System performance benchmark using different virtualization techniques

Ján Nemčík
Fakulta informatiky a informačných technológií
Slovenská Technická Univerzita
Slovenská republika, 841 04 Bratislava IV
Email: xnemcikj@stuba.sk

***Abstract*—Common benchmarking approaches often involve running a set of tasks on a uniform system, where the stress tests on single resource are used to indicate the performance of the system [2]. In this paper, we are mainly concerned with different virtualization techniques and their concept of utilizing host hardware resources. Firstly, we will define one complex problem as a benchmark use case for the guest system. Our main focus will be to point out differences in system load for particular virtualization types, primarily the dependency between used number of processor cores and execution time while performing benchmark testing. Finally, the results of the actual benchmark testing will be evaluated and conducted to find out the system density and and better performance for given testing use case.**

***Index Terms*—Virtualization, Benchmark, Software virtualization, Hardware virtualization, Containerization**

I. Introduction

Nowadays, there are a lot of discussions in field of virtualized environment, becaouse of deploying servers and applications in cloud solutions. Considering the main objective of virtualization, being effective at resource sharing on physical machines [5], [11], where the virtualized machine can be configured according to a specific purpose. However, while implementing the virtual machines, performance overhead is created because of added hypervisor layer between virtual machines and hardware resources [3]. Due to these facts, it is important to be awared of potential issues in form of virtual machines performance leaks.

Virtualization is simply abstraction of underlying hardware resources [9]. It gives the ability to allow one physical machine to serve more virtual machines or resources [11].

Performing a proper benchmark scenario leads to better identification of potential bottleneck at hardware resource layer as discovered and described by Raghavendra, *et al* [8]. Kejiang Ye, Jianhua Che, Xiaohong Jiang, Jianhai Chen, Xing Li mentioned [5], that the requirements for performance testing in virtualized environment is becoming more complex due to possible performance overhead and introduced Virtual Machine Monitor as an improvement in attempt to reduce accrued overhead.

Although, there are currently many solutions to configure and manage virtualization infrastructure, such as KVM [12], VirtualBox [13] and many others, a lot of challenges in this area are still not solved. Though, these solutions already reduced the overhead of managing the virtual machines, their quick configuration and rescaling the virtual machine instance on demand. But, the added hypervisor layer between the virtual machines and host machine remains still as an possible drawback attempting to increase the power of virtualized environment [5].

In our system load and performance benchmark testing we decided to build kernel in more virtualization environments, in order to provide structured resource load for complex single process. We will provide testing in three different virtualized systems and comparing them to actual efficiency of hosted machine, which will be a user computer.

In this paper, we selected primarily two mainly types of virtualization:

- Full virtualization
- Para virtualization

A. Full virtualization

In this type of virtualization, the hypevisor layer is operating directly over the underlying hardware on the host machine [1]. It means, that the hypervisor simulates the hardware resources and allows unchanged operating system being run in isolated layer [9], but it seems like it is directly running on the physical hardware. Big advantage of this approach is, that the operating system on virtualized machine supports all the features of the emulated hardware, where it interacts with memory, storage and hypervisor isolates the processes of the guest operating system [1]. As shown in fig. 1, the architecture of full virtualization provides the hypervisor layer installed in host operating system, which controls and monitors the running guest operating system in virtual machines.

1) Software assisted virtualization: This type of virtualization entirely relies on binary translation to virtualize the instructions execution [9]. Specifically, the guest binary codes are translated to RISC [14] binary code dynamically. The major issue of this approach is the fact, that it is running in software programmed hypervisor as well as the binary translation is done by the software, which leads to significant decrease of performance and efficiency [10]. Hongqi He, Liehui Jiang, Haifeng Chen and Weiyu Dong tried to design a dynamic binary translation based translation of X86 binaries to RISC binaries has improved the performance compared to

