Evaluating Industrial Applicability of Virtualization on a Distributed Multicore Platform

Nesredin Mahmud
Mälardalen University
Högskoleplan 1, 721 23 Västerås, Sweden
nesredin.mahmud@mdh.se

Kristian Sandström, Aneta Vulgarakis
ABB Corporate Research Center
Forskargränd 7, 722 26 Västerås, Sweden
kristian.sandstrom@se.abb.com
aneta.vulgarakis@se.abb.com

Abstract—**Adoption of virtualization technology has been limited in industrial automation due to unavailability of mature solutions, and strict timing requirements of control systems. However, current advancement in Virtual Machine Monitor, multicore technology, virtualization extension and network virtualization has led to increased interest of virtualization in industrial automation.**

So far, many related research are focused on maximizing CPU and I/O utilization, and optimization applicable to soft real-time systems (i.e., outside industrial automation domain), e.g., multimedia applications. In this research, we make use of QoS for CPU, memory and network bandwidth in pursuit of high speed and predictability on a distributed multicore platform which is constructed entirely from open source products. We evaluate the platform for latency and jitter, network throughput and CPU computation load. Finally, we analyze the result for applicability in industrial control domain.

I. Introduction

Industrial automation covers many different domains of control, e.g., process automation, motion control and discrete control, and many different applications such as pulp and paper manufacturing and assembly lines. In common for most of the systems are strict requirements on reliability and availability, and the fact that they are control systems and require timely behavior. Although the systems require real-time responses to events, the wide spectrum of applications covers timing requirements ranging from micro seconds to seconds. Industrial automation systems typically consist of large number of connected devices such as controllers, sensors, and actuators. In large process control systems the number of devices can reach many thousands, and also include powerful servers that provide data management, processing, and high level control, as well as connection to systems for manufacturing and business processes. At the controller level the interconnection between controllers are commonly realized using standard Ethernet communication. For connecting I/O devices to controllers there are a multitude of field buses using different technologies, e.g., Controller Area Network (CAN), Profibus; although, several recent field bus technologies are Ethernet based, e.g., EtherCat [1].

The industrial automation systems follow some general trends such as increased connectivity, more data, higher bandwidth networks, adoption of multicore processors, and increased amount of memory. In general there is an interest in adopting standard components and technologies where appropriate. The advances in hardware with e.g., multicore and hardware assisted virtualization, as well as in software e.g., with real-time hypervisors [2], opens up for new software architectures in industrial automation systems. A lot of new software technologies for flexible and efficient management of hardware and software resources can be found in general server systems, data centers and cloud based systems. Many of these technologies are developed as large open source projects with many contributors and stakeholders resulting in rapid development of features.Although these systems are developed for other target domain than industrial automation, many of the applications have high requirements on reliability and availability. The requirements of reliability and availability are manifested through the support for e.g., advanced features for redundancy. The timing requirements of these general systems though may not be comparable to industrial systems and it is therefore of interest to find out the applicability of these technologies in industrial automation with respect to temporal behavior.

From our experimental setup, we measure a round-trip time from initiating an action in a user space application of a control node to a user space application of another node (virtual or physical node). This measurement could be interpreted as what *cycle times* that could be reached involving two cooperating virtualized nodes. This includes time spent in the complete hardware and software stack, such as user application software, operating system, virtualization software, network virtualization, and network hardware. According to a study on timing requirements, different industrial control applications have different *cycle times* requirements. For high speed I/O applications, such as Analog-to-Digital converter, Digital-to-Analog converted, digital I/O, the *cycle times* required to perform such functionality reaches $100\mu s$. For motion control applications and fast control loop applications, the *cycle times* requirement could reach 1ms and 10ms, respectively.

The remainder of this paper is organized as follows. In Section II, we describe our evaluation platform architecture including a virtualized multicore framework and a baseline framework. In Section III, we setup the virtual multicore framework using open source software products with QoS for

CPU, memory and network bandwidth. Further, in Section IV, we conduct measurements for latency, jitter, network throughput and CPU computation on both frameworks with the setup intact. In Section V we analyze the measurement results for applicability in industrial automation. Finally in Section VI, we review the related work, and in Section VII we conclude the paper.

II. Evaluation Platform Architecture

A. Hardware components

The evaluation platform depicted in Figure 1 consists of two frameworks: *baseline framework* (Figure 1a) and virtualized *multicore framework* (Figure 1b). Each framework is distinct and comprises two computer nodes which are connected using an Ethernet switch. The virtualized multicore framework nodes (Node3 and Node4) are virtual hosts, each providing service to four virtual nodes (node3.0-3, node4.0-3) connected to the external network using virtual switch. The baseline framework is used for comparison to evaluate the relative cost of virtualization solution.

