EXPLORING NO-SQL ALTERNATIVES FOR ALMA MONITORING SYSTEM

T. Shen, R. Soto, P. Merino, L. Peña, A. Barrientos, M. Bartsch, A. Aguirre, Jorge Ibsen. ALMA, Alonso de Cordova 3107, Vitacura, Santiago, Chile

Abstract

The Atacama Large Millimeter /submillimeter Array (ALMA) will be a unique research instrument composed of at least 66 reconfigurable high-precision antennas, located at the Chajnantor plain in the Chilean Andes at an elevation of 5000 m. This paper describes the experience gained after several years working with the monitoring system, which has a strong requirement of collecting and storing up to 150K variables with a maximum sampling rate of 20.8 kHz. The original design was built on top of a cluster of relational database server and network attached storage with fibre channel interface. As the number of monitoring points increases with the number of antennas included in the array, the current monitoring system demonstrated to be able to handle the increased data rate in the collection and storage area (only one month of data), but the data query interface showed serious performance degradation. A solution based on no-SQL platform was explored as an alternative to the current long-term storage system. mongoDB has been selected. Intermediate cache servers based on Redis were introduced to allow faster streaming of the most recently acquired data to web based charts applications for online data analysis.

INTRODUCTION

The ALMA Monitoring System plays a fundamental role on recording status of all hardware devices of the observatory. This information is crucial to allow engineers to take the correct decision and to schedule efficiently a) preventive maintenance activities and in case of hardware malfunction, the system also generates alarms in order to trigger b) corrective actions by array operators or engineers.

Besides of scientific data, monitoring system makes the most intensive use of the database in term of number of transactions and data storage. Currently, around 25GB of monitoring data is collected per day from a total of 140.000 monitor points. In average, 5000 Clobs [1] is collected per second. 80% of monitor points belong to hardware located in antennas. There are 4 types of antennas in ALMA, and in average there are 2,363 variables per antenna. The rest of 20% comes from equipment in the central building.

The definition of the monitor points of each piece of hardware is based on the container/component architecture of ALMA Common Software (ACS) [2]. I.e, there is one container per antenna, and each piece of hardware within an antenna is modelled as a component [3] (there are in average 45 device components per antenna), which can have arbitrary number of BACI properties [4], and finally, from each BACI property there can be one or several monitor points. By definition, a monitor point is a scalar with a timestamp associated to it. The sampling information is defined at the BACI property level and they are part of the Telescope Monitoring & Control Database (TMCDB).

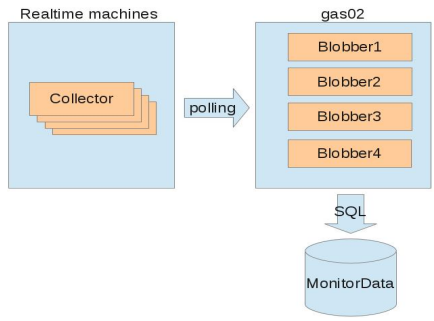

Figure 1: Monitoring system design using the relational database.

As shown in the Fig. 1, during the system start up, BACI properties are registered with a local component within the antenna container called “Monitor Collector”. This component caches temporary the sampled data, which is cleared when the data is polled by an external component called “Blobbers” in regular intervals. Each Blobber component can deal with up to 8 Collectors and in the original design they have to disaggregate the collected BACI property data into monitor points by reading the definition from the TMCDB and at the same time insert the monitor points back to the same database. In order to reduce the number of insertion in the database, the data of the same monitor point are grouped in one Character Large Object (*CLOB*).

During the early usage of monitoring infrastructure (2009/2010), two fundamental problems were detected: a) Some times, Blobbers could not keep up with the incoming data rate, b) data querying to the database was very slow. During the analysis of these problems, we learnt several lessons: a) it’s better to simplify the online section of the monitoring system and delegate as much as possible the data processing in the offline phase, b) the current database schema is highly optimized/normalized for data insertion, c) more buffering mechanisms have to be introduced in critical points of the whole pipeline in order to deal with sudden peak of data rate, and in the worse case drop the data to avoid crashing the associated components and finally d) better instrumentation has to be introduced in order to fine tune parameters of components in the whole chain.

The data insertion problem was caused by a mismatch in the versions of ODBC libraries, which caused the observed degradation in the data insertion performance.

Copyright © 2014 CC-BY-3.0 and by the respective authors

ISBN 978-3-95450-139-7

But the troubleshooting experience open new possibilities to improve the monitoring system, especially in the offline section of the pipeline: after Blobbers receives the sampling data.

