Virtualizing Programmable Logic Controllers: towards a convergent approach

Tiago Cruz, Paulo Simões and Edmundo Monteiro Department of Informatics Engineering
University of Coimbra, Portugal
Email: {tjcruz,psimoes,edmundo}@dei.uc.pt

***Abstract*—Modern Programmable Logic Controllers (PLCs) are pervasive components in Industrial Control Systems (ICS) such as Supervisory Control and Data Acquisition (SCADA), designed to control industrial processes autonomously or as part of a distributed system topology. Its success may be explained by its robustness and reliability, being one of the most enduring legacies on modern ICS, despite having evolved very little over the last years. This paper proposes an x86-based virtual PLC (vPLC) architecture that decouples the logic and control capabilities from the I/O components, while virtualizing the PLC logic within a real-time hypervisor. To demonstrate the feasibility of this concept, the topic of real-time virtualization for x86 platforms is analyzed, together with an evaluation study of the properties of real-time workloads in partitioned hypervisor environments.**

***Index Terms*—ICS, Virtualization, Converged Infrastructures**

INTRODUCTION

In recent years, SCADA ICS – a kind of systems used for controlling industrial processes, power plants or assembly lines – have become a serious concern because of manageability and security issues. This comes as a consequence of years of air-gaped isolation, together with the increased coupling of ICS and IT systems and the absence of proper management and security policies, exposing ICS to all sorts of threats. Suddenly, ICS faced a reality that has been familiar for IT infrastructure managers for decades, which led to the development of specific tools and protocols, as well as the establishment of management frameworks and security-oriented policies.

However, bridging the gap between IT and ICS is not a trivial matter of transposing technologies from one domain to the other. This is due to the fact that the primary ICS design and operation concerns are focused on reliability and operational safety, advising against any mechanism or solution with potential impact on operational performance indicators. Despite the efforts to develop domain-specific security and management capabilities for SCADA ICS, most of these solutions try to fix what is wrong without introducing significant change into existing architectures, which still struggle to deal with lifecycle operations or change management.

In this paper we propose an innovative approach for ICS infrastructure consolidation, which bridges computing and networking virtualization technologies with ICS and targets a vital SCADA ICS component: the PLC. Despite being a mature concept that incarnates a design philosophy well established across the industry, PLCs are one of the most vulnerable components on ICS, due to design (e.g., most PLCs lack redundant units such as power supplies) or cyber-security issues (as demonstrated by Stuxnet [1]). By leveraging virtualization and advanced communication technologies to decouple the PLC physical I/O and computing capabilities, we may turn it into a Real-Time Virtual Machine (VM) hosted on a real-time hypervisor, connected to I/O modules on the field using a switched deterministic and/or real-time Ethernet fabric system, with benefits in terms of resource consolidation, security, resiliency and manageability.

TOWARDS A VIRTUALIZED PLC

Modern PLCs are a class of embedded systems which incorporate technologies such as microprocessors and microcontrollers, RTOS (hosting the execution environment for the main functions and services) and communication capabilities (from serial point-to-point or bus topologies to Ethernet and TCP/IP). PLC hardware generally includes analog or digital I/O modules, fieldbus interconnects or serial communication interfaces, being occasionally coupled with Field-Programmable Gate Arrays (FPGA) or Digital Signal Processors (DSP) for real-time signal processing. Since a considerable share of these devices use commodity Instruction Set Architecture CPUs (such as x86 or ARM), the possibility of virtualizing them comes to mind, which is one of the main requirements of the vPLC architecture, discussed in this section.

A. PLCs and Real-Time Virtualization on x86 Platforms

The recent trend towards IT service and infrastructure consolidation owes much of its success to virtualization technologies. This has provided the means to effectively leverage computing and communication resources, introducing a great deal of flexibility, while also streamlining and simplifying day-to-day operations. For instance: by creating a VM snapshot before applying a security patch, changes can be rolled back in case of failure; VMs can be cloned for sandboxed testing, prior to deployment into production; also, VM instances can be live migrated, allowing for reduced downtime every time a physical device needs to be stopped.

