30/10-29-93 SS(2)
SSCL-644

SSCL-644
September 1993
Distribution Category: 400

M. Botlo
M. Jagielski
A. Romero

EPICS Performance Evaluation

**Superconducting Super Collider
Laboratory**

Disclaimer Notice

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government or any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Superconducting Super Collider Laboratory is an equal opportunity employer.

EPICS Performance Evaluation

M. Botlo, M. Jagielski, and A. Romero

Superconducting Super Collider Laboratory*
2550 Beckleymeade Ave.
Dallas, TX 75237

September 1993

*Operated by the Universities Research Association, Inc., for the U.S. Department of Energy under Contract No. DE-AC35-89ER40486.

MASTER

2/1

EPICS Performance Evaluation

M. Botlo, M. Jagielski, and A. Romero

Abstract

We report on the software architecture, some CPU and memory issues, and the performance of the Experimental Physics and Industrial Control System (EPICS). Specifically, we subject each EPICS software layer to a series of tests and extract quantitative results that should be useful to system architects planning to use EPICS for control applications.

1.0 INTRODUCTION

Consisting of six accelerators and several planned experiments, the Superconducting Super Collider (SSC)¹ has a wide variety of needs for control, monitoring, and data acquisition systems that collect information from different sources and present it to operators in comprehensive form. It is generally hoped that a single system can accommodate all requirements that the different applications might have. A prime candidate among several packages on the market is EPICS, the Experimental Physics and Industrial Control System.² In this paper we measure the performance of the individual building blocks that make up EPICS as it exists today. The goal is to give aid in the design cycle of those SSC components that will use EPICS.

EPICS can be characterized as a set of software development tools used to implement real-time distributed data acquisition systems. User interfaces, alarm handlers, and custom applications run on a Unix non-real time kernel (at present SunOS 4.1.2). The real-time single board computers (SBC) are known as the input-output channels (IOC). The SBC used in the study was the MVME167 (68040) from Motorola.³ As of June 1993, the real-time kernel supported by EPICS is VxWorks versions 5.0.2b and 5.1.⁴ EPICS versions 3.7.1 and 3.10 are used for the evaluations presented in this paper.

2.0 EPICS SOFTWARE ORGANIZATION

Figure 1 displays the various software components used to configure and run EPICS applications. Central to the EPICS system is the notion of a database. The database consists of records called process variables (PV), and each process variable has many fields. During the development of a EPICS application, a database is created using a Unix software tool (Database Configuration Tool or DCT) on the workstation. Since the physical channels are distinguished by name, PV names must be unique across all IOCs that participate in a given application. During this step of readout description the developer links the logical records (PV) to the actual hardware devices that will be read during acquisition runs. A State Notation Language (SNL) can be used to describe sequential action to be executed during data acquisition within the framework of a run-time sequencer bundled with the EPICS distribution.

EPICS also provides a Unix software tool called EDD that is used to create the operator interfaces. Programming through pictures, EDD allows one to construct user interfaces that monitor and change the database PV fields. Other Unix tools provided by EPICS include an Archiver (AR), Archive Retriever (ARR), and Alarm Handler (ALH). There is also an interface for commercially available packages such as Mathematica and Wingz.

EPICS control systems are set up so that the IOC software and the database are loaded at boot time. The IOC must be rebooted before reloading the IOC. Figure 2 shows how the EPICS IOC software is functionally organized into four layers: database access, record support, device support, and device driver layer. The mechanism used to communicate between IOCs and Unix applications (for instance, the Operator Interface (OPT)) is called Channel Access (CA).

The channel access mechanism provides a "virtual" point-to-point link between a PV and a client application running in either the real-time or the Unix kernel. This virtual point-to-point link allows the application to examine and modify all fields in the EPICS records (PVs). In addition, through CA the EPICS database is used as a point of interaction between client applications. If the ALH, for example, finds an error condition, it will post a message in a PV of type string at the IOC. The operator console is programmed to monitor the string process variables and will display the change.

Figure 1. EPICS software architecture.

