Real-time performance of virtualised protection and control software

REAL-TIME PERFORMANCE OF VIRTUALISED PROTECTION AND CONTROL SOFTWARE

Sandro SCHÖNBORN
ABB Corporate Research – Switzerland
sandro.schoenborn@ch.abb.com

Robert BIRKE
University of Torino – Italy
robert.birke@unito.it

David KOZHAYA
ABB Corporate Research – Switzerland
david.kozhaya@ch.abb.com

Thanikesavan SIVANTHI
ABB Corporate Research – Switzerland
thanikesavan.sivanthi@ch.abb.com

ABSTRACT

Substation automation is ever challenged by the integration of distributed energy resources which imposes higher deployment flexibility and adaptability for protection and control. Although virtualization helps to run software applications independent of the underlying platform in IT infrastructures and cloud computing, it is still not commonly used in the field of substation automation. This is mainly due to the real-time performance demands of substation automation protection and control applications. In this article, we present an approach for running substation automation protection and control software in virtual environments. We contrast the real-time performance of different virtualization technologies under different workloads and focus on the performance evaluation of protection and control software in container-based solutions running on Linux with PREEMPT RT. We also present additional results for performance achieved in virtual machines. Our results clearly demonstrate that it is possible to run substation automation protection and control software in virtual environments while still providing the necessary performance. This paves the way for the deployment of substation protection and control software in virtualisation environments.

INTRODUCTION

With the ubiquitous integration of Distributed Energy Resources (DERs) in the power grid, ensuring intelligence, adaptiveness, and resilience across the whole grid becomes even more challenging especially given the rigidness of deploying and adapting protection and control applications within today's substations. These substations are unprepared to deal with the new dynamics and typically operate with many different devices, some even from multiple vendors, each running applications on their proprietary hardware. As a result, immense maintenance efforts and significant costs are necessary, which often do not scale to meet the technical and business needs of the utilities. Therefore, there is a need to shift to a more flexible and rapid deployment of substation applications such that different applications can simply run on the same hardware and the update or continuous roll-out of any additional functionality is simplified. Virtualisation is a technology in the IT domain, e.g., cloud computing, that promises such flexibility by enabling applications (software) to be deployed, executed, exchanged, and migrated almost independently of the underlying platform. The core idea behind virtualisation is to allow applications to run in a "virtual" environment, which is abstracted from the actual underlying platform and isolated from other applications running on the same platform. However, unlike cloud platforms that have scalable compute, storage and networking resources, substations have low to modest scale and often require highly available operations within known and tight timing constraints.

This article addresses the real-time capability of developing virtualised protection and control software, demonstrating how virtualisation techniques can be used in time-critical systems like substation automation. We compare various virtualisation technologies, from virtual machine to container-based technologies. We then specifically demonstrate how real-time protection and control applications can be run in containers on Linux with PREEMPT RT. While we focus on containers, which provide virtualisation at operating system level, we also present results of running protection and control applications inside virtual machines. We evaluate the performance of our virtualised software by running multiple, isolated instances of industry standard substation protection and control applications on the same host. Further, we assess performance characteristics on different host machines featuring different Intel CPUs. We show that delays in virtual networking and poor resource isolation, rather than scheduling issues, are the primary causes of timing errors. Finally, we demonstrate real-time performance of the protection and control applications using tailored resource provisioning and selecting appropriate virtual networking solutions. The adherence of our virtualised control and protection applications to real-time performance is shown on testbeds and is even showcased in the field through a pilot deployment at a 10-bay MV substation [1].

BRIEF BACKGROUND ON VIRTUALISATION

Virtualisation is a technology that abstracts the underlying platform including physical hardware (such as CPU,

Authors' preprint of S. Schönborn, R. Birke, D. Kozhaya and T. Sivanthi, "Real-time performance of virtualised protection and control software," 27th International Conference on Electricity Distribution (CIRED 2023), Rome, Italy, 2023, pp. 1817-1821, doi: 10.1049/icp.2023.1028.

memory), operating system, storage, network, etc. As a result, workloads running on multiple machines can be consolidated onto fewer machines thereby resulting in better usage of hardware resources with lower overall costs. Virtualised software is typically portable since it is not tied to a specific physical hardware or custom operating system (OS) installation. Two categories of virtualisation technologies exist:

Hardware Virtualisation
This virtualises the hardware resources for guest OSs. The virtualised environment, in which guest OSs run, is called *virtual machine* (VM). It is realized using hypervisors which are the software components that virtualise the underlying hardware resources. Such hypervisors are of two types, namely, Type-1 hypervisors that run directly on the system hardware, and Type-2 hypervisors that run on top of a host OS. Type-1 hypervisors are more suitable for providing predictable performance to VMs than Type-2 hypervisors because they directly interact with the hardware and have full control on hardware resources allocated to VMs. Most modern CPUs provide direct hardware support for virtualization, e.g., Intel VT-x.

