CAFlux: A NEW EPICS CHANNEL ARCHIVER SYSTEM*

Kanglin Xu[†], Los Alamos National Laboratory, Los Alamos, USA

work, publisher, and DOI. Abstract

nust

We introduce a new EPICS channel archiver system that title of the has been developed at LANSCE of Los Alamos National Laboratory. Different from the legacy archiver system, this system is built on *InfluxDB* database. InfluxDB is an open System is ounce the series database system that provides a series Like language for fast storage and retrieval of time series data. By replacing the old archiving engine and index file 5 archiving server. On a client side, we introduce a new implementation combined with asynchronous programming and multithreaded programming. We also describe a web based archiver configuration system that is associated with .틙 our current IRMIS system. To visualize the data stored, we maint use Javascript Plotly graphing library, another open source toolkit for time series data, to build front-end pages. In addition, we also developed a viewer application with more functionality including basic data statistics and simple arithfunctionality including ba

CAFlux ARCHITECTURE AND SUBSYSTEMS

distribution of this CAFlux EPICS channel archiver system is developed to replace the legacy archiver system as it is no longer an active Åny project. CAFlux system consists of several components as shown in Fig. 1. Content from this work may be used under the terms of the CC BY 3.0 licence (© 2019).

- A Data Collection Engine is developed with Python asynchronous programming and multithreaded programming based on PyEPICS [1] and EPICS channel access.
- InfluxDB Server is used as a data storage engine. InfluxDB [2] is a database system optimized for storage and retrieval of time series data. It supports a few hundred data nodes initially and is able to scale to a few thousand nodes. Its single-node edition is free but the clustering system is sold as commercial product.
- HTTP Web Server as a logic tier provides a web service for archiver configuration, data visualization and data stream.
- MySQL Server as another data tier holds all the user configuration information.

IMPLEMENTATION OF CAFlux DATA COLLECTION ENGINE

The Data Collection Engine is the core CAFlux subsystem that collects the data through EPICS channel access, caches it in the memory and saves it to the InfluxDB server. In this section, we describe the engine in details.

Multitier Architecture

The Data Collection Engine is designed as a multitier architecture, i.e. a lower level engine, a upper level manager and at the highest level, a monitor.

- · Lower level engine for core jobs: reading configurations, collecting and caching data, and saving data.
- Lower level engine designed as a daemon and developed with the Python asyncio and threads module.
- Upper level manager for monitoring the low level engine: checking the PID file, restarting the lower level daemon process if it is dead or in zombie status, logging error messages and sending emails if any issue occurs.
- The upper level manager designed and implemented as ٠ simple as possible in order to make it more robust and stable enough to run 24 hours a day and 7 days a week with low probability for any issues.
- The highest monitor is a simple script scheduled by LINUX Cron Daemon to run every 10 minutes to check the status of the running processes. When a process is dead, it will restart it and send an email notice.

Threads and Asynchronous Task Loop

When we designed the data collection engine, an obvious approach is to use a timer thread for each record to collect data, to save data, and then to sleep for a preset time and wait for the next cycle. The drawback of this approach was that large a number of threads that do work for a short time but sleep most of the time. In addition, the number of threads is limited by resources when the record volume becomes large. Another approach is to start a thread to do work and then to let it die after the work is finished. This approach is complicated because we need a mechanism to start a thread at a preset rate and synchronize a large number of threads to make sure only one thread can access the shared resources. We also consider an *asynchronous* approach that does all the work in the main thread. But we find that one thread is not enough to meet the performance requirement for high volume records.

In this system, we developed an engine through a combination of Python asynchronous programming and multithreaded programming. In this way, we can circumvent complex implementation for synchronizing a large number of threads and still have enough working threads to handle a large amount of records with high writing frequency. Specifically, the main thread is

- · to read inputs, initialize global data containers and start up
- · to initialize CA library and create CA context

Supported by the US Department of Energy, Los Alamos National Laboratory. Managed by Triad National Security, LLC for the DOE National Security Administration (Contract 89233218CNA000001). kxu@lanl.gov



Figure 1: A diagram of CAFlux and its subsystems.

- · to split new work threads
- to include an asynchronous task to check and log the health status of each work thread every few minutes and sleep the rest of the time.