Task Name	Priority	Service
<i>timeStamp</i>	32	<i>time stamp</i>
<i>callback</i>	40	<i>async IO drivers</i>
<i>scanEvent</i>	41	<i>record processing queue (event)</i>
<i>scanPeriod</i>	53	<i>record processing queue (0.1 s)</i>
<i>scanPeriod</i>	54	<i>record processing queue (0.2 s)</i>
<i>scanPeriod</i>	55	<i>record processing queue (0.5 s)</i>
<i>scanPeriod</i>	56	<i>record processing queue (1 s)</i>
<i>callback</i>	57	<i>async IO drivers</i>
<i>scanPeriod</i>	57	<i>record processing queue (2 s)</i>
<i>scanPeriod</i>	58	<i>record processing queue (5 s)</i>
<i>scanPeriod</i>	59	<i>record processing queue (10 s)</i>
<i>callback</i>	65	<i>async IO drivers</i>
<i>CA TCP</i>	181	<i>channel access client</i>
<i>CA UDP</i>	182	<i>channel access server</i>
<i>CA online</i>	183	<i>channel access</i>
<i>taskwd</i>	200	<i>watchdog timer</i>

Figure 2. Default IOC tasks and respective priorities. The smaller number indicates higher priority.

3.0 IOC MEMORY REQUIREMENTS AND PERFORMANCE

Table 1 shows memory requirements for the EPICS IOC layered software architecture. Enough memory must be available for the VxWorks kernel (280 kB in our setup), the four IOC software modules (462 kB in our setup), and the database. The database size will vary with the type and number of PVs; the first 256 bytes of all PV types have the same fields. Table 2 lists the 26 process variable types along with their respective size.

Table 1. Memory utilization for the IOC (does not include the database).

EPICS	Size (kB)	Comments
EPICS (R.3.7.1)	462	locCore, drvCore, recSup, devSup
locCore	48	Database support and channel access
recSup	47	Record support
devSup	40	Device support
drvCore	144	Can be reduced by excluding drivers
VxWorks 5.0.2b	280	Varies based on kernel configuration

Table 2. Memory utilization for each record (PV) type in the IOC database.

EPICS Record (R.3.7.1)	Size (bytes)	EPICS Record (R.3.7.1)	Size (bytes)
aiRecord	576	permissiveRecord	288
aoRecord	656	pidRecord	496
boRecord	544	pulseCounterRecord	392
biRecord	480	pulseDelayRecord	488
calcRecord	1360	pulseTrainRecord	440
compressRecord	400	selRecord	1184
eventRecord	424	stateRecord	296
fanoutRecord	600	steppermotorRecord	584
histogramRecord	480	stringInRecord	528
longinRecord	496	stringoutRecord	584
longoutRecord	552	subRecord	1176
mbbiRecord	800	timerRecord	552
mbboRecord	856	waveformRecord	456

The IOC database layer handles the scheduling of each Process Variable. Each PV may be queued by one of four mechanisms:

1. **Periodic:** Seven predefined queues are provided with an execution interval of 0.1, 0.2, 0.5, 1, 2, 5, and 10 s, respectively. The database developer may specify other time intervals as long as they exceed the system clock. The system clock is by default 60 Hz.
2. **Event:** 256 “soft” events are provided to schedule PVs in software-controlled asynchronous mode. When a task that takes part in a run posts an event number (range 1–255), all PVs that have “Event” as the scheduling mechanism and the matching number on the EVTN field will be scheduled once.
3. **IO Interrupt:** This scheduling mechanism is asynchronous. It involves an auxiliary device driver task that either polls for new data or is triggered by VME interrupts.
4. **Passive:** PVs are not scheduled if the scan mechanism is set to passive. Passive PVs can be scheduled indirectly through input or forward links. A link is a field on the PV that points to another process variable.

