Effects of Latency, Virtualization, Frequency, and Security on Fog Computing

Faiz, Jamal, Salman, Sangeetha, and Zachary
{fabidi89, jamal93,mdsalman,bsangee,zchryb}@vt.edu
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

***Abstract*—Fog computing has gained a lot of attention in the last few years, especially now that devices like the Raspberry Pi are available in the market at an affordable price. However, concerns around the feasibility, scalability, and security of fog computing prevents its complete adoption in applications like the smart grid, smart traffic lights, and software defined networks. In this paper, we built a cluster of Raspberry Pis to measure network latency effects and the impact of virtualization while analyzing the data using Spark. We evaluated the impact of dynamic voltage and frequency scaling on a cluster of Artik boards and analyzed the security vulnerabilities of both clusters. We present our findings in this paper.**

***Keywords*—Fog computing, distributed computing, Dockers, DVFS, Internet of things, network security.**

I. Introduction

With the development of smart wearables, connected vehicles and wireless sensor networks, the Internet of Things (IoT) paradigm is considered the future of the Internet. Due to their increasing prevalence, these ubiquitously connected smart devices are now becoming a main focus of computing. The IDC 2017 report predicts that the amount of data in the world will grow ten-fold by 2020 and the IoT information will contribute 10% of the total data [1]. Although the cloud computing paradigm has been the de-facto model for processing data from sensor devices, but the cloud model falls short of the requirements for IoT applications. Examples of IoT applications include geo-distribution, location awareness, low latency, and mobility support. This was the primary motivation behind the development of the Fog Computing model [2].

Fog Computing is a distributed platform that provides compute, storage, and networking services between end devices and traditional Cloud Computing data centers usually located at the edge of the network [3]. The different elements that constitute the Fog Computing architecture are shown in Figure 1.

This architecture extends the Cloud Computing paradigm to the edge of the network. While Fog and Cloud use the same resources and share many of the same mechanisms and attributes, the extension is a non-trivial one in that there exist some fundamental differences [4] that we discuss below:

- Data and applications are processed at the network edge and a response consumes lesser time.
- Lesser bandwidth requirement as data is only stored at each of the Fog nodes.

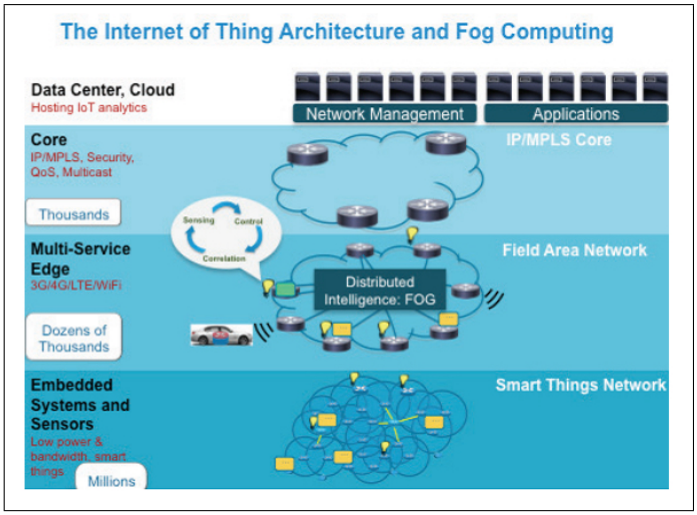

Fig. 1: Fog distributed infrastructure for IoT [4]

- Response time and scalability issues arise in cloud computing mainly due to the geo-distribution of servers which does not exist in the fog model.
- The Fog layer processing consists of distributed nodes that can process jobs as a cluster.

The architectural difference between the Fog model and the cloud computing model presents multiple challenges and there is a need to qualitatively analyze the various attributes of the Fog layer. These include a) compute capability, b) latency, c) virtualization, d) security, and e) energy efficiency.

In this project, we analyze the effects of Latency, Virtualization, Frequency, and Security on the runtime performance and energy consumption of a cluster of Fog nodes.

II. Elements of Fog Computing

A. Compute Capability

The compute capability is diverse and can vary from resource poor devices such as set-top boxes, access points, routers, switches, base stations and end devices, to resource-rich machines as Cloudlet[5] or Iox[6]. As part of this project, we evaluate two systems, namely Raspberry Pi [7] and Samsung Artik board 10 [8]. Refer to Section III-A.

