A New Generation of Real-Time Systems in the JET Tokamak

D. Alves, , A.C. Neto, , D.F. Valcárcel, R. Felton, J.M. López, A. Barbalace, L. Boncagni, P. Card, G. De Tommasi, , A. Goodyear, S. Jachmich, P.J. Lomas, F. Maviglia, P. McCullen, A. Murari, M. Rainford, C. Reux, F. Rimini, F. Sartori, A.V. Stephen, J. Vega, R. Vitelli, L. Zabeo, K.-D. Zastrow and JET EFDA contributors

Abstract—**Recently a new recipe for developing and deploying real-time systems has become increasingly adopted in the JET tokamak. Powered by the advent of x86 multi-core technology and the reliability of the JET's well established Real-Time Data Network (RTDN) to handle all real-time I/O, an official Linux vanilla kernel has been demonstrated to be able to provide real-time performance to user-space applications that are required to meet stringent timing constraints. In particular, a careful rearrangement of the Interrupt ReQuests' (IRQs) affinities together with the kernel's CPU isolation mechanism allows to obtain either soft or hard real-time behavior depending on the synchronization mechanism adopted. Finally, the Multithreaded Application Real-Time executor (MARTe) framework is used for building applications particularly optimised for exploring multi-core architectures. In the past year, four new systems based on this philosophy have been installed and are now part of the JET's routine operation. The focus of the present work is on the configuration and interconnection of the ingredients that enable these new systems' real-time capability and on the impact that JET's distributed real-time architecture has on system engineering requirements, such as algorithm testing and plant commissioning. Details are given about the common real-time configuration and development path of these systems, followed by a brief description of each system together with results regarding their real-time performance. A cycle time jitter analysis of a user-space MARTe based application synchronising over a network is also presented. The goal is to compare its deterministic performance while running on a vanilla and on a Messaging Real time Grid (MRG) Linux kernel.**

I. Introduction

In the past, real-time requirements meant that a real-time Operating System (OS) had to be used. The ratio between resource availability and demand was very low, i.e. only single-cored CPUs were available to do all the work which included servicing hardware interrupts and scheduling tasks and application threads concurrently. Undoubtedly, and in this situation, priority is the essence driving deterministic behaviour. Linux itself already implements two real-time priority based schedulers, *fifo* and *round robin*, but lacks other features such as mechanisms for prioritising interrupt service requests. Over the years, thanks to the effort of a dedicated community, many *real-time friendly* features such as nearly full kernel preemption capability and high resolution timers have gradually made their way into the Linux kernel. More recently, the ever growing number of CPUs/Cores in x86 machines has dramatically increased the ratio between resource availability and demand thus motivating the investigation of plain Linux based solutions for real-time applications.

This work is focused on the configuration of Linux based systems conforming to real-time requirements and is organised as follows: section II introduces the JET's real-time infrastructure and contextualises the use of the MARTe framework. In section III details are given on how plain Linux can be configured to conform to real-time requirements in multi-core architectures. Then section IV introduces the Linux/MRG kernel distribution and presents comparative results on the deterministic performance of a user-space application running on Linux and Linux/MRG. Section V presents details on the configuration and shows performance results of four Linux based systems recently installed at JET. Finally section VI contains a brief discussion and concluding remarks of the work presented.

II. The JET's Real-Time Infrastructure

The JET tokamak is a large and complex experimental device with various requirements for real-time activities, not only from the operational and protective point of view but also

for experimental and scientific control. For over twenty years now, JET has implemented a philosophy of distributed control where separate tasks are implemented by dedicated systems [1].

At the core of magnetic plasma control are two systems: the *shape controller* [2] and the *vertical stabilisation* [3]. The former is responsible for providing the specified plasma position, shape and current and the latter is responsible for guaranteeing the vertical stability of the elongated plasma column. Plasma density control and additional heating control are implemented via dedicated local managers.

For what concerns protection, the JET tokamak is equipped with escalating layers of protective actions. At the lower level are the most drastic but also the simplest and most reliable protective actions responsible for ensuring the machine's mechanical integrity. At a higher level, fast real-time controllers and diagnostic systems act as to avoid triggering the lower level protective actions. More recently, with the installation of the new all-metallic wall [4], new high level protection systems have been built and others have been refurbished to meet the more stringent operational demands. The vessel thermal map and the wall load limiter system, both responsible for preventing excessive thermal loads to the first wall, will be covered with more detail, respectively, in sections V-A and V-B. The *plant enable window system* is responsible for insuring safe operation of the neutral beam heating system, i.e. checking the plasma has a minimum density and guaranteeing that the maximum injected energy allowed is not exceeded, while the *real-time protection sequencer* system [5] is responsible for coordinating the responsive actions of the various controllers in order to safely terminate the experiment upon the identification of a potentially dangerous event. Coherence among systems is achieved by individual consistency checks and by the ubiquitous pulse schedule editor, the pre-pulse configuration tool [6].

In parallel with the plasma control and protection systems are real-time diagnostics, e.g. the *motional Stark effect* [7], and real-time analysis systems, e.g. *Equinox* [8]. The former provides direct measurements of plasma parameters while the latter indirect measurements by performing real-time analysis on the direct measurements. Both types of systems are potential real-time data sources for driving control systems with either protective or experimental intents [9].

At the very core of this maze of real-time nodes is the Asynchronous Transfer Mode (ATM) based Real-Time Data Network (RTDN) [10]. It provides the communication infrastructure that enables cooperation amongst all real-time systems. It was adopted by the JET in 1997. It features optical links with transmission speeds of 155 Mbps, permanent virtual connections managed exclusively at the switch level thus preventing unauthorised transmissions of affecting sensitive nodes, selective multicasting, low latency ($< 200\ \mu s$) and support by multiple OSs and hardware buses.

