EPICS Archiver Appliance

Murali Shankar, Luofeng Li*

SLAC National Accelerator Laboratory, Menlo Park, CA 94025, U. S. A.

Michael Davidsaver

Brookhaven National Lab, Upton, NY 11973, U. S. A.

Martin Konrad

Facility for Rare Isotope Beams, Michigan State University, East Lansing, MI 48824, USA.

ICALEPCS 2015
International Conference on Accelerator & Large Experimental Physics Control Systems
17-23 OCTOBER 2015 MELBOURNE AUSTRALIA

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 decimate 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.

Appliance

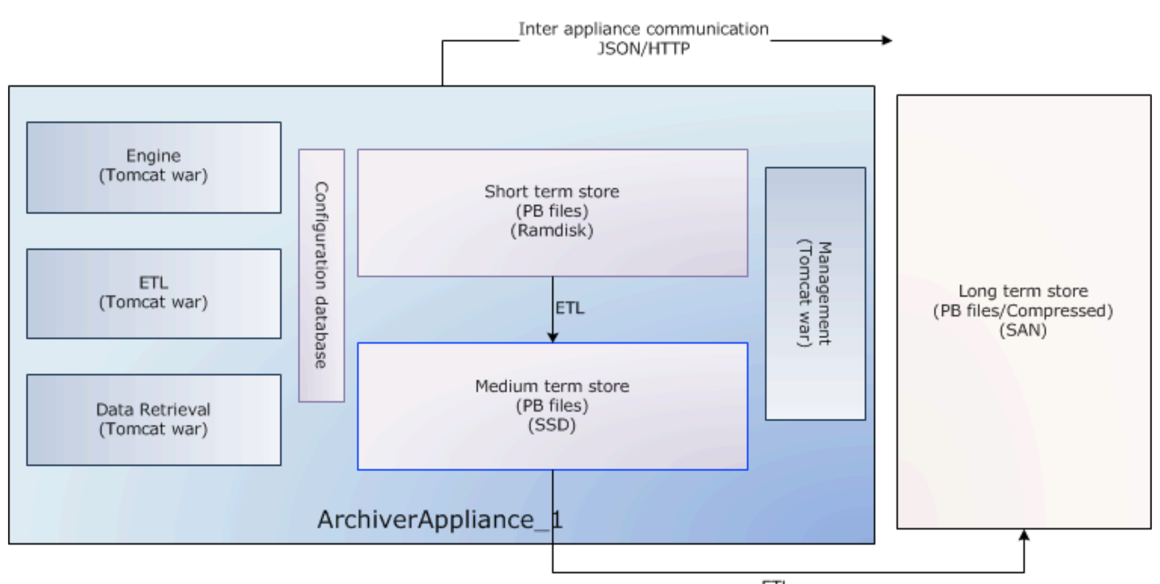

Figure 1: Each appliance has multiple storage stages and multiple processes

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 WAR 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.