[1]https://devnet.kentico.com/articles/running-kentico-in-a-docker-container/

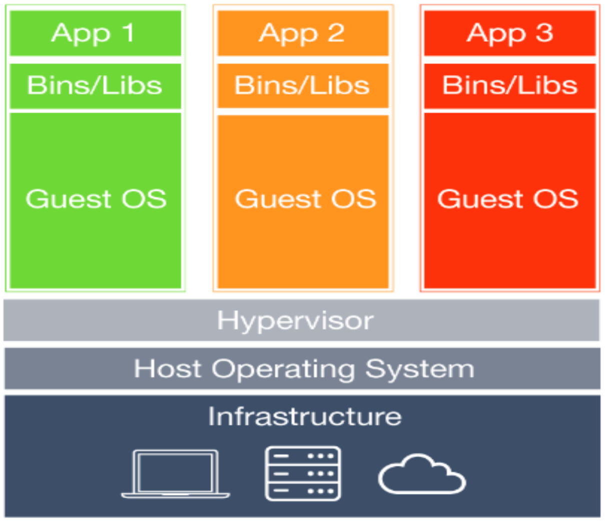


Figure 1. Layer structure of full virtualized environment[1]

fundamental binary translation but the overhead still remains [10]. Figure 2 describes the hierarchy over the layers specific for full virtualization, how the instructions are executed. As we can see, the instruction has to go through every software defined layer of the architecture.

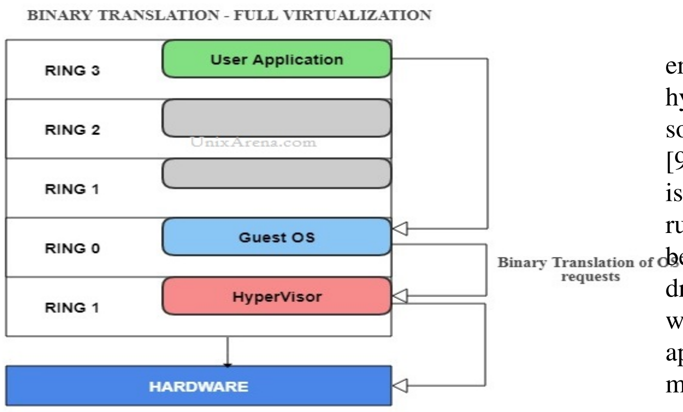


Figure 2. Binary translation execution[2]

2) Hardware assisted virtualization: Comparing to software assisted virtualization described abo seeveI-A1, hardware assisted virtualization eliminates the binary translation by interrupting directly with hardware using the virtualization technology integrated into the processors [9]. More concretely it signifies, that the privileged instructions executed on guest operating system are executed by virtual context directly on processor. By being able to simulate the complete hardware environment it is ensured, that the guest machine is using the same instruction set as the host machine [15]. Figure 3 provides an architectural overview of the instruction execution in hardware assisted virtualization. Compared to binary translation execution (I-A1), we can see that the overhead to translate the instruction is observed across all layers and the applications running in hardware assisted virtualization are directly executing the requests on hosted hardware of physical machine supported by Intel VT-x[3] or AMD-V[4] technologies.

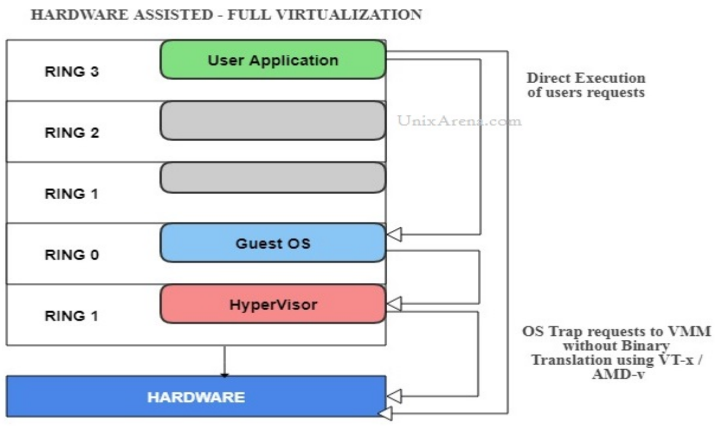

HARDWARE ASSISTED - FULL VIRTUALIZATION
RING 3
User Application
RING 2
RING 1
RING 0
Guest OS
RING 1
HyperVisor
HARDWARE
Direct Execution of users requests
OS Trap requests to VMM without Binary Translation using VT-x / AMD-v
UnixArena.com

Figure 3. Hardware assisted execution[5]

B. Para virtualization

Para virtualization is an approach in which the guest operating system is aware that it is operating directly on the hypervisor instead of the emulated underlying hardware [1], so it doesn't have to simulate hardware for the virtual machines [9] .In this type of virtualization a supporting hypervisor is installed on the host and the guest operating system is running in the host environment. The guest source codes are being modified for execution on the host and executed using drivers assured by the hypervisor [1], [9], [16]. In figure 4 we can see the both para virtualization and full virtualization approach, as well as their operations with the hardware. The main difference between these approaches is, that as already mentioned, kernel by para virtualization is modified being able to directly communicate with the hardware.