The JET's implementation of distributed real-time systems has been demonstrated successful over the years. It is flexible, i.e. adequate for meeting sudden (and often unanticipated) operational demands, scalable and simultaneously manageable, i.e. new systems can easily be added and tested in the online environment without affecting plasma operations. Commissioning activities and fault investigation of existing systems is also simplified because each one of them is functionally self contained. Being the largest nuclear fusion experimental device and explored by a vast collaborative international research community, the JET's experience and the lessons learned from it are invaluable assets to endure now that the world's fusion community is directing its major efforts onto the International Thermonuclear Experimental Reactor (ITER).

An important spin-off of the JET's distributed control philosophy is the MARTe software framework for real-time systems [11]. Just like so many other products at JET, the MARTe framework is the result of nearly two decades of experience in designing and building software for real-time systems. MARTe is a C++ multi-platform framework for the development of modular and highly configurable real-time applications. At its core is the BaseLib2 library. This library implements a layered API spanning basic low level (semaphores, mutexes, threads, filesystem interaction, high resolution timers, etc) to high level functionality such as messaging, http services and configuration parsing. It is very complete, optimised for real-time tasks and presently supports the Linux, Linux/RTAI, VxWorks, Solaris, Windows and the MacOSX OSs. Examples of the aforementioned diversity of MARTe based applications deployed at JET [12] are the *vertical stabilisation* system (Linux/RTAI) [13], the *current controller of the error field correction coils* (VxWorks) [14], the *real-time protection sequencer* (VxWorks) [5], the *vessel thermal map* (Linux) [15], the *wall load limiter system* (Linux), and the *hard X-ray and gamma-ray profile monitor* (Linux) [16]. The merits of the MARTe framework made it become, in the last few years, increasingly adopted and developed [17] by the magnetic confinement fusion community in europe [18]. It is also used in the ISTTOK [19], COMPASS [20] and FTU [21] tokamaks and under consideration for ITER's fast plant system controllers [22].

One of its main strengths is that it allows application development, testing and debugging in non real-time environments and consequent online deployment without code changes because all the OS dependent functionality is abstracted and, therefore, transparent to the developer. Development, testing and commissioning time is minimised. Applications can also be developed in user-space (e.g. Linux) and deployed in kernel-space (e.g. Linux/RTAI or VxWorks) seamlessly. Being multi-threaded and particularly optimised for exploring multi-core architectures in a manageable and configurable way, dealing with the demands of modern real-time programming, e.g. using mutex priority inheritance and real-time schedulers, MARTe is not surprisingly the framework of choice for building high performance systems.

III. Configuring Linux for Real-Time

Misconceptions often occur while trying to establish the concept of a real-time system. A real-time system is not about speed, a real-time system is about guaranteeing that timing constraints are met. And those constraints are a function of

the system's requirements thus, unique to each system. As an example, the JET's plasma shape controller [2] reads magnetic measurements, determines the plasma shape, performs sanity and limits checks and acts on 9 circuits, all within $2\ ms$, while the vertical stabilisation system executes every $50\ \mu s$.

At JET, 1990's Digital Signal Processor (DSP) based real-time systems have been traditionally replaced by VME crates hosting PowerPC processor boards running the VxWorks OS. This approach was especially adopted for fast controllers and diagnostics with hard wiring requirements.

Recently, the need to upgrade and build new systems, mainly to cope with the challenges presented by JET's new metallic wall, has motivated a reflection on the development and deployment paths and operation of some of the already existing systems. Furthermore, building on the experience of developing and testing MARTe based applications in Linux systems, the sole requirement for RTDN based I/O, the vast availability of debugging and profiling tools, the size and dedication of Linux's support community and the attractiveness of it being not only free but also open-source, motivated the investigation of its real-time capability for user-space applications. Some of the reasons why developing user-space real-time applications is more convenient than kernel-space ones are:

- in kernel-space basic user-space abstractions such as sockets are not available;
- because there is no memory protection, a bug in kernel-space code often means a total freeze of the entire system (and a power cycle);
- developing, profiling, debugging, testing and diagnosing in kernel-space is much more lengthy and difficult because tools are scarce and, again, there is no memory protection mechanism.

On the other hand, having an application running in kernel-space does eliminate the context switching overhead between both spaces thus having it closer to the hardware and the scheduler at all times, in principle, with lower latency. In single core CPUs IRQ latency is roughly of the order of tens of microseconds and user-space process latency is of the order of tens of milliseconds depending on the CPU speed. Results presented in [23] show comparable latency measurements for a kernel-space task, under no load conditions, running on plain Linux, VxWorks, Linux/RTAI and Linux/Xenomai. In this work the focus is not so much on the average latency itself but rather on the deviation from the average, the *jitter*, which is a measure of a system's determinism. For example, a system that takes $10 \pm 9\ s$ to respond to an external event is surely not considered to be deterministic while another system taking $10 \pm 0.1\ s$ might be. The former has an average *relative jitter* of 90% while the latter's 1%.

Five aspects were carefully taken into account when trying to configure a Linux system for real-time performance: BIOS setup, kernel setup, services configuration, thread affinity and IRQ affinity, all discussed below. Furthermore, systems were built without a swap partition to avoid thrashing upon page faults. A gentler approach would be just to lock the process's local memory space into RAM[1] to prevent paging to the swap area but that does not guarantee that other processes cannot swap to disk, thus creating extra load. Details of the aforementioned individual aspects follow below.

A. BIOS Configuration

In the BIOS, all features such as CPU thermal throttle and power management in general, which are able to change the CPU's clock frequency, should be disabled. Furthermore, all superfluous features such as USB control and audio should also be disabled.