Clustering

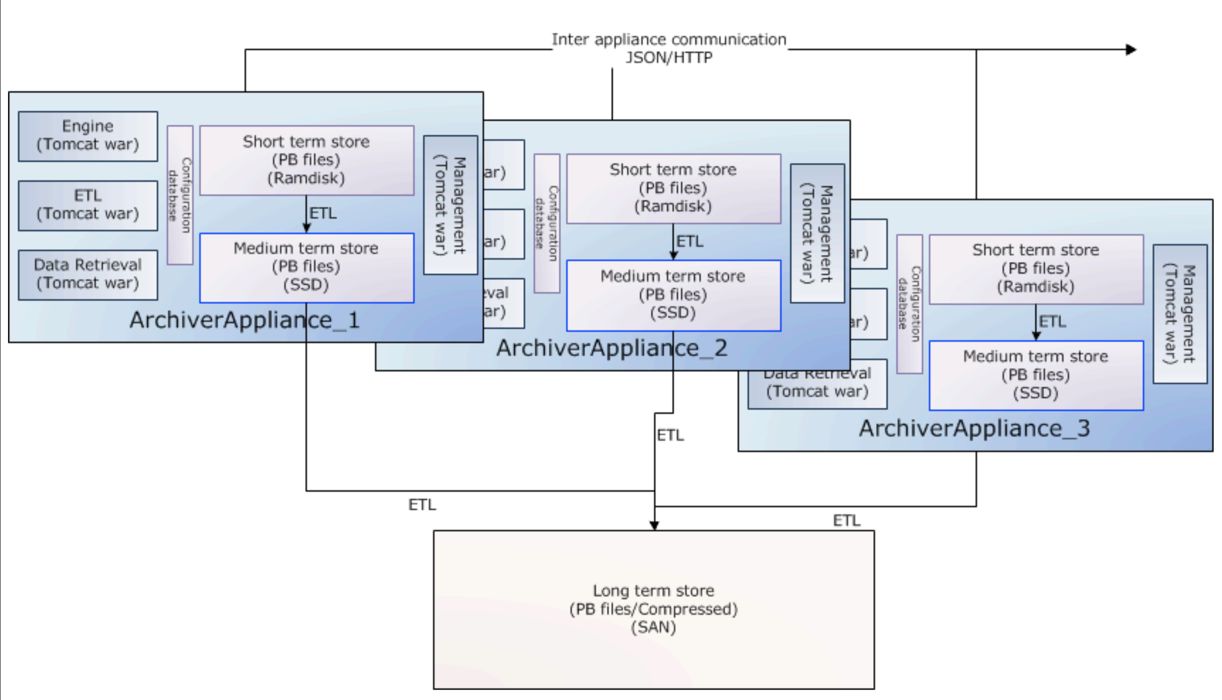

Figure 2: An installation is a cluster of appliances. Scale by adding appliances. In this setup, all appliances use the same LTS. However, you can also have each appliance use a separate LTS. The architecture is shared nothing.

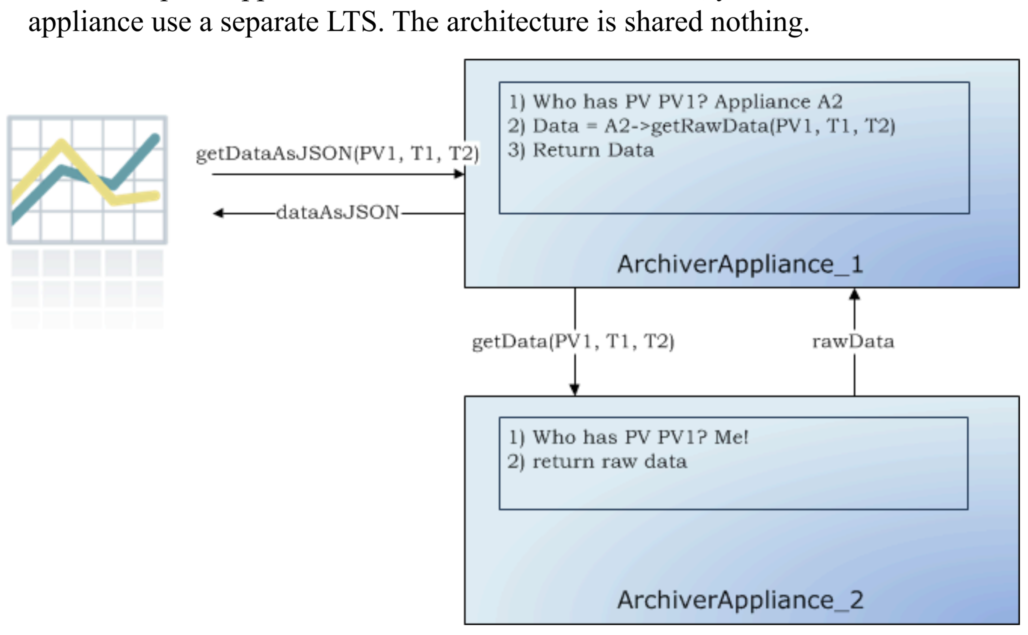

Figure 3: Requests can be dispatched to any appliance.