During initialization, the EPICS IOC spawns a total of 16 tasks that support the EPICS services:

1. Seven "scanPeriod" tasks to service the periodic queues.
2. One "scanEvent" task for the soft event queue.
3. Three tasks to support channel access services (CA TCP, CA UDP, and on-line).
4. Three callback tasks (low, medium, and high priority).
5. One timestamp task.
6. One watchdog task.

$$\text{TOTAL IOC TASKS} = (16 \text{ default tasks}) + (\text{user defined scan periodic tasks}) + (2 * \text{number of CA clients}) + (\text{number of asynchronous drivers})$$

All asynchronous device drivers are implemented as tasks. In addition, for each channel access client the IOC will start two tasks: a channel access message task (camsgtask) to service incoming requests, and a task to send packets back to the client.

As described earlier, channel access is the mechanism by which information is delivered and extracted out of the IOC. All fields in the EPICS records may be viewed or modified through CA. The clients running on remote Unix workstations, such as OPIs and the AR, are compiled and linked with a series of CA functions or a library. The interface to in-house or third-party products is accomplished using the same CA library. At the IOC, the CA mechanism uses the client-server model known as the concurrent server. A concurrent server spawns a child task to service each client application. The CA concurrent server name is "CA TCP", and the child task (one for each client) is "camsgtask". For each client a second task is started at the IOC, the "event_task". The event_task will send data and control information back to the client by consuming a buffer known as the client's "event queue" buffer. Figure 3 shows the steps involved in establishing the communication links between the IOC and a client application. The one-time setup time was measured in our lab to be in the order of 90 ms.

Figure 3. IOC task needed to set up services for a CA client application.

Figure 4 shows how the client's application buffer at the IOC is produced and consumed. Once the communication links are established, the client's buffer at the IOC is filled with control data (by the camsgtask) and PV data (by the task that schedules the PV).

Figure 4. IOC tasks needed to service a CA client application request.

For each PV scheduling mechanism there is a list of PVs and a task to service it. For a 10-Hz PV list, for example, a "scanPeriodic" task is created. Every 100 ms, the periodic task processes the PVs in its list. Processing a PV involves four software layers: database access, record layer, device support layer, and device driver. In order to understand how the various software components interact, we put together a case-study application: One analog input (ai) process variable retrieves data at 10 Hz from a XYCOM-566 DAC VME board.⁵ Figure 5 displays how the time is spent in the four software layers during record processing.

Figure 5. Time segments on various software layers during record processing.

Segment A is the time spent preprocessing in the database layer. It starts when the PV is scheduled for processing, and it ends when the record layer is called. Factors in A that will prevent further processing of the record include: if the record (PV) has been marked active (processing underway but not finished), if the value in the disable flag field (DISA) matches the disable value field (DISV), or if a record support routine (record layer) cannot be found for this process variable type. The presence of an input link on the PV will increase the duration of A. The time spent on A depends on the record type. In our test application time spent on A is:

Segment A(ai): 9.2 μ s.

Segment G ends processing in the database layer. The time in G is record-type-dependent. For the ai records we did not find any factors that influence G.

Segment G(ai): 0.6 μ s.

The database layer calls the record support layer. The record support layer is organized as a vector table. The vector table is known as the record support entry table (RSET). For each record type supported, a function pointer to an initialization routine, a read routine, and a write routine is kept in the vector table. In general, input record types (ai, bi, mbbi, etc.) have initialization and read routines, and output record types (ao, bo, mbbo, etc.) have initialization and write routines. One field in all PV types is the RECORD TYPE field. This information is used by the database layer to call the appropriate function in the record layer. Figure 6 is the vector table organization of the record layer.

	Init	Read	Write
ai	ai_init()	ai_read()	NULL
ao	ao_init()	NULL	ao_write()
bi	bi_init()	bi_read()	NULL
bo	bo_init()	NULL	bo_write()

Figure 6. Record Support Entry Table (RSET).

Segment B in Figure 5 denotes preprocessing in the record layer. It starts after arrival in the process() routine for the particular record type, and it ends just before the device support layer is called. A factor that will affect the duration of B is the use of the simulation fields.

Segment B(ai): 8.8 μ s.