Unlike what happened in the IT domain, the introduction of virtualization technologies for ICS has been a slow process (as with any other new technology), and not as straightforward. Only recently operators started virtualizing SCADA Master Stations (MS), Human-Machine Interfaces (HMI) and Historian Database servers (HDB), using Commercial Off-The-Shelf (COTS) hypervisors [2]. This was enabled by the

emergence of hardware-assisted memory management and I/O mechanisms [3], providing adequate performance guarantees while avoiding resource overprovisioning.

But PLC virtualization is a different matter, as the requirements for its RTOS environment and communications prioritize low and consistent latency. Most COTS hypervisors for x86 are designed for general-purpose workloads where throughput is priority, resorting to techniques such as hardware resource sharing or deferred interrupt processing, which have a penalty in terms of latency and determinism. For this reason, RT-sensitive applications such as servo control for Computerized Numerical Control (CNC) machinery cannot be reliably hosted within such hypervisors, as it only takes a single latency peak to create a significant positioning skew.

Achieving RT compliance may prove difficult, even for native execution. For instance, the end-to-end response latency for components on interconnected buses can be affected by aspects such as interrupt latency, message propagation delays, asynchronous periodic task overhead or RTOS task scheduling overhead. Particularly, interrupt latency and CPU overhead involved in servicing interrupts are paramount in embedded systems used for control applications. For example, [4] [5] estimate interrupt and context switch latency requirements of 280 and 800 μs for machine and process control industrial applications, respectively. For extreme cases, such as motion control applications, PLCs have to provide very low operation latencies, from 1ms to 250 μs (Class 3 RT Systems [6]).

Originally, interrupt processing in the x86 PC architecture was based on Programmable Interrupt Controllers (PIC). Cascaded 8259 PIC provided up to 15 fixed-priority interrupts channels using pointers to locate the vector entry points for the Interrupt Service Routines associated with each channel. Later, the IO-APIC (Advanced PIC) was introduced, supporting up to 24 interrupt channels, multiprocessor systems and programmable priorities – while an improvement in comparison with the dual PIC arrangement, APIC interrupt processing and routing was a latency-prone, multi-step procedure [4].

Things considerably improved with the advent of PCIe (PCI Express) and the Message Signalled Interrupts (MSI) model, which supports 224 interrupts, eliminates the need to use the IO-APIC, and allows every device to write directly to the CPU's Local-APIC, avoiding out-of-band interrupt signalling overhead, by using memory write operations. MSIs reduce the latency and CPU overhead involved in servicing interrupts, improving system performance and IO responsiveness. Latency can improve as much as 300% when compared to IO-APIC and 500% when compared with 8259-PIC [4]. Table I illustrates the results for IO-APIC and MSI modes.

Table I
INTERRUPT LATENCY COMPARISON (FROM [4])

CPU + interrupt mode	Worst Case (μs)	Average Case (μs)
Uniprocessor, IO-APIC	5.85	4.18
Multiprocessor(SMP), IO-APIC	7.16	4.14
Uniprocessor, MSI	3.60	1.66
Multiprocessor(SMP), MSI	3.56	1.58

Modern x86 CPUs provide code density and memory bandwidth, supporting Single Instruction Multiple Data (SIMD) ISA extensions such as SSE or AVX (akin to an integrated DSP, with compilers performing auto-vectorization, eliminating communication and transport overhead). However, not all x86 developments benefit real-time applications: power optimization and throughput-enhancing technologies, such as frequency-scaling or hardware threads (hyperthreading), harm deterministic behaviour – though most of them can be disabled or fine-tuned. Still, some notable (if somewhat odd) exceptions persist, such as System Management Interrupts (SMI).