B. Kernel Configuration

The Linux kernel essentially allows for two types of configuration: compilation and boot. Regarding compilation, the kernel should be configured to enable:

- **Tickless system (Dynamic Ticks)** - avoids regular timer interrupt servicing load;
- **High Resolution Timer Support** - improves POSIX timers and *nanosleep()* accuracy;
- **Preemptible Kernel (Low-Latency Desktop)** - reduces kernel latency by allowing all kernel code, not executing in a critical section, to be preempted. Selecting a preemptible kernel reduces process latency to one or two milliseconds in a typical single core machine [24].

and to disable *Power management and ACPI options* (including *CPU Frequency scaling*) which turn off, or put into a power conserving sleep, hardware components not being used. In older machines it may be required to enable *ACPI* for proper high resolution timers' support. Furthermore, the kernel should be completely striped off superfluous functionality such as graphics card support, USB and all non-required hardware support.

The kernel should boot with the following parameters:

- **isolcpus**=<CPU list> - isolates CPUs/Cores from the scheduler so that application threads can be assigned to them claiming all resources[2];
- **idle**=mwait - improves the performance of waking up an idle CPU;
- **acpi**=off - turn off power management (if possible, see above);
- **acpi_irq_nobalance** - ACPI (if enabled) will not move active IRQs;
- **pci**=noacpi - do not use ACPI (if enabled) for IRQ routing;
- **thermal.off**=1 - disable (if ACPI enabled) thermal control.

C. Service Configuration

Running services should be kept down to the bare minimum: *messagebus*, *network* and *sshd* for ethernet based non real-time

[1] This can be achieve by calling $mlockall(MCL\_CURRENT \mid MCL\_FUTURE)$.

[2] In fact there are a few OS processes running on each CPU/Core that cannot be moved (e.g. *migration* and *watchdog*) but whose priority can be lowered if causing unacceptable load.

I/O and remote administration. Services like *irqbalance* (dynamically routing IRQs among CPUs/Cores for load balancing purposes) and *cpuspeed* (dynamically adjusting the CPU's clock speed usually for reducing power consumption) should be explicitly disabled. Protection services such as SELinux and firewall should not be necessary as real-time networks should be trustworthy mediums. Arguably, X should also find no place in a real-time system.

D. Thread Affinity

The MARTe framework provides a very convenient and configurable way of specifying exactly the thread allocation distribution among CPUs/Cores. This facilitates the placement of high priority threads in isolated CPUs/Cores minimising competition for resources and context switching overheads.

E. Interrupt Requests' Affinity

Servicing IRQs is a source of non-schedulable and non-prioritisable asynchronous load in vanilla kernels. In these kernels the Interrupt Service Routine (ISR) servicing a particular IRQ preempts the CPU/Core it is allocated to and does not yield it until it finishes. This can obviously cause high priority tasks to be forced to wait until low priority interrupts are dealt with. However, this problem can easily be ameliorated by individually specifying the affinity of each IRQ, and therefore of each ISR, thus retaining control of the CPU/Core load distribution[3].

As a concluding remark for this section it is very important to stress that for applications that rely on the Time Stamp Counter (TSC) to get high resolution CPU timing information it is crucial that each CPU's tick rate is kept at a constant known value at all times. The, less reasonable, alternative is to keep track of each CPU's tick rate at all times. Failure to comply to one of these will result in corrupted time measurements.

In the following section, a series of tests are performed to assess the potential of this approach using a technology presently under evaluation by ITER as a term of comparison.

IV. Deterministic Performance Comparison of a User-Space Application Running on a Vanilla and on a Linux/MRG Kernel

In this section, the determinism of a user-space application synchronised on an external asynchronous event, is evaluated when running on top of a vanilla and a Linux/MRG kernel. The Linux/MRG kernel was chosen as a term of comparison because it is presently under evaluation by ITER and because it allows to run the exact same user-space application that also runs on a vanilla kernel.

[3]This is achieved by writing the desired CPU mask into the appropriate set of */proc/irq/xx/smp_affinity* files, where *xx* is the IRQ's interrupt vector.

A. The Linux/MRG Kernel

The Linux/MRG kernel [25], originally from Red Hat and largely inspired by the popular real-time patch (see *https://rt.wiki.kernel.org*), is specified to provide enhanced determinism to a Linux system. Its major practical enhancement, when compared to a (properly configured) vanilla kernel and from the authors' point of view, is the ability to prioritise IRQ handling by deferring the bulk of the task to ordinary kernel threads. Clearly such flexibility is crucial in case of multiple tasks competing for scarce resources.

B. Apparatus

The tests presented here aim to assess the deterministic behaviour of a user-space MARTe based application synchronising on an external event, in this case, the arrival of a network packet. Various conditions have been combined to try to establish the operational boundaries of such a system. In particular, tests were done exploring both multi and single-core architectures with both switched and direct network connections with and without simulated stress load. The test system is a x86 64bit PC with a 3GHz *AMD Phenom(tm) II X4 955 Processor* on a *Gigabyte Technology GA-MA785GM-US2H* motherboard with an onboard ethernet NIC and a *FORE PCA-200E* PCI based ATM NIC. The standard Linux ATM kernel driver module is used in conjunction with a user-space socket based implementation for ATM I/O.

The distribution of choice is Scientific Linux 6.2 (based on Red Hat Enterprise Linux) running the *3.0.9-rt26.46.el6rt.x86_64* kernel in MRG and vanilla versions for comparison.

The base setup used is shown in Fig. 1. The ATM producer publishes a datagram every $2\ ms$. This locally synchronised system ensures the datagram is sent with a worst case absolute jitter $< 5\ \mu s$ and an average absolute jitter of $25\ ns$. In the system under test, the execution of each real-time cycle is triggered by the arrival of this datagram.

A third (so called disturber) system is introduced to assist in load stress testing. It can *flood* the system under test with ethernet *ICMP ping* requests as well as simultaneously launching (recursively via *ssh*) disk usage summary (*du /*) commands. These cause high CPU process and IRQ handling loads on the test system. On the other hand, the system under test is also capable of inflicting additional load on itself by running the *dd* command (disk cloning utility), the *yes* command (output a string repeatedly) while flooding the disturber system and itself (via the loopback interface) with *ICMP ping* requests.