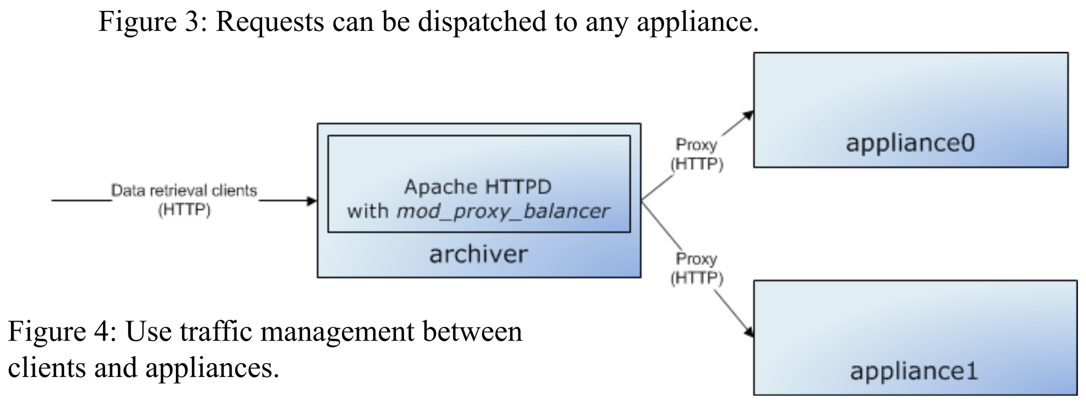

Figure 4: Use traffic management between clients and appliances.

Data Retrieval

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

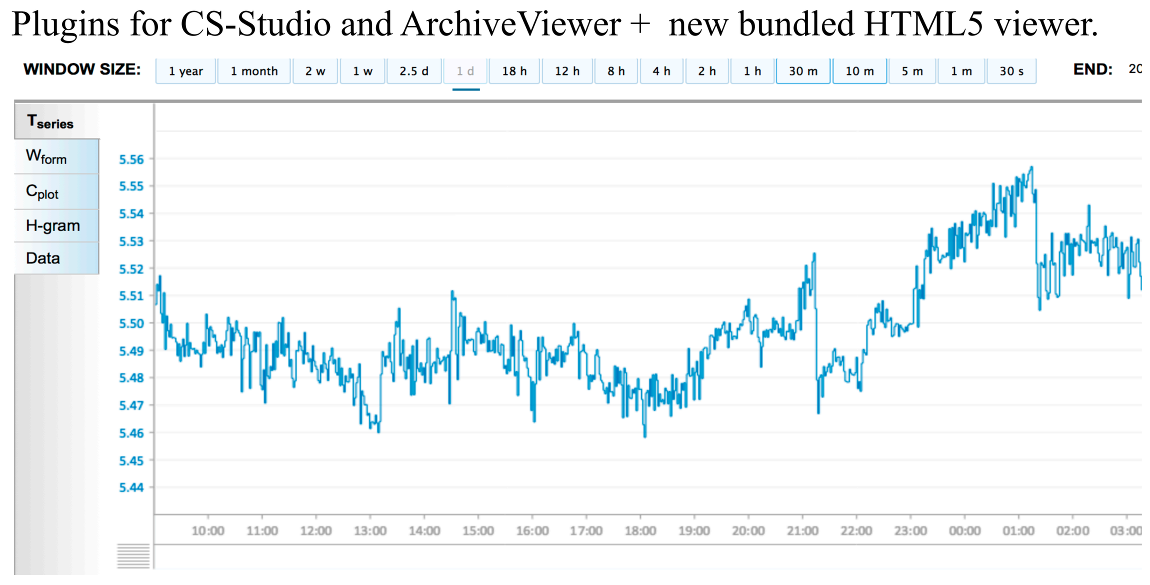

Figure 5: HTML5 viewer (in development)

Support for processing the data during retrieval using post processors.
- ✓ Mean
- ✓ Median
- ✓ Standard deviation
- ✓ Others

Use these same operators to
- ✓ Precompute as part of ETL – speed up response.
- ✓ Decimate as part of ETL – reduce data as it ages.

