ISBN: 978-3-95450-214-1 ISSN: 2673-5490 doi:10.18429/JACoW-IPAC2021-WEPAB320

RecCeiver-ETCD: A BRIDGE BETWEEN ETCD AND ChannelFinder[*]

G. Jhang[†], T. Ashwarya, A. Carriveau, Facility for Rare Isotope Beams, East Lansing, USA

Abstract

Managing EPICS Process Variables' (PVs) metadata, such as the host and the contact, is one of the important tasks for the operation of large-scale accelerator facilities with minimal downtime. Record Sychronizer (RecSync) provides a way to manage such crucial information in an EPICS Input-Output Controller (IOC). RecCeiver-ETCD is the server component of the RecSync-ETCD, or an extension of RecCeiver for ETCD. In the previous work, the client component of RecSync, or RecCaster, has been extended to RecCaster-ETCD to store the metadata into an ETCD key-value store. An important remaining step to the production use is to introduce a connection between ETCD and ChannelFinder, which is achieved by RecCeiver in the RecSync system. RecCeiver-ETCD plays the role of the original RecCeiver in the RecSync-ETCD system. RecCeiver-ETCD is designed to perform the specific operation, bridging the communication between ETCD and ChannelFinder. In addition, its simple implementation does not hold it down to ChannelFinder and makes it easy to extend RecCeiver-ETCD out to the other applications.

INTRODUCTION

An EPICS [1] Input-Output-Controller (IOC) serves Process Variables (PVs) containing numerous machine-specific information crucial to operate large-scale accelerator facilities. Managing PVs' metadata, such as the server host information and who to contact, is one of the important tasks to maintain the facility's operation time higher by resolving any issue promptly. One of the methods for managing the metadata is to use the Record Synchronizer (RecSync) [2] and ChannelFinder system [3]. ChannelFinder is a simple directory service providing REST APIs for accessing PVs' metadata stored in Elasticsearch [4].

RECORD SYNCHRONIZER

The RecSync is a project for synchronizing the information in clients with Elasticsearch via a server. The client and server are called RecCaster and RecCeiver, respectively. RecCaster is an EPICS support module built with an IOC. When the IOC starts up, RecCaster waits for the server announcement from RecCeiver for establishing a connection to RecCeiver. Upon the successful connection, RecCaster sends PVs' metadata to RecCeiver. RecCeiver is a server written in Python language with Twisted network library for relaying the metadata to ChannelFinder via ChannelFinder Python APIs [5].

[*] Work supported by the U.S. Department of Energy Office of Science under Cooperative Agreement DE-SC0000661.
[†] changj@frib.msu.edu

RECCEIVER-ETCD

RecCeiver-ETCD is designed to relay the PVs' metadata in ETCD key-value store [6] updated by RecCaster-ETCD into Elasticsearch database via ChannelFinder using ChannelFinder Java APIs [7]. It is composed of three threads internally: the UDP Broadcaster thread, the ETCD Processor thread, and the ChannelFinder Processor thread. Each thread is encapsulated by a fail-safe method that retries every configured amount of time.

UDP Broadcaster Thread

The UDP Broadcaster Thread broadcasts ETCD server information to the sub-network periodically. Both the server information and broadcast period are configured in RecCeiver-ETCD configuration file. The broadcast feature exists in the RecCaster while it was not implemented in RecCaster-ETCD by the previous work [8]. The packet broadcast by RecCeiver-ETCD has the structure described in Fig. 1.

	0	1	2	3
0x00	ID		1	0
0x04	IP address			
0x08	port		0	

Figure 1: The Packet Structure of RecCeiver-ETCD UDP Packet.

Once the packet is broadcast, the UDP listener implemented in RecCaster-ETCD recognizes the packet with its ID and version number, which are the first two bits and the third bit, respectively. ID is 2-byte characters 'RC' common for RecSync and RecSync-ETCD while the version numbers for the former and the latter are 0 and 1, respectively. Unlike RecCaster relies only on the UDP broadcast from RecCeiver to make the connection, RecCaster-ETCD first uses the server information provided in the environmental variable. The UDP listener thread is triggered and activate when configured number of connection failures occurred.

