USING EPICS REDUNDANT IOC IN UNIX ENVIRONMENT*

A. Kazakov**, SOKENDAI/KEK, Japan
K. Furukawa, KEK, Japan
M. Clausen, G. Liu, DESY, Germany

Abstract

A redundant Experimental Physics and Industrial Control System (EPICS) input output controller (IOC) is being actively developed at Deutsches Elektronen-Synchrotron (DESY) in order to achieve high availability of control software. The current development focuses on a VME vxWorks environment for cryogenic controls. However, many scientific facilities use PC-architecture and Unix-like systems such as Linux, Solaris, and Darwin. These facilities require high availability and redundancy. Therefore, this paper describes the implementation of an EPICS redundant IOC in a PC-based environment with Linux or other Unix-like EPICS-supported OSs. This is achieved by porting a redundancy monitor task (RMT) and continuous control cxecutive (CCE). The CCE is used to synchronize two RSRV-based IOC servers. The RMT monitors other parts of the system, maintains the connection with the partner, and decides when to fail-over. It is rather independent and may be used in a wide range of applications. It has been successfully employed to add redundancy to a CA gateway.

INTRODUCTION

An EPICS redundant IOC was originally developed at DESY. Two majorfields of application were defined:

1. Redundancy for cryogenic plants. In this case, the failure may be caused by malfunctioning hardware such as power supplies or fans. An automatic fail-over mechanism should guarantee system stability. However, over the years it was sometimes necessary to manually switch between the main and backup processors due to maintenance work during the runtime period (which is usually one year or more). It might be useful for a software update. While the current commercial system used at DESY allows online updates of the database, EPICS does not allow addition or deletion of records and databases during operation.

2.Redundancy for controllers in the XFEL tunnel. Although the main origin of switch-over in the first case would be a manual action, it is expected to occur automatically in the XFEL tunnel. Due to high radiation, damage to the CPU and memory is highly possible. The software update is not very important because of more frequent maintenance days when this operation may be performed.

By the design draft one major goal was set: **Any redundant implementation must make the system more reliable than the nonredundant one. Precaution must be taken especially for the detection of errors that shall initiate the fail-over. This operation should only be activated if there is no doubt that maintaining the actual mastership definitely causes more damage to the controlled system than an automatic fail-over**. The fail-over time in any case was defined to be more than several seconds and less than 15 s. The final implementation could switch in less than 2 seconds.

Originally, the project was intended to support only vxWorks and the written code was very specific to it. However, later it was observed that support for other operating system is desirable. Here at KEK we use a software-IOC on Linux which functions as a “gateway” from an old control system to the EPICS-environment. In addition, for the ILC project ATCA-based systems under Linux control will be used. Redundant IOCs are highly desirable for this project. Thus, the redundant IOC has been ported to EPICS libCom/osi; this implies that the current implementation should work on any EPICS-supported OSs.

HARDWARE ARCHITECTURE

The hardware architecture consists of two redundant IOCs controlling a remote I/O via shared media such as the Ethernet or MIL1553. The redundant pair shares two network connections for monitoring the state of health of their counterpart, where the private network connection is used to synchronize the backup to the primary and the global network is used to communicate data from the primary to any other network clients requiring the data.

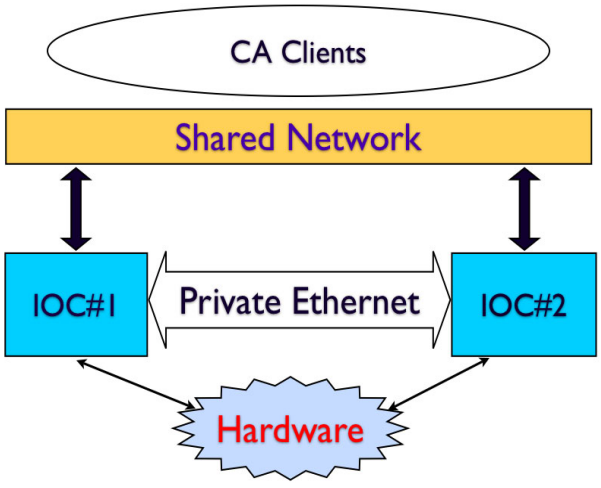

Figure 1: Hardware Architecture

SOFTWARE COMPONENTS

The current design contains three major parts: RMT (redundancy monitoring task), CCE (continuous control executive) and SNL (state notation language) executive.

