OVERVIEW OF THE MONITORING DATA ARCHIVE USED ON MeerKAT

M. Slabber[*], SKA SA, Cape Town, South Africa

Abstract

MeerKAT [1], the 64-receptor radio telescope being built in the Karoo, South Africa, by Square Kilometre Array South Africa (SKA SA), comprises a large number of components. All components are interfaced to the Control and Monitoring (CAM) system via the Karoo Array Telescope Communication Protocol (KATCP). KATCP is used extensively for internal communications between CAM components and other subsystems [2]. A KATCP interface exposes requests and sensors [3]. Sampling strategies are set on sensors, ranging from several updates per second to infrequent updates. The sensor samples are of multiple types, from small integers to text fields. As the various components react to user input and sensor samples, the samples with timestamps need to be permanently stored and made available for scientists, engineers and operators to query and analyse. This paper present how the storage infrastructure (dubbed Katstore) manages the volume, velocity and variety of this data. Katstore is comprised of several stages of data collection and transportation. The stages move the data from monitoring nodes to storage node to permanent storage to offsite storage. Additional information (e.g. type, description, units) about each sensor is stored with the samples.

INTRODUCTION

On each node in the CAM system a monitoring process is responsible for collecting sensor samples for storage. The monitor process communicates with the proxy processes that, in turn, communicate directly with the devices and other systems. The rates at which the monitor processes collect these samples are configured upfront in the central configuration system.

A sample consists of the *sensor name*, *sample timestamp*, *value timestamp*, *status* and *value*.

Sample timestamp is the time at which the CAM system received the sample reading from the sensor. The *value timestamp* is the time at which the acquisition was performed on the sensor and the *value* stored. The *status* field holds the status of the sensor. Timestamps are represented as the time in seconds since the epoch of 1 January 1970 00:00:00 UTC.

The sensor sample storage system is responsible for collecting the samples from the monitor processes. The storage system transports the samples to a central storage node from where they can be queried and archived.

[*] martin@ska.ac.za

ARCHITECTURE

The storage system is comprised out of several elements (Fig. 1). Each element performs a specific task.

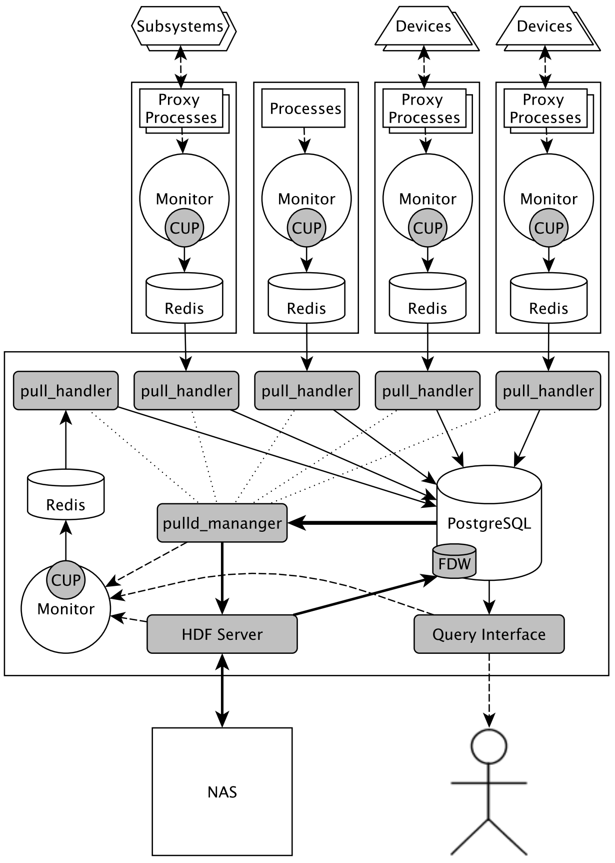

Figure 1: Connections between elements of the storage system.