On the other hand, each work thread with an asynchronous task loop in it handles hundreds of records in a "*parallel*" manner. Its work includes

- to get a record value from its cache and write it to the storage
- to set a lower priority for a channel if its status is found to be disconnected
- to repeat the above steps at a predefined rate.

Note that the number of working threads can be configured by users.

Channel Monitors by PyEPICS CA Module

The data collection engine is built on *PyEPICS* CA Module, a fairly complete wrapping of the basic *EPICS* CA library for *Python*, to save our efforts and time to write our channel monitor module. The monitor module emulates the *EPICS camonitor* application by implementing the follow steps.

- 1. Creating a channel for every record
- 2. Subscribing a connection callback function to CA and called by CA whenever a channel connection status changes
- 3. Subscribing a channel callback function to CA to monitor channel values and called by CA whenever channel value changes
- 4. The engine cache updated by the channel callback function to hold the current value

Data Structure for the Engine Cache

Since the engine cache is synchronized for both reading and writing access, it creats a bottleneck which impacts system performance. In order to allow more threads to access the cache at the same, we design a 2-level data structure for it as shown on Fig. 2.



Figure 2: Engine cache structure.

The **PV INDEX** is a *Python* dictionary mapping a channel name, e.g. *A*, to an integer. This integer is an index of **PV CACHE LIST** and used to search another *Python* dictionary contained in the **PV CACHE LIST** and having the channel *A*'s current value. The **PV INDEX** is initialized at startup and *read-only* during the program running. Therefore, there is no need for synchronization for it. The **PV CACHE LIST** is updated all the time, and thus do need to be synchronized. 17th Int. Conf. on Acc. and Large Exp. Physics Control Systems ISBN: 978-3-95450-209-7 ISSN: 2226-0358

	CAFlux	/iewer		_ ×	X Data Plotting (40376 points) for 1L .	- 0 ×
From: 2019-09-09 To:	2019-09-10 CHN Patterr	1:	1L .	- Help		
2019-09-10 10:17:46 ************************************	8: C27H_061E03 03M.001E 207H_061E03 03M.001E 207B_060000 2379.0000 3.03803 13.3159 80.060000 519.0000 900.060000 609.0000 900.060000 609.0000 900.060000 609.0000 902.060000 609.0000 902.06000 609.0000 902.06000 609.0000 902.06000 609.0000 902.06000 902.00000 902.00000 902.00000 902.00000 902.00000 902.00000 902.00000 902.00000 902.00000 902.00000 902.00000 902.00000 902.00000 902.00000 902.000000 902.000000 902.00000 902.000000 902.000000 902.000000 902.000000 902.000000 902.000000 902.00000000 902.00000000000000000000000000000000000	33 04ML001E03 33 04ML001E03 30 2978.000000 30 2978.000000 30 998.000000 30 991.000000 30 991.000000 30 992.000000 30	ssvr-net.lcs.net' ssvr-net.lcs.net' Ssvr-net.lcs.net	More		
	PI	.ot				
Prev Day	Next Day	Zoom Left	Zoom F	light	# ← → + Q 코 B	
LANSCE-RM Device Database Sy	ySLEM Zeropoure ICS CCE.Day Leg Administra ICS	re lafts: Zechtver: About	Figure	3: CA		second i long mot
Owner Group Description 1L 1L 1L 40 1L SNST Inflow Hellow	Active Channel Held 1 HFLW001D01 Held 1 HFLW001D02 Held 1 HFLW001P01 Held	IOC Rate Au	stive Update Channel h Firstox Br Dis Design Presenta	and News C Istory Weer Preferred For LANSCE-RM on For RDB	Current Values Current Values Current Values Current Values Current Values	e <mark>ws *</mark> SCE-RM DB glin updele