Some tests presented here are referred to as having been done in multi-core and others in single-core operation. Multi-core operation means setting up the system under test exactly as shown in Fig. 1. In case of Linux/MRG, the ATM IRQ thread is not only assigned to CPU 2 but its priority is also raised to 99, the maximum priority in Linux. All other IRQ threads are assigned to CPU 1 maintaining their default priority of 50. Single-core operation means that all processes, threads and IRQs are assigned to CPU 1. In case of Linux/MRG, the ATM IRQ thread's priority is raised to 99

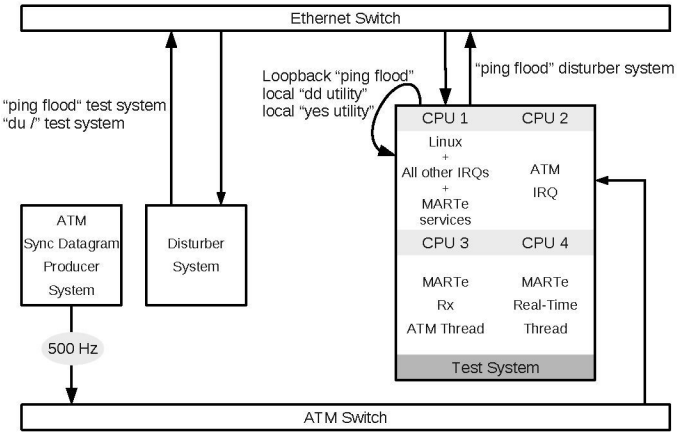

Fig. 1. The test system synchronises its real-time cycle on the arrival of an ATM datagram. The disturber system assists in load stress testing.

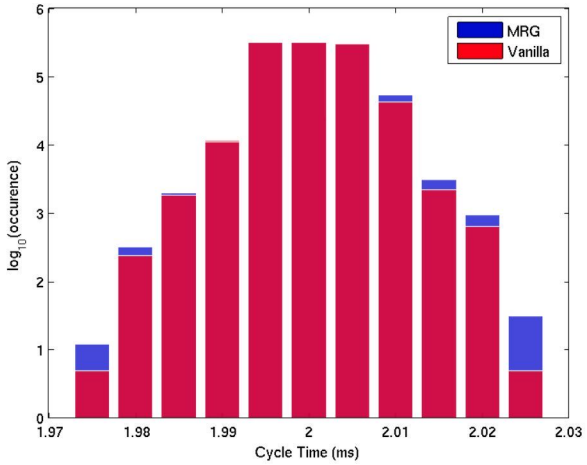

Fig. 2. In a multi-core setup with switched ATM synchronisation and simulated stress load the test application's deterministic performance is similar when running on a vanilla and on an MRG kernel.

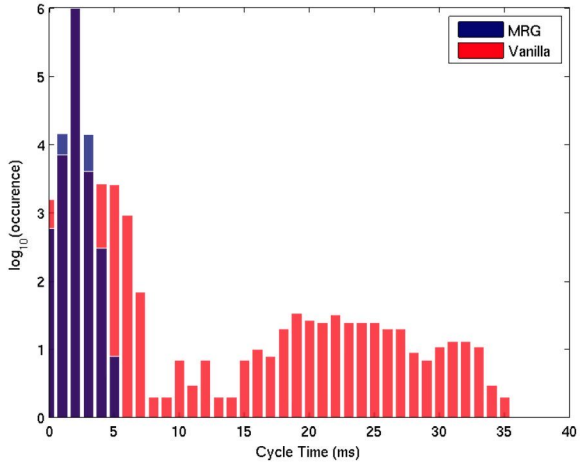

Fig. 3. In a single-core setup with switched ATM synchronisation and simulated stress load the test application's deterministic performance deteriorates more when running on a vanilla than when running on an MRG kernel.

while all other IRQ threads maintain their default priority of 50.

All cycle time measurements shown in this section are performed locally using the TSC register with a resolution of $1/3\ GHz \approx 0.33\ ns$.

C. Results

It can be seen from Fig. 2 that in multi-core operation, under simulated load stress conditions, both vanilla and MRG kernels perform similarly well. In fact, the vanilla kernel performs slightly better with a worst case jitter of $24.4\ \mu s$ and average jitter of $4.4\ \mu s$, whereas the MRG kernel yields a worst case jitter of $29.3\ \mu s$ and average jitter of $4.5\ \mu s$. In single-core operation though (see Fig. 3), and under the same simulated load stress conditions, the MRG kernel is far superior to the vanilla kernel. The worst case jitter is respectively $3\ ms$ and $55.6\ ms$ and the average jitter is respectively $202\ \mu s$ and $724\ \mu s$. Still, a $3\ ms$ absolute worst case jitter corresponds to a relative worst case jitter of 150% and, therefore, far from real-time performance. It is clear though, that prioritising IRQ handling bears a major positive impact on the application's deterministic performance. Under no stress conditions, and as anticipated because of the weak demand for resources, both kernels perform in a similar manner with an average jitter of $\approx 6\ \mu s$ and a worst case jitter of $\approx 45\ \mu s$.

It was also interesting to observe in these tests that a worst case jitter of $< 5\ \mu s$ in the producer system is propagated to give a worst case jitter of $\approx 25\ \mu s$ in the test system. Furthermore, by connecting the ATM producer and the test system directly, the average additive jitter introduced by the ATM switch[4] has been estimated to be $1.3\ \mu s$.

V. Recent Linux Based Real-Time Systems

This section presents a brief overview of four MARTe and plain Linux based real-time systems installed over the last year in JET. The Vessel Thermal Map (VTM), the WAll Load Limiter System (WALLS), the BetaLi and the Advanced Predictor Of DISruptions (APODIS). All systems run the 2.6.35.9 Linux kernel from *http://www.kernel.org* and were built using the hardware listed below, except for the BetaLi system which has a 3GHz *AMD Phenom(tm) II X4 955* CPU.

