Multi-level virtualization and its impact on system performance in cloud computing

Paweł Lubomski[1], Andrzej Kalinowski[1], and Henryk Krawczyk[2]

[1] IT Services Centre, Gdańsk University of Technology, Gdańsk, Poland
{lubomski, andrzej.kalinowski}@pg.gda.pl
[2] Faculty of Electronics, Telecommunications and Informatics,
Gdańsk University of Technology, Gdańsk, Poland
hkrawk@eti.pg.gda.pl

Abstract. The results of benchmarking tests of multi-level virtualized environments are presented. There is analysed the performance impact of hardware virtualization, container-type isolation and programming level abstraction. The comparison is made on the basis of a proposed score metric that allows you to compare different aspects of performance. There is general performance (CPU and memory), networking, disk operations and application-like load taken into account. The tested technologies are, inter alia, VMware ESXi, Docker and Oracle JVM.

Keywords: virtualization, performance, benchmark

1 Introduction

For the last few years an intensive growth of popularity in the cloud services has been observed. Nearly all hosting providers now offer such a dynamic cloud service for an affordable cost. The most popular services are Infrastructure-as-a-Service (IaaS) or Platform-as-a-Service (PaaS). There are three main types of cloud: public, private and hybrid [2].

There has also been a significant progress in technology supporting widely understood clouds. Many types of virtualization were developed. They can be divided into 3 main groups [3]. There are hardware virtualizations, containers inside systems, and programming language abstractions such as e.g. Java Virtual Machine (JVM). The last type concerns mainly large scale, service-oriented systems which use .NET or Java Enterprise Edition (JEE) platforms. More detailed classification will be described later on.

The main reason for the popularity of dynamic cloud solutions is comfort and ease of use. Dynamic clouds help share resources among many virtual systems and manage them dynamically depending on the systems' load. This way it is possible to overbook the resources. But the most important advantage is the time of a new virtual server deployment - it is incomparably shorter than buying traditional hardware, placing it in datacenter and operating system installation. Thereby, it is very easy and cheap to add some new virtual machines to a cluster for the short time of an increased system load, e.g. academic session, recruitment

for the university. When the load decreases, the resources are released and can be used for other purposes.

The deployment process is very fast thanks to, among other reasons, using some prepared templates and snapshots. It may also be a way of releasing new versions of software - the new image is only introduced. Especially Docker images [4][5] are popular for this way of releasing software [6].

There is no rose without a thorn. All these benefits come together with a cost of performance decrease. Of course, the developers of cloud solutions intensively work on the performance improvement, so that this cost is getting lower every day. There are not currently available any reliable reports comparing the costs of these solutions. The last good job was done in 2014 by IBM Research Division [1].

At our university we started to use virtualization intensively, and put it one into another. We use VMware ESXi virtualization on which we run a Debian Linux guest OS. Inside the guest OS we run a Docker image containing the release of our central system. Because our university system is very big, it was designed as a distributed JEE system written in Java language. In this way we have a three-level virtualization used. Thus, we want to check what the real cost of such an approach is.

2 Virtualization methodologies

As mentioned earlier there are three types (levels) of virtualization: hardware, containers and platform (e.g. JVM). We will briefly describe them below.

2.1 Hardware virtualization

The hardware virtualization can be divided into two main groups depending on the type of host operating system (hypervisor). The first one is a native hypervisior (also known as 'type 1' or 'bare-metal') where the guest OS works on virtualized hardware and the host OS is not a standard OS but a highly specialized solution. They are able to take advantage of all hardware capabilities, especially those supporting virtualization. The main representatives of this group are: VMware ESXi [7], XenServer [8], MS Hyper-V [9].

The second type are hosted hypervisors (also known as 'type 2'). These solutions cannot work alone - they require a standard operating system and work as a regular application in this system. For better performance they use special kernel modules and/or libraries of host operating system such as KVM (Kernel Virtualization Module) [10][11].

The intention of using this solution is not productive environments - they are rather for testing images of virtual machines (VMs) during the development and testing process. They may also be an easy and cheap solution of a multi-technology environment for testing traditional software on different operating systems, etc. The most famous solutions of this type are: VirtualBox [12], VMware Player [13], QEMU [14].

