...	...	...	...	...
tN	...	...	...	...

Event	Latitude	Longitude	Location	Date
t1	-28.39	-176.79	"KERMADEC ISLANDS REGION"	1976/02/15 21:23:22.6
t2	-14.74	167.10	"VANUATU ISLANDS"	1976/03/04 02:50:00.5
...	...	...	...	...
tN	...	...	...	...

Figure 9. Joining attributes together.

Event	Attribute Name	Value double	Value string	....
t1	a1	...	...	...
	...	...	...	...
	aM	...	...	...
t2	...	...	...	...
	...	...	...	...
	...	...	...	...
tN	...	...	...	...

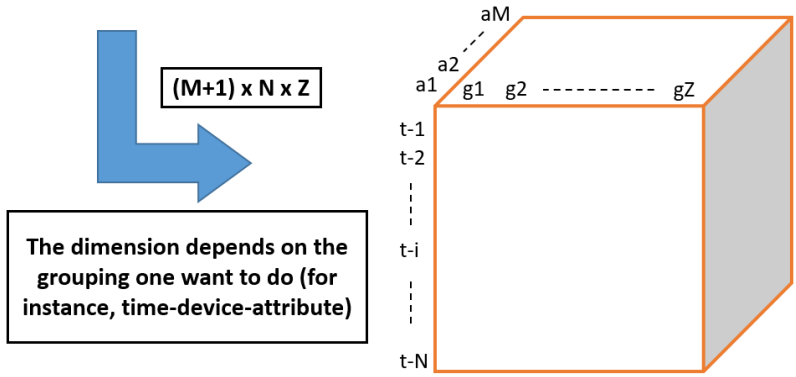


Figure 10. General transformation.

4. CONCLUSION

The present paper represents a study of the HBD++ architecture in order to understand the time needed for the customization of the archiving system with a noSql technology like ELK. The result is that the architecture allows the expansion with a new database system in a very easy way.

The system appears to be thought for archiving time series and not structured data: the difficulties of aggregating data is the tradeoff of this architectural choice. Structured data can be archived with json string, changing the system where needed (the implementation of the abstraction layer), or with array.

New development of the TANGO core model can be helpful to reduce the aggregation tradeoff, for example with a specific json data type, but, together with this, the possibility of scheduling custom archiving scripts can be beneficial too.

5. ACKNOWLEDGEMENT

This work has been made possible thanks to the financial support by the Italian Government (MEF - Ministero dell'Economia e delle Finanze, MIUR - Ministero dell'Istruzione, dell'Università e della Ricerca).

REFERENCES

[1] ELK, www.elastic.co
[2] Di Carlo, Matteo et al., TM Services: an architecture for monitoring and controlling the Square Kilometre array (SKA) telescope manager, Proc. of the SPIE Astronomical Telescopes and Instrumentation 2018, paper no. 10707-59 (this conference)
[3] Corba, www.corba.org
[4] TANGO controls, tango-controls.org
[5] HDB++ introduction, http://tango-controls.readthedocs.io/en/latest/tools-and-extensions/archiving/HDB++.html
[6] Mysql, www.mysql.com

[7] Cassandra, cassandra.apache.org
[8] Lucene, lucene.apache.org
[9] noSql, nosql-database.org
[10] CAP theorem, en.wikipedia.org/wiki/CAP_theorem
[11] Denormalization, en.wikipedia.org/wiki/Denormalization
[12] REST, en.wikipedia.org/wiki/Representational_state_transfer
[13] JSON, www.json.org
[14] REST client for C++, https://github.com/mrtazz/restclient-cpp
[15] Json for modern C++, https://github.com/nlohmann/json
[16] Dolci, M.; Valentini, A.; Ragni, M.; Di Cianno, A.; Di Rico, G.; Straniero, O.; Romano, D.; Christille, J.-M.; Piluso, A. 2012: AMICA at Dome C: results from the first year of automatic operation tests in Antarctica, Proc. of the SPIE, 8446, 844645
[17] Global CMT, http://www.globalcmt.org/
[18] DAL, https://en.wikipedia.org/wiki/Data_access_layer