ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-WE3BC001

MODULAR AND SCALABLE ARCHIVING FOR EPICS CHANNEL ACCESS AND GENERAL TIME SERIES USING SCYLLA AND RUST

D. Werder[§], T. Humar[‡], Paul Scherrer Institut, 5232 Villigen PSI, Switzerland

Abstract

To unify, simplify and improve our archiving setup we have developed a more modular archiver architecture based on an existing database with industry adoption and enterprise support, combined with additional services which ingest data from EPICS sources into the database and serve user requests for channel data from the database. The prototype is currently tested at the Swiss Free Electron Laser (SwissFEL) and planned to run in production mode from the beginning of 2024. Additional support for more input sources beside EPICS sources is also under development.

CURRENT ARCHIVING SITUATION

At the Paul Scherrer Institut we operate at the moment several different software products to archive our EPICS [1] process variables (PV) as well as data from non-EPICS sources. At the Swiss Light Source (SLS) we use the EPICS Channel Archiver [2], while at the SwissFEL [3] we use the EPICS Archiver Appliance [4] for the EPICS channels as well as the SwissFEL-Databuffer to buffer beam-synchronous data at the machine pulse rate, where the difference between the archiver and buffer being only that the buffer retains the data only for a limited time. Also HIPA, Proscan and several other smaller systems operate archivers in differing versions.

The operation of these heterogeneous setups binds resources, requires extensive expertise and makes the operation more difficult than it needs to be. Also, the current products are not easy to scale and hard to enhance for replication and high availability. Furthermore, it is often difficult, unspecified or impossible to access data from an external process in a defined, synchronized way, and also the file formats themselves are often used in rather small communities so that the available tooling can be limited.

At this time, the upgrade program for SLS 2.0 is now ongoing, which will among other changes bring a substantial increase in the number of archived channels. To meet the evolving requirements and to simplify and unify our archiving and buffering architecture, we have conducted a design study with the purpose to find a more modular and hopefully more sustainable approach.

DESIGN GOALS AND ARCHITECTURE

Instead of the current rather monolithic archiver solutions we would like to de-couple and modularize the setups. Our solution should be more easy to scale, should offer tunable replication and availability. We would like to be able to access the data concurrently in a well-defined way with more clear interfaces. The core storage engine should therefore be a dedicated database product which runs in its own process(s) and offers access via a network protocol. At the same time, we want to avoid having to maintain a custom solution and prefer an existing product which sees a wider usage also in industry.

The other components around the database to handle the different source types, including now Channel Access and in the future Beam-Synchronous-Readout and PV-Access, should be able to interact independently and concurrently with the common database. We give a brief overview of the architecture, after which we describe the individual components and their interaction.

Overview

The first component of this architecture, named Ingest service, handles the communication with the EPICS Channel Access Input Output Controllers (IOCs), maintains and monitors the channels (“virtual circuits”), and inserts the received updates from the PVs into the database.

A second component, named Retrieval service, is responsible to serve user requests for channel data and can deliver full events as well as aggregated and time-binned data. It communicates with the database as well. Other direct interaction between Ingest and Retrieval is not required, the database is the only connecting interface between the services.

As our database we have chosen Scylla [5]. Channel information and some other data which lends itself more to transactional databases is kept in Postgres [6].

Ingest service

The Ingest service [7] takes a list of channel names as input. It finds the IP addresses of the corresponding IOCs, maintains communication with the IOCs via the EPICS Channel Access Protocol [8], opens the channels on the IOCs and monitors for updates.

Updates to the PV values are inserted into the database, at which point we can also already maintain a reduced, aggregated and/or binned time series which is meant for a typically longer retention period.

The status of the channel and of the TCP connection is monitored and written as a separate time series to the database. To distinguish communication silence due to a lack of PV updates from a broken IOC connection, the Ingest service can issue a Channel Access Echo message.

§ dominik.werder@psi.ch
‡ tadej.humar@psi.ch

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-WE3BC001

The Ingest service searches for the IP address of a given channel via the usual EPICS UDP search on a given list of broadcast addresses. Found IOC addresses then get cached in the database which reduces the startup time. Cache entries are invalidated when Ingest encounters issues during an attempt to open a channel.