2.2 Containers

Another approach accompanies the development of containers. This is the oldest type, also known as para-virtualization. The most characteristic aspect of this type of virtualization is that the host OS only separates processes and resources so that it is impossible to run another operating system than the host OS. It started with *chroot*, then there were FreeBSD jails [15] and Solaris zones [16].

Nowadays there are two mainly-used solutions: OpenVZ [17] based on a customized Linux Kernel, and Docker [4] which uses LinuxContainers [18] or libcontainer library (from version 0.9).

Due to the taken architectural assumptions, they should have a lower performance overhead (than hardware virtualization solutions) but they also have some limitations (e.g. impossibility to choose another guest OS) and a worse isolation of processes and resources usage.

2.3 Programming language abstraction

In large-scale distributed e-service systems there are mainly used .NET and JEE. Both of them use a specific level of programming language abstraction - .NET Framework and JVM. Especially in the case of JVM the portability is on a high level, however it is done at the cost of a negative impact on performance.

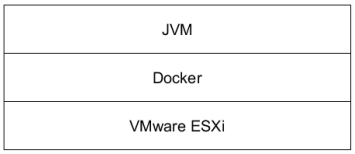

Fig. 1. Virtualization stack of our university system

Fig. 1 presents the virtualization stack of our university central system. We use either hardware virtualization (hypervisor type 1 - VMware ESXi), containers (Docker) with images containing release versions of software written in Java, so it is run inside a JVM. The use of Docker images ensure that the system is configured and behaves the same in every environment: development, testing and production.

3 Benchmark methodology and tools

The IBM research report[1] focused on the global performance of Docker, bare-metal, and pure virtualization environments. They have taken the assumption that so-called *double virtualization* is redundant and has a negative impact on performance. In our paper we want to verify this hypothesis and check if the

ease of use (in terms of maintenance, resource reallocation and scalability) of such virtualization infrastructure is worth the cost of the predicted performance drop.

The IBM research team performed comprehensive benchmarking of Docker containers, including both networking mechanisms (i.e. direct host mapping and port forwarding) and two I/O subsystems (direct mounts and *AUFS* - they have omitted: device mapper and btrfs backends). In our paper we want to focus on both the double and triple virtualization problems. In order to achieve that, we need to run our test containers in the same manner as the IBM team did. Therefore, the containers were tested in several different configurations (see table 1).

3.1 Test configurations and suites

For benchmarking purposes we have used three concepts:

test configurations - a set of all available testing environments (i.e. all combinations of virtual/bare-metal and Docker instances),
test suites - a set of *test cases* which measure a similar aspect of performance in a specified *test configuration*,
test case - an individual performance measurement.

Table 1. Benchmarked test configurations

Machine type	OS configuration	Docker configuration
bare-metal	**metal.local**	**metal.aufs**
	metal.remote	**metal.mnt**
		metal.aufs.nat
		metal.aufs.host
		metal.mnt.nat
		metal.mnt.host
virtual machine	**vm.local**	**vm.aufs**
	vm.remote	**vm.mnt**
		vm.aufs.nat
		vm.aufs.host
		vm.mnt.nat
		vm.mnt.host

Table 1 presents tested configurations. They correspond to the first two lowest levels of virtualization in our university system stack presented in fig. 1: hardware virtualization (VMware ESXi) and container (Docker). The highest layer of virtualization (JVM) is covered not by the test configuration, but by two test suites: java and rich java application (see table 2) - this way we could test java applications on every considered type of virtualization. So we benchmarked the 16 following configurations:

(metal|vm).local pure operating system with no Docker container layer, tests are performed locally,
(metal|vm).remote pure operating system with no Docker container layer, tests are performed from a remote location, thus the network layer is also being tested,
(metal|vm).aufs tests are performed on a Docker shared file system - *AUFS*, therefore I/O operations have a serious drawback,
(metal|vm).mnt tests are performed on mounted volume, therefore I/O performance is significantly improved,
(metal|vm).aufs.nat the same as **.aufs** but uses port forwarding,
(metal|vm).aufs.host the same as **.aufs** but uses direct host mapping,
(metal|vm).mnt.nat the same as **.mnt** but uses port forwarding,
(metal|vm).mnt.host the same as **.mnt** but uses direct host mapping.

