Performance Comparison of EPICS IOC and MARTe in a Hard Real-Time Control Application

Antonio Barbalace, Gabriele Manduchi, A. Neto, G. De Tommasi, F. Sartori, and D. F. Valcarcel

***Abstract*—EPICS is used worldwide mostly for controlling accelerators and large experimental physics facilities. Although EPICS is well fit for the design and development of automation systems, which are typically VME or PLC-based systems, and for soft real-time systems, it may present several drawbacks when used to develop hard real-time systems/applications especially when general purpose operating systems as plain Linux are chosen. This is in particular true in fusion research devices typically employing several hard real-time systems, such as the magnetic control systems, that may require strict determinism, and high performance in terms of jitter and latency. Serious deterioration of important plasma parameters may happen otherwise, possibly leading to an abrupt termination of the plasma discharge. The MARTe framework has been recently developed to fulfill the demanding requirements for such real-time systems that are alike to run on general purpose operating systems, possibly integrated with the low-latency real-time preemption patches. MARTe has been adopted to develop a number of real-time systems in different Tokamaks. In this paper, we first summarize differences and similarities between EPICS IOC and MARTe. Then we report on a set of performance measurements executed on an x86 64 bit multicore machine running Linux with an IO control algorithm implemented in an EPICS IOC and in MARTe.**

***Index Terms*—EPICS, plasma control, real-time control, real-time Linux.**

I. Introduction

AT the beginning of 2009, the ITER Organization announced the choice of the open-source software EPICS [1] and of the leading Linux distribution Red Hat Enterprise Linux (RHEL) 64 bits as the baseline of the ITER plant control system [2]. EPICS is used worldwide for controlling accelerators and large experimental physics facilities and its most important components are as follows.

- The channel access (CA) layer, which allows exporting the content of process variables (PVs). CA clients can read, write, and monitor PVs exported by CA servers, based only on the knowledge of the PV names.
- The input/output controller (IOC), carrying out control and supervision, based on a user-defined configuration represented by a set of interconnected Records, each one performing input/output or control computation.

IOCs are well fit for the design and the development of automation systems, typically based on PLCs, where acquired signals, sampled at a rate in the order of 0.1–10 Hz, are then used to perform a set of monitoring operations. IOCs can also be used to implement feedback systems, where the outputs sent to the plant actuators depend on the current values measured by sensors. EPICS IOCs may, however, present some limitations when used to provide hard real-time control, especially when a general-purpose operating system such as Linux is adopted. Indeed, in a Tokamak machine, there are several real-time systems, such as magnetic field controllers, that require strict determinism and high performance in terms of jitter and latency, otherwise serious deterioration of important plasma parameters may occur, possibly leading to an abrupt termination of the plasma discharge.

Several solutions are currently adopted for the real-time control in fusion devices, mostly based on Linux, possibly including real-time extensions [3]–[6]. Some systems have been developed and are used for specific fusion devices, while others represent more general frameworks and have been used in different machines. The PCS system [7], for example, has been developed at General Atomics for active plasma control and has been reused in several other experiments such as KSTAR, EAST, and MAST.

Another emerging framework is MARTe [8], originally developed for the vertical stabilization at JET, and currently adopted to fulfill the demanding requirements of a number of control systems at JET and on other smaller Tokamaks. MARTe is a newcomer if compared with PCS but presents several modern concepts combining object oriented methodology and hard real-time techniques. Its main software components areas follows:

- BaseLib, a set of C++ classes that provide operating system abstraction (thread, IPCs, files and memory accounting and management) object de/serialization, reflection and introspection, garbage collection, and math support;
- real-time threads and generic application modules (GAMs), which allow defining components for input/output and computation that can be assigned to different real-time threads and connected to exchange data in memory.

Manuscript received May 27, 2010; revised April 08, 2011; accepted May 05, 2011. Date of publication October 13, 2011; date of current version December 14, 2011.

A. Barbalace and G. Manduchi are with Consorzio RFX, Associazione Euratom-ENEA sulla fusione, 35127 Padova, Italy (e-mail: antonio.barbalace@igi.cnr.it; gabriele.manduchi@igi.cnr.it).

G. De Tommasi is with the Associazione EURATOM-ENEA-CREATE, Università di Napoli Federico II, 80125 Napoli, Italy (e-mail: detommas@unina.it).