The event and data rate from a given IOC can optionally be throttled if it exceeds a configured threshold or if the connection contains e.g. unusually large waveforms.

A HTTP API allows inspection and configuration at runtime. It is possible to add and remove channels, as well as query the status of a specific channel or a set of channels selected by a pattern. Metrics from the Ingest service are also exposed via this HTTP endpoint in the format as used by Prometheus [9]. For the development of the Ingest (and Retrieval) service, we use the Rust [10] language. This language offers a unique combination of memory safety, predictable performance without garbage collector, powerful type system and machine code generation via LLVM [11]. The services are designed in a fully futures-based fashion, taking advantage of the async/await support of the language. We use the Tokio [12] runtime for execution. In addition, we use the Tokio-Tracing [13] library for flexible logging and integration with products for the collection of application traces.

All data flow within the Ingest service, specifically where it crosses synchronization boundaries, is done via channels. We do not use manual locks, and there is no manual threading and no shared memory used.

During testing, we use Valgrind to run Ingest under a reduced random production load for additional verification.

Database

The choice of Scylla was driven by our need for availability, redundancy and easy scalability. Scylla emerged as a rewrite of the Cassandra [14] database and continues the same architecture in a refined way. More performance is also gained from its more optimal usage of modern hardware.

Each node in a Scylla cluster is equal, there is no distinction between "main" and "secondary" nodes. Data is replicated with a chosen replication factor so that a chosen number of nodes hold the same data. Clients can select the number of nodes from which a confirmation is required before a transaction is considered as committed.

At its core, Scylla (like Cassandra) is a key-value store, meaning access is optimized for the case that the client provides the specific key it wants to access, while cluster-wide scans are prohibitively expensive. Within a key, a set of rows is stored, ordered by a secondary clustering-key. To avoid confusion, the "main" key is called partition-key.

Given a partition-key that the client wants to access, it can compute the set of of nodes that, to the best of the client's current knowledge, is responsible for the given partition-key, so that in the most common case the request can be sent directly to the correct node within the cluster.

EPICS PVs contain data of a certain data type. Scylla also offers the usual basic primitive types, together with a List type. In order to not lose performance we avoid storing data from the PVs as binary blobs, but instead in separate, accordingly typed Scylla tables.

For our use case of storing time series, our partition-key consists of the series identifier and a time range bucket. The clustering-key is the offset within the given time bucket. While the width of this time bucket is often fixed in many products, we have chosen to let this width vary dynamically based on the event rate. This choice is possible when we assume that under normal circumstances only a single Ingest will insert data for a given channel. If that assumption should ever not hold true, the issue can be discovered on read and corrected.

There are many other database systems on the market, each with their own different trade-offs. For our application we have put more weight on the factors of ease of scalability, redundancy, availability and homogeneity within the cluster, which have led to our choice.

Retrieval service

The Retrieval service [15] offers a HTTP API to look up channel information and fetch recorded data. Channel information contains most importantly the scalar type of the time series, the shape (e.g. scalar or waveform) and assigns each time series a unique identifier. Event and binned data can be fetched as JSON as well as in binary formats where we for improved efficiency use streams-of-structs-of-arrays.

In addition, the Retrieval has the option to perform aggregation for min/max/avg/var, and unweighted or time-weighted binning. It is also possible to transform data on the fly using a WebAssembly program which we have tested using the Wasmer [16] WebAssembly runtime embedded into the Retrieval, even though a corresponding interface is not yet available to make this accessible to potential users.

Ingest Configuration

The configuration of the ingest service consists in the simplest case of two files. The first is the main configuration file which contains the coordinates of the involved databases and the name of a second file, which specifies the list of channels to be archived.

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-WE3BC001

Example of a configuration to archive:

```
channels: channel-list.txt
search:
  - "172.16.16.255"
  - "172.16.24.255"
  - "172.16.32.255"
scylla:
  hosts:
    - scylla-node-01:19042
    - scylla-node-02:19042
    - scylla-node-03:19042
postgresql:
  host: postgres-node
  port: 5432
  name: channeldb
  user: ..
  pass: ..
```

while the channel list referenced by the main configuration file contains all the channel names to be archived, e.g.:

```
SARCL01-DBPM120:X1
SARCL01-DBPM120:X2 (etc.)
```

Availability and Deployment

Care is taken to allow for easy deployments. The application is deployed as a single executable and links only against some of the very basic shared libraries of the operating system, which are in the case of our Redhat [17] linux environment: linux-vdso, librt, libpthread, libm, libdl, libc, libgcc_s, ld-linux-x86-64.