1) Containerization: The container virtualization is a lightweight method for constructing virtual user spaces by sharing the kernel of the host operating system, so there is reduced overhead with guest operating system and virtual machine [7]. In this approach, where each operating system kernel is modified by being able to load multiple guest operating systems [1]. The modified kernel provides proper management of hosted resources to isolate one container activity from other

[2]https://www.unixarena.com/2017/12/para-virtualization-full-virtualization-hardware-assisted-virtualization.html/

[3]https://www.intel.com/content/www/us/en/virtualization/virtualization-technology/intel-virtualization-technology.html

[4]https://www.amd.com/en/technologies/virtualization

[5]https://www.unixarena.com/2017/12/para-virtualization-full-virtualization-hardware-assisted-virtualization.html/

[6]https://www.unixarena.com/2017/12/para-virtualization-full-virtualization-hardware-assisted-virtualization.html/

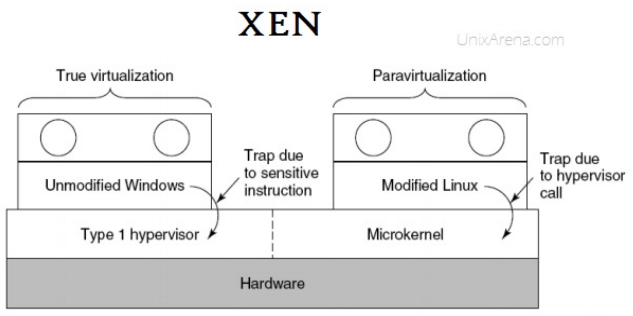


Figure 4. Para virtualization vs. full virtualization by XEN[6]

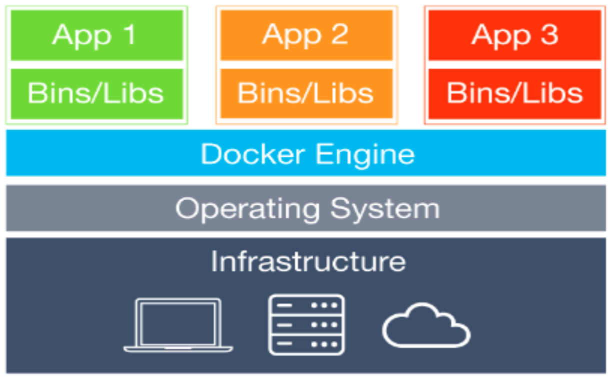


Figure 5. Layer structure of full containerized environment[7]

activities running on the host. In general, this type of virtualization has less overhead in loading the guest operating system which is in the form of containers comparing to hardware assisted virtualization, where the whole guest operating system is running in emulation. Each container has its own IP address, memory, root access and some other needed services assigned and can use them to directly execute needed instructions. Given the architecture of containerization in figure 5, we can see the difference in detail. With containerization, we run the applications only with necessary binaries in isolated processes, whereas with hardware assisted virtualization, there is need to install tho whole guest operating system and emulate the hardware of host machine.

Nevertheless, all mentioned advantages have a positive impact on performance of running containers. As Gaku Nakagawa and Shuichi Oikawa discussed [6], application running in containers can have memory leaks in their implementations which could possibly cause the failure of the host system. Therefore, if using container based environment, there is a need to be aware of described issue and constrain the usage of host resources for particular containers.

II. Virtualization Platform

For this paper, we consider following virtualization platforms to be included:

[7]https://devnet.kentico.com/articles/running-kentico-in-a-docker-container/

- Hardware assisted virtualization
- Software assisted virtualization
- Containerization

A. *Hardware assisted virtualization configuration*

To run our benchmark scenario in this virtualization environment, we will use KVM hypervisor. It is a native full virtualization solution for Linux [12] and it contains virtualization support for Intel VT-x, which is required due to physical host machine, as it has the Intel processor installed, where the virtual machine will be running. We also prefer KVM over other hypervisors, like VirtualBox or VMWare, because the installation and provisioning of virtual machines is very simple, as a result of it is native support in Linux kernel, and the provided monitoring is appropriate as well.