B. Latency

One of the biggest advantages of the Fog model is the reduced latency between devices. The data can be processed at the Fog nodes and in many cases, the Fog nodes are interconnected to each other. In general, there are three types of latencies-

1) Between sensor layer and the Fog layer.
2) Between multiple nodes at the Fog layer especially in multi-tenancy environments.
3) Between the Fog layer and the datacenter.

From the perspective of Fog-based applications, which are not frequently communicating to the cloud, the effect of latency between the Fog layer and the datacenter is minimal. As a result, we evaluate only 1 and 2 in this project.

C. Virtualization

In a typical Fog deployment, the fog nodes might be servicing multiple applications at once, each with a different QoS requirement. For example, we could have a sensor device monitoring the milk left in the refrigerator and sends a notification to a device like Amazon Echo to place an order and at the same time we can have an arrival sensor which switches room lights. By exploiting virtualization techniques at the Fog layer, each of these applications could be serviced in a different container. It is therefore important to evaluate the effect of virtualization on the runtime performance at the Fog layer.

D. Security

Security is probably the biggest hurdle for the widespread adoption of the Fog model. As we have more data being generated due to IoT applications and more devices being installed, the issue of security grows. Data collected can be abused and the devices could be taken over physcially or over the network. For example, in October 2016 internet outages happened accross the United States because of a botnot consisting of IoT devices [9].

E. Energy Efficiency

Although most IoT devices are optimized for power, as more sensors are added to the network, there is always room for reducing the energy costs. [10] summarizes energy conserving mechanisms for IoT based solutions from a wireless networking aspect. We adopt an approach similar to [11] and use Dynamic Voltage Frequency Scaling (DVFS) to vary the CPU frequency and evaluate the effect of this on the runtime performance.

1) Dynamic Voltage Frequency Scaling: Dynamic voltage and frequency scaling (DVFS) is the adjustment of power and speed settings on a computing device's various processors, controller chips and peripheral devices to optimize resource allotment for tasks and maximize power saving when those resources are not needed. DVFS allows host CPUs to dynamically change power states when resource demands are low to reduce a host's energy consumption.

The motivation behind exploring DVFS at the Fog layer is important in applications where the Fog nodes are in outdoor conditions, and need to be powered by a battery. Even in indoor conditions as the number of devices increase, vendors will be forced to optimize the device to keep energy costs down.

III. Design

A. Experimental setup

1) A cluster of Raspberry Pis: We used 10 Raspberry PI 3 model B [7] to create a Spark [12] cluster. One of the 10 Raspberry Pis was assigned as the master node while the other 9 were assigned as worker nodes. Each Raspberry Pi3 has 4 single threaded cores with 1 gigabyte of memory, and the processor runs at a frequency of 1.2GHz/sec.

Raspeberry Pis do not support setting CPU frequencies other than 600MHz and 1.2GHz. Hence, we decided not to run DVFS [13] tests on the Raspberry Pis.

To evaluate the effectd of virtualization on the Raspberry Pi cluster, we installed Docker [14] version 1.12.3 on each node, and used Docker Swarm [14] to manage the resources.

2) A cluster of Artik boards: We used 6 Artik boards [8] to create a Spark cluster. One of the 6 Artik boards was assigned as the master node while the other 5 were assigned as worker nodes. Each Artik board has 8 single threaded cores with 2 gigabyte of memory, and the processor runs at a frequency of 1.3GHz/sec.

Artik boards have a lower power DVFS [13] control hardware block that allows frequency variation in continuous steps between 500MHz to 1.3GHz. Hence, we could run experiments using the Artik boards to analyze the average energy consumption by varying the CPU frequency.

3) Algorithms run and the data used: We ran two algorithms on our Spark cluster - logistic regression and sorting.

The data that we used for running the logistic regression algorithm was airline sensor data that is publicly available on the Internet. The type of this data is CSV and it is 100 megabytes in size. There are 1 million lines and 29 columns in this data.

The data that we used for running the sorting algorithm was created using 10 million random numbers. The size of this data file is 100 megabytes.

4) Watts Up Pro: We used Watts Up Pro [15] model 99333 to measure the energy consumption on the Artik boards.

B. Spark

We installed Spark [16] version 2.0.1 on the nodes, and ran the algorithms in Spark standalone mode. We varied the amount of memory between 512, 812, and 1024 megabytes (we chose these intervals because 612, 712, and 912 megabytes of memory did not show any noticeable difference in performance) and the number of executor cores from 9, 18, 27, and 36.