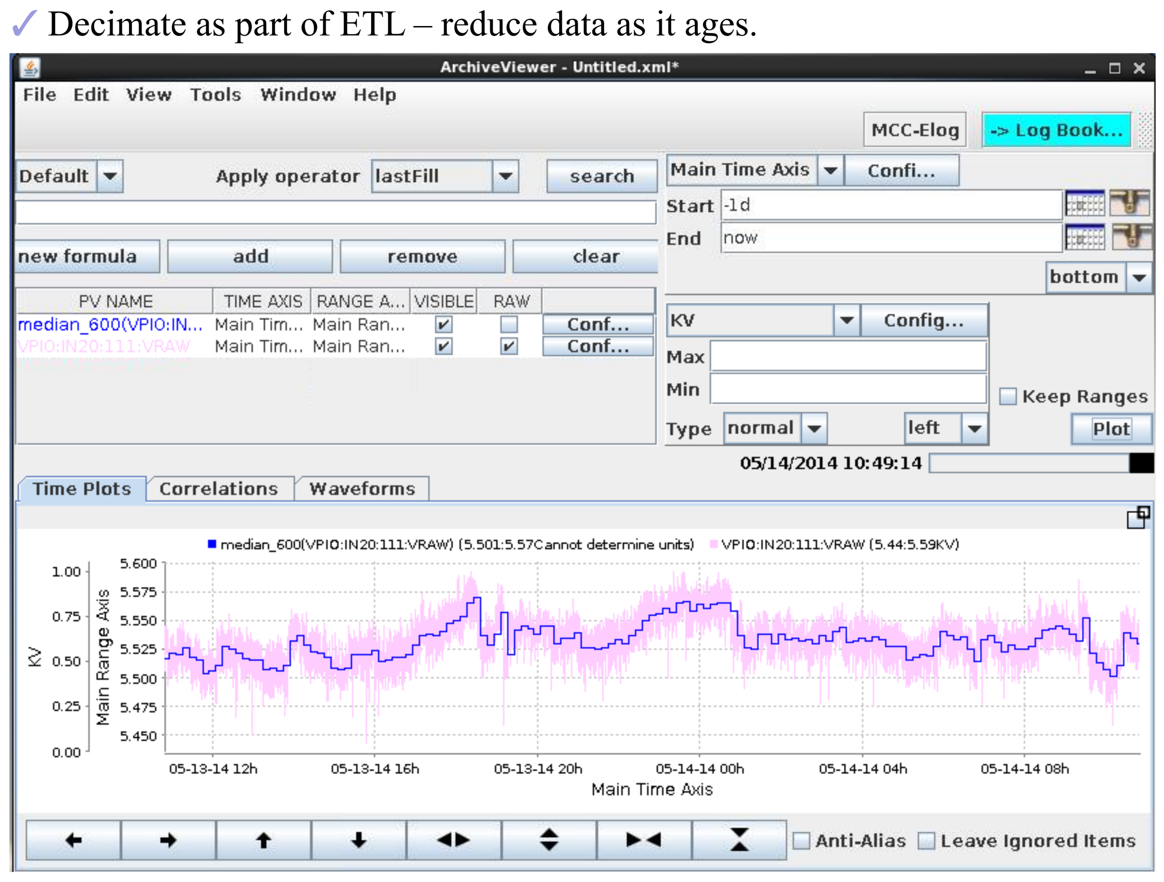

Figure 6: Processing data during retrieval; plot median and raw value on same chart