OS-level Virtualisation
This virtualises OS services such as file systems, devices, networking, and security, and provides a virtualised operating system environment to multiple isolated user-space instances. It imposes less overhead in comparison to hardware virtualisation as the instances directly use the host OS' system call interface and do not require the additional level of indirection through the hypervisor. But it limits all guests to use the same underlying OS kernel.

Table 1. Examples of Virtualisation Solutions

Solution	Virtualisation	Type
QEMU/KVM	Hardware (Type-1)	General purpose
XEN	Hardware (Type-1)	General purpose
Jailhouse	Hardware (Type-1)	Real-time
ESXi	Hardware (Type-1)	Servers
ACRN	Hardware (Type-1)	IoT/Embedded
LXC/LXD	OS-level	General purpose
Docker	OS-level	General purpose

REAL-TIME EVALUATION OF VIRTUALISATION TECHNOLGIES

In this section, we first evaluate the basic real-time capability of OS-level virtualisation techniques, since lower overhead is expected compared to hardware-based virtualisation. Namely, we compare two different and actively maintained open-source solutions Docker and LXD/LXC. However, in our performance evaluation section, we also include virtual machines.

The real-time operating system used is Linux PREEMPT RT. The basic scheduling jitter evaluation is conducted using "*cyclictest*" v1.50, a frequently used scheduling benchmark for real-time systems. *cyclictest* measures the difference between a periodic thread's intended wake-up time and the time at which it effectively wakes up. As an idle kernel running only "*cyclictest*" is not sufficient, we also loaded the kernel using "*hackbench*" which creates multiple pairs of threads that exchange data between themselves either over sockets or pipes stressing the kernel's scheduler. The experiments are conducted on two different machines equipped with Intel Xeon Silver 4208 CPU running Ubuntu 20.04 with Kernel 5.6.19 PREEMPT_RT, and Xeon Gold 6248R running Ubuntu 20.04 with Kernel 5.15.65 PREEMPT_RT, both based on Cascade Lake architecture and with 96GB of RAM.

The latency measurements (Table 2) of LXC/LXD and Docker resulted in maximum jitter of around 40µs and 20µs on the two systems with slightly higher values for LXD. The actual values correspond to those obtained natively on the OS of the same system. This shows that the real-time performances of both LXC/LXD and Docker are stable and bounded.

The maximal jitter values observed are acceptable to run protection and control applications, and the values represent the unoptimized case with high load and no resource isolation. However, such measurements are too simple to reliably assume good real-time performance. In the following, we further present results from running actual applications.

Table 2 Scheduling jitter on different systems, cyclictest running on all CPU cores at RT priority 90.

Scheduling jitter, max (avg) µs		
System	Xeon Silver 4208 5.6.19-rt12	Xeon Gold 6248R 5.15.65-rt49
Native	37 (3)	16 (2)
Docker	38 (3)	17 (2)
LXD	43 (5)	25 (2)

REAL-TIME PROTECTION AND CONTROL SOLUTION USING CONTAINERS

This section describes a container-based solution, using Docker, to virtualise a Central Protection and Control (CPC) application at the OS level. We present its realization over multiple steps including host setup, docker image preparation, container configuration and deployment. The solution is further referenced as the Virtual Protection and Control (VPC) application.

Host setup

Preparing the machine that hosts the docker containers requires taking care of three main aspects: hardware prerequisites and partition, BIOS and firmware setup, as

well as OS setup. Since containers run on the host OS, we need to configure it appropriately to support real-time operation.

Hardware Prerequisites and Partition

We performed our experiments on different host machines. We ensured that each host features an Intel CPU (> 2 GHz) with at least 8 cores to host multiple VPC instances and still provide enough resources to run the OS properly. The CPUs provide at least 11MB of L3 cache. We rely on the availability of Intel Resource Director Technology[1] to exclusively assign parts of this cache to VPC instances. In terms of memory and storage, we generously reserved 4GB per VPC plus 4GB for OS and other services, and SSD space of 20 GB for OS plus 5 GB per VPC instance. We provided at least 2 network ports with 1GbE or faster, with hardware timestamping capability and some with SR-IOV support.