Segment F is the post-processing in the record layer. Factors that affect the time in F include the presence of alarm conditions and the number of client applications requesting data from the fields in that record. The time stamp is set in F. Segment F is record-type-dependent. The results of our test for segment F are:

Segment F(ai), 0 CA client: (32.4 $\pm$ 2.5) μ s

Segment F(ai), 1 CA client: (91.1 $\pm$ 7.3) μ s

Segment F(ai), 2 CA clients: (134.2 $\pm$ 13.7) μ s.

Note that the execution times of F are non-deterministic. The number after a “ $\pm$ ” symbol is the one standard deviation fluctuation around the mean time value.

Multiple devices may be supported for a given record (PV) type. The device support layer is also organized as a vector table or device support entry table (DSET). For each device supported—for instance, a VME board—a pointer to an initialization, to a read, and to a write function is kept. One field in all PV types is the DEVICE TYPE field. This information is used by the record layer to call the appropriate function in the device support layer. Figure 7 gives the vector table organization of the device support layer.

	Init	Read	Write
dev1	dev1_init()	dev1_read()	NULL
dev2	dev2_init()	NULL	dev2_write()
dev3	dev3_init()	dev3_read()	NULL
dev4	dev4_init()	NULL	dev4_write()

Figure 7. Device Support Entry Table (DSET).

Segment C is the preprocessing in the device support layer. For the device type tested we find no time variations in C. The duration of C depends on the record type and device type. For our case-study application it is:

Segment C(ai,XY566DI): 0.3 μ s.

Segment E contains the post-processing in the device driver layer. A driver failure would prolong the time spent here (*i.e.*, raising alarms). The duration of E depends on the record type and device type. For our test example it is:

Segment E(ai, XY566DI): 1.9 μ s.

The actual work of read/write to front-end acquisition boards is performed by the driver layer. The driver layer contains regular device drivers adapted to the EPICS style. VME address assignment of Control and Status Registers (CSR), VME data buffers, and backplane interrupt lines are fixed and controlled by Central Authority. This arrangement limits hardware configurability at the front-end boards in exchange for ease of development at the back-end or user level (UI, archiver, alarm handler, database, *etc.*). The device driver used in the test application was a simple memory-mapped driver that reads one word (16 bits) from VME when a read request arrives. Segment D is the device driver. The device driver used (XY566DI) fetches a word from VME A24 space. During segment D, the data is gathered from the hardware device and is placed in the record field. Segment D depends on the device type. For the case-study application:

Segment D(XY566DI): 8.6 μ s.

Segments A through G represent the actual time of record processing. Additional time is needed to traverse and manage the list of PVs on a given scheduling list.

3.1 Time Stamp Latency

We define the time stamp latency as the time period that starts with the arrival of data from the hardware device (in segment D) and stops as soon as a time stamp is acquired for the PV (in segment F). The time stamp latency is not affected by the number of CA clients. The time stamp latency will vary with record type and device driver. In the case study we find it to be:

Time stamp latency (ai, XY566DI): 11.9 μ s.

3.2 Data Send Latency (DSL)

We define data send latency as the time starting at the arrival of data from the hardware device (in segment D) until the time the event_task returns from the sendto() routine. The sendto() routine copies the data packet (32 bytes for one ai value field) to the kernel TCP/IP driver, which in turn ships it via ethernet to Unix applications. Numerous factors affect the data send latency:

1. Number of PVs being processed in the same list or scheduling mechanism. The data send latency for the first ai increased as more PVs were processed in the same list:
DSL (1 ai PV):(604 $\pm$ 60) μ s DSL (4 ai PV):(1210 $\pm$ 51) μ s
DSL (2 ai PV):(847 $\pm$ 80) μ s DSL (5 ai PV):(1380 $\pm$ 183) μ s.
DSL (3 ai PV):(970 $\pm$ 115) μ s
2. The number of CA clients requesting data. The data send latency for successive clients has the previous clients' event_task latencies added in.
DSL (1 client, 1 ai PV): (628 $\pm$ 45) μ s
DSL (2 client, 1 ai PV): (658 $\pm$ 52) μ s (1st client)
(1153 $\pm$ 49) μ s (2nd client).
3. The number of scheduling mechanisms being used in the database. All tasks that process PVs run at a higher priority than the CA tasks in charge of sending data packets (event task).
4. Any factor that prolongs the execution of segments E, F, or G during PV processing. The reason is that PV processing tasks run at higher priority than any event_task sending data back to a client task.