We ran the algorithms on the Spark cluster with a) no network latency on the worker nodes; b) uniform network latency on the worker nodes; and c) variable network latency on the worker nodes. See Section IV for more details.

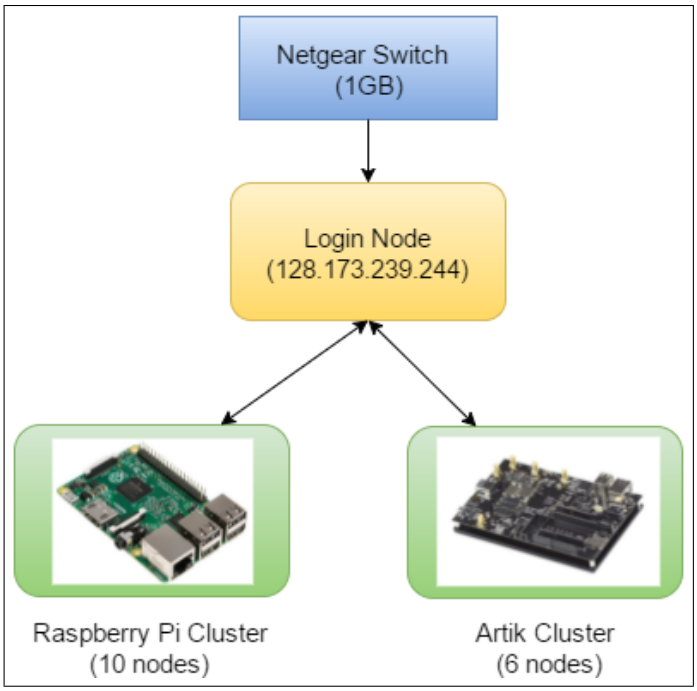

Fig. 2: Block diagram of the cluster setup we had in the lab

On the Docker containers running Spark, we only ran jobs with no network latency introduced on the worker nodes.

We installed a Ganglia [17] server on the master node to measure CPU, memory, and the time to complete the jobs.

C. Security assessment tools

For running security scans, we used Nessus [18] home edition and Open Vulnerability Assessment System (OpenVAS) [19] version 8. WireShark [20] was used to sniff the network traffic over the switch with a monitor port allowing packet inspection. A recent National Institute of Standards and Technology (NIST) special publication on IoT security standards [21] and a survey of security issues in Fog Computing [22] was used to create a security checklist.

Figure 2 shows the block diagram of the cluster of Raspberry Pis and Artik boards that we setup in the DSSL lab in Knowledge Works II. Figure 3 shows the actual setup we had in the lab. We used a 24 port 1 gigabit switch to connect all the nodes together.

D. Design Trade-offs

The very first trade off that needs to considered is the choice of hardware i.e. why Raspberry Pi 3 and Artik 10 boards were chosen. The reason for choosing Raspberry is a combination of low cost, extensive documentation, and an active community. Hence, not only is it easy to use but also has good performance per dollar. However even with its pros, the Pi is unable to run DVFS and had low computational capabilities which lead to the addition of the Artik boards which are more powerful, have more memory, and support DVFS.

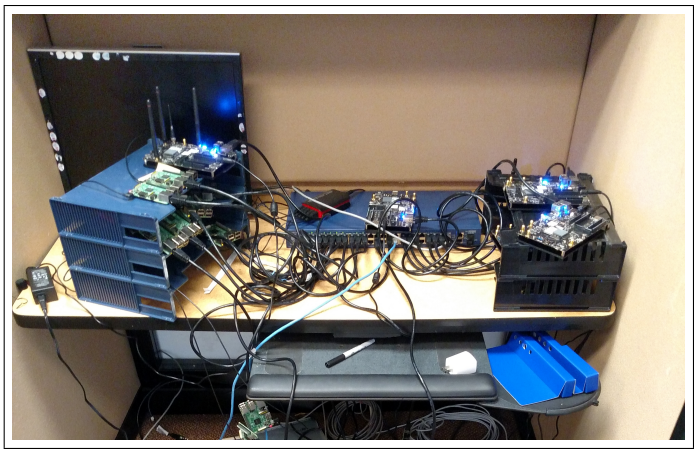

Fig. 3: Cluster of Raspberry Pi and Artik boards setup in the lab

The next question is concerned with why homogeneous clusters were implemented and not heterogeneous ones. The reasoning is quite straightforward that it allows for ease of implementation and that this is the common scenario that would be in actual deployments.