BIOS and Firmware Setup

We implemented the typical recommendations for real-time configurations such as: disabling hyper-threading, disabling speed stepping and frequency scaling, and disabling power management options.

OS Setup

Our setup is based on a standard Ubuntu Server 20.04 LTS installation plus additional packages, such as Docker and linux-ptp [2]. VPC requires the PREEMPT_RT Kernel patch [3], hence we compiled a 5.10 series vanilla kernel (5.10.65 in most of our experiments) with applied PREEMPT_RT patch and activated full pre-emption mode (`CONFIG_PREEMPT_RT_FULL`). The kernel is installed in the otherwise unchanged Ubuntu system which needs to be booted with a specific set of command line options to prepare the system for real-time operation. These include disabling power management, enabling huge pages, isolating CPU cores (isolcpus, irqaffinity) and changes to the timer (nohz, clocksource).

Time Synchronization

Substation environments rely on the IEEE 1588 Precision Time Protocol (PTP) [4] to achieve the required high level of time synchronization. Linux containers provide a namespace-based abstraction of the clock but only for CLOCK_MONOTONIC. It is therefore not possible to run full synchronization daemons within the containers. They would conflict on setting the system-wide shared CLOCK_REALTIME. Therefore, synchronization needs to be handled at the host level. In a substation, there is only a single reference time such that host-level synchronization is not a restriction and efficient. Host-level synchronization can also benefit from the availability of hardware timestamps and precision clocks on physical network cards. This feature is only scarcely available with virtualised network solutions, e.g., on some SRIOV-enabled network cards. We synchronize the host and the high-precision clocks using linuxptp [2]. We continuously monitor the synchronization state of the host since awareness of possible loss of synchronization is critical to guarantee the timely and correct intervention and coordination of the protection functions.

CPU Cache

CPU cache is essential to satisfy data access timing needs of real-time applications. It prevents orders-of-magnitude higher waiting time for data retrieval from main memory. However, on most modern CPUs, the last level cache (LLC) is shared among cores. Sharing such a CPU, even with core isolation in place, can endanger deterministic timing through unexpected waiting times introduced by cache invalidation by other applications. Modern Intel server CPUs provide cache allocation technology (CAT) to reserve and exclusively allocate portions of cache to CPU cores to improve determinism of data access. Typically, cache can be allocated in 10% chunks and is subject to certain hardware limitations in terms of overlapping assignments. CAT is part of the wider Intel Resource Director (RDT) framework which also supports memory bandwidth restrictions and monitoring of the shared resources i.e., cache and memory bandwidth. Docker containers do not support cache allocation as part of the framework. We developed our own solution as a thin layer on the host to allocate cache to the reserved CPU cores attached to the launched containers. However, the Open Container Initiative (OCI) integrated Intel RDT add this control capability into container runtime configuration [5].

VPC Docker Image

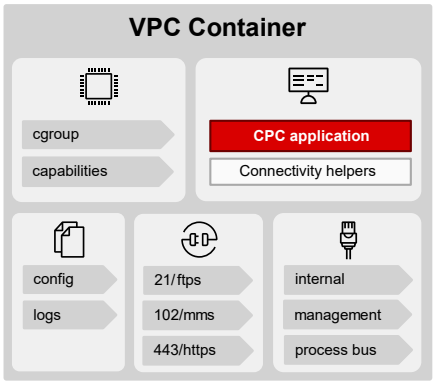

Figure 1. VPC Docker Container. Chevron shapes indicate connection points to host (resources, volumes, ports, networks).

The core of the containerized VPC is a docker image that encapsulates the application and all required dependencies as well as support utilities (Figure 1). The docker image is described by a Dockerfile, which documents the required steps for building containerized VPC and contains a base OS image, helper applications, the VPC binaries, management scripts and a root filesystem with the required

1 Intel Resource Director Technology (Intel RDT)

utilities and libraries. The container exposes various connection points to the host, such as cgroup configuration, volume mounts, network ports, and network connections.