A. Neto and D. F. Valcarcel are with Associação EURATOM/IST, Instituto de Plasmas e Fusão Nuclear—Laboratório Associado, 1049-001 Lisboa, Portugal.

F. Sartori is with F4E CODAC, Barcelona 08019, Spain.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNS.2011.2167350

The paper presents a comparison of EPICS IOC and MARTe real-time threads for hard real-time applications. Section II will summarize differences and similarities between EPICS IOCs and MARTe real-time threads and will discuss their impact in the overall performance. Section III will introduce a case study, where both EPICS and MARTe are used to provide data movement in a real-time system. Section IV will then present and discuss the performance results for both systems.

In addition to the comparison between the two frameworks, the measurements presented in the paper are intended also to assess the applicability of Linux and its real-time patches to hard real-time control. Whereas, in the past, dedicated real-time operating systems were mostly used, nowadays Linux is gaining interest in this field. The latest 2.6 Linux kernels implement in fact several techniques to reduce the portion of uninterruptible code as well as optimized scheduling mechanisms in order to reduce system latency. Moreover, real-time patches further reduce uninterruptible code segments and provide finer granularity in system clock [9]. For this reason, even if both EPICS and MARTe can run on the real-time operating system VxWorks, we have considered in the paper only the Linux version.

II. Similarities and Differences Between EPICS and MARTe

Both EPICS IOC and MARTe present a modular architecture where components can be connected to carry out the required control. Components in EPICS IOC are represented by records, while in MARTe they are represented by GAMs. Both records and GAMs comprise a number of fields which can be connected to other records and GAMs, thus providing the required real-time communication, achieved in both systems via shared memory. New record types can be created by providing a set of routines and data structures adhering to a given schema. For every record, it is also possible to define a new device support, in addition to those already supported for that Record. In this case the interface of the record is not changed, but new mechanisms for interacting with the I/O devices or carrying out computation can be defined.

New component types in MARTe are instead created by defining and implementing a new C++ class inheriting from a set of MARTe support classes providing the functionality required to properly interact with the MARTe environment. The new GAM class will then override the initialization and process methods to achieve the specific functionality.

Both in MARTe and EPICS the current configuration, i.e., the set of component instances and the way they are interconnected, is defined in a text file which is parsed (in the make procedure for EPICS, in system initialization for MARTe) to produce the target application. Although very similar in their broad concepts, EPICS and MARTe present several differences in their implementation.

Whereas in EPICS configuration files, only records can be defined, in the MARTe configuration files it is also possible to specify instances of given generic C++ classes. A new C++ class instance can be specified (and therefore created in the initialization process), provided it inherits from a set of support classes importing the ability of parsing a textual description in their initialization. In this case, it is possible to extend the functionality of the target application in a more general way, adding, for example, functionality not associated with any component in the underlying data flow model (GAM or Record). As an example, the content of the exchanged data can be exported to a Web browser by declaring in the configuration the instantiation of a Web server class accessing the named fields of the other components defined in the system.

Another important difference between the two frameworks is the computation model. In MARTe, the flow of the model is under strict control of the user which specifies in the configuration the real-time threads of the system and, for every real-time thread, the exact sequence of GAM instances to be executed in a cycle of the real-time thread. Conversely, users have no direct control of the threads created during EPICS IOC execution. The number of threads depends, in fact, on the topology of the record model for that IOC. Records will be processed in the order specified by the model semantics, but thread assignment is under full control of the IOC engine, which will try to parallelize as far as possible record processing without infringing the temporal constraints defined in the IOC configuration. As a consequence, it is not, in general, possible to tune thread priorities in order to reduce jitter and latency, except when creating user threads, like in the user code in asynchronous records. Function $epicsThreadCreate()$ allows in fact declaring the priority of the thread providing correct configuration parameters at compile time. This, however, does not help in reducing jitter and latency because it is not possible in EPICS to control thread assignment to CPU cores. MARTe instead allows assigning real-time threads to specific CPU cores, being this a parameter in the MARTe configuration file.