SMIs were originally introduced to support power management capabilities, and later used for other functions such as USB legacy peripheral device emulation. A SMI event asynchronously suspends all normal program execution in order to switch to a special System Management Mode, where specific firmware code is executed – for this reason, SMIs are a common cause of latency spikes. Explicit SMI control is not possible in all x86 platforms, as it depends on specific chipset, firmware and OEM options.

Overall, the x86 platform has become an interesting candidate to host PLC applications, despite some manageable shortcomings (e.g. several providers of RT turnkey solutions provide certified hardware lists, while some tier-1 OEMs resort to custom firmware or provide mechanisms to disable non-critical SMIs for RT usage).

B. The Virtual PLC

Recent developments, such as low-latency deterministic network connectivity for converged Ethernet (able to support robust distributed I/O) and the availability of real-time hypervisors, made it possible to virtualize PLC components [7]. The proposed vPLC architecture (Fig. 1) takes advantage of these capabilities, by decoupling the PLC execution environment from I/O modules – using a Software-Defined Networking (SDN)-enabled Ethernet networking fabric to provide connectivity to the I/O subsystem. This departs from the existing SoftPLC concept (which mostly runs on COTS x86 systems, eventually using RTOS systems, such as [8] [9]), by adopting an approach close to [10] [11] but going one step further, by leveraging converged fabric scenarios with SDN.

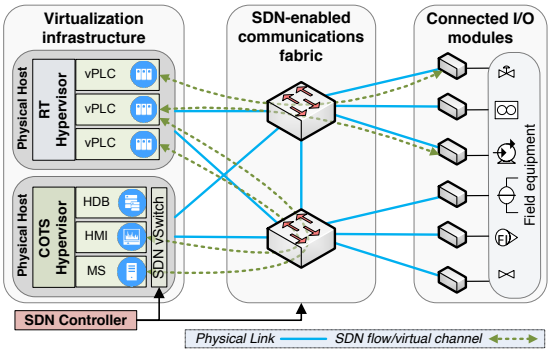

Figure 1. A vPLC deployment (adapted from [7])

In the vPLC the dedicated PLC I/O bus is replaced by a deterministic and high-speed networking infrastructure, using SDN to enable the flexible creation of virtual channels on

the I/O fabric. These channels provide connectivity between the vPLC instances and physical I/O modules, which can be implemented using FPGA or Application Specific Integrated Circuit technology. Finally, virtual channel reconfiguration is managed by means of a SDN controller, via a High-Availability server (not depicted in the figure) which monitors SDN switch statistics and path reachability, reconfiguring channel paths in case of performance degradation or failure.

This model is similar to remote or distributed I/O PLC topologies, where networked I/O modules act as extensions of the PLC rack, or even critical avionics systems, which replace legacy interconnects with Ethernet-based technologies such as Avionics Full-Duplex Switched Ethernet (AFDX) [12]. In fact, initiatives such as Converged Plantwide Ethernet (CWpE) [13] already point in this direction. Developments in cut-through switching, together with Remote Direct Memory Access (RDMA), allow for port-to-port latencies in the order of hundredths of nanoseconds in 10G Ethernet switch fabrics and application latencies in the order of microseconds [14]. Also, resources such as Intel's Data Plane Development Kit (DPDK) [15] enable low latency, high-throughput packet processing mechanisms that bypass kernels, bringing the network stack into userspace and enabling adapters to perform DMA operations. This enables single-digit microsecond jitter and restricted determinism, allowing for bare-metal performance on commodity server hardware. Additionally, Time Division-based approaches – using IEEE 1588 clock synchronization, such as Time Sensitive Networking [16] – allow for real-time requirements in the microsecond range on COTS Ethernet, compatible with strict isochronous operation needs.

