CAFIux: A NEW EPICS CHANNEL ARCHIVER SYSTEM

Kanglin Xu

Nov 14, 2018

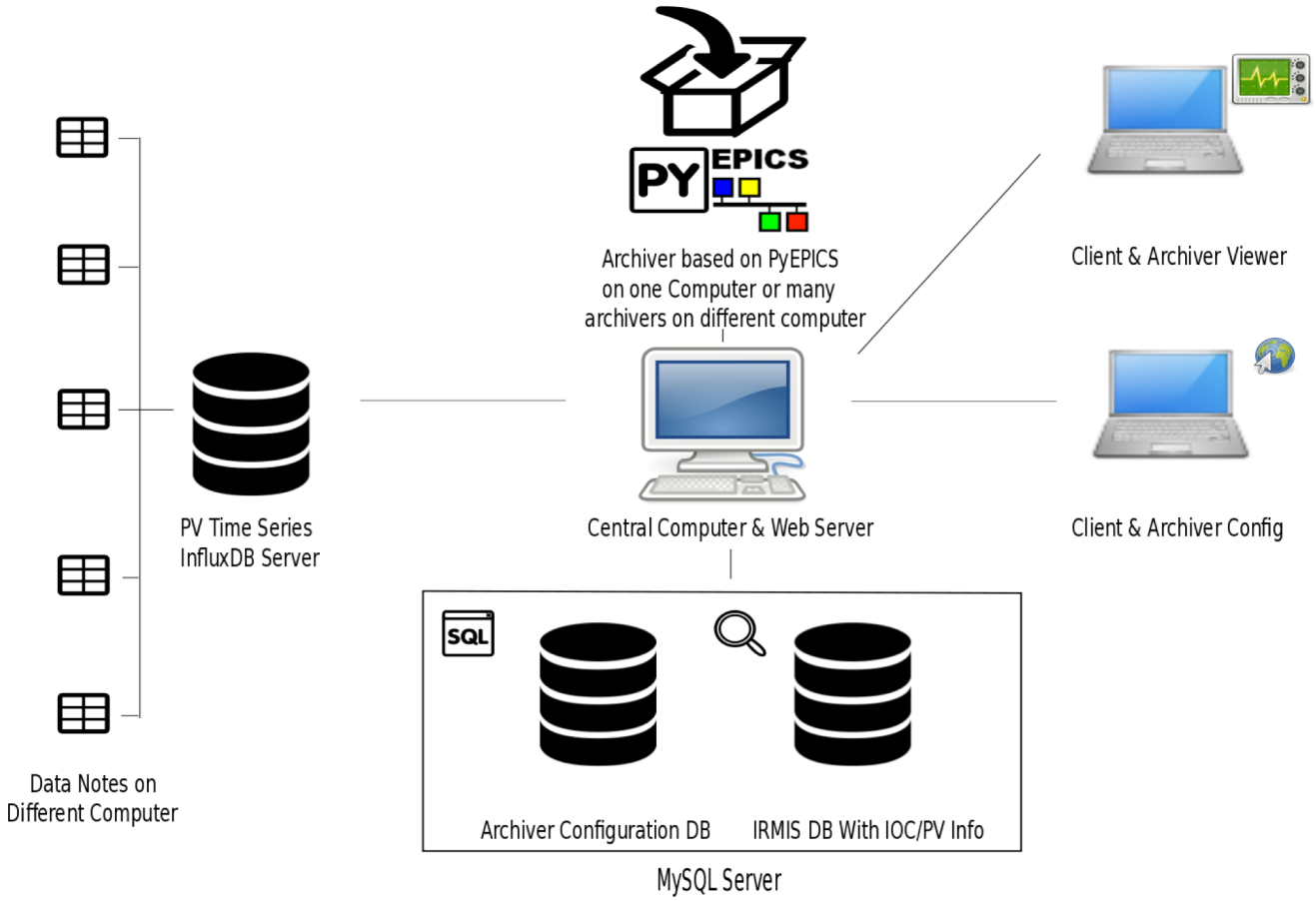
Operated by Los Alamos National Security, LLC for the U.S. Department of Energy's NNSA

CONTENTS

THE CURRENT LEGACY ARCHIVER SYSTEM AT LANSCE

- **Archiving storage and data collection**
  - Archive Engine – an EPICS channel access client
  - Archive Daemon to check archive engine status automatically
  - Several index files and large amount of data files
- **Data retrieval**
  - Java Archiver Client to browse data, to visualize data and to export data to spread sheets via a data server
  - A command line toolkits with functionality similar to the above Java client
  - Archive Data Server based on XML-RPC used to be a gateway for clients to access the data
- **Configuration**
  - Through XML configuration files
- **Released by SNS in 2006 and not an active project currently**
  - Unfortunately we do have an index file corrupted and have no idea how to fix it