3.3 Asynchronous Scheduling

A second record type (waveform) was evaluated using the Comet Card,⁶ EPICS 3.10, and VxWorks 5.1. The Comet waveform driver is asynchronous. The readout of data in asynchronous drivers is triggered externally. The process variables (PV) scheduling field (SCAN) is set to IO-INT. The device driver is notified of the arrival of new data via IRQ or by polling the status of the card. The Comet driver uses the latter mechanism.

Figure 8 shows the execution flow for EPICS asynchronous drivers.

Figure 8. Execution flow for EPICS asynchronous drivers.

The execution flow for EPICS asynchronous drivers is as follows:

1. External trigger arrives, card digitizes.
2. IOC is notified via IRQ INT or polling. The polling resolution is 32 ms.
3. Driver Layer: DoneTask wakes up and calls the device layer.
4. Device Layer: Callback routine transfers data to PV, followed by a call to the record layer.
5. Record Layer: Prepare the PV for next processing, followed by a call to the device layer.
6. Device Layer: Calls driver layer
7. Driver Layer: Arms hardware (clear and set registers). Exits to 6.
8. Device Layer: Exits to 5.
9. Record Layer: Places the data into "Event Queues" for CA clients (Segment F). If scan mechanism is IO-INT, schedules PV for next trigger, exits to 4.
10. Device Layer: Exits to 3.
11. Driver Layer: Exits Done task.

The following times were recorded in each software layer of EPICS:

Database Layer:

Segment A(wf): 9.0 μ s

Segment G(wf): 1.1 μ s

A and G are not called when the PV scan is INT-IO.

Record Layer:

Segment B(wf): 8.6 μ s

Segment B(wf): 2.4 μ s, when PV scan is IO-INT

Segment F(wf), 10 elements, 0 CA client: $(45.8 \pm 1.3) \mu\text{s}$
 Segment F(wf), 10 elements, 1 CA client: $(69.7 \pm 2.6) \mu\text{s}$
 Segment F(wf), 10 elements, 2 CA clients: $(90.5 \pm 1.7) \mu\text{s}$
 Segment F(wf), 100 elements, 0 CA client: $(45.8 \pm 1.4) \mu\text{s}$
 Segment F(wf), 100 elements, 1 CA client: $(66.7 \pm 2.3) \mu\text{s}$
 Segment F(wf), 100 elements, 2 CA clients: $(89.8 \pm 1.8) \mu\text{s}$.

Device Layer:

Segment Callback(wf,Comet): $11.6 \mu\text{s}$
 Data Transfer(wf,Comet): $(1.1 * \text{nelements}) \mu\text{s}$
 Segment C(wf, Comet): $(14.9 \pm 1.6) \mu\text{s}$
 Segment E(wf, Comet): $3.7 \mu\text{s}$.

Driver:

Segment D (Comet): $9.1 \mu\text{s}$.

Time Stamp:

Time Stamp Latency(wf,Comet): $39.5 \mu\text{s}$.

Data Send Latency:

DSL(1 client, 1 wf PV): $(541 \pm 65) \mu\text{s}$
 DSL(2 client, 1 wf PV): $(584 \pm 56) \mu\text{s}$ (1st client)
 $(1198 \pm 68) \mu\text{s}$ (2nd client).

The factors that affect the performance of an asynchronously scheduled PV include:

1. Time spent in the database layer.
2. The fact that the frequency of the scheduling is externally triggered.
3. Execution of the Done Task (interrupt or polling task), one per card.
4. Number of elements to transfer across VME.
5. Time spent arming the card for next trigger.
6. Other factors, previously discussed, that affect segments B, C, D, E, and F.