Computer nodes employed in this evaluation platform are HP Compaq Elite 8300 Ultra-Slim Desktop; and the switch is Cisco 2960 series. Each HP Compaq consists of 4x Intel(R) Core(TM), 4G RAM and 1G Network Bandwidth. The CPU type is Intel® Core ™i5-3470S Processor; and supports Intel® Virtualization Technology (VT-x) and for directed I/O (VT-d). The virtual nodes are installed with lightweight Linux distribution. And node3 and node4 are installed with Ubuntu-12-04 LTS. The kernel type employed in all Yocto and Ubuntu distribution are patched with Real-Time support.

B. Open-source Software Products

The software components used in the evaluation platform are all open source. We have used the xen hypervisor, Open-vSwitch, PTPd daemon, Yocto Linux distribution and Ubuntu distribution. We have chosen to use these products because they are mature in functionality and documentation, and the community support is active to receive and fix bugs. In addition, these products are already used in production that makes our evaluation result applicable in practice. Following this, we give a description of the application of these software products in our evaluation platform.

Xen Hypervisor

The xen® Hypervisor [3] is a type-1 (bare-metal), open source Virtual Machine Monitor (VMM) developed by then Xen Project team; also virtualization enabler in Xen® Cloud platform and Xen® ARM. It manages CPU, memory and interrupts to virtual machines. Xen hypervisor relies on a privileged virtual machine (aka driver domain) to manage I/O transactions and virtual machines management. Three types of virtual machines are supported in Xen: paravirtualized Virtual Machine (PVM), Hardware-assisted Virtual machine Machine (HVM) and Paravirtualized Hardware-assisted Virtual Macihne (PVHVM). PVM is a lightweight and efficient machine which does not require virtualization extension from a host CPU; however, it requires virtual machines with a PV-enabled kernel and Paravirtualization (PV) drivers, which makes it not suitable to adapt any type of operating system without modification. HVM, on the other hand, depends on CPU virtualization extension; and has low I/O throughput as compared to paravirtualized machines due to QEMU emulation overhead. QEMU emulation provides virtual machines services such as I/O, interrupt and BIOS services. PVHVM implements the best features of PV and HVM, e.g., improved I/O performance as compared to HVM, and support for unmodified host operating system as compared to PV.

Virtualization extensions, e.g., Intel® VT [4], ARM Architecture virtualization extension and Large Physical Address Extension (LPAE) , AMD Virtualization (AMD-V™, are improving flexibility and robustness of virtualization solution. As a result, in our evaluation, we choose PVHVM virtualization due to support for virtualization extension in favor of PV, and high I/O throughput as compared to HVM virtualization [5]. We created four virtual machines (or domUs) on a Logical Volume Manager (LVM) partition on node3 and node4.

Open vSwitch

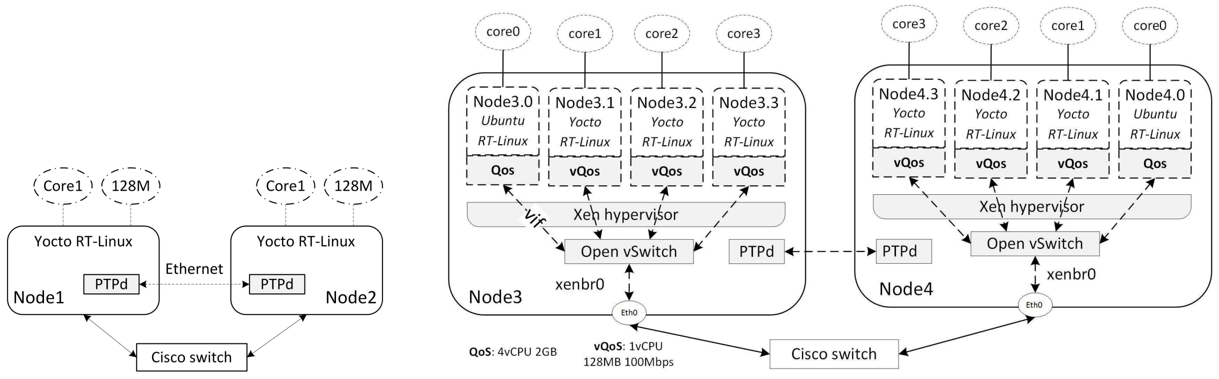
Virtualization has imposed stringent requirements on how inter- and intra-virtual machines are networked and communicated. Open vSwitch [6] is a multilayer virtual switch specially designed to address network virtualization requirements similar to VMware's vNetwork, Cisco's Nexus and Indigo's IVS. It supports advanced features found in standard switches; and features which are unique requirements for virtualized environment, such as the ability to expose external interfaces for remote and programmable automated control of distributed virtual switches placed across multiple physicals.

In this evaluation platform, an open vSwitch bridge, *xenbr0* is created over *eth0* where virtual nodes are connected to the bridge using virtual interfaces. Virtual interfaces are created and attached to virtual machines using hot-plug script provided in the xen software package. OpenvSwitch is also used to throttle bandwidth of virtual machines in the virtualized multicore framework.