A DIAGRAM OF CAFLUX AND RELATED SYSTEMS



INFLUXDB - A DATABASE SYSTEM OPTIMIZED FOR STORAGE AND RETRIEVAL OF TIME SERIES DATA IS USED AS CAFLUX STORAGE ENGINE

- **Fast READS and fast WRITES on high volume data**
- **Having a SQL like query language for RDBMS users**
- **Supporting a few hundred nodes initially and able to scale to a few thousand for future due to its clustering design**
  - Only one server node to be opened to archiving engines and application clients
- **Able to to create database, write and query data by using HTTP API**
- **But high performance on time series data at the expense of some functionalities**
  - Update functionality restricted
  - Delete functionality restricted
  - No cross table joins anymore
  - Fortunately the above functionalities rarely needed for our archiver system

INFLUXDB USED AS CAFLUX STORAGE ENGINE

- **Save our time and effort on developing a storage engine – different from legacy archiver**
- **Easy to use If you have some knowledge on SQL scripts**
  - An example for Python

```
from influxdb import InfluxDBClient
client =  InfluxDBClient(url, port, user, passwd, dbname, proxies=None)
client.write_points([ # to write two records
    {
        "measurement": "ioc",
        "tags"       : { "channel": "ABKS001" },
        "time"       : "2018-09-03 14:02:30",
        "fields"     : { "value": 0.22 }
    },
    {
        "measurement": "ioc",
        "tags"       : { "channel": "ABKS001" },
        "time"       : "2018-09-03 14:03:30",
        "fields"     : { "value": 0.25 }
    }
    ])

rs = client.query("SELECT * FROM ioc") # to red data
res = [(p["channel"], p["time"], p["value"]) for p in rs.get_points()] # get results
```

IMPLEMENTATION OF CAFlux DATA COLLECTION ENGINE – 2 TIERS

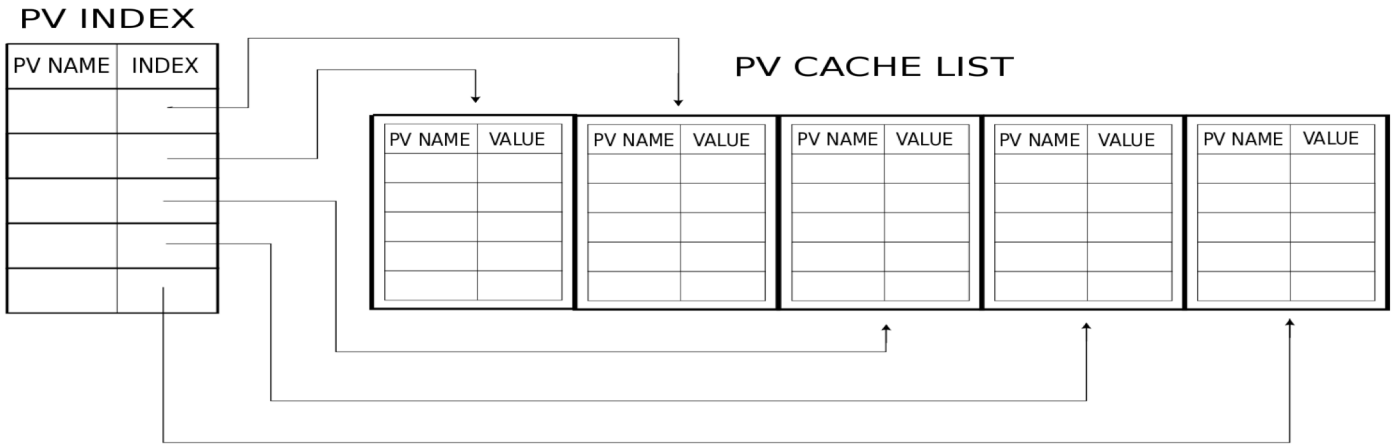
- **The 2-tier architecture for CAFlux data collection engine**
  - Lower level engine for core jobs - reading configurations, collecting and caching data, and saving data
  - Lower level engine designed as a daemon and developed with the Python asyncio and threads module
  - Upper level manager for 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 happens
  - The upper level manager designed and implemented as simple as possible in order to make it more robust and stable enough for running 24 hours a day and 7 days a week with low probability for any issues

