SCALABLE ARCHIVING WITH THE CASSANDRA ARCHIVER FOR CSS

S. Marsching*, aquenos GmbH, Baden-Baden, Germany

Abstract

An archive for process-variable values is an important part of most supervisory control and data acquisition (SCADA) systems, because it allows operators to investigate past events, thus helping in identifying and resolving problems in the operation of the supervised facility. For large facilities like particle accelerators, there can be more than one hundred thousand process variables that have to be archived. When these process variables change at a rate of one Hertz or more, a single computer system can typically not handle the data processing and storage. The Cassandra Archiver has been developed in order to provide a simple to use, scalable data-archiving solution. It seamlessly plugs into Control System Studio (CSS), providing quick and simple access to all archived process variables. An Apache Cassandra database is used for storing the data, automatically distributing it over many nodes and providing high-availability features. This contribution depicts the architecture of the Cassandra Archiver and presents performance benchmarks outlining the scalability and comparing it to traditional archiving solutions based on relational databases.

INTRODUCTION

Archiving of process-variable (PV) values is a challenge for many large experimental physics facilities (e.g. particle accelerators). While there are plenty of tools for storing experimental data, even at very high rates, these tools are typically designed to store the data acquired for well-defined experiments. However, the number of tools that are designed to store continuous time-series data is rather limited.

The Cassandra Archiver [1] has been designed in order to provide a scalable and reliable data-store for time series data, as it is typically provided by control-system PVs. The system is very scalable because its performance scales linearly with the number of participating nodes. Its built-in decimation mechanism allows very quick data retrieval, even over very long periods of time. The Cassandra Archiver is based on the Control System Studio (CSS) framework [2] and neatly integrates with it. However, it can also be used independently of CSS. The PV abstraction layer provided by CSS is used to support archiving of EPICS PVs, however the layer can be easily adapted to support PVs coming from other control systems.

* sebastian<dot>marsching<at>aquenos.com

APACHE CASSANDRA

Apache Cassandra [3] is a distributed, scalable, and highly available key-value store that is being developed and maintained by the Apache Software Foundation. The data model used by Cassandra has been inspired by the data model of Google Bigtable [4].

History

Apache Cassandra was initiated at Facebook to facilitate their need for a scalable and highly-available data store. In 2008 the software was released to the public under an open-source license. In 2009 the software became an Apache Incubator project and was promoted to an Apache top-level project in 2010.

Data Model

The data model of Apache Cassandra was inspired by the data model used by Google Bigtable [4]. A so-called cluster is built by the Cassandra instances running on one or multiple (up to a few thousand) computers. Each computer participating in the cluster is called a node. All nodes of a cluster are completely equal; there is no master node.

In contrast to a relational database management system (RDBMS), data is not organized in tables but in column families. A column family effectively is a two-dimensional map where the keys of the first layer (row keys) are not ordered and thus can easily be distributed over the cluster, allowing for linear scaling. The key-value pairs on the second level (columns), however, are ordered by their keys, and thus it is possible to efficiently retrieve a certain range of columns for a row. Unlike a table in an RDBMS, a column family does not necessarily have a fixed structure. That means that the internal structure of each row can be different.

Replication

In order to provide high availability, each piece of data stored in a keyspace (the structure which aggregates column families for an application) is stored on multiple nodes. The number of nodes used for storing a piece of data is called the replication factor and can be configured per keyspace. Therefore, the replication factor can be configured to match the availability requirements of each application. A higher replication factor increases the availability but reduces the disk-space efficiency.

Benefits

Because of its design, Apache Cassandra is extremely scalable. The throughput and the amount of data that can be stored scale linearly with the number of nodes [5]. New

ISBN 978-3-95450-139-7

nodes can be added while the cluster is running, thus it is easy to grow the cluster as the demand grows.

The on-disk storage format is optimized for a high throughput: All write operations are sequential, thus the number of seeks is minimal and the average write-rate is high. In addition to that, all data written on disk is compressed, resulting in an even higher read and write rate and saving disk space.

The replication scheme allows for the use of cheap hardware because the reliability of a single node is far less important. Availability is not ensured on a per-node basis (e.g. using RAID), but by replicating the data across multiple nodes. This increases the availability because it accounts for the failure of any type of hardware component.

When compared to an RDBMS, Apache Cassandra offers better availability and performance at the cost of features (e.g. complex queries are not supported). This makes it optimal for the storage of time-series data.

