Big Data Research 3 (2016) 24–28

Contents lists available at ScienceDirect

Big Data Research

www.elsevier.com/locate/bdr

Analysis of a Network IO Bottleneck in Big Data Environments Based on Docker Containers ☆

P. China Venkanna Varma *, K. Venkata Kalyan Chakravarthy, V. Valli Kumari, S. Viswanadha Raju

ARTICLE INFO

Article history:
Received 1 June 2015
Received in revised form 29 November 2015
Accepted 1 December 2015
Available online 4 January 2016

Keywords:
Containers
Context switching
Docker
Hadoop
Map reduce

ABSTRACT

We live in a world increasingly driven by data with more information about individuals, companies and governments available than ever before. Now, every business is powered by Information Technology and generating Big data. Future Business Intelligence can be extracted from the big data. NoSQL [1] and Map-Reduce [2] technologies find an efficient way to store, organize and process the big data using Virtualization and Linux Container (a.k.a. Container) [3] technologies.

Provisioning containers on top of virtual machines is a better model for high resource utilization. As the more containers share the same CPU, the context switch latency for each container increases significantly. Such increase leads to a negative impact on the network IO throughput and creates a bottleneck in the big data environments.

As part of this paper, we studied container networking and various factors of context switch latency. We evaluate Hadoop benchmarks [5] against the number of containers and virtual machines. We observed a bottleneck where Hadoop [4] cluster throughput is not linear with the number of nodes sharing the same CPU. This bottleneck is due to virtual network layers which adds a significant delay to Round Trip Time (RTT) of data packets. Future work of this paper can be extended to analyze the practical implications of virtual network layers and a solution to improve the performance of big data environments based on containers.

1. Introduction

Data are unorganized facts that need to be processed. When data are processed, organized and structured in a given context so as to make them useful, they are called Information [10]. Now, every business is powered with Information Technology (IT) and generating big data. There is a potential demand to process big data for rapid and accurate business decisions. Data scientists divided big data into five dimensions: Volume, Value, Velocity, Variety and Veracity (simply Big data 5V's). Volume refers to vast amounts of data generated every second. To create Value for the business, volumes of data should be processed to determine the relevance within the data sets. Velocity refers to the data processing at data gathering speed. Variety refers to managing, merging and processing of structured and un-structured data. Veracity refers to the biases, noise and abnormality in data.

Virtualization technology became de facto standard for all public and private cloud requirements. Virtualization has consolidated all the hardware components and created software redundancy layers for elastic work loads. Docker framework introduced manageable Linux containers [3] called Docker Containers. Docker is an open platform for developers and system administrators to build, ship, and run distributed applications. Hadoop is an open-source software framework for storing and processing the big data in a distributed fashion on large clusters.

Nowadays, data centers are adopting Docker to provision Hadoop [4] Clusters (HC) for elastic work loads and multi-tenant service models. Hadoop cluster nodes requires good amount of CPU cycles, Random Access Memory (RAM) and Resource IO to execute Map-Reduce operations. Virtualization solved orchestration of the dynamic resources for data centers, but virtualization overhead is the biggest penalty. "Every single CPU cycle that goes to the Hypervisor is wasted," and "Likewise every byte of RAM". Containers solved orchestration of dynamic resources within an operating system by virtualizing necessary components. If a container runs inside a VM then the network IO path has two extra virtual layers: one layer between Hypervisor and VM and another layer between VM and container. These two extra network virtual-

☆ This article belongs to Big Data Networking.

* Corresponding author.
E-mail addresses: pc.varma@gmail.com (P. China Venkanna Varma), tokalyankv@gmail.com (V. K.), vallikumari@gmail.com (V. Valli Kumari), svraju.jntu@gmail.com (S. Viswanadha Raju).

http://dx.doi.org/10.1016/j.bdr.2015.12.002
2214-5796/

ization layers adds a significant delay to the RTT of data packets in the transmission.