In order to improve the performance of the data query, we decided to experiment with different approaches. The fundamental problem was located in the database schema, which is high normalized and data are extremely atomized; therefore in order to construct a day-worth of data for a single monitor point, non-trivial queries must be submit to the engine. One solution is to use no structured way to save the monitor data.

REFACTORING OF THE MONITORING SYSTEM

The following goals were defined in order to improve the performance and usability of the monitoring system: a) provide a mechanism to access most recent acquired data, b) provide preformatted text files for each monitor point with a day-worth data, c) allow efficient historical data access and finally d) minimize access to the online control system, specially if the purpose is just to monitor the system and not to control the system. And it was mandatory to keep running the current implementation until the new approach is ready.

In the following section, we will present the solution to achieve these goals.

Implementation

The first thing we introduced is a queuing mechanism in order to multiplex the data flow, and allow processing of the monitor data in parallel by the current and the new implementations. The Apache ActiveMQ [5] was introduced into the dataflow just after the Blobber components, as shown in Fig. 2, and publisher/subscriber mode was chosen in order to feed both implementations.

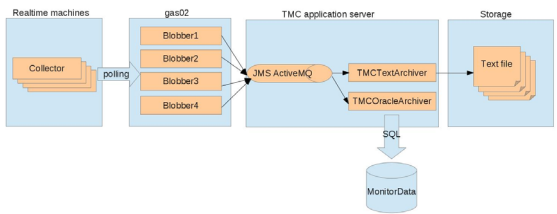

Figure 2: Apache ActiveMQ added into the dataflow.

ActiveMQ has the advantage, among others, to have a very good instrumentation based on MBean [6], which can be access through JMX [6] protocol. These are valuable information in order to understand the characteristic of the data flow and to fine-tune each involved component.

As mentioned before, in order to optimize the performance in the Blobbers, we simplified the implementation and excluded the logic to resolve BACI properties into monitor points and moved it after the ActiveMQ queue, the offline section of the data flow. A buffering mechanism is added as well in order to protect Blobber components against rush of data; basically, if the buffer is full, then new data will be dropped to protect the integrity of related components.

The original data insertion was implemented within the TMCOracleArchiver, while in parallel the TMCTextArchiver implementation is in charge to generate the preformatted text files of each monitor point with a day-worth of data. A web server hosts these text files, therefore, engineers can download them directly instead of querying the database in on-demand basis. New coming data are appended to the text files and there are few minutes of delay between the data is acquired from the hardware until it is actually persisted in a text files.

The proposed data flow allows monitor data (messages) passing from the Blobbers through the ActiveMQ to several clients. Slow clients will be properly handled in order to avoid any impact on the on-line software. ActiveMQ has proved to be reliable and be able to keep up with the required data rates. The current throughput is in average between 4,500 and 5,000 Clobs per second. Each Clob has 308 bytes in average. Peaks of 30,000 have been generated in order to stress the implementation, which was properly handled with the current available hardware (see Table 1).

Table 1: Hardware of the Monitoring System Server

Monitoring System Server	
O.S.	Redhad Enterprise 6.3, 64 bit
CPU	Intel(R) Xeon(R) CPU E5-2630 0 @ 2.30GHz, 12 cores
Memory	64 GB
Storage	30 TB in Raid5

In order to fulfil the online plotting requirement, an additional process, the TMCDistributor, was added to consume the data after the ActiveMQ queue and publish it to a Redis [7] server, as shown in Fig. 3.

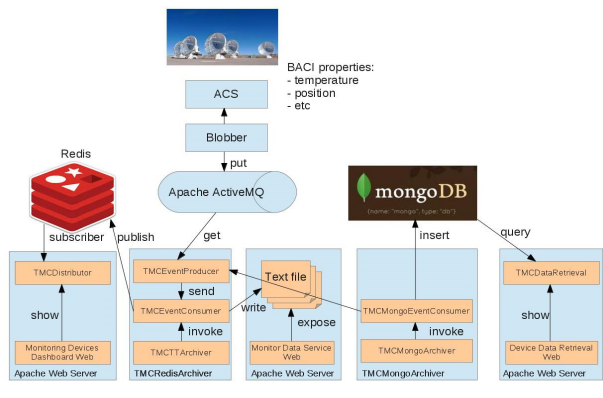

Figure 3: Redis and mongoDB added into the dataflow.