When compared to other key-value stores with a similar data model (e.g. Apache HBase, Hypertable), Apache Cassandra is easier to setup and requires less maintenance. In particular, unlike HBase and Hypertable, Apache Cassandra does not have a master node and thus does not have a single point of failure.

CASSANDRA ARCHIVER

When the Cassandra Archiver was developed at aquenos, the primary aim for its design was to create an archiving solution for EPICS PVs that would be very scalable and would have a performance that would be the same as or even better than the existing solutions.

Apache Cassandra was chosen as the storage backend for three reasons: First, it had the scalability and high availability that was required. Second, it promised a good input / output (I/O) performance when being used correctly. Third, it was easier to setup and maintain than the other products that were considered, in particular because its architecture does not have a single point of failure.

CSS was chosen as a development platform because it provides the means to support different control systems and already provides some infrastructure for archiving that has been developed for the RDB Archiver [6]. Besides, the support for CSS makes it easy for users to integrate the Cassandra Archiver into their infrastructure.

Data Model

Since version 2.0 the data model of the Cassandra Archiver has been optimized for best performance. In order to achieve this goal, a concept is needed where the number of rows and the number of columns per row are balanced. Therefore, data for the same PV and period of time (called a bucket) is aggregated in a single row [7].

When reading data, all buckets that may contain data for the given time interval are read. Typically, a bucket contains about one million samples, so that for most queries looking at two buckets is already sufficient. The start of each bucket is aligned to its size and the zero timestamp, so that it is enough to keep a list of bucket-sizes used in the past and no directory containing an entry for each bucket needs to be kept. This makes this structure very scalable. The size of the bucket used for storing live data is determined on start-up by looking at the sample rate configured for the respective PV.

Decimation

Often a user does not want to get detailed data for every sample, but rather wants to see a quick trend over a longer period of time. If such a trend is created by reading all samples for the period and then decimating them before creating the plot, this process can take considerable time, even if this process takes place on the server (like it does for the RDB Archiver), because all data has to be read from disk. Therefore, the Cassandra Archiver has the abilitiy to create archives with decimated samples on the fly while archiving the raw samples. This way, a request for an extended period of time can be served from the decimated archive and thus requests for long periods of time can be served within seconds instead of minutes. The extra disk space spent on the decimated samples is negligible because storage in an Apache Cassandra database is cheap compared to an RDBMS.

The levels used for decimation (called compression levels) can be configured for each PV. For example, a PV having a change rate of one sample per second might have compression levels of 30 seconds, 5 minutes, 1 hour, and 12 hours. In this example a request to plot the data for the last three years could be served from the 12 hours data, resulting in only 2190 samples to be read instead of 94.6 million samples for the raw data.

Decimated samples represent the actual average value for the respective period of time and also include the minimum and maximum value. Thus, the user can get a good idea about where interesting things might have happened and thus zooming in might make sense.

An additional benefit of the compression scheme is that different retention periods can be configured for different compression levels. Thus, the raw data could be kept for a limited time only, while still allowing to get a coarse trend of the data for a long period of time.

Data Access

Access to the data stored in the Cassandra database is provided through a plugin for the CSS data browser [8]. The JSON Archive Proxy [9], that provides access through a simple JSON-based HTTP web service, can be used as an alternative way of access. Therefore, applications that are not based on CSS can still access the archive data easily.

COMPARISON WITH OTHER ARCHIVERS

There are a few archiving solutions which have been developed to archive EPICS PVs. In particular three pre-existing solutions are interesting for a comparison because some of them share some concepts of the Cassandra

ISBN 978-3-95450-139-7

Archiver while others are widely used in EPICS control systems.

Channel Archiver

The Channel Archiver [10] is the oldest of the widely-used archiving solutions for EPICS. It is written in C++ and uses its own data-format for on-disk storage. The data can be accessed through an XMLRPC interface.

The data-format used is optimized for time-series data, and thus the Channel Archiver can sustain a high write-rate for samples. However, when it comes to reading data, it does not scale very well. Some of the scalability issues have been addressed recently [11], using a data-model that is similar to Apache Cassandra. However, multi-node operation is still not supported completely and thus the scalability and availability are limited.

Due to the substantial work required to maintain and improve an internal data-storage engine, some people have moved to the RDB Archiver as a replacement.

