Abstract

The EPICS Archiver Appliance was developed by a collaboration of SLAC, BNL, and MSU to allow for the archival of millions of PVs, mainly focusing on data retrieval performance. It offers the ability to cluster appliances and to scale by adding appliances to the cluster. Multiple stages and an inbuilt process to move data between stages facilitate the usage of faster storage and the ability to deduplicate data as it is moved. An HTML management interface and scriptable business logic significantly simplify administration. Well-defined customization hooks allow facilities to tailor the product to suit their requirements. Mechanisms to facilitate installation and migration have been developed. The system has been in production at SLAC for about 2 years now, at MSU for about 2 years and is heading towards a production deployment at BNL. At SLAC, the system has significantly reduced maintenance costs while enabling new functionality that was not possible before. This paper presents an overview of the system and shares some of our experience with deploying and managing it at our facilities.

Clustering

Many storage configurations are possible. This is a typical configuration.

- Short term store - The most recent 2-3 hours of data; typically a RAM disk.
- Medium term store - The most 2-3 days of data; 15k SAS drives.
- Long term store - The rest of the data.

At SLAC, this is bulk storage (with tape backups) that we rent; this is a GPFS filesystem located elsewhere and is mounted over NFS.

At MSU, this is a NetApp alliance with 2.8 TB of storage.

Each appliance has 4 processes; these are J2EE WAB files deployed on separate Tomcat containers.

- Engine - Establishes EPICS CA monitors and writes data into STS.
- ETL - Moves data from the STS to the MTS and from the MTS to the LTS.
- Retrieval - Stitches data from all the stores to satisfy data retrieval requests.
- Mgmt - Executes business logic, manages the others and holds runtime configuration state.

Each appliance has its own MySQL configuration database. Communication is mostly JSON/HTTP.

Appliance

Figure 1: Each appliance has multiple storage stages and multiple processes.

Figure 2: An installation is a cluster of appliances. Scale by adding appliances. To this strip, all appliances use the same LTS. However, you can also have each appliance use a separate LTS. The architecture is shared nothing.

Figure 3: Requests can be dispatched to any appliance.

Data Retrieval

Plugins for CS-Studio and ArchiveViewer + new bundled HTML5 viewer.

Support for processing the data during retrieval using post processes.

- Mean
- Median
- Standard deviation
- Others

Use these same operators to

- Pruncate as part of ETL - speed up response.
- Decimate as part of ETL - reduce data as ages.

Figure 5: HTML5 viewer.

Figure 6: Processing during retrieval using post processes.

Channel Archiver Integration

- Transparency proxy to the ChannelArchiver or XMREPC server
- No need to migrate data to new format
- However, MSU has developed utilities to do so if desired
- Import ChannelArchiver XML config files
- cascadeTools has two backend servers
- a2aproxy (Python)
- ardiuml
- Use as a switchable proxy between a ChannelArchiver and an EPICS Archiver Appliance

Figure 7: Response times and % of requests vs time spans of requests.

Requests are categorized in terms of their time span (midnight – starttime)

- Over 75% of requests are for < 1 day and complete within 100ms (average)
- Time spans of up to a week take an average of 250ms
- Time spans of a year with data reduction can complete in a couple of seconds
- At runtime, binning over 30 million samples into about 8000 samples.

Figure 8: HTML5 viewer (in development)

Figure 9: Many reports based on runtime static data; also accessible from Python.

Installation

System requirements - Server class machine + Recent versions of
- Linux
- Java
- Tomcat
- A browser
Other requirements
- MySQL – for config.
- Installation using
- Puppet modules – makes installation a breeze
- Some scripts
- Quickstart – Quick setup to evaluate

Table 1: Production facilities; note NSEL uses only 1 appliance for redundancy

<table>
<thead>
<tr>
<th>Name</th>
<th>Lab</th>
<th># PVs</th>
<th>GB per day</th>
<th>Years</th>
<th>appliance</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCLS</td>
<td>SLAC</td>
<td>200K</td>
<td>19</td>
<td>1.5</td>
<td>3</td>
</tr>
<tr>
<td>NSLS2</td>
<td>BNL</td>
<td>61K</td>
<td>42</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>NSCL</td>
<td>MSU</td>
<td>83K</td>
<td>1</td>
<td>2</td>
<td>2*</td>
</tr>
<tr>
<td>FACET</td>
<td>SLAC</td>
<td>34K</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>TestFac</td>
<td>SLAC</td>
<td>37K</td>
<td>1</td>
<td>2.5</td>
<td>1</td>
</tr>
</tbody>
</table>

Archive PV workflow

When users add a PV to the archiver, we
- Measure event/storage rate
- Get RTYP, NAME, ADEL, MDEL etc
- Call installation specific policy (Python script)
- Policy makes configuration decisions
- Use capacity planning to assign PV to an appliance in the cluster
- Start archiving

Figure 10: Web based UI for management

Possible to script the entire monitoring and administration of a cluster of appliances using Python scripts

Administration

All business logic is JSON/HTTP; both UI and scripting use these.
- Add/Modify/Delete PV
- Pause/Resume PV
- Redshard/Consolidate
- Many more

Figure 11: Many reports based on runtime static data; also accessible from Python

Serializaton

Configurations are on a per PV basis. Each PV’s configuration is a JSON object and includes a sequence of data stores.

"dataStore": {
  "dbHost": "localhost?name=PARTITION_YEAR...",
  "dbUser": "root",
  "dbPassword": "password",
  "dbName": "testDB"
}

"dbHost": "localhost?name=PARTITION_YEAR...",
"dbUser": "root",
"dbPassword": "password",
"dbName": "testDB"

Listing! Each PV’s config includes a list of data stores.

Each datastore is handled by a plugin, the default PlainPBStoragePlugin uses Google’s Protocol Buffers, which provides future proof serialization.

- Each EPICS V3 DHR type is mapped to a distinct PB message.
- V4 NTScalars/NTScalarArrays map to their V3 counterpart PB messages.
- All other V4 types map to a generic PB message.
- On average, a PB ScalarDouble consumes about 21 bytes

Data is stored in chunks; each chunk contains serialized PB messages
- One message per archive
- One message per line.
- Sorted by their record processing timestamps (guaranteed)

The chunk key is enough to identify the boundaries of the contained data; for example, EPICS/L0/L00/HEARTBEAT_2012_08_24_16

The PV Name
- The time partition of the chunk (partition boundaries are strictly enforced)

The PlainPBStoragePlugin uses NIO 2
- One file per chunk
- Use NIO2 to store in alternate key-value store

*Work is supported by the U.S. Department of Energy, Office of Science under Contract DE-AC02-76SF00515 for LCLS I and LCLS II.