E. Challenges faced

One of the biggest challenge we faced when we started this project was the unavailability of the hardware to run our experiments. We did have access to a cluster of Raspberry Pi 1, but because there is only 512 megabytes of memory installed on the Raspberry Pi 1, we could not run our algorithms using Spark (we needed at least 1 gigabyte of memory to efficiently run our code using Spark). After setting up Spark on the Raspberry Pi3 cluster, it was a challenge to manage the memory on the master node given that we were running a history server, a ganglia server, the Spark master server in addition to some other processes needed to run Spark. Because of all the processes running on the master node, it often broke the cluster leaving the master node inaccessible. We had to physically setup the cluster and restart the jobs many times because of this problem. Virtualizing the Raspberry Pis using Docker containers and later setting up a Docker Swarm cluster also posed many problems. There is not enough documentation available on the Spark and the Docker website that explains how to setup Docker containers on Raspberry Pis. Also, the Docker images available on the Internet do not work for the ARM processors that is installed in the Raspberry Pis. Thus, we had to build our own image according to our needs.

IV. Evaluation

A. Latency

Fog computing nodes are geographically distributed and work in tandem. There are multiple sources of network delay besides geographic distance. The impact of this delay may have some major implications in the Spark framework as the traditional framework does not take such delays into consideration. We have thus carried out the below evaluations in order to characterize the impact of latency on I/O heavy and compute intensive jobs.

We came up with a multiple linear regression model in order to analyze the trend in our observations. This model is representative of our observations and describes the correlation between our features and the runtime of the cluster jobs. We describe below the three cases that we have considered in our study along with the regression model and its Mean Squared Error (MSE). Please note that the following are the variables used in the equation:

- x: Number of executors
- y: Memory per executor
- z: Average CPU utilization

1) No latency: This is the ideal case where there is no latency. This could occur if all the worker nodes are co-located and on the same network.

Below are significant observations from the data collected:

I/O intensive job (Sorting) :

- As the number of executors are decrease from 27 to 9, the time to completion decreases as expected (Appendix Figure 14).
- With 36 executors, the performance is lower than that of 27 executors. This is expected as the dataset size is small and Spark runs jobs in an embarrassingly parallel way. 36 executors add more overhead instead of improving the performance (Appendix Figure 14).
- As the memory for each of the executors is increased from 512 megabytes to 812 megabytes, the running time decreases as expected. This is because as Spark's Resilient Distributed Datasets (RDDs) grow, more complex operations can be performed in memory. (Appendix figures 14 and 16).
- At 1024 megabytes of memory per executor, RDDs spill to disk and we see a slight increase in the run time.
- From this data, we have a multiple linear regression model with a Mean Squared Error (MSE) of 0.017:

$$6.71 + 2215 * 10^{-7}x - 1643 * 10^{-7}y - 0.07545z$$

Compute intensive job (Logistic Regression)

- In general, the time to completion decreases as the executor memory is increased.
- In most cases, the time to completion decreases as the number of executors are increased.
- Multiple Linear Regression Model (MSE of 0.60):

$$-5.13 + 0.0296x - 5487 * 10^{-7}y - 0.5423z$$

2) Uniform latency: In this case, we have emulated the scenario where there would be a uniform latency between the master node and all of the worker nodes. In order to achieve this, we introduced a delay of 100ms on the ingress bandwidth on all the worker nodes. This case corresponds to a scenario wherein the master node and the worker nodes are in disparate networks. Below are some of the observations:

I/O intensive job (Sorting) :

- Latency has no impact on I/O intensive jobs, since they have infrequent interactions with the master. The tasks run the jobs to completion before having to communicate results to the master.
- Multiple Linear Regression Model:

$$6.47 + 28823 * 10^{-6}x - 6148 * 10^{-7}y - 0.06236z$$

- With a Mean Squared Error of 0.014

Compute intensive job (Logistic Regression)

- As expected, the time to completion increases linearly with added latency.
- Multiple Linear Regression Model (MSE of 0.37:

$$-9.64 + 0.024x - 0.0021y - 0.071z$$

3) Variable latency: We introduced variable latency in the network by creating delays of 50ms, 100ms, and 150ms for two, three and four workers respectively. This is where nodes are in different locations and networks:

I/O intensive job (Sorting):