RDB Archiver

Due to the above-mentioned problems, the RDB Archiver [6] was developed. Instead of relying on its own data format for on-disk storage, it stores the archived channels in a relational database. At the moment MySQL, PostgreSQL, and Oracle are supported. The software is written in Java and uses a modern design, being based on CSS. The Cassandra Archiver reuses parts of this software.

Compared to the Channel Archiver, the write rate for samples is lower by a factor of about seven [12]. The scalability is basically determined by the scalability of the RDBMS being used. There is a clustered version of MySQL, that can be used to distribute data and to provide high-availability. However, given the poor-performance per node, the cluster has to grow quite large, which apart from being expensive can also create additional performance problems [5, 13]. When using Oracle instead of MySQL, the license costs, which typically increase with the size of the cluster, make it undesirable to deploy a huge cluster.

The main reason for the performance problems with the RDB Archiver is that RDBMS have not been designed to store time-series data, but to store relational data. Therefore, their design focuses on features like transactional consistency and complex queries rather than high write and read rates for sequential data. This affects the performance negatively for applications like time-series data, where data is usually read and written in rather large chunks.

HyperArchiver

Based on the insight that RDBMS are not optimal for storing time-series data, the HyperArchiver [14] was developed. Like the Cassandra Archiver it shares some of its code with the RDB Archiver and uses a Hypertable database for storing the archived samples. The configuration data is stored in a MySQL database, so that two database systems have to be maintained and kept available.

Unlike Apache Cassandra, Hypertable's design has a master-role and thus a single point of failure, limiting the availability. Also Hypertable is harder to setup because it relies on the Apache Hadoop File System (HDFS) for data storage.

The HyperArchiver has not left the prototype-stage so far. The newest version released is from 2010 and has properties like host-names hard-coded into the source code, making it unsuitable for production use.

PERFORMANCE EVALUATION

For evaluating the performance of the Cassandra Archiver, a small cluster of Amazon Web Service (AWS) Elastic Computing Cloud (EC2) instances was used in order to ensure that all systems used in the evaluation have the same hardware characteristics. All nodes were running the Ubuntu 12.04 LTS 64-bit operating system.

The EPICS software IOCs that produced the samples to be archived were placed on separate nodes in order to assure that the number of samples served were not a limiting factor.

In order to compare the performance of the Cassandra Archiver and the RDB Archiver, a database cluster consisting of only a single node was used. MySQL is typically running on a single node only, and the per-node performance of a clustered database cannot be expected to be better than the single-node performance.

For both the RDB and the Cassandra Archiver, two archive engines were running on the node because tests showed that two engines running in parallel had a slightly better performance than a single engine, probably due to better usage of the multiple processor cores. The sample write-rates measured were averaged over a period of five minutes.

The RDB archiver was configured to archive 2000 channels (1000 per engine) at 10 Hz to a MySQL database using the MyISAM engine. With this setup, the RDB Archiver had a write rate of 9177 samples per second. Further increasing the number of channels archived did not increase the number of samples written.

The Cassandra archiver (running on the same kind of machine) was configured to archive 8000 channels (4000 per engine) at 10 Hz. Again, further increasing the number of channels archived did not increase the number of samples written. With this setup, the Cassandra Archiver had a write rate of 79691 samples per second, which is larger by a factor of eight compared to the RDB Archiver (see Fig. 1). This means that the performance of the Cassandra Archiver is in the same order of magnitude as the performance of the Channel Archiver as it is about seven times faster than the RDB Archiver [12].

The second test focused on testing the scalability of the Cassandra Archiver when running in a multi-node cluster. For this test, additional database nodes with the same configuration were added to the cluster, increasing the number of archive engines used by two with each node.

ISBN 978-3-95450-139-7

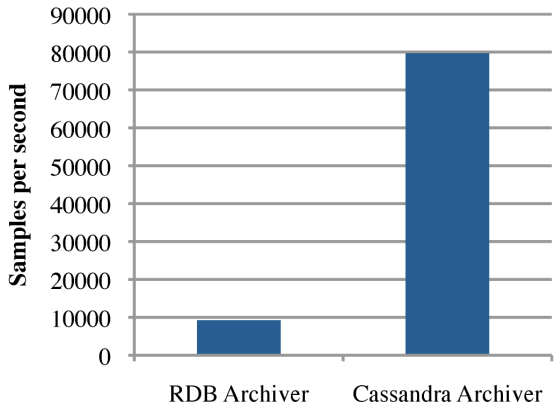

Figure 1: Comparison of RDB Archiver (MySQL) and Cassandra Archiver performance.