Yocto Linux Distribution

Industrial control systems are application specific and operate under limited hardware resources such as memory and CPU. Therefore, developing application and building an operating system for such systems is not a trivial process. Usually a simple cross compiler creates executable code for a target system in which the compiler is not running. However, this is not sufficient if we are developing for wide range of hardware architectures and platforms for two main reasons. First, due to the demand for more functionality, software packages built into control systems are growing in size which proportionally increases build time. Second, integration of tools, such as cross compilation, emulation, debugger, application toolkit, is key for a seamless build process and package management.

The Yocto Project fills these gaps found in conventional cross compilation process. It is an open source collaboration



(a) baseline framework

(b) Virtualized multicore framework

Fig. 1: Evaluation platform

project which delivers templates, tools and methods to help create custom Linux embedded systems for several hardware architecture, e.g., ARM, PPC, MIPS, x86 and x86-64. Similar systems are Openmoko, Emdebian Grip, OpenEmbedded. As compared to these solutions, Yocto project has strong collaborations with OpenEmbedded, especially on *core component recipes* which are meta-data for software packages in a Linux distribution.

In our evaluation platform, we generated a lightweight real-time Linux distribution of a size less than 100MB based on our specific CPU platform (i.e., Intel® Core$^{\text{TM}}$i5-3470S processor) using the Yocto project. Specifically, we used BitBake, which is a Yocto cross compilation build engine, and Yocto Application Development Toolkit (ADT) [7], to develop microbench applications for network performance and CPU computational load measurement. The microbenchmark applications are *syncproof* and *cpuload*; and are discussed in Section IV.

PTP daemon

Time driven communication in distributed control systems demand high accuracy of clock synchronization. Due to the inherent property of message delay in communication, it is impossible to avoid a deviation of time in distributed clocks, called *sync error*. Lundelius and Lynch [8] proved that *n* clocks cannot by synchronized to one another with an error less than *d(n-n/2)* units of sync error when there are no failures and no clock drifts; where *d* is an uncertainty delay between clocks. However, distributed clocks can be synchronized to acceptable level on a specific application domain. Similar clock synchronization algorithms are found in [9] [10] [11] [12]. For real-time distributed control applications, hight precision of clock synchronizations is vital in order to minimize jitter and increase over all performance of a control system. In our platform, we used PTP daemon (PTPd) [13], which is an open source software implementation of Precision Time Protocol (PTP), to synchronize node3 and node4.

PTPd implements both versions of PTP (i.e., IEEE 1588-2000 and IEEE 1588-2008) [14]. PTP is defined to achieve a high clock sync accuracy on a local network that could not be attained using a protocol such as the Network Time Protocol (NTP). The NTP is a widely used protocol in the Internet network; and has variation in granularity of 1 millisecond. As a result, NTP is not suitable for industrial control systems whose requirements are in microseconds. PTPd currently runs on Linux, cLinux, FreeBSD and NetBSD. The IEEE 1588 standard defines PTP as a master/slave architecture; and it consists of network segments and multiple clocks. A *network segment* identifies a communication media and provides isolation of synchronization. A *normal clock* is a master or slave clock in a single network segment; and a *boundary clock* bridges synchronization between one or more network segments. The Best Master Algorithm (BMA) [14] selects which clock acts as a master per network segment. Further, based on the average Round Trip-Time (RTT) from slave to master and back, each slave calculates the clock sync error and adjusts its clock to the master clock.

III. Evaluation Setup

In this section, we describe the setup for the virtualized multicore framework. In the context of xen virtualization, we use the term *driver domain* (aka *dom0*) to refer to the host machines (i.e., node3.0 and node4.0) on the virtualized multicore framework. Similarly, we use the term *domU* to refer to guest machines (i.e., node3.1-3 and node4.1-3) on the virtualized multicore framework.

A. Domain Pinning to CPU cores

The Xen Credit scheduler [3] is the default scheduler in Xen. It schedules Virtual CPUs (vCPUs) with proportionally fair CPU time and high throughput. It uses the concept of a *credit* to schedule vCPUs according to parameters such as *weight* and *cap*, which respectively refers to CPU allocation ratio as compared to other domains and maximum amount of CPU a domain is allowed to consume regardless of idle CPU time a host domain has. Basically, the algorithm is designed to maximize throughput and fairness in contrast to predictable execution of domains. In this setup, we pinned domains to a dedicated CPU core in order to avoid preemption of domains running on xen hypervisor. This implies, the credit scheduler is exempted from scheduling domains.

Each domain core-affinity including the driver domain is set to a single core. A core is not shared to other domains. This establishes a one-to-one mapping between a core and a domain. Furthermore, each core is scheduled with a dedicated scheduling instance in contrast to pinning a domain to a specific core with a shared scheduler. In the recent version of xen (v4.2) this concept is realized using *cpupools*. Figure 2a shows credit scheduler execution trace on node3 in the virtualized multicore framework before pinning. The driver domain (or node3.0) is represented with **A** characters; and domUs (or node3.1, node3.2 and node3.3) are represented with characters **B**, **C** and **D**, respectively. The trace on figure 2a shows how the domains are distributed across core0-3. After pinning the domains, figure 2b shows each domain running on a dedicated core without preemption from one another. The trace is captured using *xentrace* and *xenanalyzer* tools.