Only the last 20 samples are kept in memory by Redis for each monitor points. Within Redis, another layer of publisher/subscriber (channels) mechanism is used. Any subscriber can reads data and feed it to the plot dynamically. A web-based dashboards was implemented,

ISBN 978-3-95450-139-7

Copyright © 2014 CC-BY-3.0 and by the respective authors

engineers can access to them through a HTML5 compatible browsers. The dashboards were implemented with Webscokets and HighCharts [8] libraries.

The channels and the messages of each monitor point have a common custom structure:

publish application_name:device:monitor_point_name "Message"

Where the "device" parameter has the following format:

subsystem_name/antenna_name/device_name

And the "message" has the following format:

start_timestamp;end_timestamp;average_timestamp;CLOB;average

An example of a subscription to a channel of monitor points is showed in the Fig. 4.

```
redis 127.0.0.1:6379> SUBSCRIBE "TMCS:CONTROL/DV01/LORR:VDC_7"
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "TMCS:CONTROL/DV01/LORR:VDC_7"
3) (integer) 1
1) "message"
2) "TMCS:CONTROL/DV01/LORR:VDC_7"
3) "1359180856961975701359180858967092601372515751093;1359180855109319906.539683\n;6.539683"
1) "message"
2) "TMCS:CONTROL/DV01/LORR:VDC_7"
3) "1359180862968657601359180864963024901372515811093;1359180861109319906.539683\n;6.539683"
1) "message"
2) "TMCS:CONTROL/DV01/LORR:VDC_7"
3) "1359180868962650001359180870965179901372515871093;1359180867109319906.539683\n;6.539683"
1) "message"
2) "TMCS:CONTROL/DV01/LORR:VDC_7"
3) "1359180874961963601359180876962758601372515931093;1359180873109319906.5347986\n;6.5347986"
```

Figure 4: Example of channel structure in Redis.

Finally, for the persistent storage of historical data, we explored several no-SQL alternatives, such as: mongoDB [9], Cassandra, Hbase, etc. At the end the mongoDB presented the best trade-off between features and the required administration effort.

In mongoDB, three types of documents (schemas) was modelled for the monitoring data:

a) **One monitoring point per document**: The document is associated to one monitor point and it will contain only one single monitor point value. An example is shown in the Fig. 5.

```
{
    "_id" : { "$oid" : "50520ff925d8b6dfb8b4c353" },
    "date" : { "$date" : 1347554984516 },
    "location" : "TFINT",
    "componentName" : "CONTROL/CM12/DRXBBpr1",
    "propertyName" : "POWER_ALARM_REG_B",
    "monitorPointName" : "POWER_ALARM_REG_B_PSUMMARY",
    "serialNumber" : "10bfae3e010800c9",
    "monitorValue" : "255",
    "acsTime" : 135668477845168070,
    "index" : 0
}
```

Figure 5: Example of one monitoring point per document.

b) **A Clob per document**: The document is associated to one Clob of monitor point, as presented in Fig. 6.

```
{
    "_id" : { "$oid" : "50520ff925d8b6dfb8b4c353" }
    "dateStart" : "2012-12-11 11:50:28"
    "dateEnd" : "2012-12-11 11:50:38"
    "location" : "TFINT"
    "componentName" : "CONTROL/DV06/LO2BBpr2"
    "propertyName" : "POWER_SUPPLY_3_VALUE"
    "monitorPointName" : "POWER_SUPPLY_3_VALUE"
    "serialNumber" : "10bfae3e010800c9"
    "clob" : "135745302282272610|-14.713619999999999|135745302382272610|-14.713619999999999"
    "index" : "0"
}
```

Figure 6: Example of a "clob per document" schema.

c) **A monitor point per day per document**: The document is associated to a monitor point and it contains values of one day. Fig. 7 shows an example of a document using this kind of schema.

```
{
    _id : "20120901/CM12/DRXBBpr1/POWER_ALARM_REG_B_PSUMMARY",
    metadata : {
        date : "2012-09-01",
        antenna : "CM12",
        component : "DRXBBpr1",
        property : "POWER_ALARM_REG_B",
        monitorPoint : "POWER_ALARM_REG_B_PSUMMARY",
        location : "TFINT",
        serialNumber : "10bfae3e010800c9",
        index : 0,
        sampleTime : 1 },
    hourly : {
        "0" : {
            "0" : {
                "0" : 12345,
                "1" : 12345,
                "2" : 12345,
                ...
                "59" : 12345 },
            "1" : {
                "0" : 12345,
                "1" : 12345,
                "2" : 12345,
                ...
                "59" : 12345 },
            ...
            "59" : {
                "0" : 12345,
                "1" : 12345,
                "2" : 12345,
                ...
                "59" : 12345 } },
        "1" : {
            "0" : {
                "0" : 12345,
                "1" : 12345,
                "2" : 12345,
                ...
                "59" : 12345 },
            "1" : {
                "0" : 12345,
                "1" : 12345,
                "2" : 12345,
                ...
                "59" : 12345 },
            ...
            "59" : {
                "0" : 12345,
                "1" : 12345,
                "2" : 12345,
                ...
                "59" : 12345 } } }
}
```