IMPLEMENTATION OF CAFlux DATA COLLECTION ENGINE – MULTITHREADING VS ASYNCHRONOUS

- **An obvious approach to use a timer thread for each PV to collect data, to save data and then to sleep for a presetting time and wait for the next cycle**
  - Large number of threads that do work a little time but sleep most of time
  - Limited by resources considering CA itself creating a lot threads when the PV volume is large
- **Another approach to start a thread to do work and then to let it die after**
  - Hard to develop and to manage threads
  - Needing a mechanism to start a thread at a presetting rate
  - Complicate to synchronize the action of multiple threads and to make sure that only one thread can access the shared resources
- **Asynchronous approach only to do all the work in the main thread**
  - One workhorse not enough to meet a large amount of work particularly when there are a large amount of PVs with high writing frequency

IMPLEMENTATION OF CAFlux DATA COLLECTION ENGINE – MULTITHREADING AND ASYNCHRONOUS COMBINATION

- **Global data containers synchronized for all threads**
  - A PV QUEUE for holding all PVs unhandled and disconnected due to an IOC down
  - A PV cache LIST keeping many thread-safe maps which contains a hundred pairs of PV names and PV values
  - A PV index DICT mapping PV name and the PV cache LIST index to search which map this PV belongs to



Note that PV Cache list contains many PVName-Val dicts and each dict shares the same thread locker.

IMPLEMENTATION OF CAFlux DATA COLLECTION ENGINE – MULTITHREADING AND ASYNCHRONOUS COMBINATION

- **The main thread**
  - To read inputs, initialize global data containers and start up
  - To initialize CA library and create CA context
  - To split new work threads
  - To include an asynchronous task to check and log the health status of each work thread at a period of few minutes and sleep in the rest of time
- **Each work thread**
  - Have a asynchronous task loop to handle hundreds of channels in a "parallel" manner
    - Scan the PV QUEUE and get a PV to work on it
    - Update a PVName-val map/dict contained in the PV cache LIST via the PV index DICT whenever a PV value changes
    - Get a PV value from the PV cache and write it to the storage
    - Set a lower priority for a channel if its status is found to be disconnected (probably due to its IOC down) and put it back to the PV QUEUE
    - Do the above steps again at the presetting rate

IMPLEMENTATION OF CAFlux DATA COLLECTION ENGINE – PV MONITORS BY PYEPICS MODULE

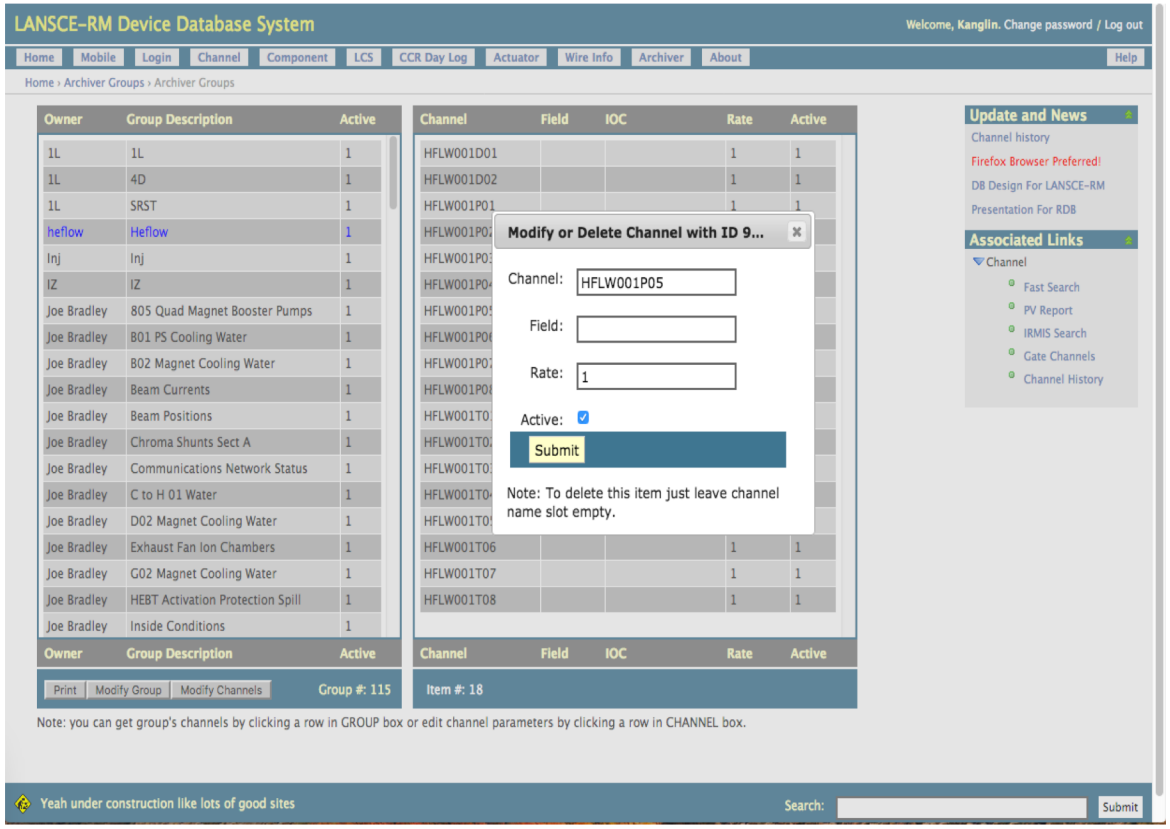
- **Creating a channel for every registered PV**
- **Subscribing a connection callback function to CA and called by CA whenever a channel connection status changes**
- **Subscribing a PV callback function to CA to monitor PV values and called by CA whenever PV value changes**
- **The PV central caches updated by the PV callback function to cache the "current" value on the channel collection server**
- **CA module of PyEPICS used to save our efforts and time to wrap the CA library and to make the implementation of the above steps easy**
- **Only trivial tweaks on the PyEPICS CA module for our multiple-threading needs, i.e. customizing PyEPICS CA module**
  - It might not be necessary if you really know how the PyEPICS CA module works and in this case you are completely dependent on Python GIL mechanism.
  - But those tweaks give us peace of mind(more details at Python workshop).