B. Virtual Computational Resource (VCR)

A Virtual Computation Resource (VCR) is a resource reservation as defined by Clara et al. [15]. In this context, VCR refers to QoS for CPU, memory and network bandwidth, which is a computing resource definition generated by partitioning the actual hardware resources for allocation, verification and analysis of application parts. A VCR allows portability of application parts to multiple hardware architecture while preserving timeliness and predictability of an application; e.g., porting an application from a single core to a multicore while maintaining previous application execution behavior. In this section, we develop a VCR and link it to virtual machines. We use the term QoS in places of *VCR* since it is more familiar.

QoS for CPU

In multicore systems, the number of vCPUs allocated to a virtual machine affects CPU utilization and I/O throughput in the system. This is largely discussed in [16], xen hypervisor evaluation for hosting IP telephony application, and best practices from the xen community. In our framework, we establish a one-to-one mapping of a virtual machine to a core; therefore, employing multiple vCPUs per virtual machine does not maximize utilization of CPU or I/O throughput of the system. To maximize predictability of applications running on each virtual machine, we avoid parallel execution of threads on a single domain.

The driver domain handles I/O communication, such as network and disk transaction of virtual machines, and is usually a bottleneck for I/O intensive and latency sensitive control applications. To avoid this bottleneck the best practice would be to allocate sufficient computing resources including CPU and memory to the driver domain [17]. By default, Xen allocates 4 vCPUs initially to driver domain; and the number grows linear to the amount of load driver domain has.

QoS for Memory

Efficient memory management is a challenging task in virtualization. The knowledge of current and future memory requirements of a virtual machine is an important input to the memory controller so that it can efficiently manage system memory. Several memory optimization techniques are implemented in the xen hypervisor, such as xen memory paging [3], paravirtualized paging [18] and dynamic memory virtualization [19]. However, these techniques do not estimate accurate memory consumption of a virtual machine over time. The main reasons to this challenge is that a working set of a virtual machine cannot be measured accurately to determine current required memory. It is also not feasible to accurately predict future memory usage of a virtual machine due to unpredictable application memory requirements. Therefore, a memory intensive application on a particular virtual machine has the tendency to raise memory thrashing, a problem which may disrupt the behavior of a real-time application running on other virtual machines. Therefore, it is important that static memory is allocated to virtual machines to minimize the problem.

Since the driver domain is usually the bottleneck for CPU and I/O performance, in our evaluation framework we allocate to the driver domain more memory (i.e., 2GBytes) than to domUs (i.e., 128MBytes). Note that Xen hypervisor implements a balloon driver [20], a technique which grabs free memory and allocates it to the driver domain. This may result in memory starvation in domUs. In order to avoid memory starvation on domUs; and increase predictability, we disable this feature in the xen hypervisor.

QoS for Network Bandwidth

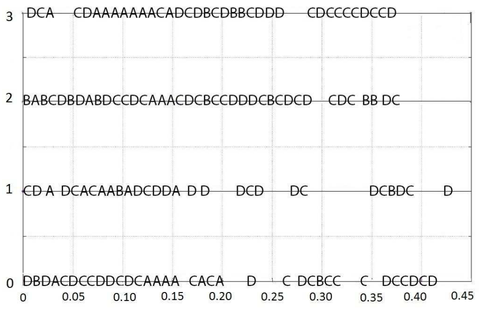
Open vSwitch is a shared resource among virtual nodes. Therefore, it is possible that a traffic intensive or misbehaving application on a particular virtual node could disrupt other virtual nodes. This could result to deadline miss of higher priority applications running on other virtual nodes. In these kinds of scenarios, QoS for network bandwidth can be configured for each virtual node on the virtual switch, which guarantees maximum bandwidth limit. The *Xenbr0* bridge has a capacity of 1000Mbps; and we have limited the virtual interfaces ingress rate policy to 100Mbps and ingress burst policy to 1Mbps (for that matter it could be any value between 0 and 1000 as long as one has managed to generate sufficient traffic to flood the interface).

IV. Result

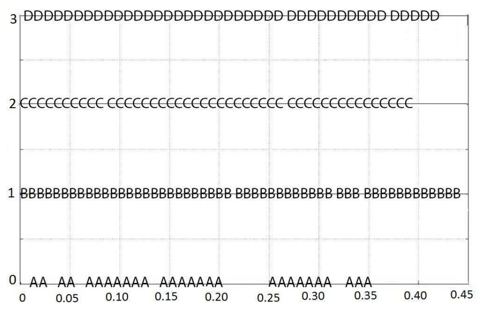
In this section we discuss our benchmark applications, workloads, and measurement procedure. Then, we show the evaluation results in a table.

A. Benchmark Application

The evaluation is focused on the frameworks as a whole in contrast to evaluation of software products in isolation. We evaluate the frameworks against performance metrics, i.e., latency and jitter, network throughput and CPU computation



(a) Default Xen scheduler



(b) Tuned Xen scheduler

Fig. 2: Domain execution trace vertical axis(cores), horizontal axis (elapsed time in millisecond).

with the help of latency-sensitive workload, bandwidth intensive workload and CPU-intensive workload, respectively. Figure 3 shows the client-server architecture of the benchmark tools (Iperf, Syncproof, Cpuload) which traverses the TCP/IP stack, operating system and physical link on the evaluation platform.