We installed a Hadoop [4] cluster in our incubation lab and executed Hadoop benchmarks such as TestDFSIO-read [5], TestDFSIO-write [5], TeraSort [5] and TeraGen [5] against the number of the containers (Hadoop cluster nodes). We considered three use cases: Native, VMs and Containers while executing benchmarks. Each use case executed and analyzed the throughput of Hadoop cluster. TestDFSIO is a storage throughput test that is split into two parts: TestDFSIO-write writes 1 000 020 MB (about 1 TB) of data to HDFS, and TestDFSIO-read reads it back in. Because of the replication factor, the write test does twice as much I/O as the read test and generates substantial network traffic. These write tests spread across 140 map tasks. TeraSort sorts a large number of 100-byte records. It does considerable computation, networking, and storage I/O, and is often considered to be representative of real Hadoop workloads. It splits into three parts: generation, sorting, and validation. TeraGen creates the data and is similar to TestDFSIO-write except that significant computation is involved in creating the random data. The Map tasks write directly to HDFS so there is no reduce phase. TeraSort does the actual sorting and writes sorted data to HDFS in a number of partitioned files. A total of 280 Map tasks was used for TeraGen, 280 simultaneous (several thousand in all) Map tasks and 70 reduce [2] tasks for TeraSort, and 70 Map tasks for TeraValidate.

Based on test results, as the number of containers increases on a VM or a physical machine, there is a significant overhead on the big data operations. This overhead is due to a raise in the network IO RTT when the CPU is busy. If the CPU is busy, the network IO operation is in pending state and waiting for its scheduled slot to execute, this will increase the RTT of the network IO packet and leads to low network throughput. Ideally Hadoop throughput is linear, overall cluster throughput increases by adding new nodes. Based on our test results, we observed that the Hadoop cluster throughput is not linear with the nodes sharing the same CPU.

The outline of this paper is as follows: (1) Explore how container networking works? (Section 2). (2) Identify the important factors of the CPU context switch latency (Section 3). (3) Identify the effects of virtual network layers on the RTT (Section 3). (4) Experiment setup, big data environment and test cases (Section 4). (5) Analysis of virtual layer overhead (Section 5). (6) Analysis of Hadoop cluster performance (Section 6). (7) Conclusion (Section 7).

2. Docker networking

Docker has a software Ethernet bridge called “docker0” build on top of Linux “bridge-utils” [6]. When docker starts, it will create a default bridge “docker0” inside the Linux kernel and a virtual subnet to exchange the packets between containers and host. Docker randomly chooses the IP address range not used by host and defined by RFC 1918. When a container is created a pair of virtual interfaces will be initialized, those are similar to both the ends of a tube where a packet sent from one end will be delivered to another end. One virtual interface given to the container and another one hooked into docker0 to communicate with host and other containers. Fig. 1 describes the docker’s docker0 and virtual interfaces. Communication between containers depends on the network topology and system firewall (iptables). By default docker0 bridge allows inter container communication. Docker never do changes to the iptables to allow a connection to docker0 bridge, it uses a flag “–iptables=false” when docker starts. Otherwise docker will add a default rule to the FORWARD chain with a blanket ACCEPT policy if you retain the default “—icc=true”, else will set the policy to DROP if “—icc=false”.

We observed that docker adds a network virtualization layer in the form of a software bridge which adds an extra hop in the network path which adds some delay in the network packet transmission. We analyzed this overhead compared to native host network IO throughput.

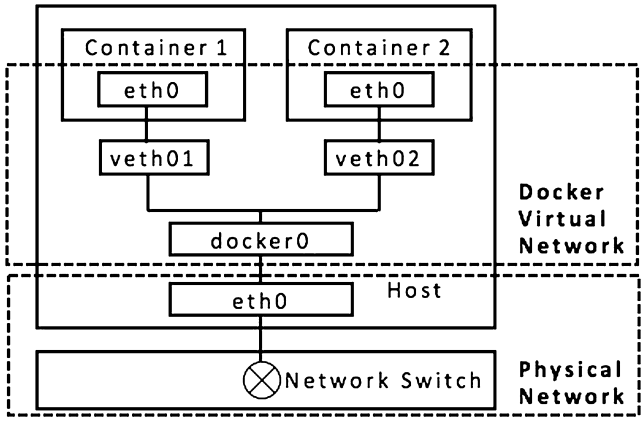

Fig. 1. Docker’s docker0 bridge and virtual network layers.