As the operating system, we chose Debian in version 10 over Ubuntu, because Debian is a lightweight distribution and significantly faster compared to Ubuntu. As we only need fundamental functions for our benchmark test, Debian with its functionalities is perfectly suitable for our approach.

The virtual machine was installed by following script displayed in listing 1 under KVM hypervisor, with default values for number of processors, amount of memory and disk size to be assigned to the virtual machine. The default *iso* image for installation was downloaded from official debian download mirror[8].

B. *Software assisted virtualization configuration*

Firstly, being able to install and run software assisted virtual machine, we needed to turn off hardware virtualization support from processor, specifically uncheck *Intel Virtualization Technology* and *VT for Direct I/O*. After setting up the *BIOS* configuration, we installed another virtual machine, which was now running as software assisted virtualized machine. For installing we also used script shown in listing 1. The main difference compared to our hardware assisted virtualized machine is, that in this case, running the machine KVM will automatically switch the execution hypevisor to *QEMU*, which supports software assisted emulation only.

Listing 1. virt-install.sh

```
cpus=2
memory=2048
disk_size=12
name="aps-vm"

virt-install \
  --name $name \
  --ram $memory \
  --vcpus $cpus \
  --disk size=$disk_size \
  --cdrom ./debian-10.2.0-i386-netinst.iso
```

[8]https://www.debian.org/distrib/

C. Containerization configuration

To perform benchmark container virtualization, we selected *Docker*[9] as our container hypervisor, because it provides all functionalities needed for our benchmark test and is straightforward to use. In order to simulate the same build conditions as in full virtualization cases described above (II-B, II-A), we used Debian as the base image for Dockerfile composition. Then we set the installation of required dependencies, download the targeted version of *Linux* kernel and unzip it. After that we copy script to build the kernel (section 5. Používateľská príručka in technical documentation). As the entry point of container we then set the building script, that immediately start the kernel build process in the Docker container. The whole Dockerfile is shown in listing 2 and script to run the container is described in technical documentation (section 5. Používateľská príručka).

Listing 2. Dockerfile

```
FROM debian

ENV URL="https://cdn.kernel.org"
ENV DIR="/pub/linux/kernel"
ENV VERSION="/v4.x/linux-4.19.80.tar.xz"

SHELL [ "/bin/bash", "-c" ]

RUN apt-get update && apt-get install -y \
    build-essential \
    libncurses-dev \
    bison \
    flex \
    libssl-dev \
    libelf-dev \
    bc \
    wget \
    time \
    curl

RUN wget ${URL}${DIR}${VERSION}

RUN unxz linux-4.19.80.tar.xz

RUN tar xf linux-4.19.80.tar

COPY ./.config linux-4.19.80/

COPY build-kernel_docker.sh .

ENTRYPOINT [ "./build-kernel_docker.sh"]

CMD ["/bin/exit"]
```

[9]https://www.docker.com/

III. Benchmark description

In this section we will discuss the proposed benchmark use case for evaluating system load and performance of virtualized environment. Considering the fact, that compilation and build of Linux kernel depends upon your system's resources, such as available number of processor cores and the current system load, we decided to perform our benchmark using this scenario. Because the build time of kernel is the output of our use case, we will primarily measure the building time of the kernel on all three chosen virtualization types with different number of processor cores, processor and memory usage. For this purpose we will use the build definition for development platform *ODROID-X2*[10]. We decided for this kernel definition, because building kernel for more complex system like Raspbian, Debian or Ubuntu would be a huge overhead in terms of size and long build time which isn't necessary in our case.

Kernel version *4.19.80* was selected, because it was used on the specified device and configuration, so to avoid complications we just stayed at this configuration. To measure all required values, we decided to use GNU version of *time* [4] library. This library allow us to get more information about currently executed process, so we used the built in functionalities of the library without any more modification needed.