Memory Buffer

Redis [4], an open source memory database server, is installed on each of the CAM nodes. Redis acts as a buffer to which the monitor process on the node writes sensor samples. On startup and at intervals, the monitor process will write metadata (e.g. type, unit of measure, description etc.) of the sensors to Redis. Internal to the monitor process, a class was developed to manage the writing to the memory database. This cache update class (CUP) runs in its own thread.

Pull Daemon - Pulld

A set of processes on the storage node pulls the samples and metadata out of Redis and stores it into the central database. These processes are collectively called Pulld. On a scheduled basis (currently daily), Pulld analyses the

ISBN 978-3-95450-148-9

database and samples older than a configured age (currently 2 days) are archived. Pulld starts up a separate process (pull_handler) for each Redis instance in the system. The pull_handler processes are managed by a parent process called pulld_manager. The pull_handler keeps a connection open to Redis and moves samples from there into the central database.

HDF Server

Samples to be archived, along with the relevant metadata of the sensors, are sent to the HDF Server process. HDF Server stores the samples into HDF5 formatted [5] files on the Network Attached Storage (NAS). A new file is created per day, per component of the CAM system. Files are stored in a hierarchical directory structure: a directory for each year contains directories for each month; which, in turn contains a directory for each day of the month. In this directory a file per component is created. The date is added to the file name to allow files to be copied and still maintain a unique name. For example the samples from 07 March 2015 of the *subarray1* component will be archived to *2015/03/07/2015-03-07_subarray1.h5*. Metadata for all of the sensors related to the samples in the file are also stored in the file.

Query Interfaces

Two independent interfaces were developed for applications, (e.g. Web GUI), components and other subsystems to query the storage system. All samples can be accessed through the query interfaces. The storage system is near real-time and lags the actual sensors by less than 1 minute. Archived samples (on the NAS in HDF5 files) are exposed in the database as a table called *samples_archived*. This table is a 'foreign table' in PostgreSQL parlance, and a foreign data wrapper (FDW) was developed for the HDF Server interface. This allows any archived sensor samples to be retrieved as if it is a row in the database. This is naturally not as fast as retrieving samples stored directly within the database system, but the performance is sufficient. The query interfaces use only the database API to access sensor samples, thus making it possible to develop powerful capabilities for the query interfaces in a few lines of Structured Query Language (SQL).

One of the query interfaces is developed to use KATCP as its access protocol. This interface is used by many of the internal CAM components to get historical sensor data and lists of sensor names. It provides filtering and produces the result in a format ready to be consumed by the components. This query interface runs on the storage node.

The second query interface runs on the portal node and provides an HTTP REST [6] -compliant interface and uses Javascript Object Notation (JSON) for encoding. This interface is mostly used by the web based graphical user interface (GUI). The GUI allows operators, scientists and engineers to create plots of any historical sensor data. The users of the GUI can search for sensor names using a regular expression syntax.

PULL VS. PUSH STRATEGY

One of the key differences in the architecture compared to other similar systems [7] is in the strategy used for moving samples from the nodes to the central storage system.

In a push system when a sample is available the sample is sent (pushed) to the central storage system. This is typically done with a distributed queue or via direct connections to the database.

In the architecture of the MeerKAT sensor samples storage system, a much simpler pull strategy was selected. This allows the monitoring processes and the storage systems to be completely decoupled. The storage system only needs to know the address of the memory buffer; it need not know which sensors or sampling strategies are associated with a monitoring process. This allows the important control and monitoring activities of the CAM system to evolve over time to best suit the telescope requirements, without needing to alter the storage system. For the monitor process this results in a very simple implementation that is unobtrusive and extremely fast.

The pull_handler implementation is also fairly uncomplicated. Samples are retrieved from Redis in as large a batch as possible and stored to the central database. Once the samples are stored, pull_handler removes the samples from Redis and retrieves another batch of samples. By batching samples it is possible to attain a much higher transfer rate and to balance the utilisation on the central database better. There are multiple pull_handlers all writing to the database in batches.