ETCD Processor Thread

The ETCD Processor Thread consists of two processes: the initialization process and the watching process. The initialization process runs only once at the beginning of the RecCeiver-ETCD. It reads all key-values from ETCD and sends them to Elasticsearch via ChannelFinder Java APIs directly. The watching process is executed at the same time of the initialization process not to lose the information while the initialization process is running. The process watches and collects the ETCD value changes of keys having "/epics/ca" as their prefix. The collected data are stored in a shared

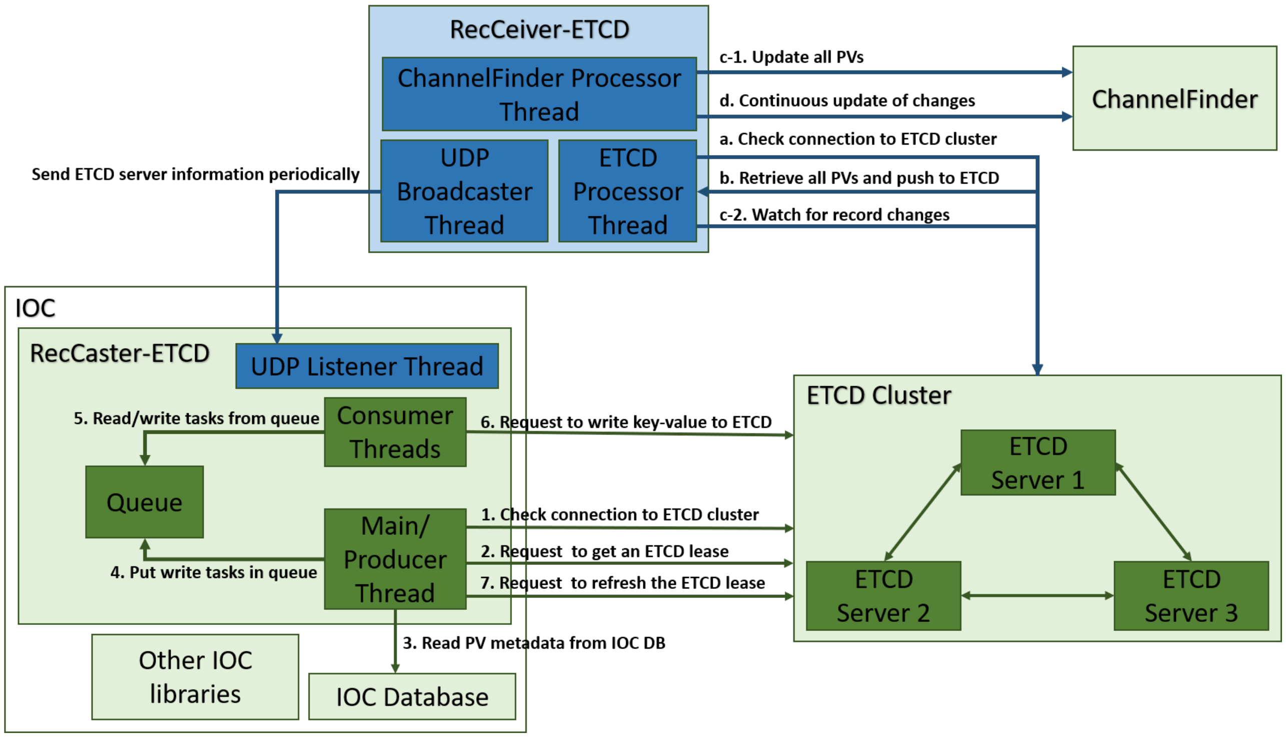
RecCeiver-ETCD
ChannelFinder Processor Thread
UDP Broadcaster Thread
ETCD Processor Thread
c-1. Update all PVs
d. Continuous update of changes
a. Check connection to ETCD cluster
b. Retrieve all PVs and push to ETCD
c-2. Watch for record changes
ChannelFinder
Send ETCD server information periodically
IOC
RecCaster-ETCD
UDP Listener Thread
5. Read/write tasks from queue
Consumer Threads
6. Request to write key-value to ETCD
Queue
Main/ Producer Thread
4. Put write tasks in queue
1. Check connection to ETCD cluster
2. Request to get an ETCD lease
7. Request to refresh the ETCD lease
3. Read PV metadata from IOC DB
Other IOC libraries
IOC Database
ETCD Cluster
ETCD Server 1
ETCD Server 2
ETCD Server 3

Figure 2: Schematic diagram of RecSync-ETCD operation with ChannelFinder. The work described in this paper is shown as the blue-colored. The operation orders of RecCeiver-ETCD and RecCaster-ETCD are labeled as 1~7 and a~d, respectively. Note that c-1 and c-2 run concurrently and the UDP Broadcaster Thread is atomic.

list for being sent until the initialization process ends. The watching process runs indefinitely and continuously updates the shared list whenever there are changes.