Finally, real-time static partitioning hypervisors, such as Jailhouse [17] or PikeOS [18], make it possible to host RTOS guest VMs for real-time and certifiable workloads, with PikeOS closely replicating the ARINC 653 [12] partitioning model for safety-critical avionics RTOS. Moreover, [19] points to the possibility of providing RT capabilities in the KVM [20] hypervisor, when combined with the Linux RT-Preempt [21] patch and specific tuning. In such environments, resources such as PLC watchdogs and system-level debugging and tracing analysis mechanisms (useful for continuous security and/or safety assessment) can be implemented at the hypervisor level, which is able to oversee partition behavior.

Evaluation

Evaluation is focused on understanding to which point partitioning techniques may prove effective for implementing real-time hypervisor environments, when used on modern hardware. The test platform uses an Intel Core i7-4770 running at 3.40GHz (Haswell family) paired with 16GB DDR3 RAM, using Debian Linux 8.4 with kernel version 3.18.29. Three versions of the kernel were used: baseline, with the standard distribution settings; host-optimized, with KVM hypervisor support; and guest-optimized. The latter two were compiled with the RT-Preempt patch, for realtime support.

Latency measurements were performed using *cyclictest* [22] to measure the response latency for four timer threads clocked at 10ms, spaced 500 μs from each other and run with a high scheduler priority (PRIO_FIFO), to emulate a RT task. *Stress* [23] was used for workload generation, instantiating 20 simultaneous threads (10 for CPU bound tasks and 10 for spinning malloc/free operations on 64MB blocks), enough to exhaust a single core. Tests were based on 120 minute runs.

The first round of tests (Fig. 2) focused on comparing a baseline system configuration using a standard kernel, with hyperthreading and power management support enabled, comparing it with an RT-optimized configuration. For the latter purpose, all power management features were disabled (c-states, dynamic core frequency, PCIe power management), as well as hyperthreading support, which have a negative impact on latency and jitter. Also, the Linux kernel was configured with core isolation, removing cores 1 to 3 from process scheduling and balancing algorithms, interrupt processing (whose affinity was manually adjusted to core 0) and other tasks, such as Read-Copy Update threads. The test workload contemplates three scenarios: idle state, load on shared core (load generator and latency test scheduled on core 1) and split core tests (latency and stress generator running on cores 1 and 2, respectively – emulating a scenario where Best Effort and RT tasks could be run on separate cores).

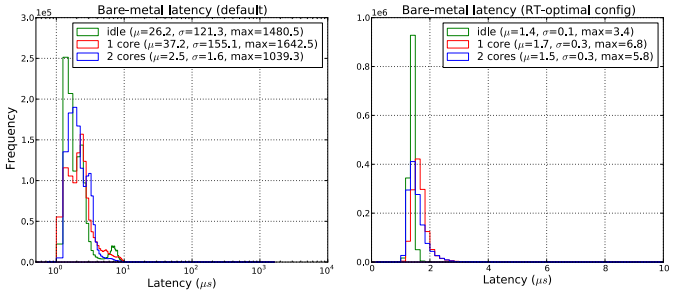

Figure 2. Bare metal test results

Results show the default configuration to be unreliable for RT purposes. While the 2 (split) core test shows improvement, there are are large latency spikes, probably due to dynamic power management and scaling, together with dynamic OS core scheduling and low core usage. Results for the RT-optimized configuration show large improvements, with the system behaving within very low latency margins, with low jitter and spikes – in fact, these margins are within the acceptable range for several motion control applications. These values could be further improved using specialized RTOS or kernel extensions such as Xenomai or RTAI [24].

The second round of tests (Fig. 3) evaluated RT performance for a single VM, using resource partitioning and two use cases: a VM with a single processor core assigned; and the same VM with three processor cores – using core affinity and memory locking on both cases. The three-core VM used the same test pattern of the bare-metal RT-optimized tests. Results show that, despite the contention effects (everything is running on the same core), the single core VM shows a controlled behaviour which is acceptable for a wide range of PLC-class applications. Nested partitioning tests (3 core VM, 2 core (split)) demonstrated the possibility of running RT applications with even stricter timings within VMs. Moreover,

there is a considerable margin for improvement in terms of hypervisor mechanisms and VM payload (e.g. we achieved a 2μs average latency with small jitter using a Xenomai co-kernel, in the same setup).