Another advantage of the decoupling of the components is that the storage system and monitoring processes can be restarted independently with no effect on one another. This is advantageous at startup, when developing and for fault-finding purposes.

CENTRAL DATABASE

PostgreSQL [8] is used for the central database. Many other database management systems were considered, most notably MongoDB [9] a NoSQL database system. It was found that PostgreSQL suited the needs of the sensor samples storage system best.

The pull_handlers write batches of samples to the database using the COPY FROM command rather than INSERT. This hits the SQL parser only once and thus write the samples more efficiently.

Horizontal partitioning (sharding) is employed and samples are written to different tables. A shardkey is calculated by taking the first part of a sensor name up to the separator character "_". This shardkey has no logical meaning within the CAM system and is only used within the database.

Child table creation in the partition is determined by three parameters; which, are also used to build up the name of the table.

1. A table contains only a days worth of samples.
2. A table is only associated with one shardkey.
3. A table is only associated with one pull_handler.

Separating tables per day and per shardkey has the advantage that when samples have been archived and need to be removed from the database, they are organised in such a manner that the table can be dropped. Using the DROP command is much less resource intensive and faster than using the DELETE command over the same samples.

The chosen shardkey distributes the usage of the tables so that only one pull_handler writes to a table. To guarantee this, the internal ID of the pull_handler is used as part of the table name. Thus there is no opportunity for write contention on any of the child tables.

When a child table is created, constraints are placed on the table. The shardkey and the minimum and maximum values of the sample timestamps are used as the constraints. These constraints help the database query parser to limit the tables used when processing a query. Well-formed queries where shardkey, minimum sample timestamp and maximum sample timestamp are given perform as well as if done directly against a standard database table.

CONCLUSION

We conducted research into many different database management systems of different types. It was concluded that no single system would fulfil all the requirements. A solution based on using Redis database as a buffer on the nodes and PostgreSQL as the central database was proposed, tested and used in the final implementation. It was found that the HDF5 file format was the preferred and most suitable format for archiving the data.

A complete system was developed to move sensor samples efficiently from the nodes where they were collected on to the central storage node where the samples can be archived and queried.

The system was designed in several independent components. Each component is concerned with a specific function. Thus while developing each component, it was possible to focus exactly on solving the problem at hand. The components make testing and fault finding easier. The components will also make it easier to improve the performance of the system as each component can be measured and improved independently of the others.

REFERENCES

[1] R. S. Booth, W. J. G. de Blok, J. L. Jonas, and B. Fanaroff, "MeerKAT Key Project Science, Specifications, and Proposals," *ArXiv e-prints*, pp. 1–16, 2009. [Online]. Available: http://arxiv.org/abs/0910.2935

[2] L. van den Heever, "MeerKAT Control And Monitoring - Design Concepts and Status," in *Proceedings of ICALEPC 2013*, ser. MOCOAAB06, 2013.

[3] S. Cross, R. Crida, T. Bennett, M. Welz, and T. Kusel, "Guidelines for Communication with Devices," 2012. [Online]. Available: http://pythonhosted.org/katcp/_downloads/NRF-KAT7-6.0-IFCE-002-Rev5.pdf

[4] "Redis homepage," Sep. 2015. [Online]. Available: http://redis.io

[5] M. Folk, G. Heber, and Q. Koziol, "An overview of the HDF5 technology suite and its applications," *Proceedings of the EDBT/ . . .*, pp. 36–47, 2011.

[6] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, Irvine, 2000. [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[7] T. Shen, R. Soto, P. Merino, L. Peña, A. Barrientos, M. Bartsch, A. Aguirre, J. I. Alma, and A. D. Cordova, "Exploring No-SQL alternatives for ALMA monitoring system," in *Proceedings of ICALEPC 2013*, ser. WECOBA06, 2013.

[8] "Postgresql homepage," Sep. 2015. [Online]. Available: http://www.postgresql.org

[9] "Mongodb homepage," Sep. 2015. [Online]. Available: http://www.mongodb.org

ISBN 978-3-95450-148-9