- 3.2 GHz AMD Phenom(tm) II X6 1090T CPU
- ASRock 890GX Extreme4 Motherboard
- Onboard Ethernet NIC
- FORE PCA-200E NIC

Except for APODIS, all other systems trigger the execution of their real-time cycle on the arrival of a particular datagram. This datagram is published in the RTDN at 500 Hz, with a worst case jitter of $50\ \mu s$, and traverses 2 ATM switches, including a shared inter-switch link fibre, before reaching their end recipients. APODIS, because it is required to have a temporal resolution of $1\ ms$, synchronises its real-time cycle

[4]MARCONI LE 155.

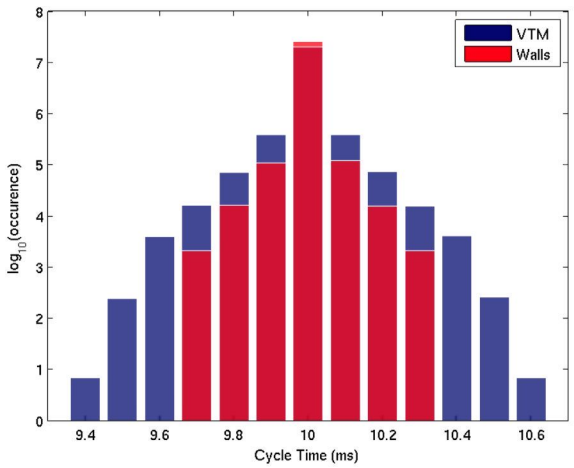

Fig. 4. VTM and WALLS's measured cycle time

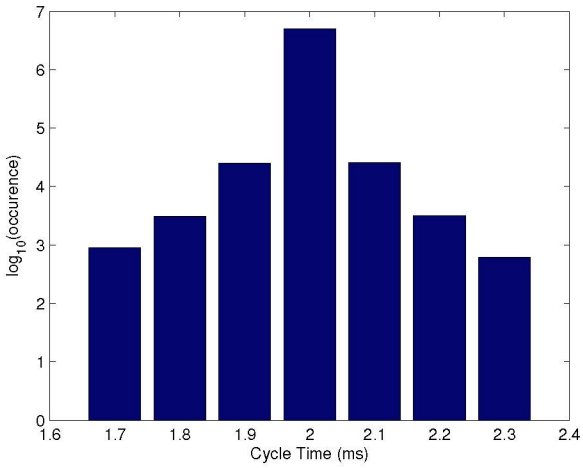

Fig. 5. BetaLi's measured cycle time

on a local timer[5].

A. VTM

The VTM system [15] was designed to collect surface temperature measurements from plasma facing components, group them according to spatial location and probable offending heat source, and raise alarms so that the appropriate protective responses can be triggered.

It receives 19 and publishes 2 ATM datagrams in total and has a 10 ms cycle time. Its thread and IRQ affinity distribution is listed below and a histogram of the measured cycle time is show in Fig. 4.

Core 1 - Linux, MARTe services and all other IRQs
Core 2 - ATM IRQ
Core 3 - MARTe thread for receiving synchronisation datagram
Core 4 - MARTe threads for receiving datagrams (x9)
Core 5 - MARTe threads for receiving datagrams (x9)
Core 6 - MARTe's real-time thread

B. WALLS

WALLS [26] was designed, just like VTM, to protect JET's wall and divertor plasma facing components. By determining the topology and location of the plasma boundary based on real-time magnetic measurements, WALLS places constraints on the plasma clearance from the wall at various locations, including the positions of the magnetic strike points in the divertor region. Furthermore, by incorporating additional information about the plasma current and non-inductive heating system's instantaneous injected power, it models the power deposition and the thermal diffusion on individual plasma facing components thus monitoring their surface and bulk temperatures.

It receives 11 and publishes 2 ATM datagrams in total and has a 10 ms cycle time. Its thread and IRQ affinity distribution is listed below and a histogram of the measured cycle time is show in Fig. 4.

Core 1 - Linux, MARTe services and all other IRQs
Core 2 - ATM IRQ and MARTe thread for receiving synchronisation datagram
Core 3 - MARTe's real-time thread
Core 4 - MARTe threads for receiving datagrams (x11)
Core 5 - Currently unused
Core 6 - Currently unused

C. BetaLi

The BetaLi system [27] was designed to estimate various plasma properties such as the *beta* (ratio between the kinetic and magnetic pressures) and the plasma's internal inductance.

It receives 7 and transmits 1 ATM datagram in total and has a 2 ms cycle time. Its thread and IRQ affinity distribution is listed below and a histogram of the measured cycle time is show in Fig. 5.

Core 1 - Linux, MARTe services and all other IRQs
Core 2 - ATM IRQ and MARTe threads for receiving datagrams (x7)
Core 3 - MARTe thread for receiving synchronisation datagram
Core 4 - MARTe's real-time thread

D. APODIS

The APODIS system [28] is a classifier based on support vector machines aiming to anticipate the occurrence of a disruptive event so that protective action can take place.

It takes 7 real-time signal inputs, of which 2 are provided by the BetaLi system, receiving 6 and publishing 2 ATM datagrams in total. It has a 1 ms cycle time and its thread and IRQ affinity distribution is listed below. A histogram of its measured cycle time is show in Fig. 6.

Core 1 - Linux, MARTe services and all other IRQs
Core 2 - ATM IRQ
Core 3 - MARTe threads for receiving all datagrams (x6)

[5]A Kalman filter scheme is used to synchronise its internal clock to JET's central timing (provided by the 500 Hz datagram) for time stamping internal signals.