The latter two are responsible for the synchronization of process variables (PVs) and internal states. The RMT is the core of the redundant system. It establishes and maintains the connection with the partner, controls other parts of the system and decides when to fail-over. All other components that have to be controlled by the RMT share the same software interface (which is defined in a header file rmtDrvIf.h). This interface defines the following functions:

1. start: Get access to the IO and start processing.
2. stop: Do not access the IO and stop processing.
3. testIO: Initiates a procedure to test access to the IO.
4. getStatus: Get status of the driver.
5. shutdown: This function is called before the IOC is rebooted. It terminates transient activities, deactivates interrupt sources and stops all driver tasks.
6. getUpdate: This routine tells the component to get an update from the redundant IOC. It is normally called by the RMT on the inactive IOC.
7. startUpdate: This routine tells the component to start updating data from the redundant IOC (monitoring). It will first read all fields (depending on the mode) from the redundant partner. It is normally called by the RMT on the inactive IOC.
8. stopUpdate: This routine tells the component to stop updating data from the redundant counterpart. This routine is normally called by the RMT on the inactive IOC.

The RMT can call these functions using an entry table that is transferred from the component to the RMT during initialization. The component calls the rmtRegisterDriver() function to register itself in the RMT and transfer the entry table.

getInfoCCE and the SNL executive are two such RMT-controlled components and they implement the RMT-driver interface mentioned above. Other components may be IO drivers, or any other piece of software. For example, the RSRV server in redundant IOC implementation is one of the RMT-controlled components and it implements the RMT-driver interface. Some of the interface functions may be unimplemented (set to NULL in the entry table) depending on the nature of the IO-driver and the tasks it performs.

PORTING

Originally, all the developments waere done only for vxWorks. The resulting code was not usable on any other OS. However, the people at KEK were interested in using a redundant IOC on Linux machines for a LINAC control system. Moreover, at DESY there was a demand for a redundant CA gateway. It appeared that the RMT has all the functionality needed for the solution. However, CA gateway runs on Linux (or other Unix-like OS), and the RMT was available only on vxWorks. Thus, it was decided to port the redundant IOC to Linux.

Porting process

All vxWorks-specific function calls were replaced with the EPICS *libCom/OSI (Operating System Independent library)*. In order to accomplish this task the following new OSI functions were introduced:

- **epicsMutex.h:** epicsMutexLockWithTimeout(id, tmo)
- **epicsMutex.h:** epicsMutexOsdLockWithTimeout(id, tmo)
- **epicsThread.h:** epicsThreadDelete(id)
- **epicsTime.h:** epicsTimeGetTicks ()

Further, some modification of the EPICS base were done to implement CCE hooks and the RMT-driver interface to the CA server (RSRV), and in database scan tasks. These modifications were also made operating system independent. The following is the list of affected source files:

- **base/src/db/dbAccess.c**
- **base/src/db/dbScan.c**
- **base/src/dbStatic/dbBase.h**
- **base/src/dbStatic/dbLexRoutines.c**
- **base/src/rec/aiRecord.dbd**
- **base/src/rsrv/camsgtask.c**
- **base/src/rsrv/online_notify.c**
- **base/src/rsrv/cast_server.c**
- **base/src/rsrv/caservertask.c**

Please contact the authors for the patch files.

Results

The ported OSI version of the redundant IOC was successfully used on vxWorks, Linux, Mac OS X, and Solaris. Tests showed that the system synchronisation speed limit was around ~5000 records/second for 2 Linux machines with 3GHz P4 1core, 2x 100Mbit Ethernet cards; functioning solely as a redundant IOC.

REDUNDANT CA GATEWAY

One of the demands at DESY was to implement a redundant CA gateway. Besides, after RMT is ported to Linux it fits very well to help this task. CA gateways are being widely used in many facilities around the globe which utilize the EPICS. The CA gateway can be used to add a security layer or to divide a network into several segments in order to reduce the amount of broadcast traffic in each segment. However, when the CA gateway is utilized, it becomes the “single point of failure” and even if some or all IOCs are redundant in the case of gateway failure some clients and IOCs would be cut-off. Therefore, it seems essential to add redundancy to applications such as the CA gateway.