After successful build of the kernel, results are published to the server in order to achieve simplicity while collecting the data, because benchmark tests are executed on several machines, so it would be very difficult to collect the results from particular machines or containers. The resulted statistics are then accessible in following *JSON* format:

```
{
  "id": 76,
  "time": 20376.45,
  "cpuname": "QEMU Virtual CPU version 2.5+",
  "memusage": 946512,
  "cpuusage": 29,
  "arch": "i686",
  "timestamp": "2019-11-24T11:21:23.548Z",
  "numofcores": 3,
  "hostname": "debian",
  "username": "root",
  "group": 1
}
```

Essentially, we wanted to perform the benchmark tests on hardware and software assisted virtual machine, in container using Docker, in para virtualized environment using Xen[11] server and on physical machine, where are all these environments running to be able to compare performance impact of virtualization. In the end we refused to run the tests in para virtualized environment, because of setup overhead of Xen on

[10]https://www.hardkernel.com/shop/odroid-x2/
[11]https://xenproject.org/

the host machine, where it would be necessary to modify the GRUB loader, while loading the operating system, because, as we already described in I-B, in para virtualized environment it is mandatory to modify kernel of hosted machine.

IV. Results

In this section we will collect measured results and interpret them in form of summarized table and graphs. Tables below display an overview of all measurements executed by running the benchmark tests. Columns represent number of cores and rows are values for particular virtualization type. In table I we can see how many times the test was executed in specific virtualization environment with number of cores given by column value and average time for each exact run.

In figure IV we can see graphical representation of above described table. It is noticeable that for a virtualization with one provided core, the physical machine has better performance, in average about 150s compared to containerization and hardware assisted virtual machine. Although, the performance of software assisted machine is significantly worse and that is s also the reason why the benchmark was performed only once for each number of cores, therefore the values are not included in comparison graphs with other virtualization types. Executing the benchmark with two and three cores, the build times are very similar. Performing the testing with four cores showed that build time in containers is a bit unstable but still comparable to physical and virtual machine, the difference is not so marginal.

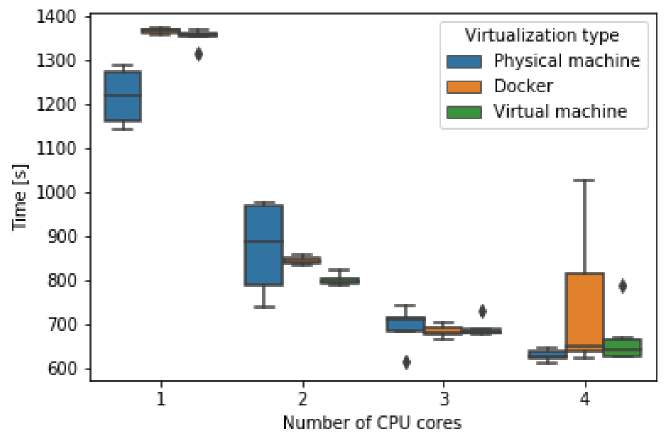


Figure 6. Comparison of build time according to number of cores

Figure IV shows average build times for each virtualization type. In general, we can confirm, that the physical machine is still more performing than virtualization environment, but there is not such a big difference and the build times are very close between each approach.

Data shown in table II, similarly as in table I, represent the count of executed benchmarks on each virtualization type with particular number of cores. The graphical representation is shown in figure 8 but again without the results from software

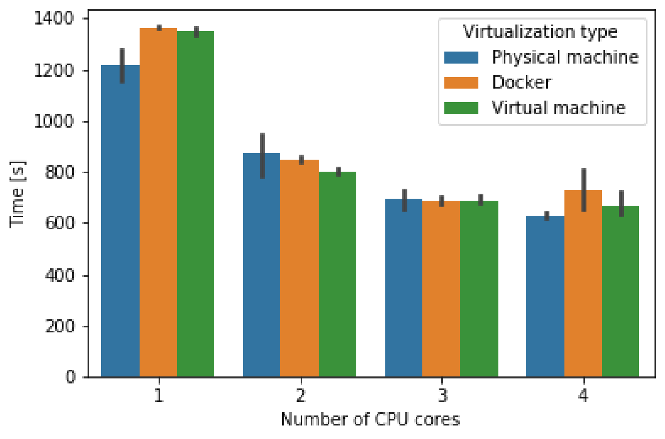


Figure 7. Comparison of average build time according to number of cores

assisted virtual machine, because the data were irrelevant in this comparison. Data representing processor usage in software assisted virtual machine are displayed at graph in figure 9.

If we look at graph in figure 8, we can see average usage of processor while running the benchmark. The pattern of processor usage is much more stable compared to average build times. The interesting thing is, that Docker environment has the least usage of processor at all. This is caused due to the hypervisor, in which the containers are running. As we described in I-B1, containers are running in isolated instances and communicate directly with the resources and are not running operating systems but only binaries, which makes this approach less encumbering.