Running the Container

Containers each get their own namespace and cgroup assignment to ensure isolation, including for example, CPU core affinity with the *--cpuset-cpus* option. Care must be taken since, contrary to LXD [6], docker does not remap core numbers to those available in the container's cgroup. Further, Linux capabilities allow to control which operations are allowed inside containers. To achieve real-time performance, we need to equip the containers with some extra capabilities such as SYS_NICE and IPC_LOCK which are required for real-time scheduling and memory locking. We explicitly drop NET_ADMIN to ensure that VPC applications cannot change the network configuration of the host as we specify everything during container creation and execution. In our experiments, we managed the containers with *docker-compose*.

PERFORMANCE EVALUATION

The studied VPC application performs power system protection and control in MV substations for distribution applications. For this evaluation, we require a threshold on maximal task execution time of *1ms* to ensure proper cyclic operation of all control tasks. In the following, we present evaluation results of the impact of various design choices and different shared setups on these critical timings of the containerized VPC in rather short experiments of a few hours to days. The application under test is running a practically relevant configuration which can handle numerous advanced protection functions simultaneously for 20 IEC 61850-9-2LE SV streams, with 4800 samples per second each (60 Hz grid). The configuration includes demanding protection functions, e.g., multi frequency admittance-based earth fault protection, and represents a case of heavy protection load in the CPC application.

The VPC software is equipped with self-monitoring capability and provides rich statistics on its various internal measurements. The desired maximal task timings are available in the log output, statistically aggregated over the monitoring time interval. In most experiments, we present the application task timings, reported either as a time series or a boxplot of aggregated duration values. The values are aggregated as maximal task duration over the monitoring interval (aggregated over 20s, yielding a total of 20k timing measurements per reported point in the series or boxplot).

Process Isolation

A lack of isolation between the application and other loads on the host system can be detrimental to real-time performance. We tested isolation properties of containerized execution with respect to other system loads which either represent non-critical OS-level tasks or another real-time application running on the same host. We represented loads with an additional VPC instance of a smaller configuration (real-time load) or the *hackbench* tool (non-rt load). We reserved 4 CPU cores for the VPC application using the kernel command line argument *isolcpus* and kept OS tasks, IRQs and other applications on the remaining cores of the CPU. With exclusive core assignment, real-time scheduling priorities (>50) and cache allocation (CAT), the application isolation is good enough to support real-time operation also under heavy load conditions (Figure 2). The experiment was run for 3 hours on two systems, a Xeon Gold 6208U CPU with 16 cores on Linux Kernel 5.10.65 and on a less capable system based on an Intel Xeon Silver 4208 with 8 cores, running an older Kernel. We observed higher performance on the Xeon Gold system, especially without real-time load due to the higher amount of cache available in those situations (note that the cache had to be split with second VPC under RT load conditions).

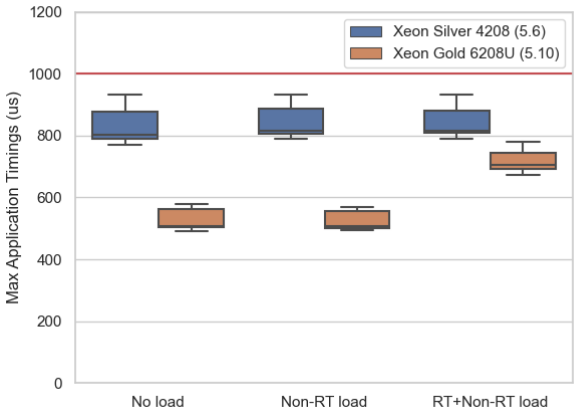

Figure 2 Distributions of maximal application timings are acceptable under 3 different load conditions on 2 different CPUs running 2 kernel versions.

Process isolation with *cpusets* was less effective. After a clean boot and setup of isolation, we still counted some kernel processes on each core isolated with *isolcpus* and even more kernel processes with *cpusets*. We therefore still applied the *isolcpus* option, despite it being marked as "deprecated".

Networking

To share the physical network interfaces and connect the VPC instances to the process bus (and management network) we need to rely on virtual networking. We tested different technologies including two pure software-based solutions (Linux bridge and MACvlan) and one hardware-assisted solution (SR-IOV). The three technologies lead to highly different latency especially under load which we reported in detail [7]. While SR-IOV achieves the best performance, it is not available for all network cards. Based on card compatibility and latency results we used MACvlan with optimized IRQ handling which avoids the performance degradations shown in [7]. With this solution we were able

to bound the worst-case receiving latency at the application level to 692µs (mean 315µs) with 20 SV streams running two full VPC instances in parallel on the Xeon Gold configuration over 24h.