3. Identify the important factors of the CPU context switch latency

“Context switch is the process of storing and restoring the state (context) of a process so that execution can be resumed from the same point at a later time” [9]. Context switch is applicable to a set of running processes in the same Central Processing Unit (CPU). There is a significant time required by the OS kernel to save execution state of the current running processes to memory, and then resume state of another process.

3.1. Process scheduling algorithms

Process Scheduling Algorithms plays a key role in the Context Switching. Each algorithm developed based on a use case. There is no ideal algorithm developed for all the known use cases. There is no choice given to end users to tune the Scheduling Algorithms. Improving the performance by tuning process scheduling algorithms is not in the scope of this paper.

3.2. Multiple-processor scheduling

Multiple-Processor Scheduling is a complicated concept. Main aim is to keep all the Processors busy to get the best throughput. But, if more and more context switch operations happened, that leads to less throughput. Improving the performance by tuning multiple-processor scheduling algorithms is also not in the scope of this paper.

3.3. Virtualization extensions

How to execute the various Virtual machine processes in the same CPU (Virtual CPU or Core), without compromising security? Is key for Virtualization extensions. To execute the process belongs to a Virtual Machine, hypervisor will add an extra layer on top of the existing Process Control Block (PCB) to identity the Virtual Machine and Process. Coding and decoding of this extra information will add a significant overhead in the process execution. Recent reports published that Round Trip Time (RTT) latency of a virtual machine process request is around 40 nm.

4. Experiment setup

Installed VMware ESXi v5.5 Operating System (OS) on a Dell PowerEdge server with Intel Core 2 Quad Processor Q9650, 128 GB

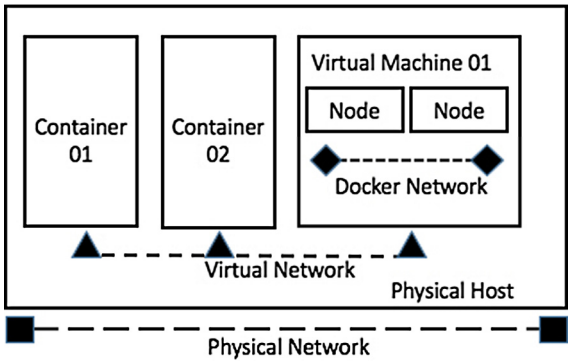

Fig. 2. Big data environment for analysis.

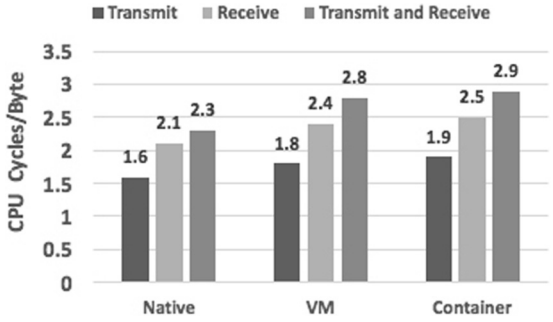

Fig. 3. CPU load and network IO efficiency.

RAM, 4 TB (4×1 TB) 7200 RPM Hard disk drive (HDD) and a Gigabit Ethernet Network Interface (NIC). Provisioned 2 virtual machines each 60 GB RAM and 2 TB Virtual HDD and 2 vNICs. Installed a Hadoop cluster using Ferry Opencore [7] frame work. Based on the use case, the number of Hadoop nodes will be increased/decreased dynamically using Yet Another Markup Language (YAML) configuration files. We have used maximum of 32 VMs or containers for all the experiments. Following are the use cases considered while analyzing the Network IO throughput.