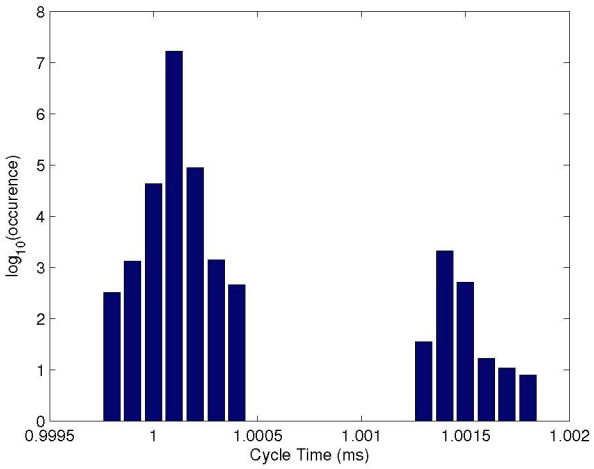

Fig. 6. APODIS's measured cycle time

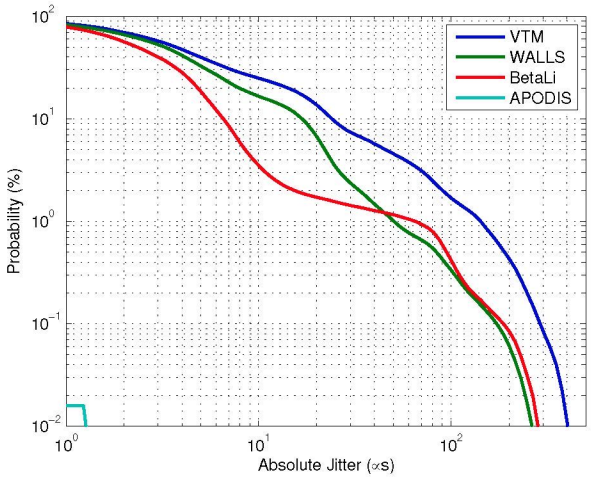

Fig. 7. Probability of occurrence of an absolute jitter larger than a given value.

Core 4 - MARTe local timer thread
Core 5 - MARTe's real-time thread 1
Core 6 - MARTe's real-time thread 2

E. Result Summary

A summary of the deterministic performance of each individual Linux based system is shown in TABLE I. It can be seen that the largest absolute worst case jitter was measured in the VTM system, 600 μs, which is equivalent to a relative worst case jitter of 6%. The largest relative worst case jitter, 15%, corresponding to an absolute worst case jitter of 300 μs was measured in the BetaLi system. The local synchronisation mechanism used in APODIS ensures relative worst case and average jitters of, respectively, 0.5% and 0.0022%.

For each system, Fig. 7 shows the probability of occurrence of an absolute jitter larger than a given value. It can be seen, for example in the case of VTM, that the probability of occurrence of an absolute jitter > 200 μs is 0.4% while for all other systems is $< 0.1\%$. Similarly, Fig. 8 presents the probability of occurrence of a relative jitter larger than a given value. It can be seen, for example, that the probability of occurrence of a relative jitter $> 1\%$ is $< 2\%$ for all systems.

It has been observed that the VTM system's absolute average and worst case jitters are roughly twice the ones of the WALLS and BetaLi systems. Although conclusive explanations for this observation cannot be presented at this point, possible motives are the differences in thread and IRQ affinity distribution among systems or eventually extra access to the CPU's shared L3 cache of the VTM system when compared to WALLS and BetaLi. Still, although intriguing, after 2645 pulses and over 21 million cycles measured, the VTM system's deterministic performance, as depicted in Figs. 7 and 8, is far from being considered unacceptable.

As a concluding remark it should be noted that the performance discrepancies between the test system of section IV and the online systems synchronising their real-time cycle on ATM is largely due to two factors. First, the absolute jitter of the online source of the synchronisation datagram is < 50 μs while in the test rig is < 5 μs, and second, the online system's datagrams must cross a shared inter-switch link fibre (sharing bandwidth with other traffic) unlike the test rig's setup.

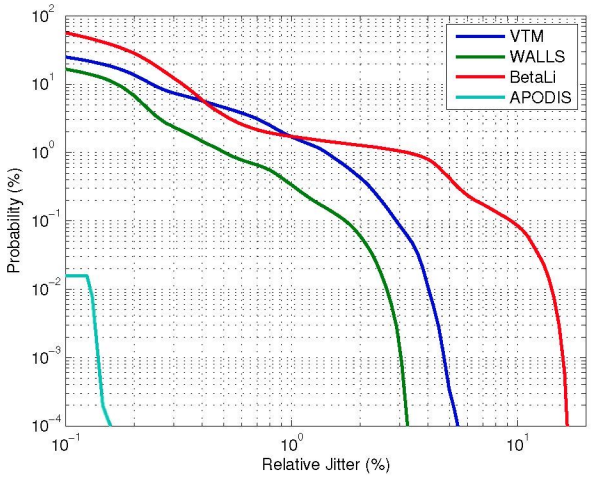

Fig. 8. Probability of occurrence of a relative jitter larger than a given value.

TABLE I
RESULT SUMMARY

System	Pulses	Cycles	Sync. Mechanism	Cycle Time (ms)	Std. Dev. (μs)	Worst Dev. (μs)
VTM	2645	> 21E6	ATM	10	30	600
WALLS	2658	> 26E6	ATM	10	15	300
BetaLi	106	> 5E6	ATM	2	15	300
APODIS	275	> 17E6	Local	1	0.022	2

VI. CONCLUSIONS

This work has demonstrated that user-space applications running under plain Linux can be made to meet timing constraints in a deterministic manner. Four systems recently deployed in the RTDN of the JET tokamak, two of which fundamental protection systems, were shown to conform to real-

time requirements after statistically relevant measurements of several millions of cycles.

When compared to the real-time oriented Linux/MRG kernel, the vanilla kernel was shown to be equally deterministic in multi-core configuration. Tests demonstrated that a real-time OS becomes a requirement when the competition for resources is exceedingly demanding. However, because competition for resources is inversely proportional to the current trend of ever increasing number of cores in present CPUs, real-time OSs may, in many situations, be an avoidable complication. Linux/MRG in particular introduces an overhead in terms of kernel threads that might be considered a bit overwhelming. In single-core operation and under simulated load stress conditions though, the Linux/MRG kernel demonstrated its superiority when compared to a vanilla kernel but, with a relative worst case jitter of 150%, still failed to deliver acceptable performance.