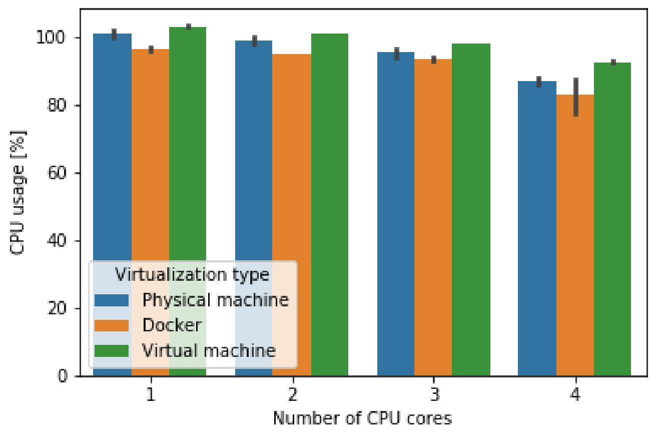


Figure 8. Comparison of average build time according to CPU usage

Considering the graph demonstrated in figure 9, we can recognize huge drawbacks between particular benchmark execution. As we already mentioned in I-A1, running code in software assisted virtualization responds to huge performance leaks due to complicated process of binary translation, which

Table I

RESULTS SUMMARY ACCORDING TO AVERAGE BUILD TIME

Number of cores /	1		2		3		4	
Virtualization type	**Tests count**	**Avg. time [s]**	**Tests count**	**Avg. time [s]**	**Tests count**	**Avg. time [s]**	**Tests count**	**Avg. time [s]**
Hardware assisted	5	1350	5	800	5	690	5	667
Software assisted	1	37898	1	20750	1	20376	1	19185
Containerization	5	1364	5	844	5	684	5	725
Physical machine	5	1215	5	873	5	694	5	630

Table II

RESULTS SUMMARY ACCORDING TO AVERAGE CPU USAGE

Number of cores / Virtualization type	1		2		3		4	
	Tests count	**Avg. CPU usage [%]**	**Tests count**	**Avg. CPU usage [%]**	**Tests count**	**Avg. CPU usage [%]**	**Tests count**	**Avg. CPU usage [%]**
Hardware assisted	5	103.1	5	101	5	98	5	92.66
Software assisted	1	98	1	97	1	29	1	39
Containerization	5	96.2	5	95	5	93.5	5	83.1
Physical machine	5	101	5	99.2	5	95.4	5	86.8

is necessary in this concept. Displayed drawbacks could have many causes, but if we consider the fact, that the code execution is done by software, current performance of hosted machine comes to word and leads to long execution times.

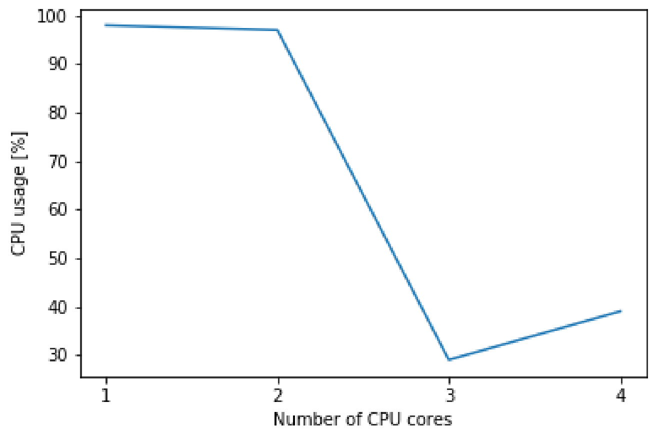


Figure 9. Comparison of CPU usage according to number of cores in software assisted virtualization

During benchmark tests execution we also measured memory usage of the process. Based on result data we declare, that the memory usage is the same in every virtualized environment and the number of attached processors doesn't impact the usage as well. Average memory usage during our benchmark testing was **946402.75** MBytes.

V. EVALUATION

Taking into account bad performance of software assisted virtual machines, we tried to find out, how many number of processor would be needed to create a software assisted virtual machine, which can run our benchmark test in approximately same time as hardware assisted virtual machine running with one processor core based on given data.