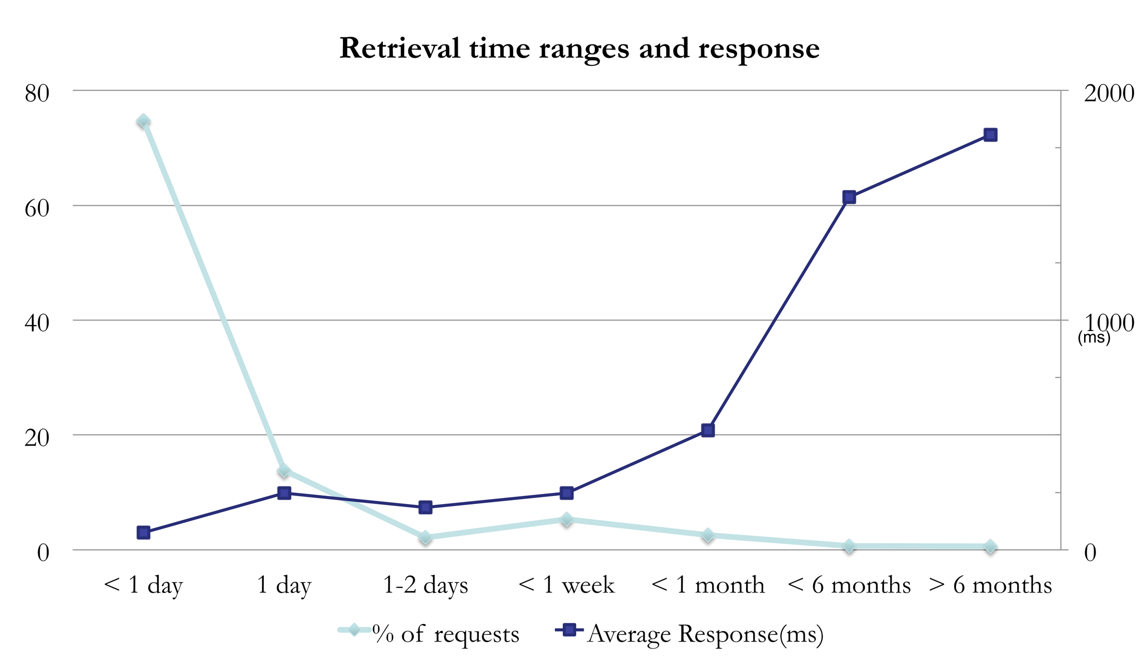

Figure 7: Response times and % of requests vs time span of requests

Requests are categorized in terms of their time span (endtime – 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.

Data retrieval over HTTP in multiple formats; URL has just pvName + times.
- ✓ JSON
- ✓ CSV
- ✓ MAT
- ✓ RAW
- ✓ TXT
- ✓ SVG
- ✓ Easy to add new MIME types.

Common client tools
- ✓ Python
- ✓ Matlab
- ✓ JMP
- ✓ Excel
- ✓ Others.

carchivetools – Python based command line tools
- ✓ Get data for a time range – this uses the RAW response.
- ✓ Search for matching PV's

Channel Archiver Integration

Transparently proxy the ChannelArchiver XMLRPC server
- ✓ No need to migrate data to new format
- ✓ However, MSU has developed utilities to do so if desired.

Import ChannelArchiver XML config files
carchivetools has two backend servers
- ✓ a2aproxy
- ✓ archmiddle

Use as a switchable proxy between a ChannelArchiver and an EPICS Archiver Appliance

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

Name	Lab	# PVs	GB per day	Years	#appliance
LCLS	SLAC	200K	19	1.5	3
NSLS2	BNL	61K	42	0.5	1
NSCL	MSU	83K	1	2	2*
FACET	SLAC	34K	1	2	1
TestFac	SLAC	37K	1	2.5	1

Table 1: Production facilities; note NSCL uses 2 appliances for redundancy

Archive PV workflow

When users add a PV to the archiver, we
- ✓ Measure event rate/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