1. Varying the number of Containers (Hadoop cluster nodes)
2. Varying the number of Virtual Machines (Hadoop cluster nodes)
3. Varying the number of Containers (Hadoop cluster nodes) on a VM

4.1. Hadoop cluster benchmarking testbed

Installed Hadoop cluster version 2.7.0 with Hive (version 0.12) using Opencore Ferry framework [7]. Following is the configuration of the Hadoop cluster [8] and Fig. 2 describes Hadoop cluster environment for network IO throughput analysis.

backend: {-storage : { personality: “hadoop”, instances: 2, layers: hive }, connectors: {personality: “hadoop-client”} }
node: { Distribution: Ferry Hadoop 2.7.0, NameNode, JobTracker: node0, Secondary NameNode: node1, Workers: Dockers,
Non-default parameters: fs.datanode.max.xcievers=4096, dfs.replication=2, dfs.block.size=134217728, io.file.buffer.size=131072, mapred.child.java.opts=”-Xmx4096m –Xmn512m” (native), mapred.child.java.opts=”-Xmx4096m –Xmn512m” (virtual), Cluster topology:Native, 1 VM per host: all nodes in the default rack}

4.2. CPU load and network IO efficiency testbed

On top of the above Hadoop cluster environment some of the containers are designated as Servers or Clients to measure the CPU load and network IO efficiency against 3 network scenarios.

1. Client-to-server acts as the transmitter.
2. Server-to-client acts as the receiver.
3. Client-to-server and Server-to-client acts as transmitter and receiver.

5. Analysis of virtual layer overhead

5.1. CPU virtualization layer overhead

We measured CPU virtualization layer overhead using “CPU load and Network IO efficiency” testbed. In such an I/O-bound scenario, we estimated overhead by measuring the amount of CPU cycles required to transmit and receive the data. Fig. 3 shows system-wide CPU utilization, measured using a Linux command “perf stat -a”. The network virtualization in docker i.e. Docker’s bridging and NAT noticeably increases the transmit path length; Containers that do not use NAT have identical performance to the native Linux. In real network-intensive workloads, we expect such CPU overhead will reduce the performance. Results shows that there is a significant overhead added by the virtualization layer. Native application always performs well compared to the application on the virtualized environments. Both Process scheduling and virtual CPU architecture are the important factors for the network IO throughput. If the number of processes/virtual machines/containers increases on the same CPU, the context switch latency will be increased proportionately.

5.2. Network virtualization layer overhead

We measured network bandwidth between the system under test using “CPU load and Network IO efficiency” testbed. An identical machine connected using a direct Giga byte Ethernet. Docker attaches all containers on the host to a bridge and connects the bridge to the network via NAT. We used the netperf request-response benchmark to measure round-trip network latency using similar configurations. In this case the system under test was running the netperf server (netserver) and the other machine ran the netperf client. The client sends a 100-byte request, the server sends a 200-byte response, and the client waits for the response before sending another request. Thus only one transaction is in flight at a time. Fig. 4 describes that the Docker’s Virtualization layer, increases the latency around 5 µs compared to the Native and Virtual environments. Virtual environment adds 30 µs of overhead to each transaction compared to the native network stack, an increase of 80%. TCP and UDP have very similar latency because in both the cases a transaction consists of a single packet in each direction.

5.3. Number of containers on a virtual machine

The same use cases were executed against varying the number of containers within the same VM. Each docker node is given 4 GB of RAM and performed 1000 Map-reduce [2] operations per minute. Fig. 5 describes that as the number of containers increases in the same VM, the RTT of the Network IO packets also increases proportionately. Test results describes that adding more nodes to Hadoop cluster will not increase the performance of the system linearly.

6. Analysis of Hadoop cluster performance

We executed Hadoop benchmarks TestDFSIO-read, TestDFSIO-write, TeraSort and TeraGen on the Hadoop cluster testbed. Following are the use cases considered while analyzing the Network IO performance.