The performance of each test configuration was separately measured in the following aspects: CPU, memory, I/O and network. The final test suite covered a typical web application resource usage. This way, we adopted 7 test suites:

1. CPU - overall computation performance tests
2. multicore - specific tests run to verify multiple CPU cores' performance
3. memory - overall memory performance tests
4. java - CPU utilization during java application execution
5. disk - overall disk performance tests
6. network - overall network layer performance tests
7. application - complex performance tests

Table 2 presents the combinations of test configurations and test suites which were measured during our benchmarking. One can easily notice that only the most representative and not-redundant combinations were taken into account.

Table 2. Test configurations and suites matrix.

Test configuration	Test suites						
	CPU	**multicore**	**memory**	**java**	**disk**	**network**	**application**
(metal|vm).local	✓	✓	✓	✓	✓		✓
(metal|vm).remote						✓	✓
(metal|vm).aufs	✓	✓	✓	✓	✓		✓
(metal|vm).mnt					✓		✓
(metal|vm).aufs.nat						✓	✓
(metal|vm).aufs.host						✓	✓
(metal|vm).mnt.nat							✓
(metal|vm).mnt.host							✓

3.2 Hardware and software configuration

Our tests were launched on Dell R210 hardware with one Intel Xeon E31270 @3.39GHz (8 cores) processor and 16 GiB of RAM. Network throughput was 1 Gbps. And we used 1 TB Western Digital WD1003FBYX-0 disk.

Our tests were based on a standard GNU/Linux Debian 8.2 distribution which was used as a host and guest OS and also as a base for Docker containers. As for Docker daemon we used a currently available version 1.9. For hardware virtualization tests we used *VMware ESXi 6.1*. For Java applications we used *Oracle Java 1.8.0_45-b14*.

We did not use any limitations like *ulimit*, *cgroups* or ESXi mechanisms, hence all tests were run on all available resources. On ESXi we used *Hardware version 11* with all default options, for disk storage we used *Local Storage* with thin provisioning and standard 1 MiB data blocks.

To measure our tests we used mainly *Phoronix Test Suite 6.0.1* which was run on *PHP 5.6.14*. Additionally, for measuring network performance we used *netperf 2.6.0*. And finally, for overall testing we used *IBM Daytrader 3* web application with *Apache Jmeter 2.13*.

3.3 General and disk performance

As mentioned earlier we divided our tests into 7 test suites. For suites 1-5 we used *Phoronix Test Suite*, with the following standard test suites:

1. pts/cpu (27 different test cases)
2. pts/multicore (19 different test cases)
3. pts/memory (9 different test cases)
4. pts/java (6 different test cases)
5. pts/disk (17 different test cases)

Each of these test cases consisted of three runs. The average score of those runs was selected as the result. The results of each test case were normalized, and finally summed to provide a total score for each test suite.

3.4 Network performance

For test suite no. 6 we used two test cases:

1. netperf in tcp mode
2. netperf in udp mode

Each test case contained 10 identical tests. After running them, the average score was computed, and afterwards the total score was calculated in the same manner as in the previous test suites. In all cases the tested machine was used as a server.

3.5 Application stack performance

For test suite no. 7 we used a complex application which tries to utilize all previous test suites to provide a final performance result. It consists of *IBM Daytrader 3* web application which is *a Java EE 6 application built around an online stock trading system. The application allows users to login, view their portfolio, look up stock quotes, buy and sell stock shares, and more*[19]. We deployed it on a *WildFly 9.0.2.Final Application Server*, *H2* standalone database and with EJB3 Mode enabled in the *Daytrader* application (which uses EJBs with JPA to connect to the database).

To measure the total performance of this test suite we used an *Apache JMeter* which was run within the tested system, and also from a remote host, to check network performance. For each run we started with a clean database and a 180 seconds dry-run. After that, we measured the performance for 180 seconds, in 6-second intervals.

3.6 Score calculation

We propose the following score calculation methodology. Each completed test case *produces* an average result. This value is normalized with all average results collected from all test configurations. The acquired values are called test scores. After completion of a particular test suite, those test scores are summed up to present a final performance score in each of the 7 test suites. Most of values collected by test runs are presented as HIB (higher is better) but some (especially in multicore test suite) are presented as LIB (lower is better). In the second case the test score must be subtracted from the final performance score.