A further difference, affecting hard real-time performance, can be the possibility in MARTe of executing GAMs in kernel space. Such feature, missing in EPICS is important for extremely high demanding systems in term of latency and jitter because it allows removing the latency introduced by user process context switches. This feature has not been further considered in this paper, concentrating only in user mode operation.

III. The Case Study

As we intended to measure the fingerprint of the two frameworks, we set up a sample real-time application handling the required data movement for a feedback control application, but not carrying out any additional computation. Basically, the sample application acquires four ADC channels, transfers the samples to a dummy computation component, and then transfers the single resulting channel (a copy of the first of the four channels) to a DAC module. By measuring the delay between the input waveform feeding, the ADC, and the DAC output with an oscilloscope, it is possible to measure the overall delay of the system. In our application, we have used the National Instrument NI 6255 board [10], able to perform both ADC and DAC conversions. The NI6255 module is configured to acquire four channels, and the conversion time for each channel is 1.4 μs per channel, as suggested in the device datasheet, and confirmed by our measurements. It is, in fact, possible to program the module to generate a digital output upon the occurrence of a wide variety of conditions, including the start of the AD conversion for each channel. The NI6255 module is mounted on a PXI rack, connected through a PCIe bus extender to a PC workstation (HP Compaq dc 7900 CMT, x86 Intel Core 2 Duo E7300@2.66 GHz with 3 MB of cache memory and 3 GB of RAM). The used Linux version is 2.6.29.6 RT patched (rt-24), where the bare kernel has been compiled with PREEMPT_RT option enabled and system clock set at 1 kHz. The time required for transferring each sample from the ADC FIFO into the computer memory has been computed using the time-stamp counter register of the processor for measuring the time required to read the sample from the FIFO, which turns to be approximately 1.3 μ s per sample. This time refers to the PCI readout, including crossing the internal PCI bridges of the PC. No DMA has been used in this test since our purpose was not to speed ADC readout, rather to accurately compare the performances of the two frameworks. In order to measure more precisely the fingerprint of the EPICS and MARTe, we developed a C program whose unique operation is to reads four ADC channels and to output the first one. Such an application has been optimized to the best of our effort so that the difference in latency between EPICS or MARTe implementation and the straight one can be considered a reliable measurement of the fingerprint for both frameworks. In order to optimize the reference application, its thread of execution has been statically assigned to a given CPU core (using $set\_affinity()$ system call), which has been shielded from (almost) all sources of interrupts, except the ADC interrupt. This has been achieved by defining some kernel parameters ($isolcpu$, $acpi\_irq\_nobalance$, $acpi = no\_irq$, $no\_irqdebug$, $hpet = disable$) and then, after the boot, by setting the affinities for various interrupt sources, using the utility $irqbalance$, so that they have been removed from the shielded core. The program is then executed at the highest priority. Finally, the priority of the kernel thread handling the ADC interrupt has been set as the highest among the other kernel threads (using the RT patches, interrupt service routines are executed in kernel thread context). Despite all the taken actions, a number of system tasks, mostly kernel threads, cannot be avoided on the shielded core. Their priority is however less than that assigned to the thread handling data I/O in our test program. This program represents, to the best of our efforts, the fastest way to achieve the sample control loop in user mode, the difference in performance between EPICS and MARTe and thus reveals the true fingerprint of the two frameworks. The graphical representations of the module organization for MARTe and EPICS are shown in Figs. 1 and 2, respectively.

In the MARTe configuration, the following modules are defined:

— Input GAM: when processed waits for ADC data availability and then reads the ADC samples;
— Output GAM: outputs the passed sample to the DAC register of the ADC module;
— Simple GAM: it is the computation template, here it simply copies the first input to the output;
— Web Statistic: not representing a GAM, exports data and accessory information to the Web. It is somewhat similar to the EPICS CA server ability provided by EPICS IOCs. It is worth noting that the actions done by this component during operation are limited to the copy of few variables in memory. The HTTP server exporting such information is typically executed on another core and therefore does not influence loop execution time.