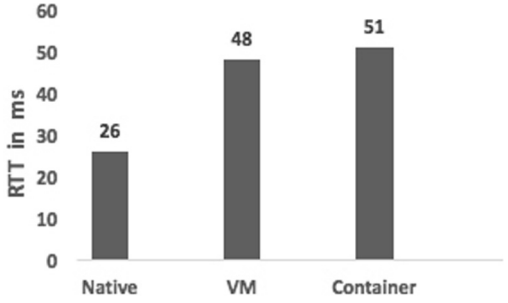

Fig. 4. Describes the transaction latency for native, hypervisor and container environments.

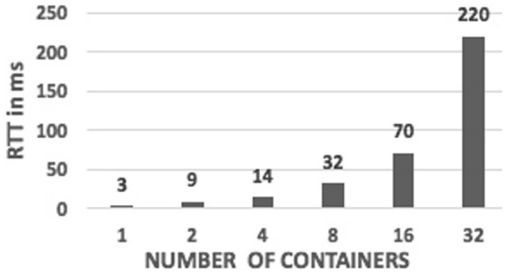

Fig. 5. Describes the increase of RTT with increase of the containers in same virtual machine.

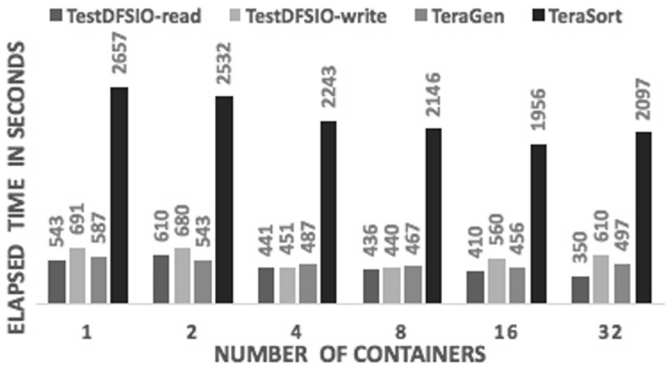

Fig. 6. Hadoop benchmark – elapsed time in seconds (lower is better) against number of containers on a physical machine.

1. Varying the number of Containers (Hadoop cluster nodes)
2. Varying the number of Virtual Machines (Hadoop cluster nodes)
3. Varying the number of Containers (Hadoop cluster nodes) on a VM

6.1. Number of containers (Hadoop cluster nodes) on a physical machine

We executed Hadoop benchmarks TestDFSIO-read, TestDFSIO-write, TeraSort and TeraGen with varying number of containers on a physical machine installed with CentOS 7 x64 OS. For all benchmarks we observed the elapsed time in seconds (lower is better). Fig. 6 describes that all benchmark results getting better with number of containers. But, the elapsed time of TestDFSIO-write and TeraGen operations are not linear with number of containers.

6.2. Number of virtual machines (Hadoop cluster nodes) on a physical machine

We evaluate the performance with the same benchmarks with varying number of VMs on a physical machine installed with VMware ESXi version 5.5 OS and observed the elapsed time in seconds (lower is better). Fig. 7 describes that TestDFSIO-read and TeraSort operations getting better with number of containers. But, TestDFSIO-write and TeraGen operations are blocked somewhere and did not get the expected performance. Test results describes that elapsed time increases along with the number of containers.

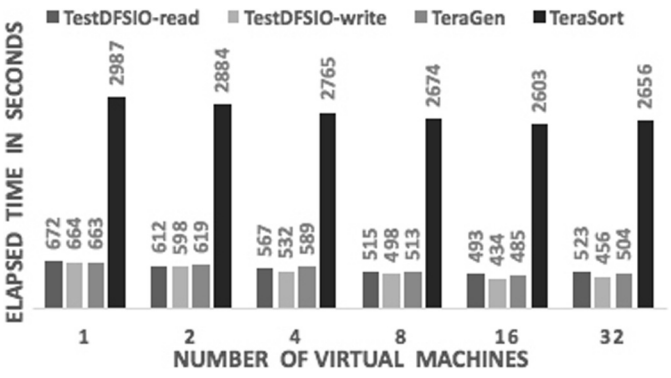

Fig. 7. Hadoop benchmark – elapsed time in seconds (lower is better) against number of virtual machines on a physical machine.