All these Linux based user-space applications have proven to be notoriously stable. Having MARTe as a standard greatly benefits maintenance, fault investigation and upgrade activities not only by decreasing effort and time requirements but also because the knowledge can be spread instead of concentrated exclusively in the people responsible for (or even just the developers of) specific applications.

Finally it is important to point out that the conjugation of the increasing number of CPU core availability, the implementation of a distributed real-time control, measurement and analysis philosophy whilst having the MARTe framework executing these activities in the form of user-space, tailored, flexible and data driven applications is one of the keys to the success of present JET operations.

ACKNOWLEDGMENT

This work was supported in part by the European Communities and the Instituto Superior Técnico under a Contract of Association between EURATOM and IST and in part by Fundação para a Ciência e Tecnologia under a Contract of Associated Laboratory. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

REFERENCES

[1] F. Sartori, T. Budd, P. Card, R. Felton, P. Lomas, P. McCullen, F. Piccolo, L. Zabeo, R. Albanese, G. Ambrosino, G. De Tommasi, and A. Pironti, "Jet operations and plasma control: A plasma control system that is safe and flexible in a manageable way." in *Fusion Engineering, 2009. SOFE 2009. 23rd IEEE/NPSS Symposium on*, june 2009, pp. 1 –6.

[2] F. Sartori, G. De Tommasi, and F. Piccolo, "The Joint European Torus," *Control Systems, IEEE*, vol. 26, no. 2, pp. 64 – 78, april 2006.

[3] M. Lennholm, D. Campbell, F. Milani, S. Puppin, F. Sartori, and B. Tubbing, "Plasma vertical stabilisation at JET using adaptive gain adjustment," in *Fusion Engineering, 1997. 17th IEEE/NPSS Symposium*, vol. 1, oct 1997, pp. 539 –542 vol.1.

[4] G. M. et al, "JET ITER-like Wall - Overview and experimental programme," in *Proceedings of the 13th International Workshop on Plasma-Facing Materials and Components for Fusion Applications, Rosenheim, Germany*, 05 2011.

[5] A. Stephen, G. Arnoux, T. Budd, P. Card, R. Felton, A. Goodyear, J. Harling, D. Kinna, P. Lomas, P. McCullen, P. Thomas, I. Young, K.-D. Zastrow, A. Neto, D. Alves, D. Valcárcel, S. Jachmich, S. Devaux, and J. E. Contributors, "Centralised Coordinated Control To Protect The JET ITER-like Wall." 13th International Conference on Accelerator and Large Experimental Physics Control Systems, October 2011.

[6] H. van der Beken, B. Green, C. Steed, J. Farthing, P. McCullen, and J. How, "Level 1 software at JET: a global tool for physics operation," in *Fusion Engineering, 1989. Proceedings., IEEE Thirteenth Symposium on*, 2-6 1989, pp. 201 –204 vol.1.

[7] D. Alves, A. Stephen, N. Hawkes, S. Dalley, A. Goodyear, R. Felton, E. Joffrin, and H. Fernandes, "Real-time motional Stark effect in JET," *Fusion Engineering and Design*, vol. 71, no. 14, pp. 175 – 181, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0920379604000420

[8] D. Mazon, J. Blum, C. Boulbe, B. Faugeras, M. Baruzzo, A. Boboc, S. Bremond, M. Brix, P. DeVries, S. Sharapov, L. Zabeo, and E. J. contributors, "EQUINOX: A Real-Time Equilibrium Code and its Validation at JET," in *Proceedings of the 4th International Scientific Conference on Physics and Control*, September 2009.

[9] R. Felton, E. Joffrin, A. Murari, L. Zabeo, F. Sartori, F. Piccolo, J. Farthing, T. Budd, S. Dorling, P. McCullen, J. Harling, S. Dalley, A. Goodyear, A. Stephen, P. Card, M. Bright, R. Lucock, E. Jones, S. Griph, C. Hogben, M. Beldishevski, M. Buckley, J. Davis, I. Young, O. Hemming, M. Wheatley, P. Heesterman, G. Lloyd, M. Walters, R. Bridge, H. Leggate, D. Howell, K.-D. Zastrow, C. Giroud, I. Coffey, N. Hawkes, M. Stamp, R. Barnsley, T. Edlington, K. Guenther, C. Gowers, S. Popovichef, A. Huber, C. Ingesson, D. Mazon, D. Moreau, D. Alves, J. Sousa, M. Riva, O. Barana, T. Bolzonella, M. Valisa, P. Innocente, M. Zerbini, K. Bosak, J. Blum, E. Vitale, F. Crisanti, E. de la Luna, and J. Sanchez, "Real-time measurement and control at JET experiment control," *Fusion Engineering and Design*, vol. 74, no. 14, pp. 561 – 566, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0920379605003352

[10] R. Felton, K. Blackler, S. Dorling, A. Goodyear, O. Hemming, P. Knight, M. Lennholm, F. Milani, F. Sartori, and I. Young, "Real-time plasma control at JET using an ATM network," in *Real Time Conference, 1999. Santa Fe 1999. 11th IEEE NPSS*, 1999, pp. 175 –181.

[11] A. Neto, F. Sartori, F. Piccolo, R. Vitelli, G. De Tommasi, L. Zabeo, A. Barbalace, H. Fernandes, D. Valcarcel, and A. Batista, "MARTe: A Multiplatform Real-Time Framework," *Nuclear Science, IEEE Transactions on*, vol. 57, no. 2, pp. 479 –486, april 2010.