Let us assume that our results can be stored in a large 3 dimensional array - $\boldsymbol{t}$ containing all test suites, test cases and test configurations with s, i, c as their indices. Therefore, $t_{s,i,c}$ is a single result of test configuration c on test case i in suite s. $\boldsymbol{t}_{s,i}$ is a vector containing the results of test case i in suite s and $|\boldsymbol{t}_{s,c}|$ is the number of test cases in the test suite s for the particular configuration c. In such a case we can use the following formulae:

$$\text{test\_score}_{s,i,c} = \frac{t_{s,i,c}}{\max(\boldsymbol{t}_{s,i})} \tag{1}$$

$$\text{total\_score}_{s,c} = \sum_{i=1}^{|\boldsymbol{t}_{s,c}|} \text{sgn}_s(i) \cdot \text{test\_score}_{s,i,c} \tag{2}$$

where $\text{sgn}_s(i)$ is '-1' when LIB and '$+1$' in HIB case.

4 Benchmark results

4.1 CPU and memory performance

Table 3 presents aggregated results for CPU, java and memory test suites. The final results were predictable and are similar to those presented in the IBM report. Docker performance drop in single core configuration is around 3% for both

bare-metal machine and virtualized environment. The multicore performance has a higher drop of 8%. Java performance tests were based on *scimark*, *bork file encrypter* and *sunflow* rendering system. Therefore, they mainly focused on CPU and memory efficiency. The outcome is similar to the CPU performance. Memory drop is insignificant. ESXi virtualization layer produces around 15% of CPU performance drop and lowers java application efficiency by 10%. Memory drop is lower than 2%.

When testing Docker CPU/memory efficiency we were really benchmarking the cgrups/namespaces efficiency, thus the Docker overhead is almost nonexistent. The differences between metal-docker test cases were minimal (less than 1%), but while summing 27 results we ended up with an overhead of 3%.

Table 3. CPU and memory performance - total scores

Test suite	**metal.local**	**metal.aufs**	**vm.local**	**vm.aufs**
CPU	7.43	7.29	6.36	6.08
multicore	−4.31	−4.64	−4.79	−5.16
java	2.17	2.03	1.96	1.93
memory	9	9	8.83	8.82

4.2 Network performance

In table 4 there is presented the overall network performance. The Docker in direct host mapping configuration (**.host**) has no significant performance drop. The port forwarding configuration (**.nat**) lowers performance by around 4%. The highest efficiency drop (14%) can be observed in the ESXi environment.

Table 4. Network performance

Test suite	**metal. aufs.host**	**metal. aufs.nat**	**metal. remote**	**vm. aufs.host**	**vm. aufs.nat**	**vm. remote**
netperf tcp	1.00	0.93	1.00	0.86	0.84	0.86
netperf udp	1.00	0.97	1.00	0.88	0.84	0.87
Total score	2.00	1.9	2.00	1.72	1.68	1.73

4.3 Disk performance

The disk performance results are shown in tables 5 and 6. Oddly, the overall disk performance is significantly greater in the ESXi environment (roughly, by 39%). This is probably the effect of VMware *VMFS* filesystem features[22], notably

the huge block size of 1 MiB. The *ext4* file system used on bare-metal system has a maximum block size of 4 KiB. Therefore, file system buffers will sync more frequently than the *VMFS* counterpart, leading to the observed performance drop.

Table 5. Bare-metal configurations disk performance test scores. In brackets there are in-normalized results denominated in given units.

Test case	**Order**	**Unit**	**metal.local**	**metal.mnt**	**metal.aufs**
AIO-Stress	HIB	MB/s	1.00 (2637.12)	0.92 (2430.07)	0.89 (2336.56)
SQLite	LIB	Seconds	1.00 (524.31)	0.88 (463.31)	0.86 (448.47)
FS-Mark	HIB	Files/s	0.34 (15.67)	0.40 (18.43)	0.40 (18.43)
Dbench1	HIB	MB/s	0.13 (39.22)	0.14 (41.72)	0.15 (43.72)
Dbench2	HIB	MB/s	0.13 (66.93)	0.14 (72.27)	0.14 (69.34)
Dbench3	HIB	MB/s	0.28 (67.5)	0.30 (72.43)	0.24 (58.63)
Dbench4	HIB	MB/s	0.36 (9.68)	0.41 (10.76)	0.40 (10.5)
IOzone1	HIB	MB/s	1.00 (8059.31)	0.97 (7780.49)	0.38 (3032.21)
IOzone2	HIB	MB/s	0.60 (69.02)	0.97 (111.48)	0.96 (110.45)
Threaded I/O Tester1	HIB	MB/s	1.00 (13595.01)	0.99 (13453.36)	0.83 (11330.63)
Threaded I/O Tester2	HIB	MB/s	0.42 (0.55)	0.41 (0.53)	0.45 (0.59)
Compile Bench1	HIB	MB/s	0.89 (487.95)	1.00 (548.2)	0.71 (388.47)
Compile Bench2	HIB	MB/s	0.95 (261.22)	1.00 (274.74)	0.56 (154.23)
Compile Bench3	HIB	MB/s	0.80 (1460.93)	0.74 (1343.32)	0.29 (533.12)
Unpacking	LIB	Seconds	0.89 (8.61)	0.89 (8.61)	0.92 (8.95)
PostMark	HIB	TPS	0.99 (5208.0)	1.00 (5282.0)	0.60 (3178.0)
Gzip Compression	LIB	Seconds	0.94 (12.85)	0.94 (12.93)	0.95 (13.05)
Apache Benchmark	HIB	RpS	1.00 (29040.6)	0.89 (25830.51)	0.80 (23352.62)
Total score	HIB		7.06	7.57	5.07