Latency-sensitive workload

In this context, *latency* is the amount of time elapsed for an Ethernet frame to make a round-trip from a client node to a server node, and back to the client node. *Jitter* is a variation of delay (measured in microseconds) due to a clock synchronization error, operating system overheads and application level overheads. We use the *syncproof* network performance tool to measure latency and jitter of Ethernet based communication. The tool is developed in-house and is based on client-server architecture. It emulates real-time network communication. The client and server applications are guaranteed to run with higher priority compared to other processes running in the Linux operating system. As such, preemption by other processes is minimized; and latency measurement accuracy is increased.

In figure 1a, the *syncproof* client runs at node1; and the *syncproof* server runs on node2. Likewise, in figure 1b, the *syncproof* client runs at node3.1; and the *syncproof* server runs on on node4.1. The measurement procedure is describe in the subsection IV-B.

Bandwidth and CPU intensive workload

Iperf [21] is an open source network performance tool used to measure the maximum TCP bandwidth and the UDP communication behavior, e.g., delay jitter and packet loss. Using this tool, we perform two experiments. First, we measured maximum TCP bandwidth of the baseline (see Figure 1a) and the virtual multicore (Figure 1b). Second, we run a bandwidth-intensive workload on virtual nodes, node3.2 and node3.3 using the *iperf* tool; and evaluate its impact on latency-sensitive application running on the virtual node, node3.1.

Similarly, with a CPU intensive workload, we measure the amount of time (measured in seconds) required to do a floating-point computation on node1 and node3.1 (see Figure 1).

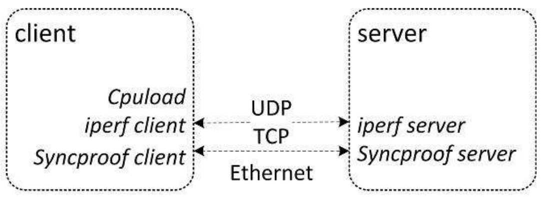


Fig. 3: Logical toplogy for running microbenchmark application

node	core	vCPU	Memory	BW Limit
node1-2	0	N/A	2G	no
node3.0	0	4	2G	no
node3.1-3	1-3	1	128M	100Mbps
node4.0	0	4	2G	no
node4.1-3	1-3	1	128M	100Mbps

TABLE I: Platform setup summary

B. Measurements

Measurements are taken from figure 1 according to our setup. For clarity we summarize the setup in Table I. We perform three experiments:

- Experiment 1: Latency and jitter measurement
- Experiment 2: Network throughput measurement
- Experiment 3: CPU load measurement

Experiment 1: Latency and jitter measurement

In Figure 3, the Syncproof client sends an Ethernet frame of Maximum Transmission Unit (MTU) equals to 1500 bytes to the Syncproof server. The number of frames sent is 10,000 with an inter-arrival time of 100 ms, a sufficient round-trip time before the client tries to send the next frame. Then, the average latency and the jitter are computed at the client side. This test is repeated 400 times. Table II shows the average latency and the average jitter on the baseline framework and virtualized multicore framework.

We compared our result to a similar work presented by Patnaik et al. [16] on the performance evaluation of Xen for a telephony application with a similar xen scheduler and CPU configuration. In contrast to our work, Patnaik et al. allow multiple virtual machines to run on a single core. Our experimental result for the RTT latency is 30% lower on

	Avg. Latency	Avg. Jitter
Baseline	265.66	5.72
PVHVM	344.96	16.90

TABLE II: Latency and jitter in microseconds

the PVHVM which shows that one-to-one mapping of virtual machine to core has better performance as compared to other xen scheduler and CPU configuration setups used in [16] [22].

A one-way latency distribution for both frameworks is show in figure 4. The Average latency (RTT) can be calculated by multiplying a one-way latency by two. The distribution is modeled as a frequency distribution with a sampling period of 1 millisecond. This distribution tells how often latency values occur within a range of latency values. Throughout the latency test for as long as ten hours or 400 request-response times using the *sycproof* tool, some long tail latency is observed, especially in the virtualized multicore framework. As compared to the baseline framework, the virtualization multicore framework experiences some long tail latency, e.g., as shown in the far right of figure 4b. However, the latency distribution of the virtualized multicore framework closely follows the frequency distribution of the baseline framework; and it also shows improvement over results gained by Xu et al. [23]. And this improvement is due to boosting the responsiveness of latency-sensitive application running on a virtual machines. This is accomplished in two ways in our setup. First the virtual machines are not preempted by another virtual machines in the xen hypervisor; second, the *syncproof* client and server instances which simulate latency-sensitive applications run as real-time Linux processes, hence higher priority as compared to non real-time processes running in the Linux operating system of the virtual machines.