AN ONLINE CONFIGURATION SYSTEM – A 3-TIER WEB APPLICATION

- **A replacement of the legacy configuration files and a CGI web application**
- **Web browser as a platform for users to INSERT, READ, UPDATE and DELETE configuration information**
- **Web site developed with Python Django web framework**
  - Linux HTTPD ver 2.4 used for a web server
  - Apache mod_wsgi module to host the Python Django application
- **Configuration database built on a MySQL server**
  - Not on the same InfluxDB server for the data collection since UPDATE and DELETE functionalities needed for configuration but restricted on InfluxDB
  - The IRMIS system on the same MySQL server to provide detail information of archived PVs, e.g. their IOC information
- **A tool to load existing XML configuration files into the new system**

AN ONLINE CONFIGURATION SYSTEM – A 3-TIER WEB APPLICATION

- The frontend using JQuery AJAX and JQuery UI for a dynamic interface and web forms
- The backend implementing GET and POST methods to handle requests and send data
- The database consisting of a GROUP table and a CHAN table with 1:n relationship as shown



DATA RETRIEVAL AND VIEWERS

- **Data retrieval using SQL-like query language shipped with InfluxDB**
  - With help from the Python open-source InfluxdbClient module
  - Through a SELECT clause
- **Online archived data viewers**
  - Archived data streaming to view current data stored in real time
  - Archived data viewer for historical data stored
- **Viewer architecture and implementation - also 3-tier web applications**
  - Frontend developed with plotly.js - Javascript Graphing Library and JQuery AJAX
  - Backend developed with Python Django to query data and response to requests from web browsers
  - InfluxDB to manage data sources in addition to the data collection storage
- **Other free or commercial dashboards available for InfluxDB**
  - For example, Grafana shipped with data source plugin for InfluxDB

DATA RETRIEVAL AND VIEWERS – ARCHIVED DATA STREAMING TO VIEW CURRENT DATA STORED IN REAL TIME

DATA RETRIEVAL AND VIEWERS – ARCHIVED DATA VIEWER FOR HISTORICAL DATA

ARCHIVED DATA BACKUP AND RESTORE TO KEEP DATA IN SAFE AND CONSISTENT STATE

- **InfluxDB features for free (Not on the enterprise version)**
- **Data located at /var/lib/influxdb/data by default or somewhere you point to**
  - Time series data under the data directory
  - Meta data including user information, database and shard metadata, subscriptions, and etc. under the meta directory
- **Backup command line**
  - Run command “influxd backup –portable path-to-backup”
- **Restore command line**
  - Run command “influxd restore -portable path-to-backup”
- **Please refer to the manual at https://docs.influxdata.com/influxdb/v1.6 for details**

FUTURE WORK

- **Add data analysis and statistics like averages and standard deviations in the viewers**
  - With the help of the Python numpy and pandas modules
- **Monitor IOC status and adjust channels in a task loop accordingly**
  - Remove the channels of an IOC from a task loop if its status is OFF
  - Add the channels of an IOC into a task loop if its status if ON
- **Improve performance and availability with clustering nodes if really necessary**
  - InfluxDB OSS not support clustering
  - Clustering with InfluxDB Enterprise is not free
- **Develop data collection plugins for other database systems**