In some test cases the efficiency of the Docker *AUFS* file system is greater than pure OS. This is the effect of the *COW* (copy on write) mechanism which writes changes to the disk only when it is needed [20, 21]. Similar *COW* mechanisms are used while mounting volumes inside Docker (**metal.mnt** and **vm.mnt**).

The bare-metal supremacy performance can be observed in asynchronous AIO-Stress, IOzone1 (8GiB read test), Threaded I/O tester1 (64 MiB random read by 32 threads), Gzip Compression (2 GiB file) and Apache Benchmark. While analysing the results one can notice that the bare-metal environment shows some serious performance drops on writes (e.g. IOzone2, Threaded I/O tester2). This is the effect of the aforementioned *COW* mechanisms on Docker and *VMFS* buffers.

4.4 Application stack performance

The results of the most comprehensive test suite are presented in table 7. We ran this test suite on 16 test configurations, thus benchmarking all useful combinations. The most important finding is the overall low performance of the ESXi

Table 6. Virtualized configurations disk performance test scores. In brackets there are in-normalized results denominated in given units.

Test case	Order	Unit	vm.local	vm.mnt	vm.aufs
AIO-Stress	HIB	MB/s	0.85 (2239.05)	0.79 (2094.9)	0.75 (1969.52)
SQLite	LIB	Seconds	0.26 (136.46)	0.28 (144.8)	0.28 (144.39)
FS-Mark	HIB	Files/s	1.00 (46.03)	0.97 (44.87)	0.98 (44.97)
Dbench1	HIB	MB/s	1.00 (301.23)	0.98 (295.08)	0.90 (272.4)
Dbench2	HIB	MB/s	1.00 (508.15)	0.93 (472.84)	0.84 (428.83)
Dbench3	HIB	MB/s	1.00 (244.02)	0.80 (194.63)	0.80 (196.13)
Dbench4	HIB	MB/s	1.00 (26.55)	0.96 (25.51)	0.95 (25.24)
IOzone1	HIB	MB/s	0.94 (7559.72)	0.96 (7714.91)	0.69 (5569.87)
IOzone2	HIB	MB/s	1.00 (115.35)	0.99 (114.35)	0.98 (112.72)
Threaded I/O Tester1	HIB	MB/s	0.91 (12390.95)	0.89 (12157.88)	0.81 (11076.75)
Threaded I/O Tester2	HIB	MB/s	0.90 (1.17)	1.00 (1.3)	0.95 (1.24)
Compile Bench1	HIB	MB/s	0.66 (360.75)	0.65 (356.42)	0.61 (336.42)
Compile Bench2	HIB	MB/s	0.73 (199.35)	0.76 (208.75)	0.61 (166.58)
Compile Bench3	HIB	MB/s	1.00 (1825.35)	0.75 (1367.49)	0.76 (1389.03)
Unpacking	LIB	Seconds	0.99 (9.64)	0.98 (9.52)	1.00 (9.69)
PostMark	HIB	TPS	0.89 (4716.0)	0.90 (4746.0)	0.72 (3807.0)
Gzip Compression	LIB	Seconds	1.00 (13.66)	1.00 (13.63)	1.00 (13.69)
Apache Benchmark	HIB	RpS	0.77 (22259.71)	0.67 (19492.16)	0.59 (17089.18)
Total score	HIB		11.4	10.74	9.66

network stack (around 25%, thus lower than in network performance benchmark).