4.0 EPICS RECORD PROCESSING OVERHEAD (RPO) AND CHANNEL ACCESS (CA)

We define the record processing overhead as the time needed to process a record, excluding the device driver layer. The technique used to measure record processing overhead was indirect. In each test a single PV type was chosen, and a single scan mechanism was used. The DEVICE TYPE field in each PV was "Soft Channel". The soft channel device type exercises the EPICS software layers without talking to a real device driver. The first and last record in the queue was a subroutine record that generated VME activity. A VME tracer measured the time elapsed between processing of the first and last record in a given queue. The process variables have a PHASE field that allows one to prioritize (highest 0, lowest 100) the order in which PVs are scheduled within a given queue.

Figure 9 illustrates the database configuration used in the RPO measurements. The SCAN field was used to prioritize the order in which the PVs were scheduled: first a subroutine record to start timing; next, one or more ai records; finally, the subroutine to stop timing. The two subroutine records added 140 μ s to each measurement. Consequently, the results reported were corrected accordingly.

<i>Record (Field,)</i>	<i>Comment</i>
<i>sub_tstart(SCAN=1s,PHASE=0)</i>	<i>Start timing with VME tracer</i>
<i>ai1(SCAN=1s,PHASE=50,DTYPE=Soft,RTYPE=ai)</i>	
<i>ai2(SCAN=1s,PHASE=50,DTYPE=Soft,RTYPE=ai)</i>	
<i>ai3(SCAN=1s,PHASE=50,DTYPE=Soft,RTYPE=ai)</i>	
<i>⋮</i>	
<i>aiN(SCAN=1s,PHASE=50,DTYPE=Soft,RTYPE=ai)</i>	
<i>sub_tend(SCAN=1s,PHASE=100)</i>	<i>Stop timing with VME tracer</i>

Figure 9. Database configuration to measure record processing overhead.

The test was first executed using analog input (ai) process variables. As described earlier, there are numerous record types (PVs), so the test was done with bi (binary input), multibinary input (mbbi), and the waveform record. Most record types have a single value field, with the exception of waveforms. The number of elements in a waveform record is specified in the NELEM field of the PV. Increasing this number does not increase the record processing overhead. However, in the device driver layer the total record processing time will increase with the number of elements in the waveform record to account for the buffer copy. Table 3 lists the record processing overhead in μ s, for various record types as a function of the number of records. The average record processing overhead per PV scales well with the number of PVs:

RPO(ai): 61 μ s
RPO(bi): 51 μ s
RPO(mbbi): 52 μ s
RPO(wf): 52 μ s.

Table 3. EPICS record processing overhead for various record types. Time in μ s, scan queue running at 1 Hz, no channel access.

Number PV	ai	bi	mbbi	waveform (diff NELEM)				CPU Idle
				10	100	500	1000	
10	762	660	670	668	668	668	*	99.0
50	3174	2693	2728	2750	2750	2750	*	98.5
500	30576	25333	25989	27500	27500	27500	*	96.0
900	55480	46305	46960	46960	46960	46960	*	93.0
1000	*	*	*	*	*	*	*	----

The next test was to develop an application with one operator interface (channel access client) and one IOC. The database contained 100 ai records on a queue at 10 Hz. The first and last records were subroutine records that accessed VME for timing purposes. An additional subroutine record was added at the end of the queue to change the value field in all PVs. The last subroutine record provided a “fake” device driver by changing the value field in all 100 PVs, forcing the Channel Access mechanism to execute at 10 Hz. Channel access clients receive data only when the PV value changes. The last subroutine record was implemented using two calls provided by EPICS:

dbNameToAddr() Find the address of a process variable in memory
dbPutField() Change the value field of the process variable.

The last subroutine record had a loop over 100 elements to change all the PVs. It executed in 13.7 ms. It should be noted that this record was at the end of the 10-Hz queue and did not affect the RPO measurement. However, the CPU idle time was reduced by 14%.