Archive fields HIHI, LOLO as part of regular PV

Administration

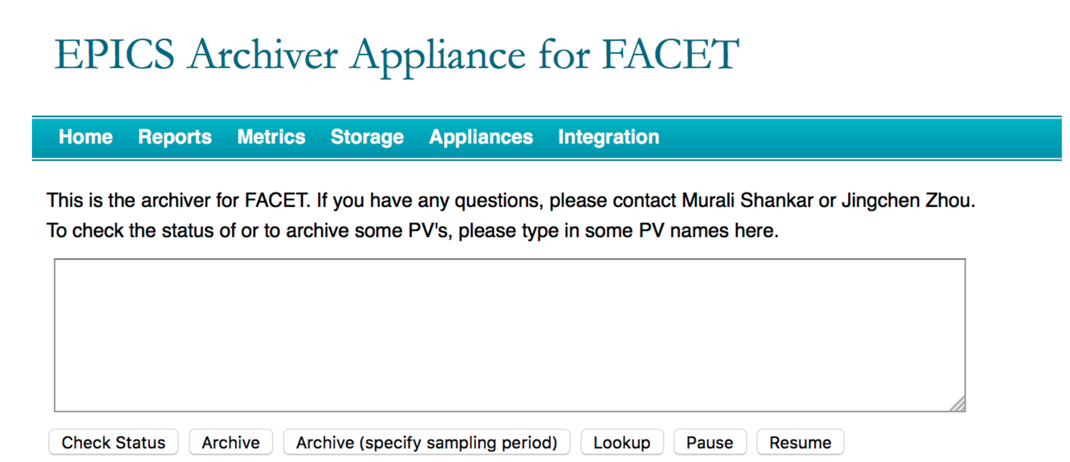

Figure 8: Web based UI for management

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

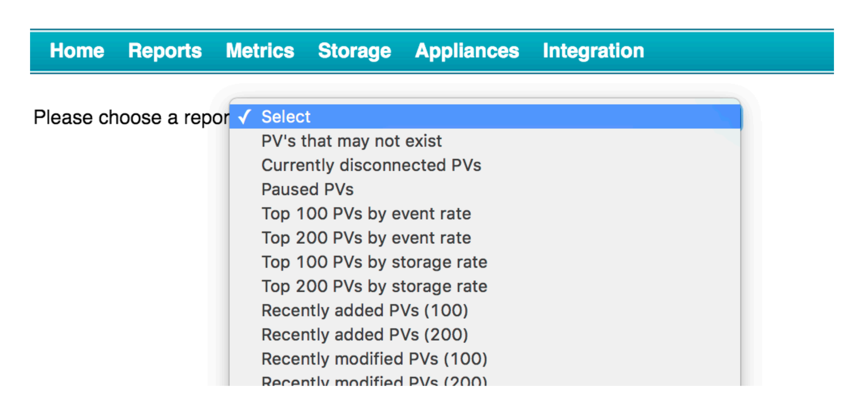

Figure 9: Many reports based on runtime+static data; also accessible from Python

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

Serialization

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

"dataStores": [
"pb://localhost?name=STS&...&partitionGranularity=PARTITION_HOUR...",
"pb://localhost?name=MTS&...&partitionGranularity=PARTITION_DAY...",
"pb://localhost?name=LTS&...&partitionGranularity=PARTITION_YEAR..."
],

Listing1: 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 DBR 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 contain serialized PB messages
- ✓ One message per archive sample
- ✓ One sample per line.
- ✓ Sorted by their record processing timestamps (guaranteed)

The chunk key is enough to identify the boundaries of the contained data; for example, *EPIOC/LI30/MP01/HEARTBEAT:2012_08_24_16*.pb.
- ✓ The *PV Name*
- ✓ The time partition of the chunk (partition boundaries are strictly enforced)

The PlainPBStoragePlugin uses NIO.2
- ✓ One file per chunk
- ✓ Use NIO.2 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.