Figure 2 shows the write rate for a growing number of nodes (up to seven): The actual write rate (blue diamonds) scales nearly linearly (dashed line), resulting in a write rate of about 420k samples per second for seven nodes. This result is consistent with the expectations for an application using the Apache Cassandra database [5].

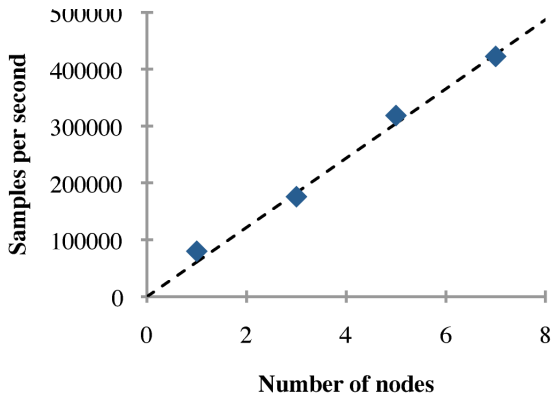

Figure 2: Expected linear and actual scaling of the Cassandra Archiver as nodes are added.

This test shows that a small cluster consisting of less than ten nodes, each using cheap hardware, can already archive several hundreds of thousands of samples per second.

SUMMARY

The Cassandra Archiver provides an archiving solution for EPICS channels that outperforms the RDB Archiver and neatly integrates into the CSS infrastructure. It benefits from the design of the Apache Cassandra database, that provides a data store that stays available in the event of a single node failure. The built-in decimation feature of the Cassandra Archiver allows users to quickly browse through the data for extended periods of time.

In the future, further developments of the Cassandra Archiver may concentrate on automatically distributing processing of PVs over the cluster and automatically adjusting the configuration settings to the actual change rate and sample size of a PV.

Due to the use of a widely-used data store, the Cassandra Archiver will benefit from further developments of Apache Cassandra and can scale to very large cluster sizes, potentially storing hundreds of tera-bytes or even peta-bytes of data.

REFERENCES

[1] aquenos GmbH, "Cassandra Archiver for CSS", http://oss.aquenos.com/epics/cassandra-archiver/.

[2] K. Kasemir, "Control System Studio Applications", ICALEPCS'07, Knoxville, October 2007, http://www.jacow.org/.

[3] The Apache Software Foundation, "Apache Cassandra", http://cassandra.apache.org/.

[4] F. Chang et al., "Bigtable, A Distributed Storage System for Structured Data", OSDI'06, Seattle, November 2006, http://research.google.com/archive/bigtable.html.

[5] J. Ellis, "2012 in review: Performance", DataStax Developer Blog, January 2013, http://www.datastax.com/dev/blog/2012-in-review-performance.

[6] K. Kasemir, "RDB Channel Archiver", March 2010, https://ics-web.sns.ornl.gov/css/docs/RDBChannelArchiver.doc.

[7] T. Hobbs, "Advanced Time Series with Cassandra", DataStax Developer Blog, March 2012, http://www.datastax.com/dev/blog/advanced-time-series-with-cassandra.

[8] K.U. Kasemir, "Control System Studio (CSS) Data Browser", PCaPAC'08, Ljubljana, October 2008, TUP009, p. 99, http://www.jacow.org/.

[9] aquenos GmbH, "JSON Archive Proxy for CSS", February 2013, http://oss.aquenos.com/epics/json-archive-proxy/.

[10] K.U. Kasemir and L.R. Dalesio, "Overview of the Experimental Physics and Industrial Control System (EPICS) Channel Archiver", October 2001, http://arxiv.org/abs/cs.oh/0110066.

[11] J. Rowland et al., "Algorithms and Data Structures for the EPICS Channel Archiver", ICALEPCS'11, Grenoble, October 2011, MOPKN006, p. 96, http://www.jacow.org/.

[12] "Control System Studio - RDB Archive", October 2010, http://sourceforge.net/apps/trac/cs-studio/wiki/RDBArchive.

[13] J. Shute et al., "F1: A Distributed SQL Database That Scales", Proceedings of the VLDB Endowment, Vol. 6, No. 11, Trento, August 2013, http://research.google.com/pubs/pub41344.html.

[14] M. Giacchini et al., "HyperArchiver: An EPICS Archiver Prototype Based on Hypertable", ICALEPCS'11, Grenoble, October 2011, MOPKN012, p. 114, http://www.jacow.org/.

ISBN 978-3-95450-139-7