ChannelFinder Processor Thread

The ChannelFinder Processor Thread sends the information in the shared list mentioned in the *ETCD Processor Thread* periodically to Elasticsearch via ChannelFinder using ChannelFinder Java APIs. Before sending, the shared list is cloned not to block the shared list from being written by the *ETCD Processor Thread* for a long time. As soon as the cloning process ended, the shared list is cleared and blocking is released to continue collecting the changes from the watching process.

Operation Flow

Figure 2 shows the schematic diagram of the communication among RecCaster-ETCD, RecCeiver-ETCD, ETCD cluster, and ChannelFinder. The green-colored components in the IOC box are the implementation of the previous work in Ref. [8] and the rest green-colored components are the 3rd party software packages needed by RecSync-ETCD system. The blue-colored components indicate the implementation of RecCeiver-ETCD described in this paper.

Start-up stage RecCeiver-ETCD tries to connect the ETCD server configured in the environmental variable. Failure to connect makes the process retry periodically every configured period. Upon the successful connection, the periodic connection checking process is started and clean-on-startup and clean-on-shutdown are scheduled as configured in the configuration file. The UDP Broadcaster Thread is initiated to send out the packet described in Fig. 1 every 30 minutes (configurable). The *ETCD Watching Thread* is started. If clean-on-startup is configured, all the channels in ChannelFinder are set to PV status property “Inactive”. Then, the start-up process requests all the key-value sets from ETCD and pushes those to Elasticsearch database via ChannelFinder.

Watch and update stage The main purpose in this stage is to update the changes in the ETCD key-value store to ChannelFinder. The *ETCD Watching Thread* watches the changes with keys having the prefix “/epics/ca” and stores them in a shared list. Each insertion, update, and deletion change in ETCD is regarded as a single entry in the shared list. Each entry is processed sequentially. For example, the deletion of a PV happens right after the insertion of the same PV. The insertion update is processed first, then deleted afterward resulting that the PV’s final PV status property becomes “Inactive”.

Shutdown stage Once the shutdown signal is received, all processes stop. Then, entries remaining in the shared list are updated to ChannelFinder. If clean-on-stop is configured, all the channels in ChannelFinder are set PV status property “Inactive” before the shutdown.

OUTLOOK

In this paper only described is the usage of RecCeiver-ETCD with ChannelFinder. However, RecCeiver-ETCD can easily be extended for the other applications which provide REST APIs or Java APIs. Furthermore, RecSync-ETCD can be implemented with the authentication methods ETCD provides to ensure secure transactions as already mentioned in the previous work [8].

SUMMARY

RecCeiver-ETCD is a standalone application written in Java to build a bridge between the ETCD server and ChannelFinder. RecCeiver-ETCD provides fail-safe threads for ETCD connection, ChannelFinder connection, and updating information so that retries any event on failure until it succeeds. Although RecCeiver-ETCD has been developed to communicate with ChannelFinder, its simple implementation makes it easy to adopt for other needs beyond ChannelFinder.

REFERENCES

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

[2] RecSync, `https://github.com/ChannelFinder/recsync`

[3] ChannelFinder, `https://github.com/ChannelFinder/ChannelFinder-SpringBoot`

[4] Elasticsearch, `https://www.elastic.co/elasticsearch`

[5] ChanneLFinder Python API, `https://github.com/ChannelFinder/pyCFClient`

[6] ETCD, `https://etcd.io`

[7] ChannelFinder Java API, `https://github.com/ChannelFinder/javaCFClient`

[8] T. Ashwarya, E. T. Berryman, and M. G. Konrad, "Rec-SyncETCD: A Fault-tolerant Service for EPICS PV Config-uration Data", in *Proc. ICALEPCS'19*, New York, NY, USA, Oct. 2019, pp. 714–718. `doi:10.18429/JACoW-ICALEPCS2019-TUBPL05`