Figure 7: A monitor point per day per document schema.

Several tests were done to determine if the proposed designs were able to meet the required performance. The results showed that the scheme of "**a monitor point per day per document**" provided the best balance between amount of documents within single a collection (the equivalent of table in relational database) and the granularity of the data of a single monitor point (in real life, no body is interested in just a single monitor point but in a range of monitor points)

In the case of one day-worth data of a monitor point, this schema actually managed to retrieve the required data within couples of milliseconds.

Another advantage to save the monitor data in mongoDB is the simplicity of the queries. For example, to retrieve single data of a monitor point named "FrontEnd/Cryostat/GATE_VALVE_STATE", with seconds-level of granularity can be achieve by using the following excerpt (in the case we queried for the data at 2012-09- 15T15:29:18).

```
db.monitorData_[MONTH].findOne(
{"metadata.date": "2012-9-15",
  "metadata.monitorPoint":
"GATE_VALVE_STATE",
  "metadata.antenna": "DV10",
  "metadata.component": "FrontEnd/Cryostat"},
  { 'hourly.15.29.18': 1 }
);
```

Copyright © 2014 CC-BY-3.0 and by the respective authors

ISBN 978-3-95450-139-7

To retrieve a range of values of the same monitor point at 2012-09-15T15:29 (all samples acquired during the the minute 29), we can use the following example:

```
db.monitorData_[MONTH].findOne(
{"metadata.date": "2012-9-15",
  "metadata.monitorPoint":
"GATE_VALVE_STATE",
  "metadata.antenna": "DV10",
  "metadata.component": "FrontEnd/Cryostat"},
  { 'hourly.15.29': 1 }
);
```

CONCLUSION

We presented the current implementation of the monitor system of ALMA, a system capable to handle up to 150k monitor points. We described the lessons learnt during the early stage of the deployment. Improvements introduced in critical sections of the data flow were shown, as well as the alternative way to deal with monitor data instead of the de-facto structured database implementations.

No-SQL database proved to be a valid solution for a monitoring system, in which, no-SQL is a perfect paradigm for storing big and heterogeneous amount of data. In our experience, mongoDB, a document-oriented solution, has demonstrated to be a good alternative for permanent data storage. The chosen schema, "A monitor point per day per document", fulfilled most of the use cases in the operation with regards to the monitoring data, especially, it allows queries to be returned in range of milliseconds.

Redis is an appropriate key-value solution for caching short period of monitoring data. Redis channels are well designed for publishers/subscribers of events in pseudo real time environment.

Finally, we believe that we can also achieve the same results by de-normalized the schema of TMCDB and a better definition of indexes in the relational database. But we believe that it is better and more nature to use no-SQL database to deal with no structured data instead of the very complex entity-relational scheme. At the end, why do you bother to structure the data while it will be used only in no structured way at the end of the day?

REFERENCES

[1] Shen, T., Ibsen, J., Soto, R., Other. "Status of ALMA software", ICALEPCS, MOPMU024. (2011).

[2] Schwarz, J., Farris, A., Sommer, H., "The ALMA Software Architecture", Proceedings of SPIE, 5496, p. 190 (2004).

[3] Farris, A. and Juerges, T., Device Driver Code Generation Framework. ALMA (2007).

[4] Chiozzi, G. and Sekoranja, M., ALMA Common Software Overview (2006).

[5] ActiveMQ website http://activemq.apache.org

[6] Java Management Extensions (JMX) website http://www.oracle.com/technetwork/java/javase/tech/javamanagement-140525.html

[7] Redis website http://redis.io

[8] Highcharts website http://www.highcharts.com

[9] mongoDB website http://www.mongodb.org

Copyright © 2014 CC-BY-3.0 and by the respective authors

ISBN 978-3-95450-139-7