- As explained earlier, latency has no impact on I/O intensive jobs.
- Multiple Linear Regression Model (MSE of 0.029):

$$5.74 + 3148 * 10^{-7}x - 4089 * 10^{-7}y - 0.048z$$

Compute intensive job (Logistic Regression)

- As expected, latency increases the time to completion.
- Multiple Linear Regression Model (MSE of 0.37):

$$-9.79 + 0.015x - 0.0019y - 0.051z$$

B. Virtualization

1) No Latency: We used Docker containers to virtualize the Raspberry Pi nodes, and ran logistic regression jobs with no network latency between the worker nodes. Below are some of the observations:

1) Because of virtualization, the maximum available memory in the Docker container was only 862 megabytes instead of 1 gigabyte.
2) The time to completion of the job increased by 2 minutes when compared with the no virtualization case. See Figures 5 and 23 in the appendix for more details.
3) Compared to no virtualization, we saw an increased CPU percentage on the Docker containers for the same job (almost 3%). See Figures 6 and 24 for more details.
4) Compared to no virtualization, we saw a little increase in memory utilization in the Docker containers for the same job (almost 200 megabytes). See Figures 7 and 25 for more details.
5) Multiple Linear Regression Model (MSE of 0.44):

$$-4.37 - 0.021x - 0.0019y - 0.336z$$

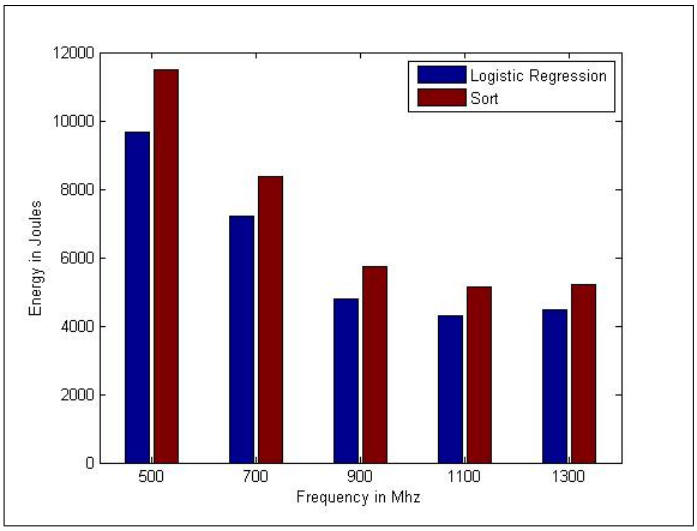

Fig. 4: Average Energy Consumption vs Frequency in MHz

C. DVFS

Our initial hypothesis was that the lowest energy consumption would be at the highest frequency.

As shown in 4, our experiments showed a trend of increasing power consumption and at the same time decreasing execution time as the frequency goes on increasing. The total energy consumed by a certain job is simply the area under the graph which can be computed using the integral. We ran the experiments for both compute intensive jobs and I/O intensive jobs and found the best performance at 1.1GHz.

These results point to the fact that when I/O operations are involved, running at the highest frequency isn't always the best idea as it does not ensure best performance in terms of total energy consumed. The reason being that, in I/O tasks the CPU is not always involved because the Memory Management Unit (MMU) handles them and during these cycles if the CPU is running at a high frequency it consumes more energy. Hence there should be a sweet spot where the energy consumption of the CPU balances during I/O is minimum without impacting the performance of I/O.

D. Security

Risk analysis is dependent on identifying quantitative and qualitative risks and analyzing them to justify whether a system is in a low, medium, to high state of risk. Quantitative information was obtained using the network scanners and qualitative information obtained by documentation for the Pi and Artik boards.

The quantitative analysis returned results showing that there were security issues in both of the clusters. The Raspberry Pi cluster had known vulnerabilities with OpenSSH and DNS, caused by delayed updates for the Raspbian image. The Artik cluster was flagged with a critical level issue for using an outdated operating system. The scanners found out the management(Ganglia) and history web servers(Spark) and crawled them, finding that there was no HTTPS support or authentication interface. These issues all have known solutions. The Pi cluster can be updated if the community is given some time. The Artik boards can have the latest Fedora patches applied manualy. The servers can be encapsulated in an HTTPS tunnel with authentication proxy. From this information, it can be drawn that the default security of these devices is not optimal. The evidence suggests that there is plenty of room for the security of these devices to improve in particular.

Next is to consider the qualitative analysis by comparing the information from the NIST checklist [21] and the Fog security survey [22] with the information from the specification of the Pi [7] and Artik [8] clusters. A Fog cluster needs to address issues with authentication, data storage, and verifiable computation. Authentication means that the cluster needs to be able to assure data exchanges are confidential. This issue is addressed in the Artik cluster with the Secure Element (SE).It allows fast encryption and secure boot for the Artik device. The Pi cluster has to suffice with the state of the art encrpytion changes in software. Secure data storage means that information taken by a network or physical attack is not usable by the attacker. A solution for this is to encrypt data when it is not in full use. Verifiable computation or computational integrity is the abulity to determine whether the data coming from a node is malicious or trusted data. Our current setup does not guarantee verifiable computation. The above mentioned threats have the potential for causing medium to high levels of harm to the operators of the Fog cluster and there are little to no implemented defenses.

The Fog cluster setup has a number of addressable high risk vulnerabilities.The absence of a well-defined Fog Computing standard coupled with the traditional approach towards security contributed to the inablity to maintain a set of secure standards. From a qualatative standpoint, there are risks to deploying these boards as part of a large Fog cluster because there are fundamental problems that have few to no good answers. These threats would qualify as high, since an attack vector that is known to be indefensible would be attractive to those who wish to exploit the Fog cluster. Overall there is a security risk to Fog computing that could be addressed with further research.

V. Related Work

There have been efforts to understand the effects of the factors being considered in this study on the Fog Network, Cloud, and IoT deployments but in contrast to this previous work, our study focuses on multiple factors together to provide a wholesome picture and deeper understanding.

A comprehensive survey of current research on Fog computing has been done in [23] by S. Yi et al., which gives a birds eye view of the pervasive problems and the solutions being developed. Some systems proposed to leverage virtualization to achieve higher performance. For example, ThinkAir [24] parallelizes method computation using multiple virtual machine (VM) images to introduces elasticity and scalability to the cloud and hence enhances the power of mobile cloud computing. Another system named COMET [25] uses VM synchronization primitives and distributed shared memory to augment smart devices such as smartphones and tablets with machines that are already available on the network.

Few papers have focused on security and/or privacy issues in Fog computing. This combined with the fact that privacy has become a serious concern because of the increasing number of IoT deployments and will become even more so with increasing popularity of Fog warrants more research. Privacy-preserving techniques have been proposed in many scenarios including wireless network [26], smart grid [27], and cloud [28]. These algorithms can be run between the Fog and the Cloud to prevent privacy leakages.

VI. Conclusion

This project has been an attempt to formally define the Fog Layer and conduct a qualitative and quantitative evaluation of the elements. The devices we have used in no way encompass the varied nature of the IoT devices. However, we do believe that since Raspberry Pi is one of the most commonly used devices, it will affect the design of later Fog Nodes. We selected Artik because of the hardware based DVFS block which allows us greater granularity over the frequency. There is a correlation between the run time performance (CPU and execution time) with respect to the number of executors and the memory allocated. In terms of virtualization, the run time performance decreases, but we believe the isolation offered by virtualization will be an important factor in terms of security and compartmentilization of the Fog Layer. DVFS results shows a saddle point at 1.1GHz for both the compute intensive and I/O intensive jobs, leading us to conclude that there exists a saddle point which is not necessarily at the highest possible frequency for the jobs. This lends credence to finding out the most optimum frequency dynamically. From the security analysis, we are able to conclude that Fog nodes have known vulnerabilities with un-applied solutions and that there are threat vectors for the Fog architecture that do not have a developed solution. This means that Fog node security needs more research into cost effective solutions and an active stance towards fixing known issues.

VII. Acknowledgement

We would like to thank Dr. Ali R. Butt for his continued guidance on the project. Also, we would like to appreciate our mentor Arnab for providing technical inputs and help in procuring the hardware.

References

[1] G. Press, "Top 10 tech predictions for 2017 from idc." Online, Nov. 2016.

[2] Cisco, "Fog computing and the internet of things: Extend the cloud to where the things are," 2015.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, (New York, NY, USA), pp. 13–16, ACM, 2012.

[4] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 27–32, Oct. 2014.

[5] R.-P. M. Hardware, "The case for vm-based cloudlets in mobile computing,"

[6] "Cisco devnet: Iox - technical-overview." (Accessed on 12/13/2016).

[7] R. Pi, "What is raspberry pi," 2016. [Online; accessed 7-December-2016].