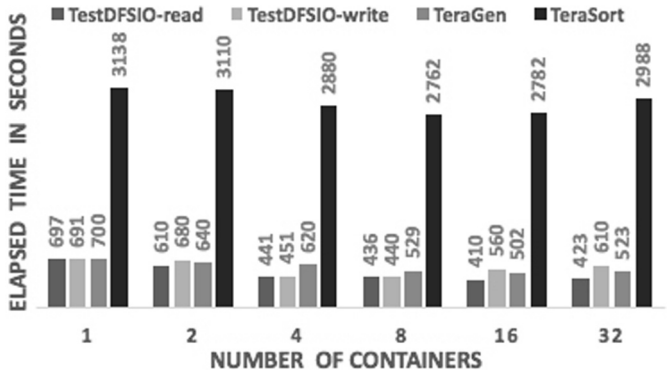

Fig. 8. Hadoop benchmark – elapsed time in seconds (lower is better) against number of containers on a virtual machine.

6.3. Number of containers (Hadoop cluster nodes) on a virtual machine

We evaluate the performance with the same benchmarks with varying the number of the containers on the same VM and observed the elapsed time in seconds (lower is better). Fig. 8 describes that the Hadoop cluster throughput is linear up to 16 nodes and degraded from 20th node onwards. Test results describes that elapsed time increases along with the number of containers. After 20th node, we observed that the context switch latency for each node increased significantly and decreased the network IO throughput, hence the overall system performance was not linear with the number of nodes. If the CPU is busy in context switching, then the IO operations will be blocked and creating a bottlenecks.

Finally, we found that Hadoop environment suffers from low network IO throughput. For better Data center operational margins CPU always kept busy. Busy CPU will increase the context switch latency, which leads to a raise in the network IO RTT of the data packets.

7. Conclusion

We have analyzed the factors that effects performance of the Hadoop cluster. We found that the network IO throughput is inversely proportional to the number of cluster nodes (sharing same CPU) on a VM. If the CPU is busy in the context switching it will add a significant latency to the RTT of the data packets. A raise in

the RTT will reduce the throughput of the entire system. Ideally, adding nodes to the Hadoop cluster will improve the throughput of the system linearly. But, beyond a number (cluster nodes running on containers), adding nodes to Hadoop cluster will decrease the performance due to a raise in the RTT of data packets of the containers.

Future work of this paper would be on studying the docker network bridge and come up with a solution to improve the network IO throughput of the big data environments. We will study the TCP work flow and identify the overhead caused by the virtual network layers. Then we will try to build a solution to reduce the RTT of the data packets passing through the virtual network layers. The solution would be a counter logic in the docker network bridge and transparent to the containers and virtual machines.

References

[1] http://www.sersc.org/journals/IJDTA/vol6_no4/1.pdf.
[2] http://static.googleusercontent.com/media/research.google.com/en/us/archive/mapreduceosdi04.pdf.
[3] http://www.ijarcsse.com/docs/papers/Volume_4/1_January2014/V4I10330.pdf.
[4] http://zoo.cs.yale.edu/classes/cs422/2014fa/readings/papers/shvachko10hdfs.pdf.
[5] https://www.vmware.com/files/pdf/techpaper/VMWHadoopPerformancevSphere5.pdf.
[6] https://wiki.aalto.fi/download/attachments/70789083/linux_bridging_final.pdf.
[7] https://github.com/jhorey/ferry.
[8] http://hortonworks.com/wpcontent/uploads/downloads/2013/06/Hortonworks.ClusterConfigGuide.1.0.pdf.
[9] https://en.m.wikipedia.org/wiki/Process_switching_latency?previous=yes#LATENCY.
[10] http://www.wikiind.com/2011/04/what-is-difference-between-data-and/.