Figure 10 shows the database configuration used in the test. Table 4 lists the RPO and CPU idle time measured. The numbers in the "corrected" column are the CPU idle times corrected by 14%. RPO and CA CPU utilization increase with the number of clients.

Using a network scope, we observed that the network (Ethernet) load was 40 kB per client when 100 ai PVs were changing values at 10 Hz. If we assume that 4 bytes of application data are required to represent the 12-bit values being read from the ai devices, the network overhead is $(36/40)*100 = 90\%$.

To observe the effect of CPU overload at the IOC, seven CA clients were started on three Sun workstations. Data was archived for 500 000 data points. We observed a 0.25% data loss with error messages at the IOC, but the channel access clients were not notified (*i.e.*, operator interface). In addition, it appears that the new data packets are discarded in favor of the ones already queued. When an eighth client was added to the application, the Archiver (one-channel access client) lost connection with the IOC.

In another test to study the behavior of CPU usage by channel access, we used one client (operator interface) monitoring the ai PV queued with 10 Hz. Again, the values of all variables were forced to change in each cycle. Table 5 gives the results of CPU utilization as the number of variables increased.

<i>Record (Field,)</i>	<i>Comment</i>
<i>sub_tstart(SCAN=0.1s,PHASE=0)</i>	<i>Start timing with VME tracer</i>
<i>ai1(SCAN=0.1s,PHASE=50,DTYPE=Soft,RTYPE=ai)</i>	
<i>ai2(SCAN=0.1s,PHASE=50,DTYPE=Soft,RTYPE=ai)</i>	
<i>ai3(SCAN=0.1s,PHASE=50,DTYPE=Soft,RTYPE=ai)</i>	
<i>⋮</i>	
<i>ai100(SCAN=1s,PHASE=50,DTYPE=Soft,RTYPE=ai)</i>	
<i>sub_tend(SCAN=0.1s,PHASE=100)</i>	<i>Stop timing with VME tracer</i>
<i>sub_fillai(SCAN=0.1,PHASE=100)</i>	<i>Fill all ai values (214ms)</i>

Figure 10. Database configuration, single periodic queue at 10 Hz, 100 ai PV plus a single subroutine record that fakes the device driver, changing the values of all 100 variables at 10 Hz.

Table 4. EPICS Record Processing Overhead (PRO) and CPU utilization for 100 ai PV at 10Hz. The third column is corrected to account for a subroutine used to change the value field on every iteration (approx. 14 ms).

	RPO (0.1 s)	Corrected CPU Idle (%)	Observed CPU Idle (%)
No CA Request	6.615 ms	92.3	78.3
One CA Client 100 PV	8.786 ms	82.0	68.0
Two CA Client 100 PV	10.342 ms	72.5	58.5
Three CA Client 100 PV	11.857 ms	62.0	48.0
Four CA Client 100 PV	13.101 ms	53.0	39.0

Table 5. Application with 100 ai PVs at 10 Hz in the IOC, and one operator interface monitoring N variables.

Number of Process Variables	CPU Idle (%)
10	79
20	78
30	77
40	76
50	75
100	68

5.0 DATA OVERHEAD

We briefly looked at the EPICS Data Presentation Overhead. We ran the AR (Archiver) connected to an IOC that processes 100 analog input records at a frequency of 10 Hz with the VAL fields changing synchronously. No other record fields were touched. The AR logged the data in a file, and we measured an increase in file size of 999 kB in 109 s, or 9.17 kB/s. Next, we compressed the file with the Unix compression utility (compress(1)) and found that the data set was reduced to 9% of its original size. From that we conclude that a different data representation in EPICS could, in certain cases, reduce storage requirements by a factor of 10.

6.0 EPICS IOC RESOURCE UTILIZATION

The total CPU usage by the EPICS IOC is:

$$\text{CPU} = \text{TRP} + \text{TCA} + \text{tKernel},$$

where TRP is the sum of the time spent processing all the records in the database;

TCA is the sum of the time spent by all the tasks that service clients' event queues, which include all the "camsgtasks" and "event_tasks"; and

tKernel is the time spent by the VxWorks kernel during task management.