[12] G. De Tommasi, D. Alves, T. Bellizio, R. Felton, A. Neto, F. Sartori, R. Vitelli, L. Zabeo, R. Albanese, G. Ambrosino, and P. Lomas, "Real-Time Systems in Tokamak Devices. A Case Study: The JET Tokamak," *Nuclear Science, IEEE Transactions on*, vol. 58, no. 4, pp. 1420 –1426, aug. 2011.

[13] T. Bellizio, G. De Tommasi, P. McCullen, A. Neto, F. Piccolo, F. Sartori, R. Vitelli, and L. Zabeo, "The Software Architecture of the New Vertical-Stabilization System for the JET Tokamak," *Plasma Science, IEEE Transactions on*, vol. 38, no. 9, pp. 2465 –2473, sept. 2010.

[14] D. Alves, R. Vitelli, L. Zaccarian, L. Zabeo, A. Neto, F. Sartori, P. McCullen, and P. Card, "The new Error Field Correction Coil controller system in the Joint European Torus tokamak," *Fusion Engineering and Design*, vol. 86, no. 68, pp. 1034 – 1038, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0920379610005181

[15] D. Alves, R. Felton, S. Jachmich, P. Lomas, P. McCullen, A. Neto, D. F. Valcárcel, G. Arnoux, P. Card, S. Devaux, A. Goodyear, D. Kinna, A. Stephen, and K.-D. Zastrow, "Vessel thermal map real-time system for the JET tokamak," *Phys. Rev. ST Accel. Beams*, vol. 15, p. 054701, May 2012. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevSTAB.15.054701

[16] A. Fernandes, R. Pereira, A. Neto, D. Valcárcel, D. Alves, J. Sousa, B. Carvalho, V. Kiptily, B. Syme, P. Blanchard, A. Murari, C. Correia, C. Varandas, and J.-E. contributors, "Real-time processing system for the JET Hard X-ray and Gamma-ray profile monitor enhancement." IEEE-NPSS 18th Real-Time Conference, June 2012.

[17] D. Valcárcel, D. Alves, A. Neto, C. Reux, B. Carvalho, R. Felton, P. Lomas, J. Sousa, L. Zabeo, and J. E. Contributors, "Parallel Task Management Library for MARTe." IEEE-NPSS 18th Real-Time Conference, June 2012.

[18] A. Neto, D. Alves, L. Boncagni, P. Carvalho, D. Valcarcel, A. Barbalace, G. De Tommasi, H. Fernandes, F. Sartori, E. Vitale, R. Vitelli, and L. Zabeo, "A Survey of Recent MARTe Based Systems," *Nuclear Science, IEEE Transactions on*, vol. 58, no. 4, pp. 1482 –1489, aug. 2011.

[19] P. Carvalho, P. Duarte, T. Pereira, R. Coelho, C. Silva, and H. Fernandes, "Real-Time Tomography System at ISTTOK," *Nuclear Science, IEEE Transactions on*, vol. 58, no. 4, pp. 1427 –1432, aug. 2011.

[20] D. Valcarcel, A. Neto, I. Carvalho, B. Carvalho, H. Fernandes, J. Sousa, F. Janky, J. Havlicek, R. Beno, J. Horacek, M. Hron, and R. Panek, "The COMPASS Tokamak Plasma Control Software Performance," *Nuclear*

Science, IEEE Transactions on, vol. 58, no. 4, pp. 1490 –1496, aug. 2011.

[21] L. Boncagni, Y. Sadeghi, D. Carnevale, G. Mazzitelli, A. Neto, D. Pucci, F. Sartori, F. Piesco, S. Sinibaldi, V. Vitale, R. Vitelli, L. Zaccarian, S. Monaco, and G. Zamborlini, "First Steps in the FTU Migration Towards a Modular and Distributed Real-Time Control Architecture Based on MARTe," *Nuclear Science, IEEE Transactions on*, vol. 58, no. 4, pp. 1778 –1783, aug. 2011.

[22] B. Gonçalves, J. Sousa, B. Carvalho, A. Rodrigues, M. Correia, A. Batista, J. Vega, M. Ruiz, J. Lopez, R. Castro Rojo, A. Wallander, N. Utzel, A. Neto, D. Alves, and D. Valcarcel, "Engineering Design of ITER Prototype Fast Plant System Controller," *Nuclear Science, IEEE Transactions on*, vol. 58, no. 4, pp. 1439 –1446, aug. 2011.

[23] A. Barbalace, A. Luchetta, G. Manduchi, M. Moro, A. Soppelsa, and C. Taliercio, "Performance Comparison of VxWorks, Linux, RTAI, and Xenomai in a Hard Real-Time Application," *Nuclear Science, IEEE Transactions on*, vol. 55, no. 1, pp. 435 –439, feb. 2008.

[24] D. Abbott, *Linux for Embedded and Real-Time Applications*. Newnes, 2003.

[25] R. H. Enterprise, "Red Hat Enterprise MRG - Realtime Whitepaper," Red Hat Enterprise, Tech. Rep., 2007. [Online]. Available: http://pt.redhat.com/f/pdf/mrg/mrg_realtime_whitepaper.pdf

[26] A. Cenedese, F. Sartori, V. Riccardo, and P. Lomas, "JET first wall and divertor protection system," *Fusion Engineering and Design*, vol. 6668, no. 0, pp. 785 – 790, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092037960300317X

[27] O. Barana, A. Murari, E. Joffrin, F. Sartori, and contributors to the EFDA-JET Workprogramme, "Real-time determination of internal inductance and magnetic axis radial position in JET," *Plasma Physics and Controlled Fusion*, vol. 44, no. 10, p. 2271, 2002. [Online]. Available: http://stacks.iop.org/0741-3335/44/i=10/a=312

[28] J. Lopez, J. Vega, D. Alves, S. Dormido-Canto, A. Murari, J. Ramirez, R. Felton, M. Ruiz, G. de Arcasand, and J. E. contributors, "Implementation of the disruption predictor APODIS in JET real time network using the MARTe framework." IEEE-NPSS 18th Real-Time Conference, June 2012.