Table 7. Application stack performance - total scores

	Test configuration	Total score
Local execution	metal.local	0.95
	metal.aufs	0.94
	metal.mnt	0.95
	vm.local	0.95
	vm.aufs	0.91
	vm.mnt	0.95
Remote execution	metal.remote	1
	metal.aufs.host	0.96
	metal.aufs.nat	0.95
	metal.mnt.host	1
	metal.mnt.nat	0.99
	vm.remote	0.79
	vm.aufs.host	0.75
	vm.aufs.nat	0.74
	vm.mnt.host	0.75
	vm.mnt.nat	0.76

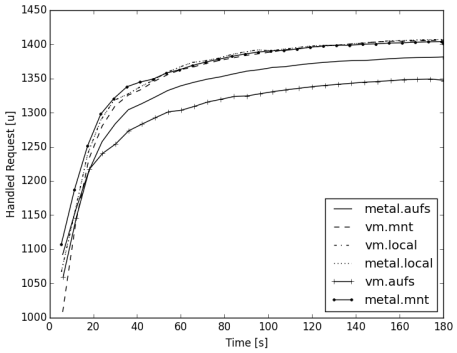

Fig. 2. Application stack performance - Local JMeter executions

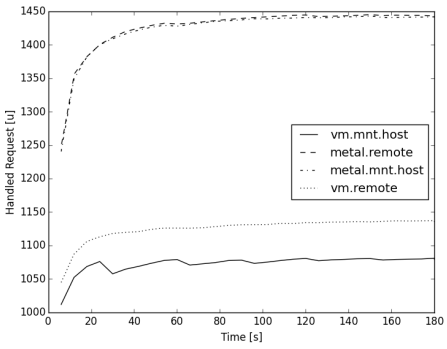

Fig. 3. Application stack performance - Remote JMeter executions

If we compare the local executions of JMeter (thus removing the network stack) we can see that the overall performance of the virtualized environment is almost identical to the bare-metal one (see figure 2), with some minor fluctuation of the *AUFS* filesystem (4%). Docker with mounted volume and direct network mapping is almost identical in performance to its bare-metal counterpart. The only difference can be observed in the virtualized environment where pure OS versions are faster by 6% (see figure 3).

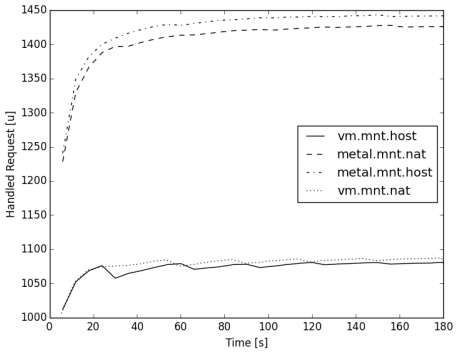

Fig. 4. Docker port forwarding vs direct host mapping comparison

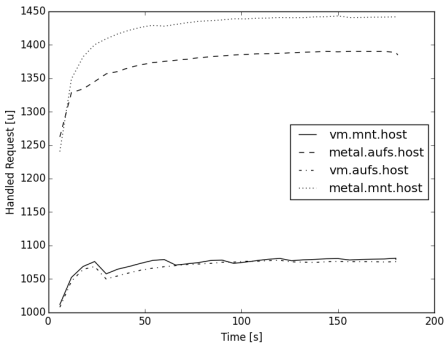

Fig. 5. Docker AUFS vs mounted volumes comparison

There are no significant differences between Docker network modes (NAT and direct network mapping) in this test suite (see figure 4). The performance of *AUFS* and mounted volumes is almost identical to ESXi, some slight differences appear in the bare-metal environment(see figure 5).

5 Conclusions and future work

The most significant benchmarking results were obtained in disk and application stack performance measurements. The overall results show that there is no significant decrease in performance while using three-level virtualization configurations. The decrease is from 1 to 9% with the exception of ESXi network layer overhead. The ESXi network layer is significantly slower when using a standard configuration. The observed network bottleneck in ESXi environment can probably be reduced by changing the default network driver.

Container layer (Docker) has a negligible impact on performance independently of using hardware virtualization or not. This refers to medium loaded systems such as our university system. The Docker default settings are production ready - the overall performance drop is insignificant.