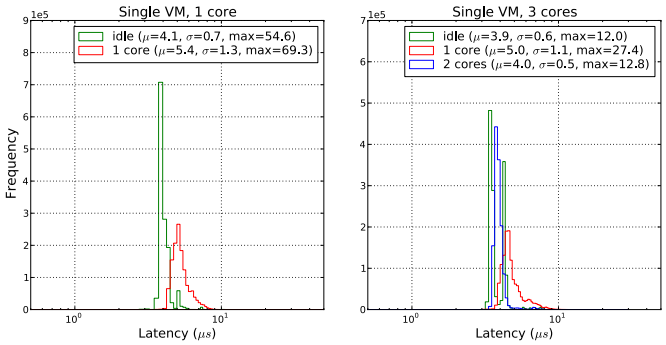

Figure 3. Test results for 1 VM

The third round of tests (Fig. 4) evaluated concurrent real-time VM performance, using resource partitioning for 3 VMs with a 1:1 core assignment ratio. Results show a uniform and consistent behaviour pattern across the 3 VMs, demonstrating the performance isolation of the partitioning approach.

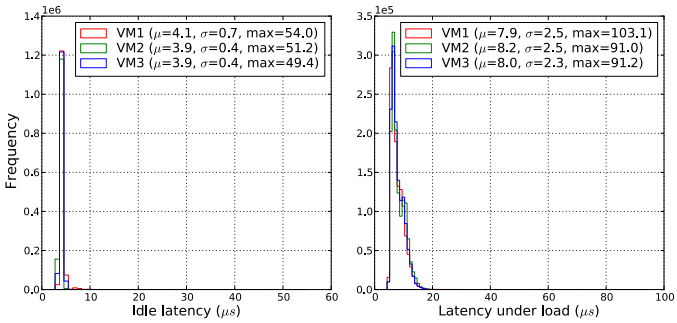

Figure 4. Test results for 3 VMs

Overall, the tests show that optimized resource partitioning provides performance guarantees for isolated workloads, while ensuring adequate graceful degradation under heavy load (a situation that nonetheless must be avoided for RT workload cores). Results can be further improved by software optimization and enhancements such as device affinity (using IOMMU [3] mechanisms) or even cache partitioning (e.g. Intel's Cache Allocation Technology), which extend the partitioning concept down to the CPU L3 cache, achieving better latency and further containing and isolating partitioned workloads.

Conclusion

The proposed vPLC constitutes a convergent approach in the sense that isolated PLC devices are virtualized and co-hosted on the same physical equipment, with distributed I/O being consolidated on the networked I/O fabric. This constitutes a convergence of computing and communication resources towards a unified infrastructure, much in the same way as it happened with datacenter architectures in the IT domain.

Evaluation results show that the vPLC is feasible from a systems virtualization perspective, with a considerable margin for further improvement on x86 platforms. Nonetheless, this paper is focused on the presentation of the vPLC concept and evaluation of the suitability of x86 virtualization for concurrent PLC workloads. Next developments include the implementation and validation of the SDN-based I/O orchestration and RT communications capabilities (using Xenomai's Real-Time Driver Model framework) that provide the coupling of the vPLC instances with the physical infrastructure – in order to comply with determinism and latency requirements.

Finally, it should be stressed that, despite its name, the vPLC is more than the simple virtualization of the PLC device, constituting an integrated approach where the device seamlessly merges with the infrastructure, providing potential benefits in terms of manageability, cost and security.

References

[1] R. Langner, "To kill a centrifugue: A technical analysis of what Stuxnet's creators tried to achieve," The Langner Group, Tech. Rep., 2013.