Experiment 2: Network throughput measurement

The Network Interface Card (NIC) bandwidth capacity of the client and server nodes is 1Gbps. The open vSwitch is configured to handle 1Gbps traffic so that it will not be a bottleneck. The Iperf client sends frame of MTU (1500 bytes) to the Iperf server, as shown in Figure 3. A total of 195 GBytes is transmitted from the client to the server. The network throughput is 934.26 Mbps and 932.05 Mbps for the baseline and the virtualized multicore frameworks, respectively. According to a study by Aravind Menon et al. [24] on performance overhead in the xen hypervisor, the reason for a degradation of a network throughput on the xen hypervisor was due to higher instruction cost, higher L2 cache misses and lower Translation Lookaside Buffer (TLB) miss rates which led to more instruction stalling as compared to a non virtualization Linux environment.

To observe the degree of virtual nodes isolation on the virtualized multicore framework (presented in figure 1b), we run bandwidth-intensive workload that is capable of generating around 700 Mbps using TCP Iperf running on virtual nodes, node3.1 and node3.2. Then we conduct *Experiment 1* once more to measure latency and jitter. The result turned out to be the same as in Table II. We expected this result to be the same due to the fact that node 3.1 and node3.2 are throttled to 100 Mbps using open vSwitch.

Experiment 3: CPU load measurement

We run compute-intensive workload using a *cpuload* application on a non-virtualized node (i.e., node1), and virtualized node (i.e., node3.1) to measure CPU load computation. The average amount of time required to compute the workload is 10.16 seconds in node1, and 12.13 seconds in node3.1. Even though node1 and node3.1 have the same CPU and memory resource allocation, node1 performed better with 16% advantage. The Xen virtualization overhead is discussed in this paper [24].

V. Discussion

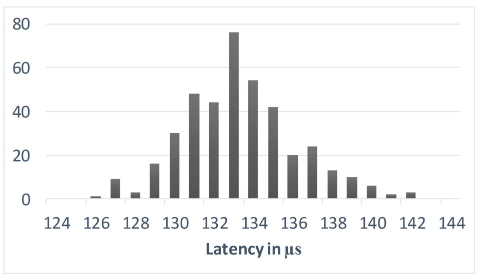
Our evaluation results from Section IV-B show the performance of the virtualized multicore framework (Figure 1b) relative to the baseline framework (Figure 1a). The results indicate the additional cost in terms of latency, jitter, bandwidth throughput for introducing a virtualization technology. This additional cost does not however directly say anything about the applicability of the technology from a performance perspective in industrial automation systems. In this section, we present a discussion on how the results from the evaluations can be interpreted with respect to applicability in industrial automation. We compare the results from the experimental set-up with cycle times found in different industrial application types as summarized in Table III. As stated in section I, the cycle times describes a round-trip time from initiating an action in a user space application on one node to the user space application of another node (virtual or physical node).

A. High speed I/O control applications

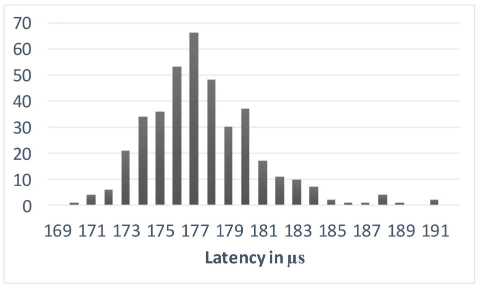
Speed is vital in high speed I/O control systems to perform, e.g., A/D, D/A, digital I/O, counter/timer, communication and other functions. Observe Table III, the cycle times required to perform such functionality reaches $100\mu s$. In this case, the applicability of virtualization technology is unlikely, as we can compare it to the PVHVM in Table II. High speed control applications are usually hosted on dedicated I/O cards, DSP or FPGA.

B. Motion control applications

Motion control applications serve to control the position or velocity of an object using devices such as electric motors. They are applied in robotics, semiconductor production and industrial processes, such as packaging and assembly industries, and in other automation fields. Precision motion control applications, such as in the field of semiconductor for wafer probing and memory repair, and in optical manufacturing require high-speed, and sometimes ultra-precision positioning functionality, usually in the range of 1ms ($1000\mu s$). Our evaluation results show such precision. Therefore it is possible to satisfy the real-time requirement of motion control applications on Ethernet based cooperating nodes using a xen PVHVM virtualization



(a) Baseline



(b) Xen PVHVM

Fig. 4: Frequency distribution of one-way latency, sampling period 1 millisecond

Cycle time	Description
100μs	*High speed I/O applications* - often managed by dedicated I/O systems, FPGAs, or DSPs
1ms	*motion control applications* - Commonly found in semiconductor production, industrial process, e.g., packaging, printing, industrial assemblies
10ms	*fast control loops application* - usually found industrial automation domains and applications, e.g., steam turbine control systems
10-100ms	Common, e.g., in process automation
>100 ms	Slow control systems

TABLE III: Cycle time requirements for various industrial control applications.

solution with 345μs as compared to what is needed for these types of applications, that is 1000μs.