Superior ESXi disk performance may be obtained by using external disk storage. In the future there should be used some larger files (bigger than RAM) for benchmarking disk operations. For additional disk performance measurement we should also focus on a large database.

Finally, we can say that the cost (understood as system performance decrease) of using multi-level virtualization in our university system is acceptable, especially while taking into account the advantages of such configuration (see Introduction). This cost can be balanced by introducing horizontal scaling (clustering) when the system load is higher. It is very quick and easy to do thanks to the usage of virtualization on the first and second levels. There is an overwhelming need to check non-default ESXi network drivers because effective networking is very important while clustering large scale distributed e-service systems, which is the point of cloud configurations.

References

1. Felter W., Ferreira A., Rajamony R., Rubio J., "An Updated Performance Comparison of Virtual Machines and Linux Containers. IBM Research Report" (2014)
2. T. Mather, S. Kumaraswamy, and S. Latif, "Cloud Security and Privacy. An Enterprise Perspective on Risks and Compliance" OReilly, 2009.
3. R. Perez, L. Van Doorn, and R. Sailer, "Virtualization and Hardware-Based Security", IEEE Security Privacy Magazine, vol. 6, no. 5, pp. 24-31, 2008.
4. D. Merkel, "Docker: lightweight Linux containers for consistent development and deployment", Linux Journal, vol. 2014, no. 239, p. 2, 2014.
5. D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes", IEEE Cloud Computing, vol. 1, no. 3, pp. 81-84, 2014.
6. P. Di Tommaso, E. Palumbo, M. Chatzou, P. Prieto, M. L. Heuer, and C. Notredame, "The impact of Docker containers on the performance of genomic pipelines", PeerJ PrePrints, vol. 3, p. e1428, 2015.
7. VMware, "Performance Best Practices for VMware vSphere 6.0" `http://www.vmware.com/files/pdf/techpaper/VMware-PerfBest-Practices-vSphere6-0.pdf`
8. P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, "Xen and the Art of Virtualization" `http://www.cl.cam.ac.uk/research/srg/netos/papers/2003-xensosp.pdf`

9. Microsoft, "Why Hyper-V? Competitive Advantages of Microsoft Hyper-V Server 2012 over the VMware vSphere Hypervisor" http://download.microsoft.com/download/5/7/8/578E035F-A1A8-4774-B404-317A7ABCF751/Competitive-Advantages-of-Hyper-V-Server-2012-over-VMware-vSphere-Hypervisor.pdf
10. A. Kivity, U. Lublin, A. Liguori, Y. Kamay, and D. Laor, "kvm: the Linux virtual machine monitor", Proceedings of the Linux Symposium, vol. 1, pp. 225-230, 2007.
11. M. Fenn, M. A. Murphy, J. Martin, and S. Goasguen, "An Evaluation of KVM for Use in Cloud Computing", Proceedings of the 2nd International Conference on the Virtual Computing Initiative - ICVCI08, vol. V, pp. 1-7, 2008.
12. https://www.virtualbox.org/
13. http://www.vmware.com/products/player/
14. D. Bartholomew, "QEMU: a Multihost, Multitarget Emulator" http://www.ee.ryerson.ca/~courses/coe518/LinuxJournal/elj2006-145-QEMU.pdf
15. P. Kamp, R. Watson, "Jails: Confining the omnipotent root." http://phk.freebsd.dk/pubs/sane2000-jail.pdf
16. Oracle, "Oracle Solaris 11.1 Administration: Oracle Solaris Zones, Oracle Solaris 10 Zones, and Resource Management" http://docs.oracle.com/cd/E26502_01/pdf/E29024.pdf 2013
17. https://openvz.org/Features
18. R. Rosen, "Resource management: Linux kernel Namespaces and cgroups" http://www.haifux.org/lectures/299/netLec7.pdf 2013
19. J. McClure, "Measuring performance with the Daytrader 3 benchmark sample" https://developer.ibm.com/wasdev/docs/measuring-performance-daytrader-3-benchmark-sample/ 2014
20. S. Kasampalis, "Copy On Write Based File Systems Performance Analysis And Implementation" Kongens Lyngby 2010
21. https://docs.docker.com/engine/userguide/storagedriver/aufs-driver/
22. https://www.vmware.com/pl/products/vsphere/features/vmfs