Development of Ingest Prototype at SwissFEL

Our development environment consists of a four-node Scylla cluster assembled from decommissioned servers having Xeon E5-2680v2 Ivy Bridge CPUs (year 2013), spinning disks and 130 GB RAM assigned to Scylla. The Ingest service runs separately on a fifth node.

We configure Ingest with the 301k channels of SwissFEL, which are spread over 1290 IOCs. The total incoming traffic from the IOCs is on average 20.4 MB/s and results in 267 k/s inserts into the database on average. The CPU utilization is evenly spread over the assigned cores and without spikes on individual cores. The memory footprint of Ingest is negligible. The longest consecutive test run extended over 68 days, ended on purpose by user command.

Monitoring and Metrics

Besides log messages, Ingest, Retrieval and also Scylla are instrumented with a wide range of counters, histograms and values. The metrics of all three components can be scraped by Prometheus [8] and visualized for example in Grafana [18]. Some basic variables include the total rate of inserts by Ingest into the Scylla cluster (Fig. 1)

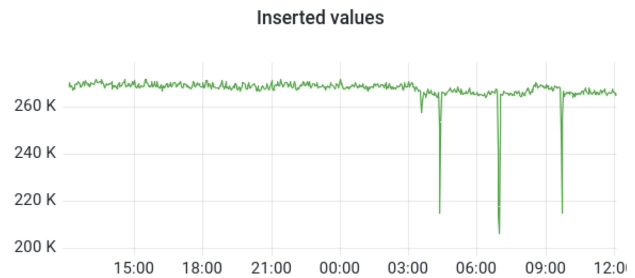

Figure 1: Rate of inserts by Ingest service into database.

as well as the percentiles of the total time that the average Channel Access event spends inside the Ingest service from the receive on the network to the commit acknowledged by the Scylla cluster (Fig. 2).

Figure 2: Quantiles of the duration between the receive of a Channel Access event update from the network and the commit confirmation for that update by the database server. Shown are 0.999, 0.99, 0.90 and 0.50 quantiles.

STATUS AND OUTLOOK

Started from a feasability study, we have demonstrated that the design works well and ingests our Channel Access data from all SwissFEL IOCs. The goal is to reach neccessary quality for basic production operation at the end of 2023. To that end some refactoring and simplification is in order based on the observations so far, user-friendliness has to be improved and commands have to be added to the HTTP API to allow for long-term 24/7 operation. From beginning of 2024, we plan to have the new hardware available and to run Ingest continuously at SwissFEL in parallel with the existing archivers and verify correctness with a more detailed comparison study.

REFERENCES

[1] EPICS, https://epics.anl.gov/

[2] EPICS Channel Archiver, https://epics.anl.gov/docs/GSWE/starttools/channelarchiver.htm

[3] SwissFEL, https://www.psi.ch/en/swissfel

[4] EPICS Archiver Appliance, https://slacmshankar.github.io/epicsarchiver_docs/index.html

[5] ScyllaDB, https://www.scylladb.com/

[6] PostgreSQL, https://www.postgresql.org/

[7] DAQ Ingest, https://github.com/paulscherrerinstitute/daqingest

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-WE3BC001

[8] EPICS Channel Access Protocol, https://epics.anl.gov/base/R3-16/0-docs/CAproto/index.html

[9] Prometheus, https://prometheus.io/

[10] Rust programming language, https://rust-lang.org/

[11] LLVM, https://llvm.org/

[12] Tokio, https://tokio.rs/

[13] Tokio-Tracing, https://tokio.rs/tokio/topics/tracing

[14] Cassandra, https://cassandra.apache.org/

[15] Retrieval, https://github.com/paulscherrerinstitute/daqbuffer

[16] Wasmer, https://wasmer.io/

[17] Redhat Linux, https://redhat.com/

[18] GitHub Actions, https://github.com/features/actions/

[19] Grafana, https://grafana.com/

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI