Aalto University
School of Science
Department of Mathematics and System Analysis

Testing Performance of Two Virtualization Software

Bachelor's thesis
31.10.2017

Jani Laine

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi

Abstract of bachelor's thesis

Author Jani Laine

Title of thesis Testing Performance of Two Virtualization Software

Degree programme Engineering Physics and Mathematics

Major Mathematics and System Analysis **Code of major** SCI3029

Supervisor Kai Virtanen

Thesis advisor(s) Jimmy Kjällman

Date 31.10.2017 **Number of pages** 29 **Language** English

Abstract

To match the growing needs of better data handling in todays' networked society various solutions have been invented, many of which utilize remote servers called clouds for computing. For small networks of devices it is often beneficial to perform computing relatively close to the data source. This called edge computing, where the device controlling the components of the network is called the edge device. On edge devices, different kinds of software are ran to create software-based computers which are called virtual machines. Running those kinds of software is called virtualization and it is highly useful for controlling the components of a small network since every component can be assigned to a specific virtual machine on the edge device.

Different virtualization software have different attributes when it comes to performance, energy efficiency and scalability. This thesis focuses on the performance side of those software and compares two virtualization software, KVM (Kernel-based Virtual Machine) and Docker, to find out which one is better to be used as an edge device in a small network of devices. Various performance measures were used for comparing the two platforms. These measures involve evaluating the speed of the processor, writing to and reading from hard disk, RAM speed and network bandwidth. The result data is gathered with software specified for testing the capabilities of devices. The data is then analyzed by calculating basic statistics from the data and by utilizing t-test.

The conclusions of the comparisons is that Docker is better for the virtualization purposes of edge computing. Docker is clearly faster when it comes to CPU and network performance. For disk writing and reading and RAM performance, there is no clearly faster platform. However, the conclusion drawn from the latter two testing areas is that the dd tool, which is a basic linux command line tool, should probably not be used for perfomance testing since the results from those test were often not very logical.

Keywords cloud, edge device, virtualization, KVM, Docker, performance

Aalto-yliopisto
Perustieteiden
korkeakoulu

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi

Tekniikan kandidaatintyön tiivistelmä

Tekijä Jani Laine

Työn nimi Testing Performance of Two Virtualization Software

Koulutusohjelma Teknillinen fysiikka ja matematiikka

Pääaine Matematiikka ja systeemitieteet **Pääaineen koodi** SCI3029

Vastuuopettaja Kai Virtanen

Työn ohjaaja(t) Jimmy Kjällman

Päivämäärä 31.10.2017 **Sivumäärä** 29 **Kieli** Englanti

Tiivistelmä

Internettiin yhdistettyjen laitteiden määrä kasvaa kiihtyvällä vauhdilla verkottuneessa yhteiskunnassa. Tämän ilmiön myötä syntyneisiin datan säilytys- ja käsittelyhaasteisiin on kehitetty erilaisia ratkaisuja, joista useat käyttävät hyväkseen etäisiä palvelimia eli pilvipalveluita datan käsittelyyn. Pienien muutamien laitteiden verkkojen datan käsittely suoritetaan yleensä hyvin lähellä datan lähdettä eli verkon reunaa niin sanotulla reunalaitteella, jolla myös ohjataan verkon laitteita. Laitteiden hallintaan reunalaiteella luodaan erilaisia teknologioita käyttäen ohjelmallisesti toteutettuja tietokoneita eli virtuaalikoneita. Tätä kutsutaan virtualisoinniksi ja se mahdollistaa verkon laitteiden eristämisen toisistaan, koska jokaista laitetta voidaan tällöin hallita eri virtuaalikoneella.

Eri virtualisointiteknologioilla on vahvuuksia ja heikkouksia toisiinsa nähden muun muassa suorituskyvyn, virrankäytön ja sovellusmahdollisuuksien suhteen. Tämä työ keskittyy kahden eri teknologian suorituskyvyn mittaamiseen samalla reunalaitteella. Vertailtavat virtualisointiohjelmat ovat KVM (Kernel-based Virtual Machine) ja Docker. Tavoiteena on selvittää, kumpi on parempi virtualisointiohjelmisto reunalaitteelle suhteellisen pienessä laiteverkostossa. Ohjelmistojen suorituskykyä vertaillaan prosessorin nopeuden, kiintolevyn kirjoitus- ja lukunopeuden, keskusmuistin nopeuden sekä verkkoliikenteen nopeuden suhteen. Mittaukset tehdään siihen suunnitelluilla ohjelmilla ja saatu mittausdata analysoidaan laskemalla siitä tilastollisia tunnuslukuja sekä t-testillä. Tulokset olivat suurimmilta osin järkeviä ja niistä voidaan päätellä, että Docker on parempi virtualisointiohjelmisto pienen laiteverkoston reunalaitteelle.

Työn tulokset osoittavat muun muassa sen, että Dockerilla prosessorin käyttö sekä verkkoliikenne ovat selkeästi nopeampia kuin KVM:llä. Kiintolevyn tai keskusmuistin käyttönopeuksille ei voitu tulosten perusteella määrittää nopeampaa ohjelmistoa. Kahden viimeksi mainitun käyttönopeuden mittauksissa kuitenkin havaittiin, että mittauksiin käytettyä linuxin komentorivityökalua dd:tä ei pitäisi käyttää, koska sen tuottamat tulokset olivat useimmiten epäloogisia.

Avainsanat pilvi, reunalaite, virtualisointi, KVM, Docker, suorituskyky

Contents

1 Introduction

In recent years, the increasing number of devices connected to the Internet and the large amounts of data they produce have forced researchers to find better and faster solutions for data storage, computation and other types of data handling. The most successful solution has been performing all that in a remote computing server called a cloud. Although cloud computing is an effective way to increase computing power and convenience it has some challenges it has to face, too.

For heavy computing, needs there is centralized infrastructure that is cloud-based but for example a home entertainment center does not need that much computing power. In that kind of situations the computing is usually performed relatively close to the data source, in other words, at the edge of the network and is therefore called edge computing [4]. One solution for edge computing needs is to have a small single board computer (SBC) to be the platform that controls the components of the network as the edge device. This can be achieved by running virtual machines on the device which are basically software-based computer instances. Virtual instances can be ran with different virtualization software.

The Raspberry Pi models [10] and the ODroids [9] are examples of possible edge devices. Virtualization software include the hypervisors Kernel-based Virtual Machine (KVM) and Xen as well as Docker which uses containers to run instances. The Odroid C2 with KVM and Docker were used in these tests.

The objective of the measurements conducted in this thesis is to compare the performance of KVM to Docker and also to the native platform which is the host operating system without any virtualization. The comparisons are conducted using specific performance measures. These measures include testing of processor, writing to and reading from hard disk, RAM and network performance. Measuring performance means measuring either speed or execution times. After the measurements the data is analyzed using statistical analysis. Various software were used as tools in the measurements and analysis.

Performance testing and evaluation for virtualization software has been done before but not that much for SBCs. Virtualization using SBCs can be regarded as a rather new concept and not very much research has been done in that area yet. However, Roberto Morabito in his paper [2] evaluated the performance of different SBCs when running Docker on them. On top of performance evaluation, also power consumption was evaluated in that paper. For the actual testing he used among others the tools sysbench, mbw and iperf. These tools are also used in this thesis. The conclusions of

Morabito's paper included the fact that Odroid C2 outperformed all other devices in most of the test. It is therefore logical to use Odroid C2 in these experiments.

Another paper by Morabito [1] evaluates the overhead that Docker produces when compared to a native platform. These test were done on Raspberry Pi which is the most well-known SBC. The conclusion was that the overhead produced by Docker is negligible. Ismail et al. [5] also evaluated Docker for edge computing and came to a similar conlusions that Docker is a good solution for edge computing. There is not a lot of experiments done with KVM as virtualization software on an edge device but Christoffer Dall and Jason Nieh confirmed in their paper [7] that KVM is applicable on an SBC such as Odroid C2 and also has potential as virtualization software. It, however, requires a little setting up in terms of enabling certain properties of the device [6]. This could be one reason why KVM has not been tested that much.

This thesis consists of six sections. The second section is "Methodology" in which the experimental setup is explained as well as the performance measures and statistical analysis used in the thesis. The third section "Results and Analysis" presents the results and provides the data analysis. The fourth section "Conclusions" sums up the main conclusions of the thesis and analyses the used methods and tools. It also presents some ideas for future experiments on the research field. The fifth section "References" has all the references and the sixth "Appendix" has the tables of the thesis.

2 Methodology

The methods used in the tests included the usage of different tools and setups. All other test were performed using only the Odroid board itself except network performance tests since they require a device that sends traffic to and receives traffic from the Odroid. A Dell laptop, that was connected to the same local are network (LAN) with the Odroid, was used for network testing. Both of the devices were connected to the LAN via ethernet cables.

2.1 Virtualization setup

The virtualization software evaluated in this paper, KVM and Docker, have different approaches towards virtualization: KVM is a hypervisor, which runs complete virtual machines and requires hardware-level support for virtualization whereas Docker uses containers. In computing, certain software are designed to enable one computer, which in this case is called host computer,

to act like another computer. The other computer is called guest and the process is called emulation. In these experiments QEMU (Quick Emulator) was used as hardware emulator for KVM virtual machine.

The containers that Docker uses for virtualization are similar to virtual machines, only more lightweight. Rather than running an operating system on an emulated virtual hardware, Docker's approach only modifies the host operating system in a way that provides isolation for the specific processes. This is achieved for example by restricting what operating system resources a process can see and use as well as by assigning access control checks to all calls that the process makes to the system.

The principles used for configuring the virtualized environment were basically the same for both KVM and Docker as far as operating system is concerned. The guest operating system in both cases was an Ubuntu 16.04 xenial cloud OS. Docker version 1.12.6 was used. Version 2.5.0 of QEMU needed by KVM for managing guests was used in these tests.

In virtualization, it is possible to dedicate a particular physical CPU to a single virtual instance. This is called vCPU pinning or processor affinity and it affects the performance of the device. This way multiple different vCPU configurations can be assigned for the test. In the test performed for this thesis, however, a random setup was used in which no specific configuration was assigned. This kind of a configuration allows CPU to be utilized better than with a specific setup [3].

2.2 Experimental Setup and Design

Table 1: The Odroid C2's hardware features.

Features ↓ Device →	Odroid C2
Chipset	Amlogic S905
CPU	ARMv8 Cortex-A53 Quad Core 2GHz
GPU	3 x ARM Mali-450 MP2 700 MHz
Memory	2GB DDR3 912MHz
Flash Storage	MIcroSD, eMMC5.0
Ethernet	10/100/1000 Mb/s
USB connectivity	4 x USB 2.0 Host, 1 x USB 2.0 OTG
OS	Linux, Android

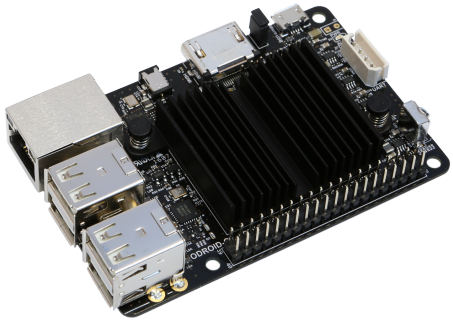

Figure 1: The Odroid C2 [8].

Figure 1 shows what an Odroid C2 looks like. It is a small single-board computers published by the company Hardkernel. The model C2 is quite powerful compared to other SBCs. It's hardware features are shown in Table 1. The Odroid was running Linux 3.14.79 at the time of the tests. An Intel Core i5 PC was used in the network test for sending and receiving network traffic. The PC was running Linux 4.4.0 and had an Intel Ethernet Connection I219-LM network adapter.

The sysbench CPU tests are ran with four different configurations on Docker and KVM: each with different number of CPUs enabled (1, 2, 3 and 4). Native platform is tested only with all four CPUs enabled. Integer calculations are ran with all CPUs enabled for all of the platforms. Both sysbench and dd disk I/O tests are ran with 1 GB and 2 GB of allocated RAM for all three platforms. RAM performance measurements with dd tool are done only with the full 2 GB of allocated memory for every platform. Mbw RAM tests are ran first with 1 GB of allocated memory and then with 2 GB of memory. Mbw tests evaluate RAM performance on every platform in three different areas: memcpy, dumb and mcblock.

Network testing, excluding web server tests, is done for every platform without any special configurations. The web server related tests, though, are done for every platform with varying number of connecting clients (50, 100 and 200 clients). Also, the total number of requests made by the clients ranges from 5000 to 100000.

2.3 Performance measures

Processor

The tools that were used to test CPU performance were a diverse benchmarking tool sysbench and a basic commandline command time. The number of threads was set to 4 so the command will start 4 threads. The maximum

prime number was set to 9999. This way the command starts 4 threads and verifies prime numbers until 9999 by doing standard division of the number by all numbers between 2 and the square root of the number. This makes the tool stress the cpu(s) and then print out statistics from which the execution time was recorded in seconds.

This initiates a calculation of all integers from 0 until 9999999. The command also prints out the time it took to calculate the integers and that was recorded in seconds.

Disk input and output

Disk input and output was tested with sysbench and the "dd" tool. Sysbench is used for stressing the board by writing or reading sequentially one 4 GB file and printing out statistics. From the printed statistics the execution speed is recorded in Mb/s. The dd tool is used for writing or reading 256000 16000 kB blocks which equal to 4.2 GB file. The tool also prints out results from which the writing or reading speed is saved in Mb/s.

RAM

RAM performance was tested with the Memory BandWidth benchmark or mbw and with the command "dd". The test with "dd" first creates a folder and then mounts a temporary filesystem to the folder that was made. After that the tool copies data to the temporary filesystem. It also prints out statistics from which the writing or reading speed is recorded in Mb/s.

The mbw tool copies arrays of data in memory and determines available memory bandwidth that way. The array size was set to 300 MB. The test consisted of ten loops of using three different methods of copying data: memcpy, dumb and mcblock. Every method prints out the speed or bandwidth of memory in the operation and also calculates and prints the average of those results. The average of every method was recorded in Mb/s.

Network

Two different methods were used for testing network performance. The other one was a quite basic test that tested TCP and UDP traffic both sending and receiving. This test was performed using a laptop which was connected to the same LAN as the board and a tool called Iperf which is a widely used tool for network testing. The other test included hosting a certain web server on the board and testing network performance when sending traffic to the web server from the laptop. A tool called Apache benchmarking (ab) tool was used for this test. The web servers used were Apache, Lighttpd and Nginx.

The Iperf TCP traffic testing was done via port 80 in the LAN. Testing time was limited to 20 seconds. When the test was finished Iperf printed out the bandwidth of the network traffic which was recorded in Mb/s. For UDP traffic the test was quite similar. It was performed also with Iperf and it had the same port. The difference between testing TCP and UDP traffic was that with UDP the bandwidth had to be assigned first as a parameter to Iperf and after the test it printed out the achieved bandwidth in Mb/s and if there occurred any packet loss during the test. Those statistics were then recorded.

Web servers

The web servers were hosted on the ODroid device on either native platform, KVM or Docker and traffic was sent from the laptop to the server. The ab tool allows one to send traffic to the web servers and then prints out statistics from the test. For all of the web servers the test was the same where the number of clients connecting to the server and the number of requests sent were varied: number of clients were 50, 100 and 200 and the number of requests were 5000, 10000, 20000, 30000, 40000, 50000, 70000 and 100000. The recorded statistics were the execution time of the test and the number of requests made per second. Failed requests were not allowed except in the test for KVM with 200 clients. The test were performed until the statistics didn't show any failed requests.

2.4 Statistical analysis

The collected sets of data were analyzed by calculating basic statistics for example means and by utilizing t-test. The t-test is used for testing if two sets of data are significantly different from each other from a statistical perspective by comparing the means of the data sets. The data sets being significantly different basically means that the difference in the means does not come from single or a few results that are far from the other results, thus changing the mean. To be significantly different most of the values of a data set have to be different from the other data set. There are various forms of t-tests for different initial conditions. In these tests, however, all the t-tests were for two samples and the variances of the tested sets of data were assumed to be equal.

When conducting a t-test first a null hypothesis is assigned, that is the original hypothesis before the actual testing. The null hypothesis in these experiments were that there are no differences in KVM's and Docker's performances. After that a significance level is determined. The significance

level indicates what percentage of certainty, that the two tested sets of data have no significant differences, is accepted. In these tests, a commonly used significance level of 5 % was used.

T-test mathematically represented

$$t = \frac{\bar{x_1} - \bar{x_2}}{\sqrt{\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}}}, \tag{1}$$

where $\bar{x_1}$ and $\bar{x_2}$ are the means, s_1 and s_2 the standard deviations and n_1 and n_2 the sizes of the first and second data sets, respectively.

The tools, that were used to analyze the collected data, were Microsoft Excel and LibreOffice Calc. The figures were created using both Excel and LibreOffice Calc. Excel was used for analyzing the data which included calculating statistics for data sets and t-testing.

3 Results and Analysis

3.1 Processor

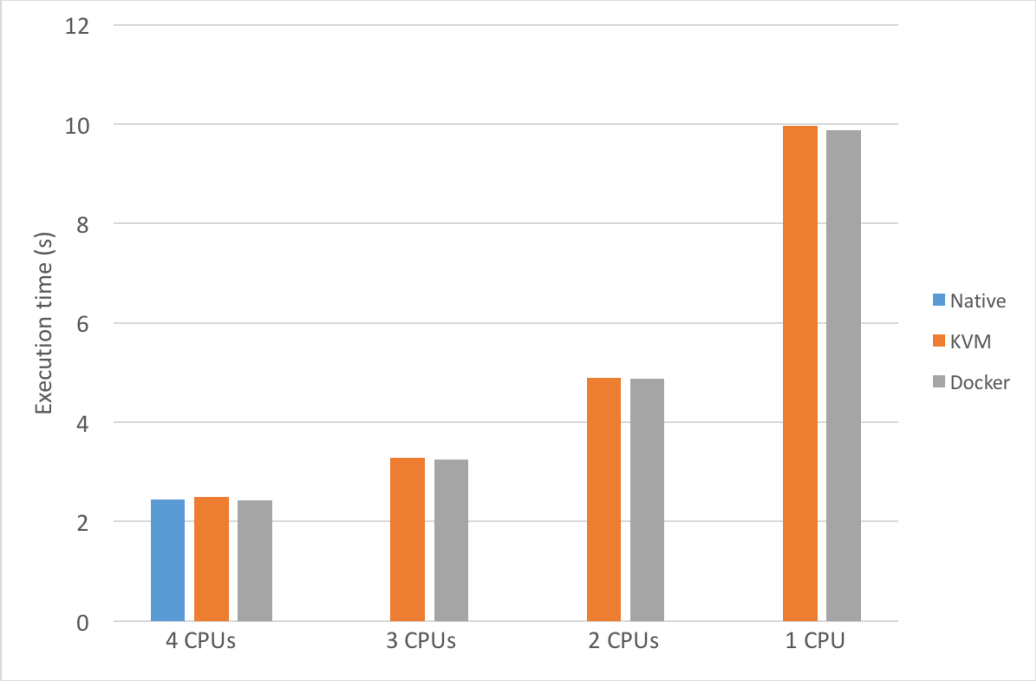

Figure 2: Sysbench CPU performance for different amounts of CPUs.

Figure 2 shows the processing time in seconds for different platforms and CPU amounts. As far as CPU performance goes Docker always outperformed KVM. Four measurements were performed with varying amount of CPUs used (1-4 CPUs). Native performance was measured with 4 CPUs only. Though the differences were relatively small, on average Docker was faster than KVM in every scenario. For cases with 3 and 4 CPUs enabled the p-values are smaller than the threshold (0.0031 and 0.000038) and over it for the rest. The p-values therefore confirm that Docker was faster in this area, at least when multiple CPUs were enabled. The graph shows that Docker was even faster than native with 4 CPUs enabled. P-value for that is 0.035 so it is under the threshold. However, the result is still questionable since it should not be possible for native platform to be slower than Docker.

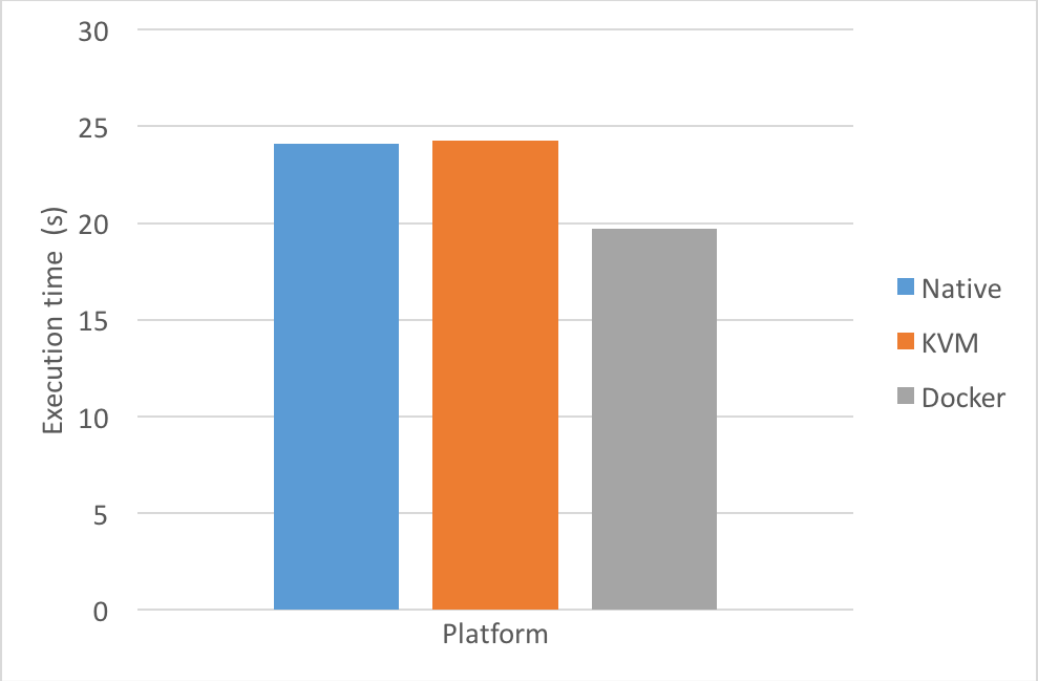

Figure 3: Integer calculation times for different platforms.

In Figure 3 there is presented the execution times for integer calculation for every platform. The integer calculation was performed in the way described in Methods.

These results are contrary to the results from the sysbench CPU results. In Figure 3 it seems like Docker would be considerably faster than either of the two other platforms. KVM still showed the slowest performance which was expected. However, the result that on Docker it would be much

faster to calculate integers than on native, makes these results unreliable. That is because Docker is ultimately ran on the native platform and it has to produce some overhead even though it could be negligibly small.

3.2 Disk input and output

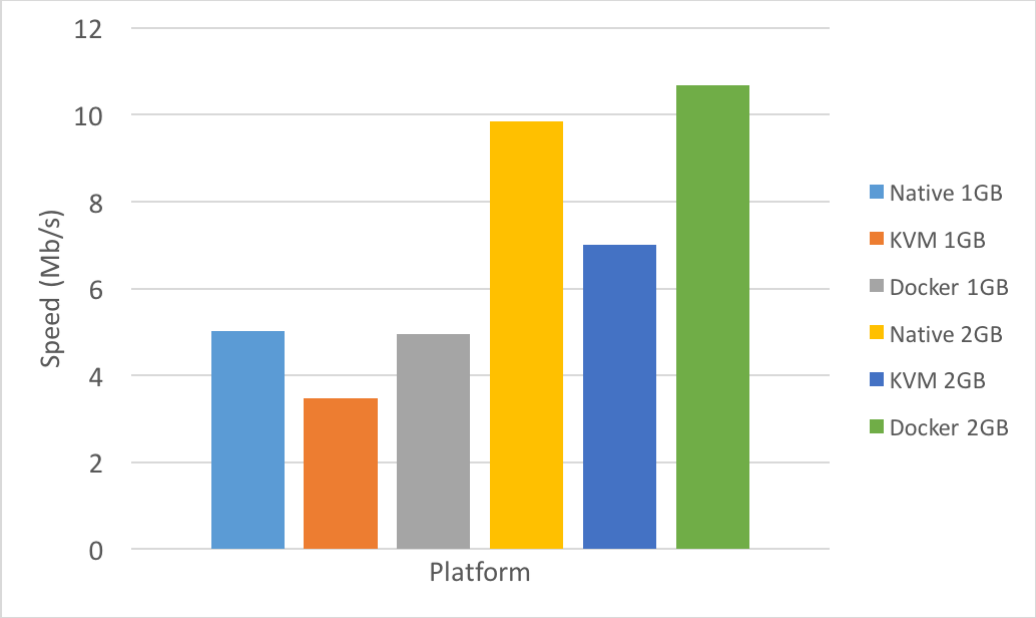

Figure 4: Sysbench disk writing speed for different amount of allocated RAM.

Figure 4 shows the disk writing speed in megabytes per second for different platforms and amounts of memory allocated. The different amounts of memory were 1 GB and 2 GB. The graph shows that the platforms native and Docker performed clearly better than KVM but differences between the two were pretty small. Native had slightly better performance with 1GB of memory and Docker with 2 GB of memory. The p-values for those two cases are 0.62 and 0.12 respectively. They both are way over the threshold, thus making the result, that either one would be clearly faster, unreliable. Every platform's performance increased as more memory was allocated.

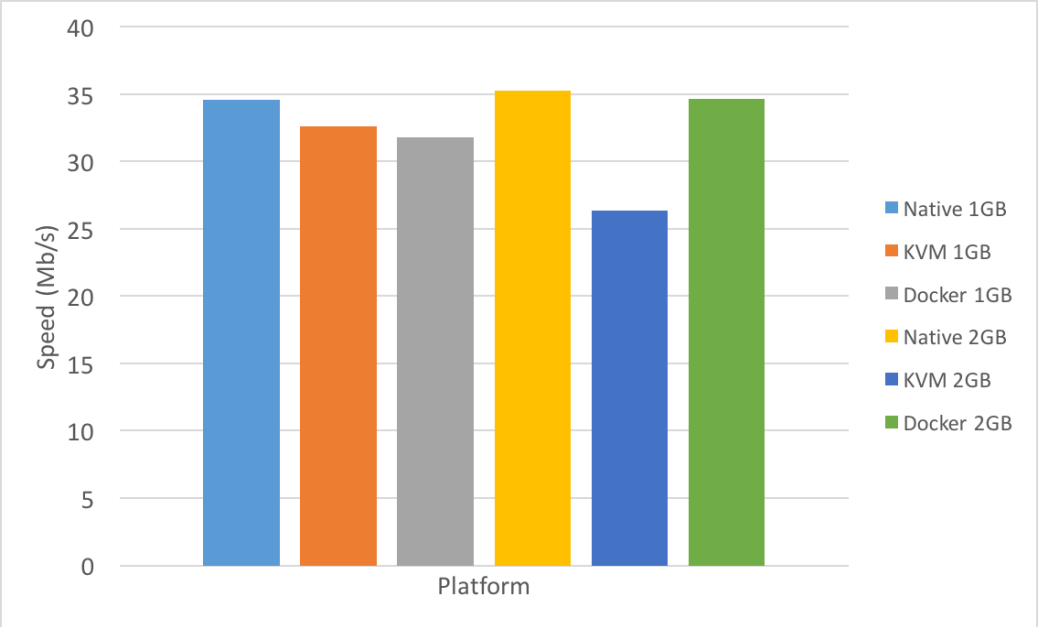

Figure 5: Sysbench disk reading speed for different amounts of allocated RAM.

Figure 5 shows the disk reading speed in megabytes per second for different platforms and amounts of memory allocated. The graph shows that native had the best performance overall. Native and Docker increased their performances when 2GB of memory was allocated instead of 1GB. On the other hand KVM's performance decreased, for some reason, when memory allocation was increased (32.6 to 26.4 Mb/s). The graph shows that would have performed better with 1 GB of memory than Docker but the p-value for that case is 0.90. Therefore it is clear that the result is not reliable and that KVM is not faster than Docker in that case.

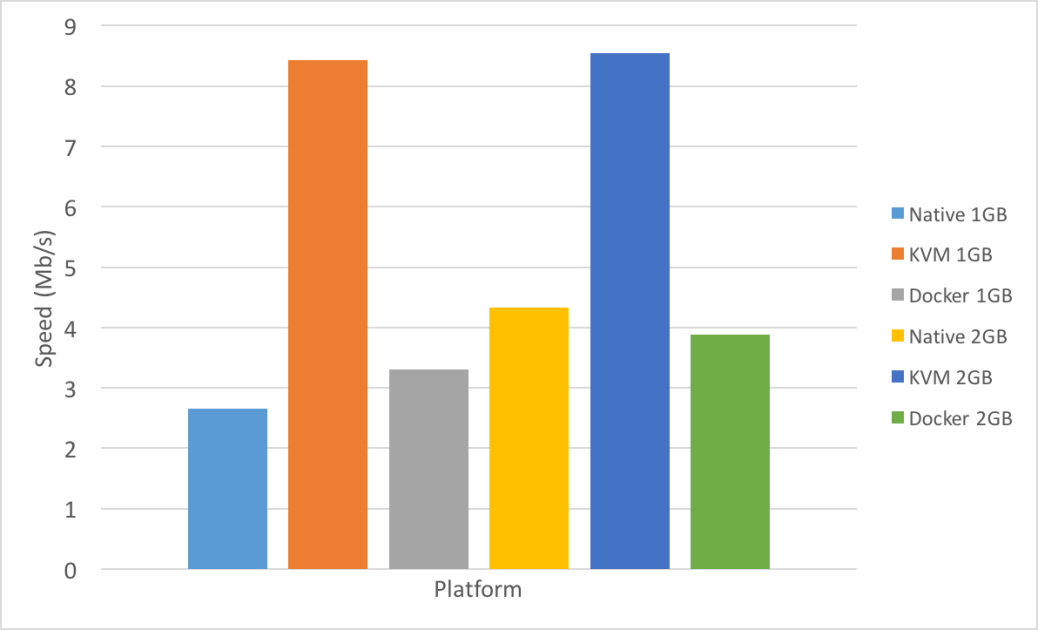

Figure 6: Disk reading speed for different amounts of allocated RAM measured with the dd tool.

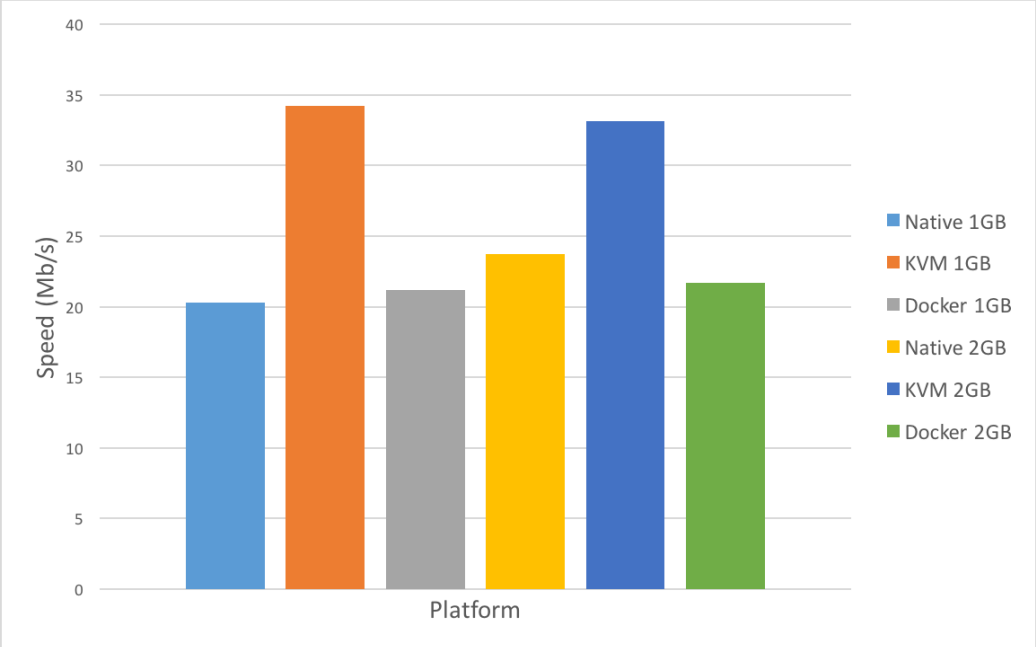

Figure 7: Disk reading speed for different amounts of allocated RAM measured with the dd tool.

Figures 6 and 7 present the results from disk writing and reading performance testing, respectively, with the dd tool. The results are in conflict with the conclusion from the other disk I/O test that were done with sysbench. In fact, the results show that KVM would be quite clearly faster than the other platforms, both native and Docker. This does not make any sense since KVM should produce at least some overhead. These kind of results could be some fluctuation in the measurements but because the results shown in the figures are averages it cannot be random error. Also, the p-values for the comparisons of native and KVM with 1 GB and 2 GB memory allocations are $4.60576 * 10^{-}24 \approx 0.000$ and $4.98974 * 10^{-}14 \approx 0.000$. The p-values are clearly under the threshold thus it can be concluded that the data sets are significantly different. However, KVM cannot be faster than native platform because of the fact that KVM is ultimately ran on native and must therefore produce at least some overhead.

3.3 RAM

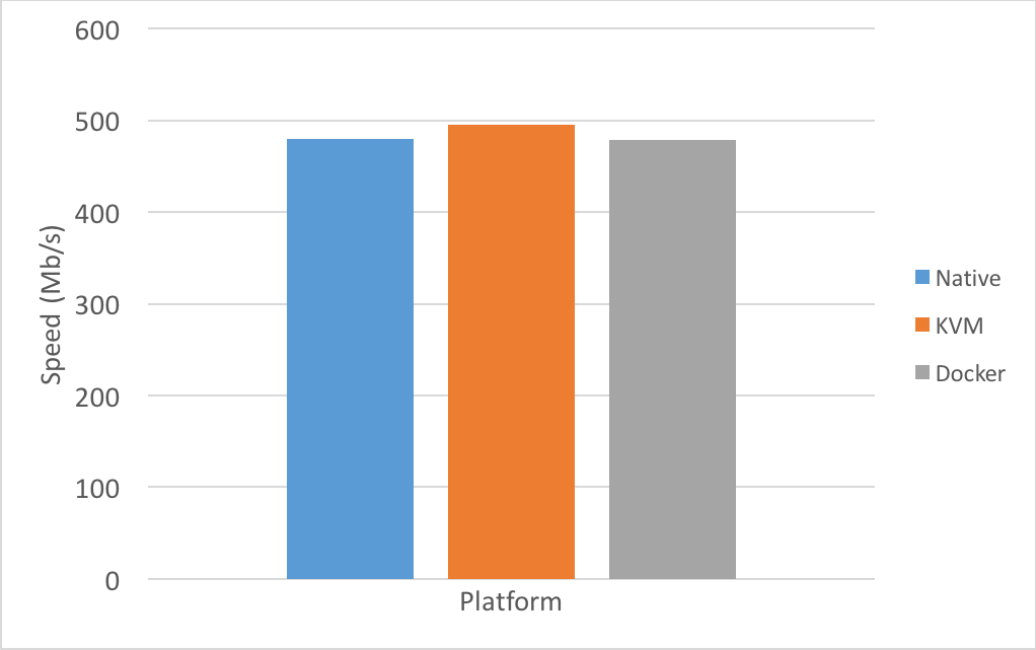

Figure 8: The average RAM writing speeds for different platforms measured with the dd tool.

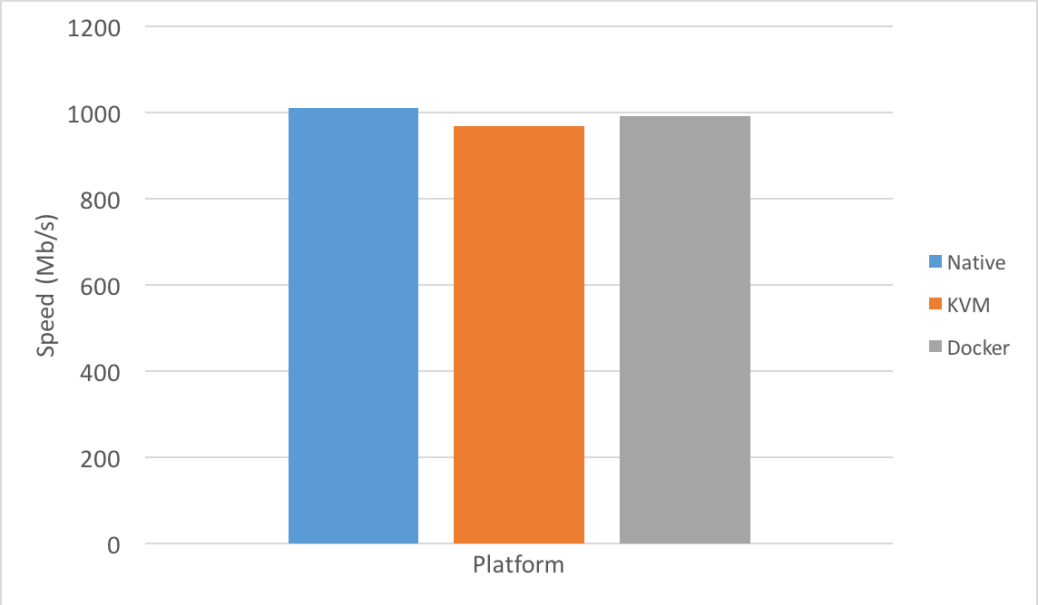

Figure 9: The average RAM reading speeds for different platforms measured with the dd tool.

Figures 8 and 9 present the results from RAM writing and reading performance testing, respectively, with the dd tool. It can be seen from the results that all of the platforms' reading and writing speeds are very similar. It seems, though, that KVM would be the fastest platform at RAM writing speed. Comparing native platform to KVM in that case gives a p-value of 0,0399. The data sets can therefore be considered to be significantly different. As mentioned before, KVM cannot be faster than native. Consequently, the results from RAM writing speed are unreliable.

On the other hand, the results from RAM reading tests seem logical. The graph shows that native platform would be the fastest, then docker and then KVM. However, the p-values for every three comparisons are over the threshold. Thus, no real conclusions can be drawn from the results gotten with dd.

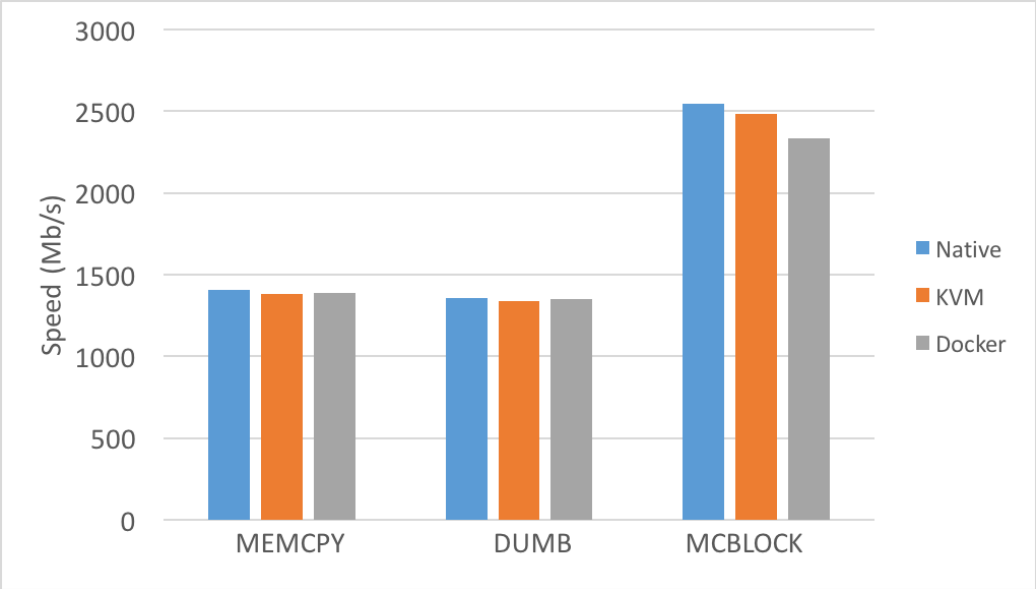

Figure 10: Memcpy, dumb and mcblock speeds in megabytes per second for 1 GB of memory allocated.

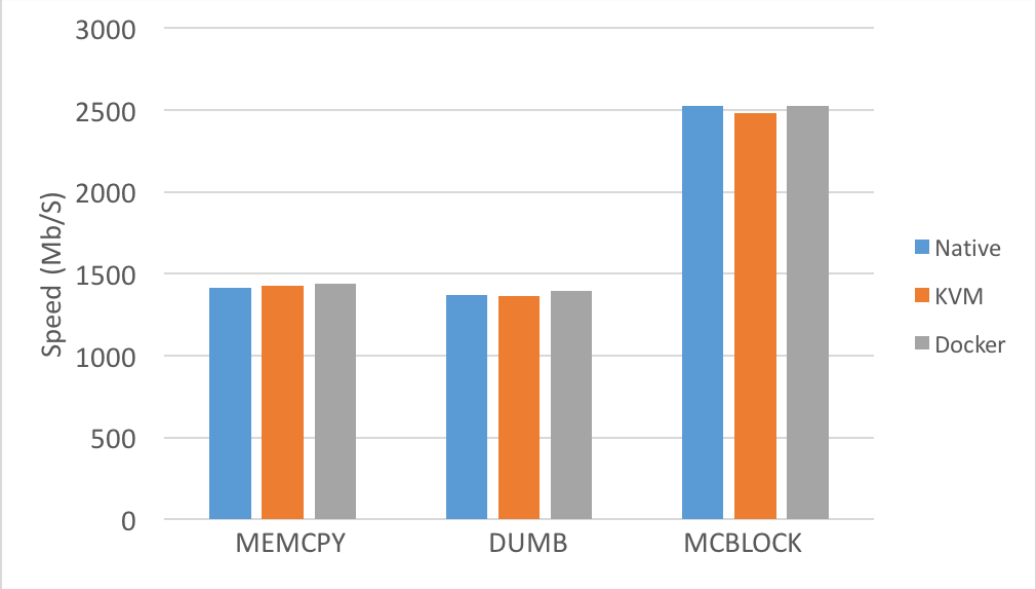

Figure 11: Memcpy, dumb and mcblock speeds in megabytes per second for 2 GB of memory allocated.

Mbw test results are shown in the Figures 10 for 1 GB of allocated memory and 11 for 2 GB of allocated memory. The results show the speed

of the different methods in megabytes per second which were memcpy, dumb and mcblock.

It is shown in the graph that when 1GB of memory was allocated native platform had the best performance in all areas. Docker performed slightly better than KVM at memcpy and dumb and KVM had better performance in mcblock.

On the other hand, when 2GB of memory was allocated all the results were very close and the platform with the fastest speeds varied. Thus, none of the platforms was clearly the fastest.

3.4 Network

TCP

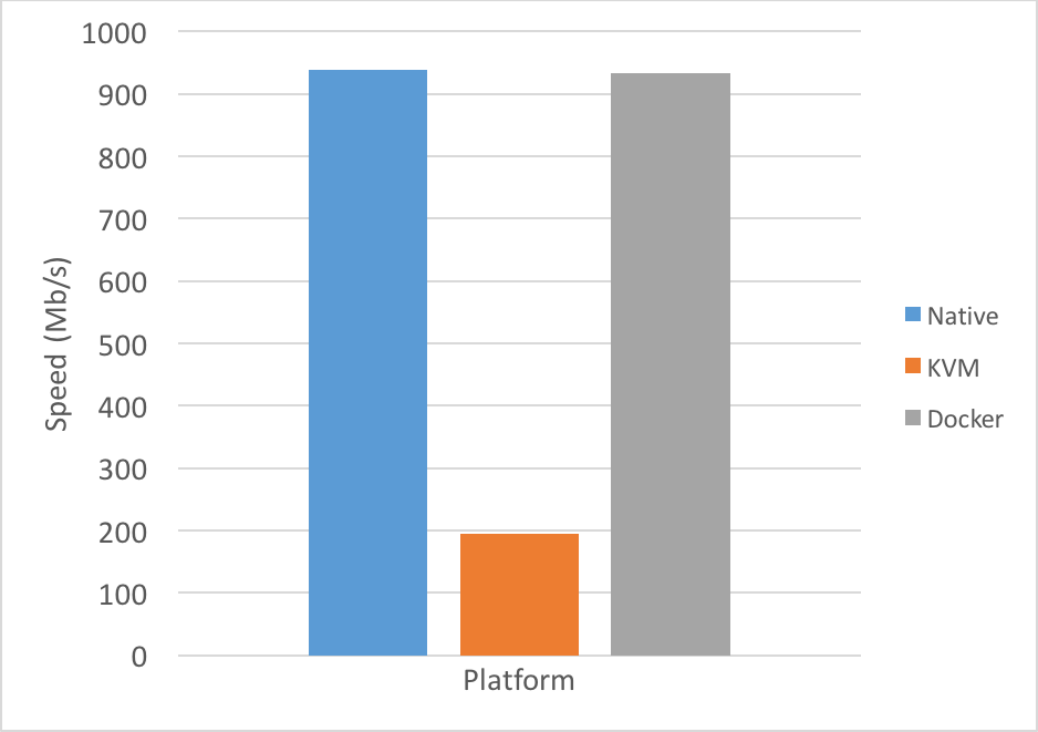

Figure 12: Network bandwidth on different platforms when sending traffic.

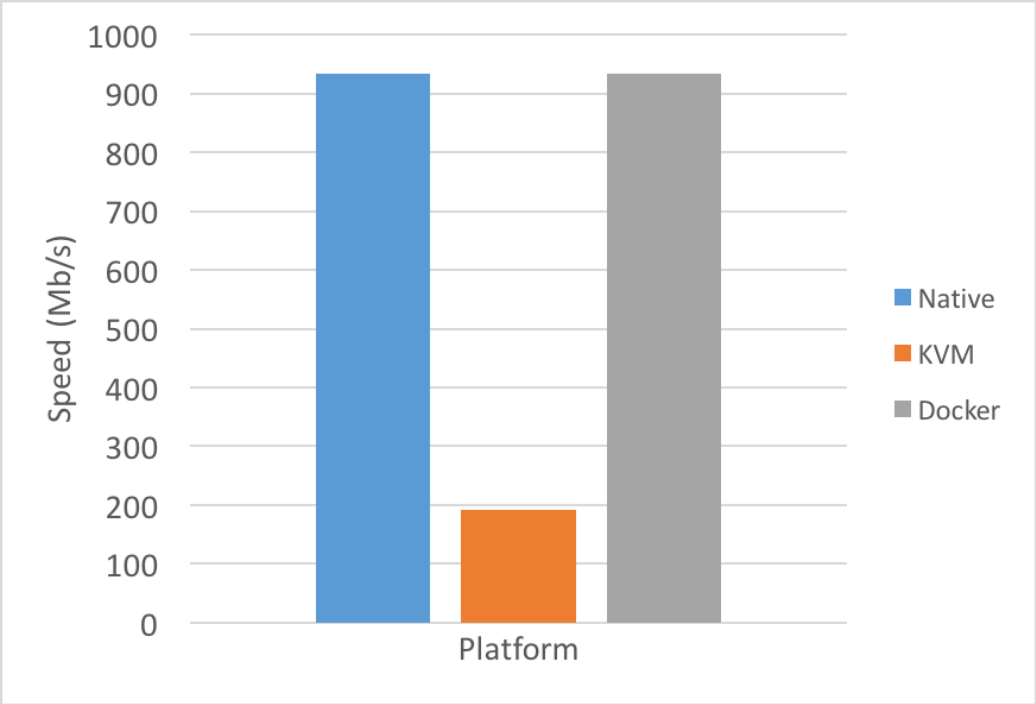

Figure 13: Network bandwidth on different platforms when receiving traffic.

Figures 12 and 13 show the network bandwidth in megabytes per second for both sending traffic from the board and receiving traffic to the board. The bandwidths shown in the figure are averages.

Both of the graphs show clearly that the network performance of KVM fell far behind native and Docker. The p-values also confirm this conclusion. For sending traffic KVM had an average of 192 Mb/s whereas native and Docker had the same average of 940 Mb/s. For receiving traffic KVM had an average of 195 Mb/s, native had 939 Mb/s and Docker had 933 Mb/s.

UDP

The data gathered from the UDP tests is shown in Tables 2 and 3, which are found in Appendix. Docker managed to experience 0% loss on sent packets up to the speed of 50 Mb/s and even at 100 Mb/s there was a negligible loss of 1 packet out of 84672. The maximum bandwidth for Docker was 386 Mb/s and that came with a packet loss of 0.22%. The results for KVM are very similar to the results for Docker. No loss up to 50 Mb/s and very little, though more than Docker, loss at 100 Mb/s (0.079%). The maximum bandwidth for KVM was at 310 Mb/s and the packet loss at that speed was 0.14%.

Docker managed to receive all sent packets with 0% loss up to 100 Mb/s speed. The next measurement was done with the option 999 Mb/s which resulted to 132 Mb/s and 84% loss. KVM managed to receive all packets with 0% loss up to a speed of 50 Mb/s. At 100 Mb/s there was some loss (0.042%) which is quite small but noticeable. It also more than the loss that happened with Docker (0%). The last measurement was done with the option 999 Mb/s which resulted to 156 Mb/s and a loss of 80%. KVM managed to receive packets with higher bandwidth than docker and with less packets lost. However, KVM also experienced packet loss at lower bandwidth than Docker.

Web servers

The Apache web server was able to receive 100% of the sent packets on every configuration when it was running on either native platform or Docker. With KVM the test had no loss only when the number of clients was 50. When there were 100 clients only one measurement with 10000 sent requests was completed without any loss and when there were 200 clients there also some loss on every measurement.

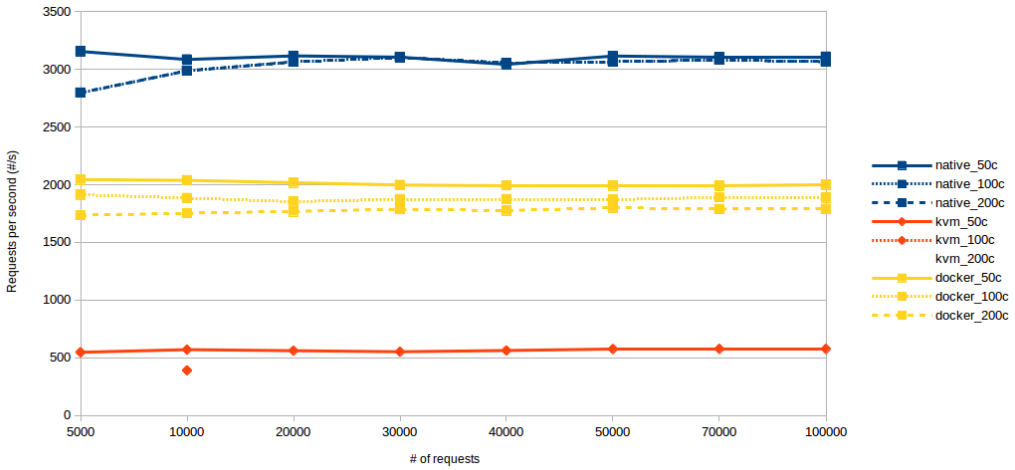

Figure 14: Number of requests per second in the test on the Apache web server.

Figure 14 shows the number of requests per second for different numbers of total requests and platforms. The graph only shows one complete line for KVM because KVM was not able to finish enough measurements without packet loss.

For the native platform it seems that neither the total number of requests or the number of clients have any significant effect on the number of requests per second. For Docker there is pretty subtle but clear drop in performance when the number of clients is increased but nothing when the total number of requests is increased. For KVM there is not enough data to draw any conclusions since packet loss occurred as the number of clients was increased. Allowing the loss of packets of course led to an increase in "performance".

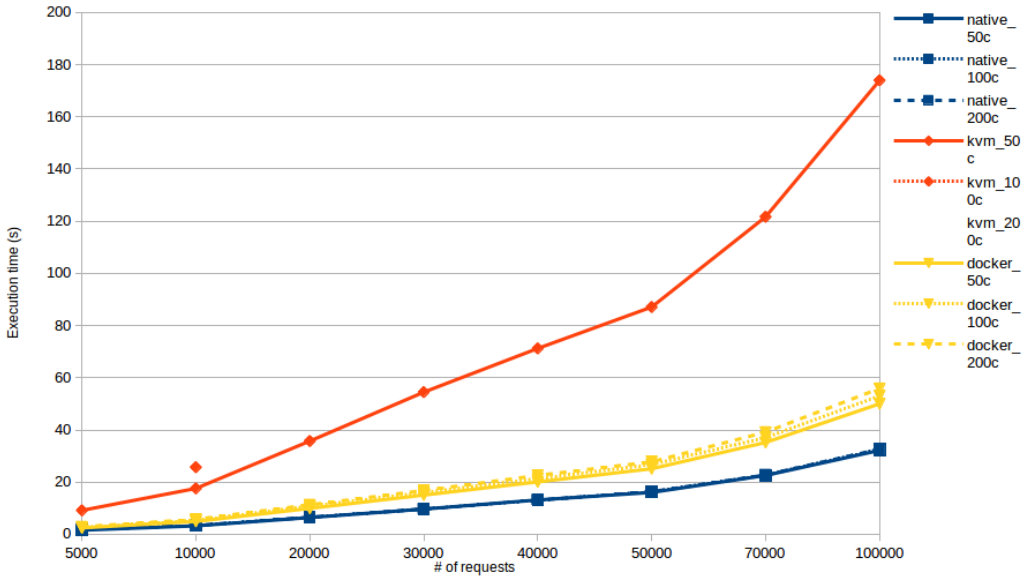

Figure 15: The execution time of the test on the Apache web server.

In Figure 15 there is shown the execution time of the test in seconds for different numbers of total requests and platforms. The results in execution time are very similar to the results in the number of requests per second as one could expect.

Native platform had the best performance and basically no effect when either the number of clients or the total number of requests were increased. For Docker there was again a little drop in performance when the number of clients was increased. For KVM there is no reliable data available because of the loss of packets.

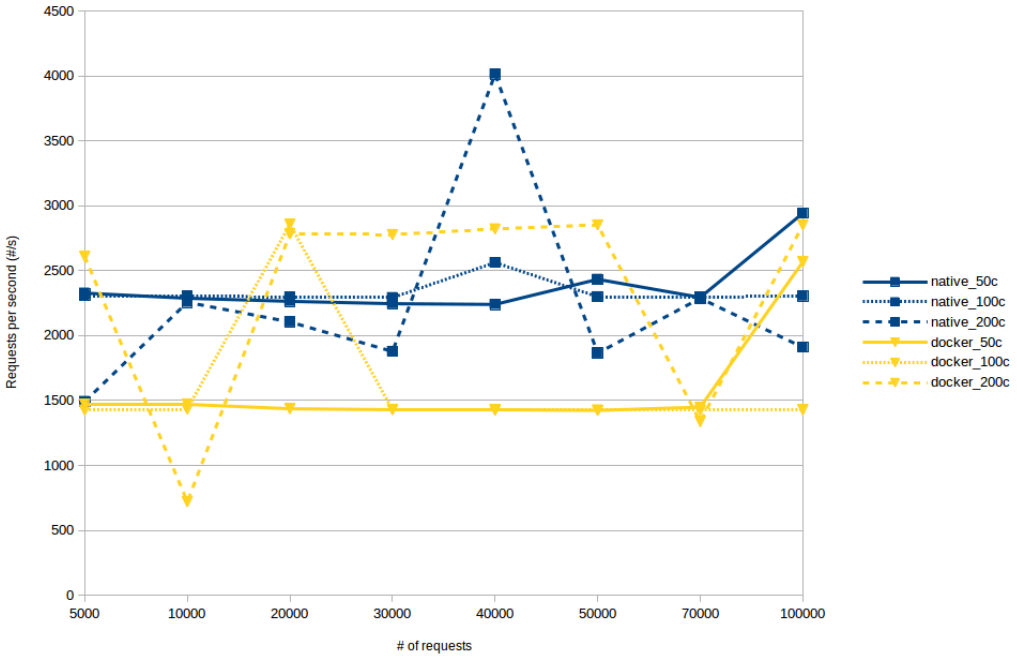

Figure 16: Number of requests per second in the test on the Lighttpd web server.

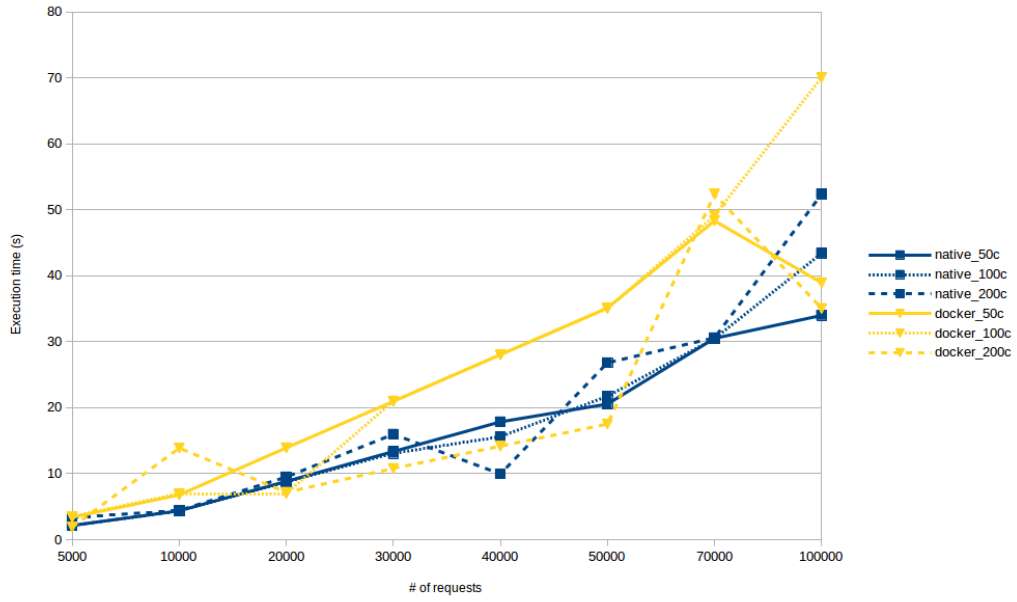

Figure 17: The execution time of the test on the Lighttpd web server.

Figure 16 shows the number of requests made per second on the Lighttpd web server with a different number of total requests made in the test. Figure 17 shows the execution time of the performed test on the Lighttpd web server in seconds. Both of the graphs show some scattering but also that native platform is on average faster than Docker. Not one measurement was able to be completed on KVM so no data of KVM is displayed on the graphs.

The graphs also show clearly that when 50 clients were connecting to the server native platform is far faster. When there were more clients connecting to the server there was also more scattering and the results are not that clear. It seems though that also when 100 clients were connecting to the server the native platform was faster. The data with 200 clients is too scattered to have any conclusions drawn from.

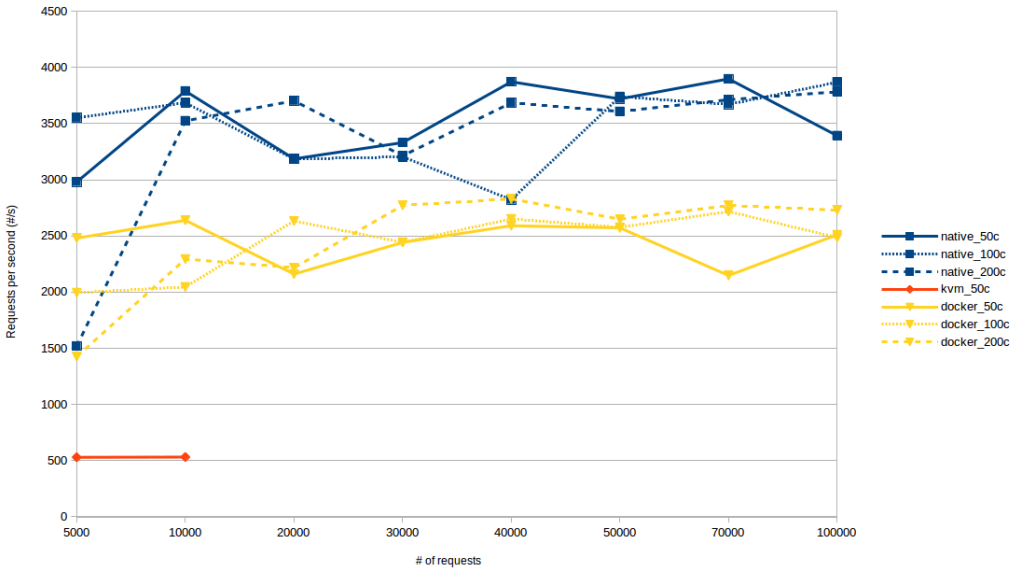

Figure 18: Number of requests per second in the test on the Nginx web server.

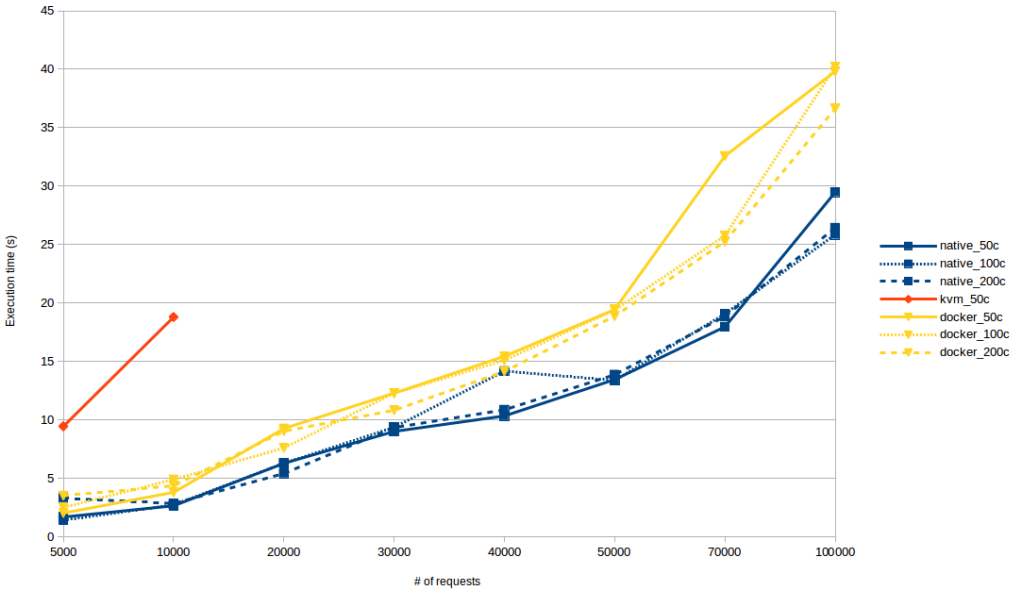

Figure 19: The execution time of the test on the Nginx web server.

Figure 18 presents the number of requests made per second on the Nginx web server with a different number of total requests made in the test. Figure 19 presents the execution time of the performed test on the Nginx web server in seconds.

The results from testing the web server Nginx were pretty logical but had also some scattering. Native and Docker were able to complete all the measurements but KVM only finished 2 of them both of which were with 50 clients. The results also follow the same pattern as the results from other web server tests: native is the fastest then Docker and then KVM. KVM doesn't really have enough data to draw solid conclusions but the results from the two finished tests show so much slower performance that it is safe to say that KVM is the slowest platform when it comes to this test.

When comparing the results from with the same platform but different number of clients connecting they show no clear significant variation. As in all of the web server test results the execution time rises fairly steadily when the total number of requests is increased.

3.5 Summary

As far as CPU performance is concerned Docker outperformed KVM. The results indicated that docker would have been even faster than native platform but due to the negligible difference they are considered almost equally

fast. This results points out the fact that Docker does not produce hardly any overhead when it comes to CPU stress. On the other hand KVM had systematically slower processing time thus it is safe to say that CPU performance on Docker is better than on KVM.

No real conclusions can be drawn from disk I/O testing. The results were either not possible or not clear enough. The results from testing with dd tool showed that KVM would be considerably faster than Docker and even native platform. KVM being faster than Docker could of course be possible but KVM being faster than native is not possible due to the fact that KVM has to produce some overhead. Therefore, the results from testing with dd are unreliable and cannot be used for drawing conclusions. Sysbench produced more logical results but the calculated p-values showed that the data sets are not likely to be significantly different from each other. Consequently, no clear conclusions can be drawn from those results either.

The results for RAM performance have very few statistically significant differences between the platforms, if any. The fastest platform cannot be determined with enough clarity. Also, there is again illogical results from testing with dd tool. The RAM writing speed seems to be the fastest on KVM but that cannot be the case. The conclusion for RAM performance is that all the platforms utilize RAM equally well.

For network performance the results are the clearest. The results from TCP traffic testing show significantly better performance on Docker compared to KVM. Also native platform showed expected performance since it was the fastest platform. UDP traffic shows almost equally good performance for both of the platforms as far achieved bandwidth and the loss of packets go.

The results from web server testing show the server performing a lot better when running on Docker rather than on KVM, thus supporting the TCP test results. All web servers running on Docker were able to complete all the test without any packet loss whereas on KVM only the Apache web server was able to complete one full test with 50 clients. Also the speed at which the 50 client test was completed differed significantly when comparing docker to KVM further proving that Docker has better network performance. The p-values are shown in Table 4.

4 Conclusions

The purpose of this thesis was to determine which one the two virtualization software is better to be used on an edge device of a small network. In this experiment, the comparison of the software was based only on performance. Different areas of performance were evaluated with specific performance measures that tested CPU, disk I/O, RAM and network performance. These performance measures consisted of programs that are designed to stress and test different parts of computers. The data gotten from the measurements was then analyzed using statistical analysis for example t-test. The analysis provided the necessary information for determining which platform is better for virtualization in the given conditions.

The performance measures used in the experiment produced, in most cases, rational results. The tools were selected according to earlier usage in performance testing. The only tool that produced unreliable results was dd. Thus, dd should probably not be used for performance testing in the future experiments. The goal was that two different tools would be used to test each area except for network testing. That goal was only reached for CPU testing since the results from dd were not rational and dd was used for testing disk and RAM. More accurate results could be achieved by using more performance measures and taking more measurements.

The most of the data analysis is based on t-testing. T-test is an easy way to compare two data sets. It compares the means of the data sets and determines whether or not the differences in the data sets are statistically significant. For this experiment's purposes, t-test provides good enough information about the significance of the differences of the data sets. T-test was not necessary in the analysis of network performance because the test that were ran on KVM did not produce enough data.

The overall conclusion drawn from the results is that Docker is better for virtualization when comparing it to KVM. CPU and network performance are clearly better when using Docker. Though the difference between the two platforms was not large when it comes to CPU performance, it was systematically consistent and thus very clear. On the other hand, the difference in network performance was shown in multiple ways. Docker was significantly faster and finished all of the tests whereas KVM could not finish a single test with 100 or 200 connecting clients. Neither disk I/O nor RAM tests declared clearly which is the faster platform. In both sets of tests, dd tool was used and it produced unreliable results that could not be used for drawing any conclusions. Sysbench's results from disk I/O showed no significant differences in performance between the platforms. Same applies for mbw's results from RAM testing.

The next step in testing the differences of Docker and KVM could be to measure the power consumption of these two technologies. Especially when using virtualization on SBCs the power consumption should be minimized. That is because they would mainly be used in small networks of devices for example home entertainment systems which are not supposed to consume power heavily.

5 References

[1] Roberto Morabito, "A Performance Evaluation of Container Technologies on Internet of Things Devices.", 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2016

[2] R. Morabito, "Virtualization on Internet of Things Edge Devices with Container Technologies: a Performance Evaluation.", IEEE Access, 2017

[3] C. Xu, Z. Zhao, H. Wang, R. Shea, J. Liu, "Energy Efficiency of Cloud Virtual Machines: From Traffic Pattern and CPU Affinity Perspectives", IEEE Systems Journal, 2015

[4] H. Chang, A. Hari, S. Mukherjee, T. V. Lakshman, "Bringing the Cloud to the Edge", 2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2014

[5] B. Ismail, E. Goortani, M. Ab Karim, W. Tat, S. Setapa, J. Luke, O. Hoe, "Evaluation of Docker as Edge Computing Platform", IEEE Conference on Open Systems (ICOS), 2015

[6] G. Deka, P. Das, "Design and Use of Virtualization Technology in Cloud Computing", IGI Global, 2017

[7] C. Dall, J. Nieh, "KVM/ARM: the Design and Implementation of the linux ARM Hypervisor", Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems ASPLOS'14, 2014

[8] Odroid C2, Available: http://www.hardkernel.com/main/products/prdt_info.php?g_code=G145457216438

[9] Odroid models, Available: http://www.hardkernel.com/main/products/prdt_info.php

[10] Raspberry Pi models, Available: https://www.raspberrypi.org/products/

6 Appendix

Table 2: The achieved bandwidth in Mb/s and the packet loss percentage when sending traffic to the hosted web server for each assigned bandwidth for both platforms (bandwidth/percentage).

Platform ↓ Bandwidth (Mb/s) →	10	20	50	100	999
KVM	9.98/0	20/0	49.8/0	98.5/0.079	311/0.14
Docker	9.98/0	19.9/0	50/0	99.5/0.0012	386/0.22

Table 3: The achieved bandwidth in Mb/s and the packet loss percentage when receiving traffic to the hosted web server for each assigned bandwidth for both platforms (bandwidth/percentage).

Platform ↓ Bandwidth (Mb/s) →	10	20	50	100	999
KVM	10/0	20/0	50.1/0	100/0.042	156/80
Docker	10/0	20/0	50/0	101/0	132/84

Table 4: The p-values of all of the conducted t-tests. Values below 0,05 are bolded.

Test ↓ Platforms →	Docker-KVM	Native-Docker	Native-KVM
Sysbench (1 CPU)	0,256	-	-
Sysbench (2 CPU)	0,443	-	-
Sysbench (3 CPU)	**0,000**	-	-
Sysbench (4 CPU)	**0,000**	**0,035**	**0,012**
Time (CPU)	**0,000**	**0,000**	0,591
Sysbench Read 1GB (Disk)	0,895	0,257	**0,000**
Sysbench Write 1GB (Disk)	**0,000**	0,617	**0,000**
Sysbench Read 2GB (Disk)	**0,000**	0,304	**0,000**
Sysbench Write 2GB (Disk)	**0,000**	0,119	**0,000**
dd Read 1GB (Disk)	**0,000**	**0,000**	**0,000**
dd Write 1GB (Disk)	**0,000**	**0,000**	**0,000**
dd Read 2GB (Disk)	**0,000**	**0,001**	**0,000**
dd Write 2GB (Disk)	**0,000**	0,505	**0,000**
dd Read (RAM)	0,442	0,579	0,121
dd Write (RAM)	**0,042**	0,828	**0,040**
mbw memcpy 1GB (RAM)	0,852	0,073	0,376
mbw dumb 1GB (RAM)	0,480	0,103	0,264
mbw mcblock 1GB (RAM)	**0,000**	**0,000**	**0,000**
mbw memcpy 2GB (RAM)	**0,031**	0,302	0,103
mbw dumb 2GB (RAM)	**0,001**	0,560	0,046
mbw mcblock 2GB (RAM)	0,431	0,483	0,183
Network receive	**0,000**	1	**0,000**
Network send	**0,000**	**0,016**	**0,000**