By the nature of the gateway task, it has no internal status needed to be synchronized, and thus, it is desirable to use both redundant gateway at the same time (load-balance them). It will reduce the load on each of them and increase the peak system throughput. Further, in the case of failure of one of them only half of all the clients will notice the problem and will reconnect. Hence the task is divided into two steps.

Redundancy without load-balancing

Redundancy without load-balancing was implemented by using the RMT as a stand-alone application, which runs separately from the CA gateway. It provides the benefit of not modifying the source of the gateway, but raises the problem of how to control the gateway. Therefore, the chosen solution was to use firewall rules to block replies from the Gateway process. Only in the case of load-balancing it is possible to block incoming search requests from CA clients. However, when we add load-balancing it will not work (*see next section for details)*.

In order to add and delete firewall rules, the so called "script-driver" was implemented as the RMT-driver. Upon becoming a master this driver makes a call to an external "start-script" (which may be any executable file), and upon becoming a slave, it calls an external "stop-script". This script driver may be used in other applications and may call any external application on a specified status change. In our case, stop-script adds a firewall rule that blocks replies from the gateway to the clients, while start-script removes this firewall rules, making it possible to send replies to the clients. Hence, at any given time there is only one replying gateway that replies to the clients. The other gateway is on standby. This scheme worked well and hence we proceeded to the next step.

Load-balancing

This task appeared to be more tricky. Before we decided to use RMT as a standalone application, in order to reduce impact on the original gateway. However, to introduce load-balancing the RMT needs to have more control over the gateway. A load-balancing CA gateway has to be informed of the current status of its partner. I our approach of having them as separate processes it was decided to use the Unix signals mechanism to inform the gateway process about the status change. Some modification was done to the gateway source, which allows the reception of two signals. One of them signifies that the partner has become "alive", and the other signifies that the partner has become "dead".

The logic of a load-balancing gateway is simple: when the partner is alive, every other reply from the gateway includes its partner's IP address (this is the same functionality of the CA which is used in the CA directory service). When the partner is dead, the Gateway replies normally. Hence, both the gateways perform the same function if they are alive at the same time. Upon receiving a PV search request from a CA client they perform a search on the IOC network; if the search is a success, they create a "virtual-circuit connection" to the corresponding IOC. At this stage, the CA gateway adds this PV to the list of known PVs. Subsequently, a reply to the client is sent (containing either the partner's IP address or it's own IP address). However, on the slave side, the RMT's "script-driver" adds a firewall rule blocking replies. Therefore, the clients receive only one reply from the master. The master load-balances the further requests between itself and its partner. Therefore, after receiving a reply from the master gateway, the CA client establishes a connection to one of the two gateways. If we would block the incoming request from the client instead of the reply from the gateway, the client cannot connect to the slave gateway later because its list of know PVs would be empty and it will deny all incoming connections.

As a result of putting all this together, we achieve a load-balancing redundant CA gateway. Some minor changes to the CA Gateway source code are required. These changes include signal handling, load-balancing functionality and new command line options for configuring the IP address of the partner and signal numbers.

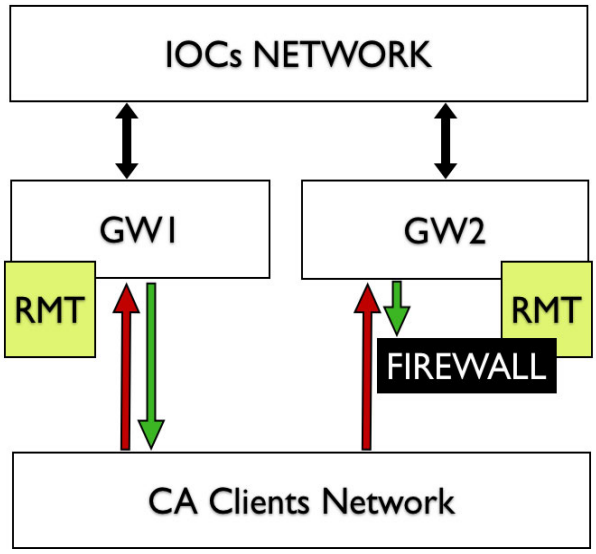

Figure 2: Redundant CA gateway

SUMMARY

Porting redundant IOC to Unix-environment allowed a much wider application of this system. Moreover, it was shown that the RMT functioning as the core of the redundant system can be utilized to add redundancy to other software.

*Work supported by SOKENDAI, Japan
**kazakov@gmail.com