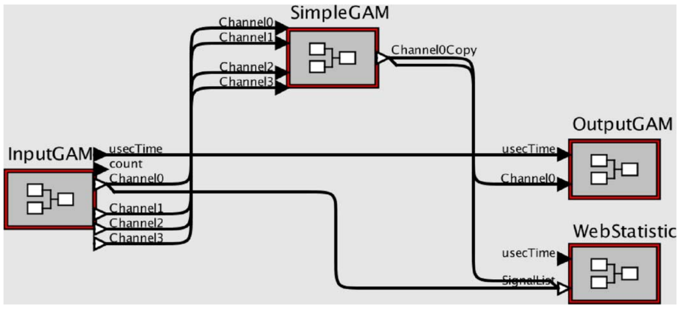

Fig. 1. MARTe configuration of the sample application.

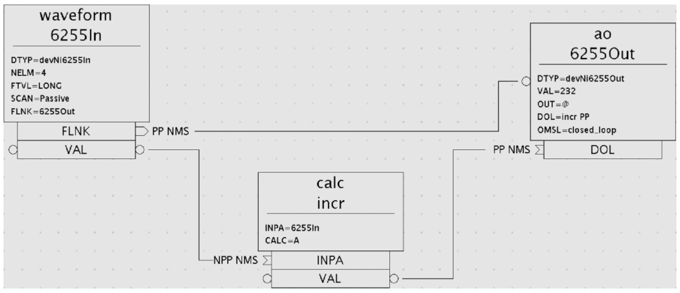

Fig. 2. EPICS configuration of the sample application.

The three GAMs are assigned to a MARTe RealTime thread, which executes them in sequence in an infinite loop. MARTe Real Time threads are C++ classes whose instances are associated with a POSIX Linux thread and whose configuration defined the thread characteristics, including CPU core assignment, scheduling policy, and priority. The MARTe RealTime thread has been configured to run on the shielded CPU core, setting its priority as the highest for that core.

In the EPICS configuration, a waveform record reads ADC samples. The associated device works asynchronously and the created thread will wait for the availability of ADC data, processing then the record. A forward link propagates the record processing to an analog output (AO) record, which first reads, via a process passive (PP) link from a calc record which, in turn, reads from the VAL field of the waveform record and returns the first sample. The device associated with the AO record will copy the passed value to the DAC register of the NI 6255 module. In order to reduce latency, we enabled the POSIX thread scheduling by setting EPICS configuration parameter

$$\mathrm{USE\_POSIX\_THREAD\_PRIORITY\_SCHEDULING=YES}$$

in the overall EPICS compiling configuration. The priority of the thread waiting for ADC data availability has been set to the highest in the system by passing the max priority level to EPICS function $epicsThreadCreate()$ and the priorities of the other system threads have been lowered to guarantee that all the

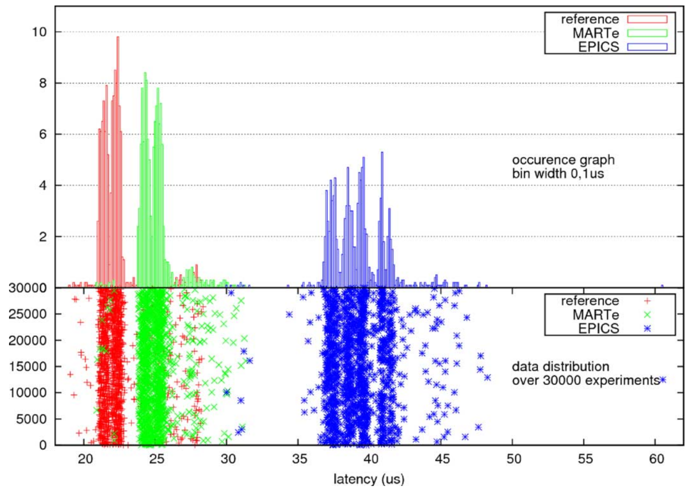

Fig. 3. Measure latencies in the reference program, MARTe and EPICS IOC.

TABLE I
MIN, MAX AND AVERAGE LATENCIES (μs)

	Reference Prog..	MARTe	EPICS
Min	19.0	20.9	30.0
Max	28.3	31.3	50.5
Avg	22.3	25.0	39.3

EPICS threads have a higher priority. It has not been possible to lock execution to a given shielded core since this functionality is not available in EPICS. Moreover, due to the larger number of threads created by EPICS, locking all them to the same core would have possibly led to a non optimal core assignment, because this would have prevented Linux to assign other cores for the concurrent execution of EPICS threads.