C. Fast control loop applications

High speed control systems, e.g., turbine controllers, operate with *cycle times* in the range of 5-10ms. This is the amount of time required to sense speed, compute and adjust the level of steam in case of steam turbine. It is therefore possible to host such application on a xen PVHVM virtualization technology (see Table II for latency or cycle times comparison). Also Xen PVHVM virtualization in our virtualized multicore platform shows applicability for the majority of the control loop systems whose cycle times reach 100ms; and for slow control systems with cycle times greater than 100ms.

VI. Related Work

Somani et al. [25] evaluated the xen hypervisor isolation strategy by running a CPU intensive and I/O intensive applications on a separate domains deployed on a single server. The test is conducted on a default configuration of the xen credit scheduler and SEDF. In our research, we pinned each domains to a dedicated CPU core; and applied QoS for CPU, memory and network bandwidth to virtual nodes before conducting evaluation process.

Lee et al. [17] and Patnaik et al. [16] independently worked on tuning the xen scheduler for optimal performance of a multimedia application. Lee et al. emphasized measurement metrics such as sched_count, sched_latency and sched_timeslice to identify bottleneck in the xen hypervisor, and to describe the performance of the media application. On the other hand, Patnaik et al. used metrics such as UDP/TCP packet delay and loss to describe performance of telephony application on a multicore platform. However, neither of the experiments considered one-to-one mapping between a core and a virtual machine, and apply QoS (i.e., memory and network bandwidth) to virtual machines.

To improve performance of I/O intensive processing in the the xen virtualization, Ongario et al. [22] enhanced the xen credit scheduler, e.g., minimizing driver domain preemptions during processing of I/O transactions; and introduced a two-level hierarchy of bit vectors to speed up processing of event channels. In addition, the latency measurement in our setup is between two virtual nodes hosted on different physical location.

In [26] [27] the authors have already analyzed the impact of CPU overload on I/O throughput and vice versa for different xen schedulers and CPU configurations. Other interesting works on a real-time scheduler for the xen hypervisor are compositional scheduling in virtualization [28] and RT-Xen [29].

VII. Conclusion

Due to constrained temporal and computing intensive resource requirement in embedded and industrial control systems, adoption of virtualization technology had been hindered for a long time. Nowadays, advancement in multicore technology, Virtual Machine Monitoring (VMM), CPU virtualization extension technology (e.g., Intel-VT and AMD-V), and network virtualization technology (e.g., openvSwitch, VMWare NSX) has boosted virtualization performance. In this research, we explore the potential for applying mature and commonly used open source virtualization solutions to industrial control systems. We apply the concept of virtual computational resource (VCR) and define QoS for CPU, memory and network bandwidth to minimize latency and jitter, and maximize predictability of virtual node execution in distributed multicore environment. To realize the introduced concept, an evaluation platform has been developed based on the xen hypervisor, open vSwitch, Yocto real-time Linux distribution, and PTP daemon. The performance of the platform is evaluated with respect to latency, jitter, and network bandwidth, and CPU computation.

Finally, we analyzed the result for industrial control applications, such as high speed I/O application, motion control application and fast control loop application measured in cycle times. The analysis showed possible application of then xen PVHVM virtualization for many industrial control systems with average cycle time (Round Trip Time) 345μs. The drawback of the proposed solution is underutilization of CPU. Further research direction could be to address this challenge through a lightweight VMM scheduler which is applicable in industrial control systems.

Acknowledgment

This work is conducted at ABB Corporate Research, Sweden. We would like to gratefully acknowledge the Software Architecture department/ABB CRC for providing us the necessary equipment, and we would also like to acknowledge VINNOVA, Sweden, for supporting this work under the project VeriSpec, number 16335.

References

[1] H. Buttner and D. Jansen. Real-time Ethernet: the EtherCAT solution. *Computing and Control Engineering*, 15(1):16–21, February 2004.

[2] Kristian Sandstrom, Aneta Vulgarakis, Markus Lindgren, and Thomas Nolte. Virtualization technologies in embedded real-time systems. In *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–8. IEEE, September 2013.

[3] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5):164–177, October 2003.

[4] R. Uhlig, G. Neiger, D. Rodgers, A.L. Santoni, F.C.M. Martins, A.V. Anderson, S.M. Bennett, A. Kagi, F.H. Leung, and L. Smith. Intel virtualization technology. *Computer*, 38(5):48–56, May 2005.

[5] Roger Pau Monnè. Performance tunning xen. Xen.org, April 2013.

[6] Ben Pfaff, Justin Pettit, Keith Amidon, Martin Casado, Teemu Koponen, and Scott Shenker. Extending networking into the virtualization layer. In *Hotnets*, 2009.

[7] Michael Lauer. Building embedded linux distributions with bitbake and openembedded. In *Proceedings of the Free and Open Source Software Developers European Meeting (FOSDEM), Brussels, Belgium*, 2005.

[8] Jennifer Lundelius and Nancy Lynch. An upper and lower bound for clock synchronization. *Information and Control*, 62(2-3):190–204, August 1984.