In figure 10 we see a graph, which shows build time per number of cores. We were able to perform testing only till four processor cores because of time and hardware limitations, so the prediction will be only informative. When we look at the line chart, we recognize that the benchmark execution time is really outlined compared to others, so we didn't count it into our prediction function.

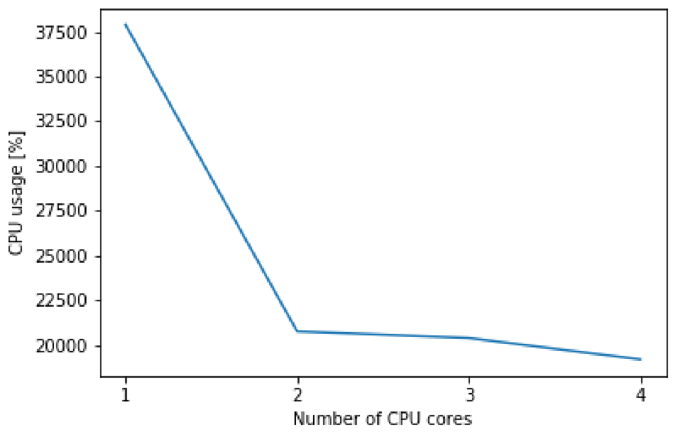


Figure 10. Comparison of build time according to number of cores in software assisted virtual machine

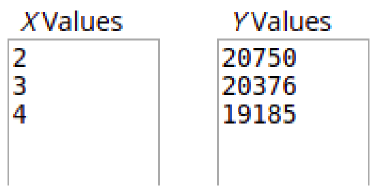
Given in table I we took the value **1350** as the target time in seconds into our prediction, because it's the average time to perform benchmark in hardware assisted virtual machine. For our prediction we decided to use basic linear regression prediction function. The linear regression base equation is defined as follows:

$$\hat{y} = bx + a$$

In order to get our regression equation, we used verified linear regression calculator[12]. We provided our values to their

[12]https://www.socscistatistics.com/tests/regression/default.aspx

calculation as follows:



X Values	*Y* Values
2	20750
3	20376
4	19185

As a result we got calculated linear regression formula:

$$f(x) = -782.5x + 22451.166667$$

After getting the formula, we calculated the approximate number of cores needed in order to run the benchmark in 1350s with following steps:

1)
$$f(x) = -782.5x + 22451.166667$$

2)
$$1350 = -782.5x + 22451.166667$$

3)
$$x = 26.96$$

The resulted value tells us, that according to our linear regression function, in order to achieve benchmark execution time around 1350s in software assisted virtualized environment, we would need approximately 27 processor cores to reach targeted time.

Graphical representation of our research attempt is shown in figure 11, where we can see the linear decrease of time and increase of number of cores. We also included the time value for single core result in order of data correctness, although we didn't count it into to the regression calculation.

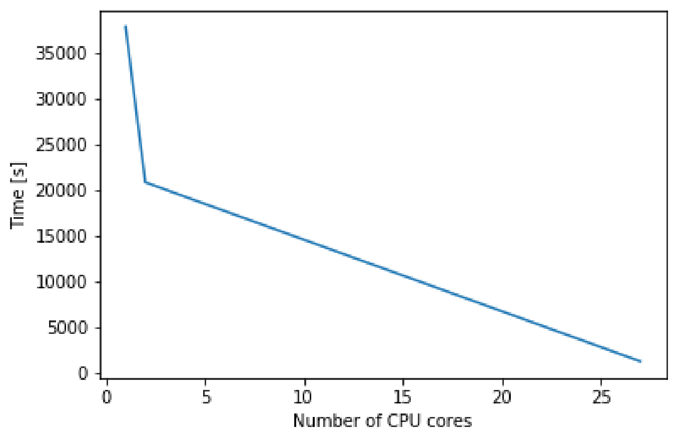


Figure 11. Prediction chart reaching target time

VI. Conclusion

In this paper we firstly introduced different types of virtualization approaches. Then we analyzed, the exact architecture of introduced virtualization types, in order being able to explain the benchmark results later. We also described the major differences between particular virtualization concepts and tried to point out advantages and disadvantages of them. Then we choosed the virtualization platform, on which we performed our benchmark testing and concretely described configuration of testing environment.

After that we presented our benchmark testing scenario, in which we built Linux kernel in configured virtualized environments and measured systems parameters during the benchmark. Then we collected the results of benchmark testing and compared significant attributes across virtualization environments. We also presented the results in comparison graphs and overview tables.