Field Pilot Installation

To validate our solution, we extensively tested the prototype in a laboratory with hardware-in-the-loop setup by connecting the VPC to an RTDS system simulating different faults and externally verifying the correctness and timing of the different protection functions. The VPC passed all tests even with multiple instances hosted on the same host (Xeon Gold) with no fault test pattern triggering any overruns and all observed timings were within the required deadlines. However, these tests have a limited time length of a few days. To test the VPC's long-term behaviour in a real-world setup we collaborated with a Finnish utility to install a prototype VPC in a MV substation in western Finland. Here two VPC instances have been running with flawless operation for over one year with a perfect match between the behaviours of VPC instances and the physical CPC controlling the substation. More details can be found in [1].

VPC in Virtual Machines

Containers provide near-native performance and already offer a good level of isolation between applications at low virtualisation overhead. However, they still require the applications to share the host OS' kernel. To achieve additional deployment flexibility and isolation, moving from container-based virtualisation to hypervisors, we also evaluated VPC inside virtual machines. To compare performance, we collected metrics from VPCs in VMs running on KVM (5.15.55-rt48 PREEMPT RT) and VMware ESXi (7.0u3). The VMs run a more recent version of the VPC application which handles even 30 SV streams simultaneously, based on a custom Linux distribution with PREEMPT_RT patch. The performance is evaluated on a Xeon Gold 6208U host system with hardware virtualisation support VT-x enabled. We also enforce resource reservation setups as described above.

The results (Figure 3) show that the virtualisation efforts of the VPC, such as resource reservation and networking, also enable real-time operation of protection and control applications inside full VMs thus offering even higher flexibility in deployment configurations and increased isolation.

SUMMARY AND CONCLUSION

We have demonstrated the feasibility of running protection and control software for MV substation automation across different virtualisation environments on different hardware platforms. Despite the increased overhead of virtualisation, we have been able to achieve the necessary real-time performance in the VPC application. Through many trials, we determined that poor resource isolation and virtual network delays, rather than scheduling, were the primary causes of timing failures of VPC in virtual environments. With careful resource provisioning and virtual networking, we keep the 1ms application task execution time limit across a multitude of tests and even over a 1-year pilot in the field.

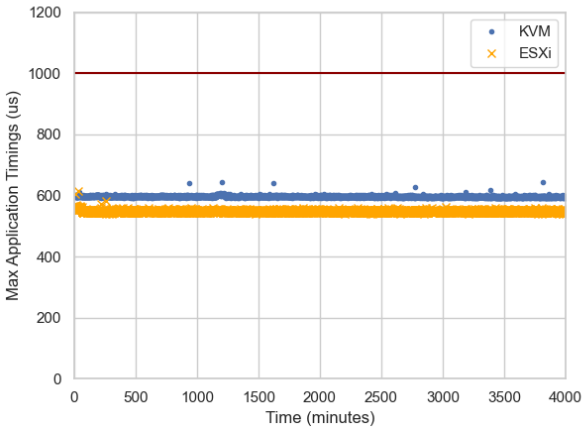

Figure 3 Maximal application task timings of VPC running inside VMs on two different hypervisors for 70 hours.

REFERENCES

[1] J. Valtari, A. Kulmala, S. Schönborn, D. Kozhaya, R. Birke, J. Reikko, "Real-life Pilot of Virtual Protection and Control – Experiences and Performance Analysis", The 27th International Conference and Exhibition on Electricity Distribution (CIRED), 2023

[2] Richard Cochran, et. Al. "The Linux PTP Project", available online: https://linuxptp.sourceforge.net/

[3] Thomas Gleixner, et. Al. "The RTL Collaborative Project", available online: https://wiki.linuxfoundation.org/realtime/start

[4] Precise Networked Clock Synchronization Working Group, "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems", IEEE 1588v2

[5] Open Container Initiative, "Linux Container Configuration", available online: https://github.com/opencontainers/runtime-spec/blob/main/config-linux.md

[6] linuxcontainers.org, "LXCFS", available online: https://linuxcontainers.org/lxcfs/introduction/

[7] G. Albanese, R. Birke, G. Giannopoulou, S. Schönborn, T. Sivanthi, " Evaluation of Networking Options for Containerized Deployment of Real-Time Applications". In IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2021.