[9] Jennifer Lundelius Welch and Nancy Lynch. A new fault-tolerant algorithm for clock synchronization. *Information and Computation*, 77(1):1–36, April 1988.

[10] Hermann Kopetz and Wilhelm Ochsenreiter. Clock Synchronization in Distributed Real-Time Systems. *IEEE Transactions on Computers*, C-36(8):933–940, August 1987.

[11] Flaviu Cristian. Probabilistic clock synchronization. *Distributed Computing*, 3(3):146–158, September 1989.

[12] Mikls Marti, Branislav Kusy, Gyula Simon, and kos Ldeczi. The flooding time synchronization protocol. In *Proceedings of the 2nd international conference on Embedded networked sensor systems - SenSys '04*, page 39, New York, New York, USA, November 2004. ACM Press.

[13] Kendall Correll, Nick Barendt, and Michael Branicky. Design considerations for software only implementations of the ieee 1588 precision time protocol.

[14] K Lee, John C Eidson, Hans Weibel, and Dirk Mohl. Ieee 1588-standard for a precision clock synchronization protocol for networked measurement and control systems. In *Conference on IEEE*, volume 1588, 2005.

[15] Clara Otero Pérez, Martijn Rutten, Liesbeth Steffens, Jos van Eijndhoven, and Paul Stravers. Resource reservations in shared-memory multiprocessor socs. In *Dynamic and Robust Streaming in and between Connected Consumer-Electronic Devices*, pages 109–137. Springer, 2005.

[16] Devdutt Patnaik, AS Krishnakumar, Parameshwaran Krishnan, Navjot Singh, and Shalini Yajnik. Performance implications of hosting enterprise telephony applications on virtualized multi-core platforms. In *Proceedings of the 3rd International Conference on Principles, Systems and Applications of IP Telecommunications*, page 8. ACM, 2009.

[17] Min Lee, a. S. Krishnakumar, P. Krishnan, Navjot Singh, and Shalini Yajnik. XenTune: Detecting Xen Scheduling Bottlenecks for Media Applications. *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, pages 1–6, December 2010.

[18] Daniel J Magenheimer, Chris Mason, Dave McCracken, and Kurt Hackel. Paravirtualized paging. In *Workshop on I/O Virtualization*, 2008.

[19] XiaoLin Wang, YiFeng Sun, YingWei Luo, ZhenLin Wang, Yu Li, BinBin Zhang, HaoGang Chen, and XiaoMing Li. Dynamic memory paravirtualization transparent to guest OS. *Science China Information Sciences*, 53(1):77–88, February 2010.

[20] Carl A. Waldspurger. Memory resource management in vmware esx server. *SIGOPS Oper. Syst. Rev.*, 36(SI):181–194, December 2002.

[21] Ajay Tirumala, Feng Qin, Jon Dugan, Jim Ferguson, and Kevin Gibbs. Iperf: The tcp/udp bandwidth measurement tool. *htt p://dast. nlanr. net/Projects*, 2005.

[22] Diego Ongaro, Alan L. Cox, and Scott Rixner. Scheduling i/o in virtual machine monitors. In *Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '08, pages 1–10, New York, NY, USA, 2008. ACM.

[23] Yunjing Xu, Michael Bailey, Brian Noble, and Farnam Jahanian. Small is better: Avoiding latency traps in virtualized data centers. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 7:1–7:16, New York, NY, USA, 2013. ACM.

[24] Aravind Menon, Jose Renato Santos, Yoshio Turner, G. (John) Janakiraman, and Willy Zwaenepoel. Diagnosing performance overheads in the xen virtual machine environment. In *Proceedings of the 1st ACM/USENIX International Conference on Virtual Execution Environments*, VEE '05, pages 13–23, New York, NY, USA, 2005. ACM.

[25] G. Somani and S. Chaudhary. Application performance isolation in virtualization. In *Cloud Computing, 2009. CLOUD '09. IEEE International Conference on*, pages 41–48, Sept 2009.

[26] G. Somani and S. Chaudhary. Application performance isolation in virtualization. In *Cloud Computing, 2009. CLOUD '09. IEEE International Conference on*, pages 41–48, Sept 2009.

[27] Padma Apparao, Srihari Makineni, and Don Newell. Characterization of network processing overheads in Xen. In *First International Workshop on Virtualization Technology in Distributed Computing (VTDC 2006)*, pages 2–2. IEEE, November 2006.

[28] Jaewoo Lee, Sisu Xi, Sanjian Chen, L.T.X. Phan, C. Gill, Insup Lee, Chenyang Lu, and O. Sokolsky. Realizing compositional scheduling through virtualization. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2012 IEEE 18th*, pages 13–22, April 2012.

[29] Sisu Xi, J. Wilson, Chenyang Lu, and C. Gill. Rt-xen: Towards real-time hypervisor scheduling in xen. In *Embedded Software (EMSOFT), 2011 Proceedings of the International Conference on*, pages 39–48, Oct 2011.