IV. RESULTS

In its execution, the EPICS IOC created 20 threads. The highest priority is assigned to the thread created by the Waveform device. Conversely, the number of threads created by the MARTe application is 9, and the highest priority thread corresponds to the MARTe RealTime thread handling the execution of the GAMs.

In a first test, we ran the system by providing an external sampling clock of 1 kHz. The latency of the three considered systems, i.e., the reference program, MARTe and EPICS IOC are shown in Fig. 3. In the lower part of the figure, the latency distribution is shown, considering 30 000 cycles, summarized by the histograms in the upper part. The minimum, maximum and average latencies are listed in Table I.

The added latency due to MARTe in respect of the reference program is on average 2.7 μs, and overall latency is bounded to 31.3 μs. The added latency due to EPICS is 17 μs and the latency distribution is smoother than in the other two systems. Observe that the maximum measured overall latency is much higher, around 50 μs.

Very similar results have been obtained at higher sampling speeds. As a consequence, the maximum cycle frequency for such an application, for which a negligible loss of data samples may occur would be 35.7, 32.2, and 20 kHz for the optimized application, MARTe and EPICS, respectively.

TABLE II
CPU LOAD

Sampl. Speed:	1kHz	2 kHz	5kHz	10kHz
Reference Prog:	2%	4%	8%	14%
MARTe:	4%	7%	14%	28%
EPICS:	4%	4%	14%	29%

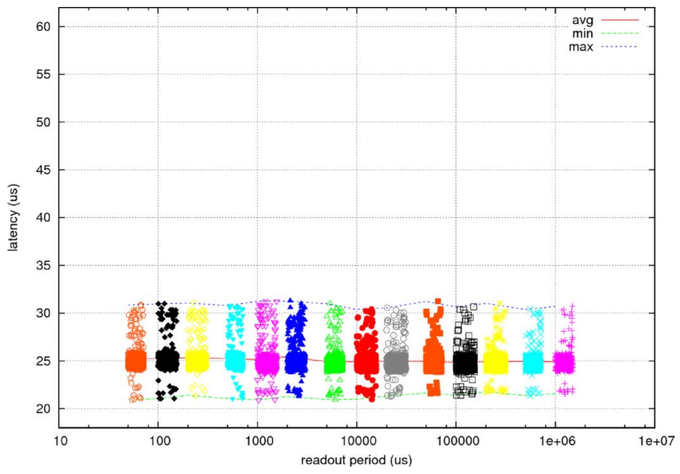

Fig. 4. MARTe latencies at different readout frequencies. The overall latency figures are unaffected by the external disturbances to the system.

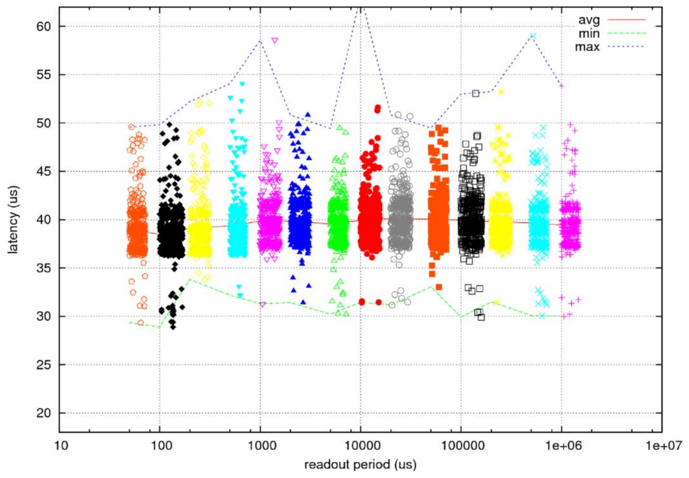

Fig. 5. EPICS latencies at different readout frequencies.

The CPU load for the three systems at various sampling speeds is shown in Table II. The additional load for the EPICS and MARTe frameworks is likely due to internal copies of data in memory.

In a final test, the effects of network data readout carried out by a remote data client has been considered. For MARTe, data readout consists of a HTTP request for data, while for EPICS is a CA get operation. In both cases, GBit Ethernet connection via switches is used. The latencies are shown in Figs. 4 and 5, for various read access rates, for MARTe and EPICS, respectively.