[2] J. Reeser, T. Jankowski, and G. M. Kemper, "Maintaining HMI and SCADA systems through computer virtualization," *IEEE Transactions on Industry Applications*, vol. 51, no. 3, pp. 2558–2564, May 2015.

[3] M. García-Valls, T. Cucinotta, and C. Lu, "Challenges in real-time virtualization and predictable cloud computing," *Journal of Systems Architecture*, vol. 60, no. 9, pp. 726–740, 2014.

[4] L. Kean, "Microcontroller to Intel architecture conversion: PLC Using Intel Atom Processor," Intel Corporation, White Paper, 2010.

[5] Venture Development Corporation, "The embedded software strategic market intelligence program 2002/2003," Tech. Rep., 2003.

[6] C. Pereira and P. Neumann, *Industrial Communications Protocols*, S. Y. Nof, Ed. Springer-Verlag, 2009.

[7] T. Cruz, R. Queiroz, P. Simões, and E. Monteiro, "Security implications of SCADA ICS virtualization: survey and future trends," in *Proc. of 15th European Conference on Cyber Warfare and Security (ECCWS 2016)*.

[8] Codesys GmbH. Codesys control – the controller functionality software. Codesys GmbH. [Online]. Available: https://www.codesys.com

[9] ISaGRAF. Isagraf overview. [Online]. Available: http://www.isagraf.com

[10] Intel Corporation, "Reducing cost and complexity with industrial system consolidation," white paper, 2013.

[11] IntervalZero, "A soft-control architecture: Breakthrough in hard real-time design for complex systems," white paper, 2010.

[12] C. Fuchs, "The evolution of avionics networks from ARINC 429 to AFDX," in *Proc. of Innovative Internet Technologies and Mobile Communications and Aerospace Networks*, vol. 65, 2012, pp. 65–76.

[13] P. Didier and F. M. et al., "Converged plantwide ethernet (cpwe) design and implementation guide," Cisco Press, Tech. Rep., 2011.

[14] M. Beck and M. Kagan, "Performance evaluation of the RDMA over ethernet standard in enterprise data center infrastructure," in *Proc. of 3rd wrk. on Data Center – Convergent and Virtual Eth. Switching*, 2011.

[15] W. Zhang, T. Wood, K. Ramakrishnan, and J. Hwang, "Smartswitch: Blurring the line between network infrastructure & cloud applications," in *Proc. of 6th USENIX Work. on Hot Topics in Cloud Computing*, 2014.

[16] IEEE. Time-sensitive networking task group. [Online]. Available: http://www.ieee802.org/1/pages/tsn.html

[17] Siemens AG. Jailhouse partitioning hypervisor,. [Online]. Available: https://github.com/siemens/jailhouse

[18] C. Baumann and et al., "Proving memory separation in a microkernel by code level verification," in *Proc. of 14th IEEE Int. Symp. on Obj./Comp./Service-Oriented RT Dist. Computing*, 2011, pp. 25–32.

[19] R. Riel, "Real-time KVM from the ground up," in *KVM Forum 2015*. [Online]. Available: http://www.linux-kvm.org/images/2/24/01x02-Rik_van_Riel-KVM_realtime.pdf

[20] KVM Project. [Online]. Available: http://www.linux-kvm.org

[21] H. Fayyad-Kazan, L. Perneel, and M. Timmerman, "Linux PREEMPT-RT v2.6.33 versus v3.6.6: Better or worse for real-time applications?" *SIGBED Rev.*, vol. 11, no. 1, pp. 26–31, Feb. 2014.

[22] RT Linux wiki. [Online]. Available: http://rt.wiki.kernel.org

[23] Stress project. [Online]. Available: http://people.seas.harvard.edu/~apw/stress

[24] A. Barbalace, A. Luchetta, and et al., "Performance comparison of VxWorks, Linux, RTAI and Xenomai in a Hard Real-time application," in *Real-Time Conference, 2007 15th IEEE-NPSS*, April 2007, pp. 1–5.