The Total Record Processing time is the sum of the Record Processing (RP) time spent in each individual process variable. Based on our observations, the RP for each PV can be estimated:

$$\text{RP} = [\text{tDb}(\text{recType}) + \text{tRec}(\text{recType}, \text{caReq}) + \text{tDev}(\text{recType}, \text{devType}) + \text{tDriv}(\text{devType}, \text{nElem}) + \text{tFetch}] * \text{schFreq},$$

where tDb is the time used by the database layer (segments A and G). It depends on the record type. Another factor that increases tDb is the presence of input links.

tRec is the time used by the record layer (segments B and F). This time is a function of the record type and number of pending channel access requests for fields in the process variable. Other factors that increase tRec are the presence of alarm conditions and the size of the data being copied to the event_queue buffers (*i.e.*, number of elements in waveform).

tDev is the time used by the device support layer (segments C and E); it depends on the record type and device type.

tDriv is the time spent by the device driver layer (segment D); it depends on the device type and number of elements (typically one) to be read or written.

tFetch is the time spent by the EPICS task fetching or traversing the list of process variables to get the next record (in the order of 10–18 μ s).

schFreq is the frequency at which the process variable is scheduled for processing.

The Total Channel Access (TCA) time is the time used by the “CA UDP,” “CA TCP,” “CA online”, and the two tasks necessary per client application, which include one “camsgtask” and an “event_task.”

The TCA tends to fluctuate during a run, depending on the number of client tasks, number of outstanding requests for fields of the PV, the frequency with which the value fields change, the number of requests for control data, and the size of the data and control packets being sent to each client application.

The tKernel is the time spent by the operating system switching between tasks and is determined by the number of active tasks and the scheduling frequency during the interval being measured.⁷

7.0 CONCLUSIONS

EPICS allows prototyping and development of distributed monitoring and control applications that use VxWorks for the front-end processors and Unix (SunOS) for storage and operator interfaces. The amount of front-end processors needed is application-specific and can be predicted.

The time stamp latency varies with the record type, a fact that should be considered if the time stamp is to be used for algorithms other than ordering and sequencing of data packets.

The Data Send Latency is non-deterministic with a mean in the millisecond range. It depends mainly on the number of PVs and clients. This latency is independent of the carrier.

The current carrier, TCP/IP over Ethernet, is affected by network load.

The network overhead, defined as the ratio of useful data (timestamp and data value) to total data on the carrier, is 20%. The ratio is better (increases) with the use of array data (waveform). Under overload conditions the channel access mechanism discards newer packets first, with no notification to the client application. The data is sent to client applications only when they change above certain programmable thresholds.

The data compression rate achieved on archived data is 90%.

The data packet send mechanism runs at the lowest priority and may halt under overload conditions. Therefore, mission-critical corrective algorithms should run on the IOC.

The EPICS application layer has no mechanism to pass configuration data to device drivers. Every desired hardware functionality has to have its own device support software.

ACKNOWLEDGEMENTS

We used a CPU utilization program developed by Carl Kalbfleisch, and ported to VxWorks by Lee Miller, who also helped in preparing this document. Database developments were done by James Hayes. Device Driver utilization work was carried out by Adrian Stipe and Huan Nguyen.

REFERENCES

1. SSC Site Specific Conceptual Design, SSCL-SR-1056.
2. Los Alamos National Laboratory, *EPICS Users Manual*.
3. Motorola Inc., *MVME166/MVME167/MVME187 Single Board Computers Programmers Reference Guide*, October 1992.
4. Wind River Systems, *VxWorks Programmers Guide*.
5. XYCOM, *XVME-566 Manual*, May 1987.
6. OMNIBYTE Corp., *COMET Reference Manual*, April 1992.
7. M. Botlo *et al.*, "VxWorks 5.1 Benchmark Tests," SSCL-627, June 1993.

END

**DATE
FILMED**

12/13/93