It can be shown that both systems handle very well external data access, with no interference in overall I/O latency.

V. Conclusion

By means of a simple case study, the real-time performances of MARTe and EPICS have been compared, using a highly optimized ad hoc program as reference. It is worth noting that the overall latencies do not represent the minimal obtainable latencies. Better performance can, in fact, be achieved using DMA for ADC data readout, but this is of no interest in this context, since the purpose of the work has been highlighting the fingerprint of the two systems.

MARTe provides a shorter and, above all, more deterministic latency, whose distribution shape is comparable to that of the optimized reference application. This is, most likely, the consequence of the fact that MARTe is more oriented towards hard real-time applications, using fewer threads and providing a direct control of the thread and scheduling parameters. However, for nonhighly demanding real-time applications, both in term of latency and system complexity, EPICS can well be used, despite the fact that it is mostly used for control systems where the cycle time is much larger. On the other side, for more complex control systems, the current lack in thread core assignment in EPICS may lead to further uncertainty in system latency.

Finally, the current Linux distribution including real-time patches appears to be well suited also for hard real-time applications, making Linux on multicore CPUs a strong candidate for real-time control in large systems.

Even if not covered in this paper, the possibility of defining MARTe GAMs running in kernel space would further reduce the system latency as done, for example, in the vertical stabilization system at JET [11].

References

[1] EPICS Home Page [Online]. Available: http://www.aps.anl.gov/epics/

[2] CODAC Core System Overview, 1.4 ed. 2010.

[3] K. Kurihara, J. B. Lister, D. A. Humphreys, J. R. Ferron, W. Treutterer, F. Sartori, R. Felton, S. Brémond, and P. Moreau, "Plasma control systems relevant to ITER and fusion power plants," *Fus. Eng. Des.*, vol. 83, no. 7–9, pp. 959–970, Dec. 2008.

[4] K. Behler, H. Blank, A. Buhler, R. Drube, H. Friedrich, K. Förster, K. Hallatschek, P. Heimann, F. Hertweck, J. Maier, R. Merkel, M.-G. Pacco-Düchs, G. Raupp, H. Reuter, U. Schneider-Maxon, R. Tisma, and M. Zilke, "Review of the ASDEX upgrade data acquisition environment—present operation and future requirements," *Fus. Eng. Des.*, vol. 43, no. 3–4, pp. 247–258, Jan. 1999.

[5] M. Lennholm, T. Budd, R. Felton, M. Gadeberg, A. Goodyear, F. Milani, and F. Sartori, "Plasma control at JET," *Fus. Eng. Des.*, vol. 48, no. 1–2, pp. 37–45, Aug. 2000.

[6] A. Luchetta and G. Manduchi, "General architecture, implementation and performance of the Digital Feedback Control in RFX," *IEEE Trans. Nucl. Sci.*, vol. 47, no. 2, pp. 186–191, Apr. 2000.

[7] B. G. Penaflor, J. R. Ferron, M. L. Walker, D. A. Humphreys, J. A. Leuer, D. A. Piglowski, R. D. Johnson, B. J. Xiao, S. H. Hahn, and D. A. Gates, "Worldwide collaborative efforts in plasma control software development," *Fus. Eng. Des.*, vol. 83, no. 2–3, pp. 176–180, Apr. 2008.

[8] A. C. Neto, F. Sartori, F. Piccolo, R. Vitelli, G. De Tommasi, L. Zabeo, A. Barbalace, H. Fernandes, D. F. Valcárcel, and A. J. N. Batista, "MARTe: A multiplatform real-time framework," *IEEE Trans. Nucl. Sci.*, vol. 57, no. 2, pp. 479–486, Apr. 2010.

[9] Linux Real Time Kernel Patches Home Page [Online]. Available: http://rtwiki.kernel.org

[10] NI 625x Specifications [Online]. Available: http://www.ni.com/pdf/manuals/371291h.pdf

[11] F. Sartori, A. Barbalace, A. J. N. Batista, T. Bellizio, P. Card, G. De Tommasi, P. Mc Cullen, A. Neto, F. Piccolo, R. Vitelli, and L. Zabeo, "The PCU JET plasma vertical stabilization control system," *Fus. Eng. Des.*, 2010, to be published.