Data exchange protocol for the INFN-LNS accelerators beamlines

G. Vecchio\textsuperscript{1}, L. Cosentino\textsuperscript{1}, S. Pulvirenti\textsuperscript{1}, E. Furia\textsuperscript{1}, S. Cavallaro\textsuperscript{1}, B. Diana\textsuperscript{1}

\textsuperscript{1} INFN Laboratori Nazionali del Sud, via S. Sofia 62, 95135 Catania

Abstract – This paper describes design and development of a data exchange protocol for the management of the INFN-LNS accelerators beamlines. We illustrate the data structure server adopted, Redis, and the hardware and software architecture of the whole system.

INTRODUCTION

This paper presents the software platform aiming to store, send and manage all the data coming from and to the field devices. The beamline is composed by different devices from different vendors and for this reason a standard protocol for data exchange is strongly required.

Compared to traditional data acquisition systems, we developed a new concept of control system and data acquisition framework, based on a data structures server which so far had never been used for supervisory control.

We have chosen Redis \cite{1} as a highly scalable data store, shared by multiple and different processes and applications \cite{2}. This system easily allows cross-platform, cross-server and cross-application communication, using extremely lightweight libraries.

REDIS OVERVIEW

Redis is part of the NoSQL databases, efficient and powerful tools for storing and manipulating large quantities of data \cite{3}.

In particular, Redis is a data structures server supporting different kind of values, unlike traditional key-value stores that only associate string keys to string values. In Redis values are not limited to strings, but can also hold more complex data structures like lists, sets and hashes.

Redis stores everything in primary memory which is very fast in read and write of data. For this reason it is perfectly suited to store small information which needs to be accessed, modified and inserted at a very fast rate.

REDIS ARCHITECTURE

Redis architecture contains two main processes: Redis client and Redis server. Redis server is responsible for storing data in memory. It handles all kinds of management and forms the major part of architecture.

Clients are available for the most popular programming languages, and a command line tool is also available to directly send queries to the server.

Since Redis stores everything in RAM, depending on the user requirements it might be needed to put data on a non-volatile storage media. There are three ways to make the in-memory database persistent: RDB mechanism, AOF and SAVE command.

RDB makes a copy of all the data in memory and stores it in a permanent storage when certain criteria are met. AOF logs all the write operations received by the server and therefore can be used to reconstruct the entire dataset. Finally, SAVE command can force Redis server to create a RDB snapshot anytime using the client \cite{4}.

Redis Replication

Replication is a technique involving many computers to enable fault-tolerance and data accessibility. In a replication environment many computers share the same data with each other so that even if some computer goes down, all the data will be available.

Redis replication is very simple to use and to configure master-slave replication, that allows slave Redis servers to be exact copies of master servers (Fig. 1).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig1.png}
\caption{Redis replication}
\end{figure}

When a new slave is added to the environment, the master automatically synchronizes all data with the slave. In master-slave configurations the master should have persistence turned off, so that in case of a server crash a slave is usually promoted to master role. This is meant to prevent data loss, since setting up a brand new master would start with its own dataset.

Redis Clustering

Clustering is a technique by which data can be shared among many computers. The main advantage is that more data can be stored in a cluster because it’s a combination of computers. Redis Cluster provides a way to run a Redis installation where data is automatically shared across multiple Redis nodes, as shown in Fig. 2.
Redis Protocol

Redis clients communicate with the Redis server using a protocol called RESP (REdis Serialization Protocol), that is simple to implement, fast to parse and human readable.

RESP can serialize different data types like integers, strings, arrays. There is also a specific type for errors. Requests are sent from the client to the Redis server as arrays of strings representing the arguments of the command to execute. Redis replies with a command-specific data type.

RESP is binary-safe and does not require processing of bulk data transferred from one process to another, because it uses prefixed-length to transfer bulk data.

A client connects to a Redis server creating a TCP connection to the port 6379. While RESP is technically non-TCP specific, in the context of Redis the protocol is only used with TCP connections (or equivalent stream oriented connections like Unix sockets).

LNS Beamline: Redis configuration

The keys used on our Redis server have this format:

StationName_DeviceName:Attribute|Type

This kind of attribute separation ensures a more efficient use of network bandwidth, in addition to a logical separation of attributes for each device.

Moreover, every station has a variable that contains a timestamp for all of the values associated to the keys, useful for the graphic representation of the data.

If a field-level computer hangs up or loses connectivity with the server, its variables are deleted from Redis after a certain timeout, so that every client trying to read that data can properly handle this situation.

We tested our system with a polling time of 100 ms and a data exchange of tens of kiloBytes, and we have observed an excellent management of CPU and memory.

CONCLUSIONS

In this paper we presented a data exchange protocol for the management of the INFN-LNS accelerators beamlines.

We explored a new way to exchange data between devices and computers inside special environments, where high performance and reliability are needed.

After some early tests, we can say that the whole system is very efficient with a fast data exchange and Redis is an excellent solution even for industrial applications.

REFERENCES