After all we evaluated accumulated results in form of prediction, where we tried to conclude hardware requirements of software assisted virtual machine being able to hit the Linux kernel build time on hardware assisted virtual machine running on single core. In this prediction attempt we used linear regression given the measured values.

As a result of this paper we confirm, that the virtualization in general point of view, has comparable performance with physical machines, especially containerization seems to be applicable. We also proved, that software assisted virtualization builds a big overhead in meaning of execution computing tasks, where the resources needed to ability to execute challenging tasks is enormous. But, as we proved with building Linux kernel, the system load was meaningful and collected results are representing the actual performance of discussed environments.

References

[1] A. B. S., H. M.J., J. P. Martin, S. Cherian and Y. Sastri, "System Performance Evaluation of Para Virtualization, Container Virtualization, and Full Virtualization Using Xen, OpenVZ, and XenServer," 2014 Fourth International Conference on Advances in Computing and Communications, Cochin, 2014, pp. 247-250.

[2] V. Soundararajan, B. Agrawal, B. Herndon, P. Sethuraman and R. Taheri, "Benchmarking a virtualization platform," 2014 IEEE International Symposium on Workload Characterization (IISWC), Raleigh, NC, 2014, pp. 99-109.

[3] K. Ye, Z. Wu, B. B. Zhou, X. Jiang, C. Wang and A. Y. Zomaya, "Virt-B: Towards Performance Benchmarking of Virtual Machine Systems," in IEEE Internet Computing, vol. 18, no. 3, pp. 64-72, May-June 2014.

[4] Differences between physical CPU vs logical CPU vs Core vs Thread vs Socket, http://man7.org/linux/man-pages/man1/time.1.html, Accessed: 24.11.2019.

[5] K. Ye, J. Che, X. Jiang, J. Chen and X. Li, "vTestkit: A Performance Benchmarking Framework for Virtualization Environments," 2010 Fifth Annual ChinaGrid Conference, Guangzhou, 2010, pp. 130-136.

[6] G. Nakagawa and S. Oikawa, "Behavior-Based Memory Resource Management for Container-Based Virtualization," 2016 4th Intl Conf on Applied Computing and Information Technology/3rd Intl Conf on Computational Science/Intelligence and Applied Informatics/1st Intl Conf on Big Data, Cloud Computing, Data Science & Engineering (ACIT-CSII-BCD), Las Vegas, NV, 2016, pp. 213-217.

[7] Y. Tachibana, J. Kon and S. Yamaguchi, "A Study on the Performance of Web Applications Based on RoR in a Highly Consolidated Server with Container-Based Virtualization," 2017 Fifth International Symposium on Computing and Networking (CANDAR), Aomori, 2017, pp. 580-583.

[8] K. T. Raghavendra, S. Vaddagiri, N. Dadhania and J. Fitzhardinge, "Paravirtualization for Scalable Kernel-Based Virtual Machine (KVM)," 2012 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), Bangalore, 2012, pp. 1-5.

[9] Para virtualization vs Full virtualization vs Hardware assisted Virtualization, https://www.unixarena.com/2017/12/para-virtualization-full-virtualization-hardware-assisted-virtualization.html/, Accessed: 3.10.2019.

[10] H. He, L. Jiang, H. Chen and W. Dong, "Hardware/software co-design of Dynamic Binary Translation in X86 emulation," 2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE), Zhangjiajie, 2012, pp. 283-287.

[11] N. Jain and S. Choudhary, "Overview of virtualization in cloud computing," 2016 Symposium on Colossal Data Analysis and Networking (CDAN), Indore, 2016, pp. 1-4.

[12] Kernel Virtual Machine https://www.linux-kvm.org/page/Main_Page, Accessed: 3.10.2019.

[13] Oracle VM VirtualBox https://www.virtualbox.org/

[14] RISC vs. CISC https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc/

[15] Q. Xin and C. Hao, "The Research of Inter-domain Communication Optimization under Xen Hardware-assisted Virtualization," 2011 International Conference on Business Computing and Global Informatization, Shanghai, 2011, pp. 585-588.

[16] Yun Chan Cho and Jae Wook Jeon, "Sharing data between processes running on different domains in para-virtualized xen," 2007 International Conference on Control, Automation and Systems, Seoul, 2007, pp. 1255-1260.