hefow	Hefow	1	HFLW001P0	Modify or Delete Channel with ID	9 ×		Associated Links	1		0.9708279	FLW001P08		HPLW001P08	Please slop by for update
Inj	inj	1	HFLW001P0			- 1	Thered Version	Submit Pre Day Next Day		0.9354442	FLW001D01			Associated Links
12	2	1	HFLW001P0	Channel: HFLW001P05			Fast Search			0.718256	FLW001D02		HFLW001706	A commenter
Joe Bradley	805 Quad Magnet Booster Pumps	1	HFLW001P0				Pv Report	0.5					HPLW001P05	Software Team
Joe Bradley	801 PS Cooling Water	1	HFLW001P0	Field:			BMS Search						HPLW001P04	Mana Library
Joe Bradley	802 Magnet Cooling Water	1	HFLW001P0	Bate:			^o Gate Oransels			-0.1554116	HFLW001P02			· Resolved SW MR
Joe Bradley	Beam Currents	1	HFLW001P0				Channel History	D		-0.2433508				Search
Joe Bradley	Beam Positions	1	HFLW001T0	Active: 0						-0.2549047	HFLWC01P01	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~		▼ Others
Joe Bradley	Chroma Shunts Sect A	1	HFLWCOITC	Fridaya						-0.2877712	HELW001P07			Centrols Dec
Joe Bradley	Communications Network Status	1	HFLW001TD	ausine				-0.5		-0.3037527	HFLW001P03			* Resources
Joe Bradley	C to H 01 Water	1	HFLW001TC	Note: To delete this item just leave ch	annel			-		-0.2810668	HFLW001P08			Network Status
Joe Bradley	D02 Magnet Cooling Water	1	HFLW001T0	name slot empty.				-1		-0.9251431	HELW001P05			
Joe Bradley	Exhaust Fan Ion Chambers	1	HFLW001TD	6 1	1				15:31:30	15:31:40 Oct 23, 2018, 15:31:49	15:32:00	15:32:10 1	5:32:20	
Joe Bradley	G02 Magnet Cooling Water	1	HFLW001TD	7 1	1				00.23,2010					
Joe Bradley	HEBT Activation Protection Spill	1	HFLW001TC	8 1	1									
Joe Bradley	Inside Conditions	1												
			Channel											
Print Made	y Group Madity Channels G	irosp #: 115	iten #: 18											
Note: you can g	et group's channels by clicking a row	In CROUP be	e or edit channel	parameters by clicking a row in CHANNEL box.										
	nstruction like lots of good sites				Sea	irch:	Submit	🏟 Yeah under construction like lots of good sites						Search

Figure 4: CAFlux configuration and management Figure 5: Online data retrieval and data stream.

But only the same dictionary needs to be synchronized rather than the entire cache. If two channels are located at different dictionaries, they can be touched parallelly. CAFlux archiver system allows users to tune the number of dictionaries in the PV CACHE LIST to meet their different requirements.

PV CACHE LIST to meet their different requirements. DATA RETRIEVAL AND VIEWER Data saved in the *Data Storage Engine*, actually an *In-fluxDB* database system, can be retrieved by SQL-like query language shipped with *InfluxDB*. Many popular program-ming languages have interfaces to connect, query, insert and \succeq manipulate data in *InfluxDB* systems. For example, *Python* ^O has a module called *InfluxDBClient* for users to read and e write data easily.

To display and plot data retrieved, an application called *CAFlux Viewer*, as shown in Fig.5, nas been developed *Python matplotlib* module. In addition to data waveform viby and simple arithmetic for record values.

CAFlux CONFIGURATION

The CAFlux configuration and management system is a \approx 3-tier web application that allows users to insert, update and delete records they want to archive. Other management features like Enable or Disable records, and archiving rate update are also available. The front-end uses JQuery AJAX and JQuery UI for a dynamic interface and web forms. The back-end implements GET and POST methods to handle requests and send data. The database is hosted on MySQL server. One example page of the web system for users to update record configuration is shown in Fig.4.

The online data retrieval and stream is also a web application as shown in Fig.5. Its front-end is developed with Javascript Graphing Library and JQuery AJAX while the back-end is built with Python Django to query data from *InfluxDB* data storage and response to requests.

CONCLUSION

Built on InfluxDB system, CAFlux has a safer, more compact and faster storage and is currently running continuously at Los Alamos Neutron Science Center of Los Alamos National Laboratory. A balance is achieved between archiver scalability and complexity with Python multithreaded and asynchronous programming.

REFERENCES

- [1] PyEPICS, http://https://cars9.uchicago.edu/ software/python/pyepics3/ca.html
- [2] InfluxDB, https://www.influxdata.com/