[8] Samsung, "Samsung - artik," 2016. [Online; accessed 7-December-2016].

[9] B. Krebs, "Hacked cameras, dvrs powered todays massive internet outage." https://krebsonsecurity.com/2016/10/hacked-cameras-dvrs-powered-todays-massive-internet-outage/, 2016.

[10] Z. Abbas and W. Yoon, "A survey on energy conserving mechanisms for the internet of things: Wireless networking aspects," *Sensors*, vol. 15, no. 10, pp. 24818–24847, 2015.

[11] P. Abrahamsson, S. Helmer, N. Phaphoom, L. Nicolodi, N. Preda, L. Miori, M. Angriman, J. Rikkilä, X. Wang, K. Hamily, *et al.*, "Affordable and energy-efficient cloud computing clusters: the bolzano raspberry pi cloud cluster experiment," in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, vol. 2, pp. 170–175, IEEE, 2013.

[12] Wikipedia, "Apache spark — wikipedia, the free encyclopedia," 2016. [Online; accessed 8-December-2016].

[13] Wikipedia, "Dynamic voltage scaling — wikipedia, the free encyclopedia," 2016. [Online; accessed 20-October-2016].

[14] Wikipedia, "Docker (software) — wikipedia, the free encyclopedia," 2016. [Online; accessed 7-December-2016].

[15] P. M. Store, "Watts up pro portable power meter," 2016. [Online; accessed 8-December-2016].

[16] Wikipedia, "Apache spark — wikipedia, the free encyclopedia," 2016. [Online; accessed 8-December-2016].

[17] Wikipedia, "Ganglia (software) — wikipedia, the free encyclopedia," 2016. [Online; accessed 6-October-2016].

[18] Tenable, "Nessus vulnerability scanner," 2016. [Online; accessed 11-December-2016].

[19] OpenVAS, "Open vulnerability assesment system," 2016. [Online; accessed 11-December-2016].

[20] Wireshark, "About wireshark," 2016. [Online; accessed 11-December-2016].

[21] J. O. Ron Ross, Michael McEvilley, "Nist special publication 800-160, systems security engineering," 2016.

[22] S. Yi, Z. Qin, and Q. Li, "Security and privacy issues of fog computing: A survey," *WASA*, 2015.

[23] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*, Mobidata '15, (New York, NY, USA), pp. 37–42, ACM, 2015.

[24] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *2012 Proceedings IEEE INFOCOM*, pp. 945–953, March 2012.

[25] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen, "Comet: Code offload by migrating execution transparently," in *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, (Berkeley, CA, USA), pp. 93–106, USENIX Association, 2012.

[26] Z. Qin, S. Yi, Q. Li, and D. Zamkov, "Preserving secondary users' privacy in cognitive radio networks," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pp. 772–780, April 2014.

[27] A. Rial and G. Danezis, "Privacy-preserving smart metering," in *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society*, WPES '11, (New York, NY, USA), pp. 49–60, ACM, 2011.

[28] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for data storage security in cloud computing," in *2010 Proceedings IEEE INFOCOM*, pp. 1–9, March 2010.

VIII. Appendix

Because of space constraints, we are including the graphs that we got while running our study in this section.

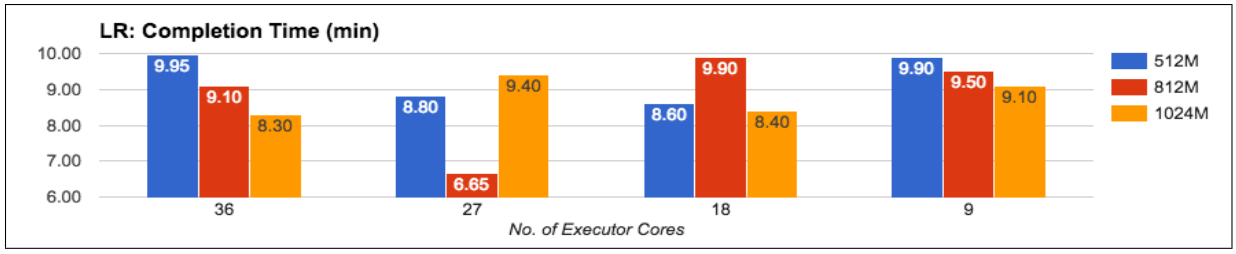

Fig. 5: Average time to complete the logistic regression job running with no network latency

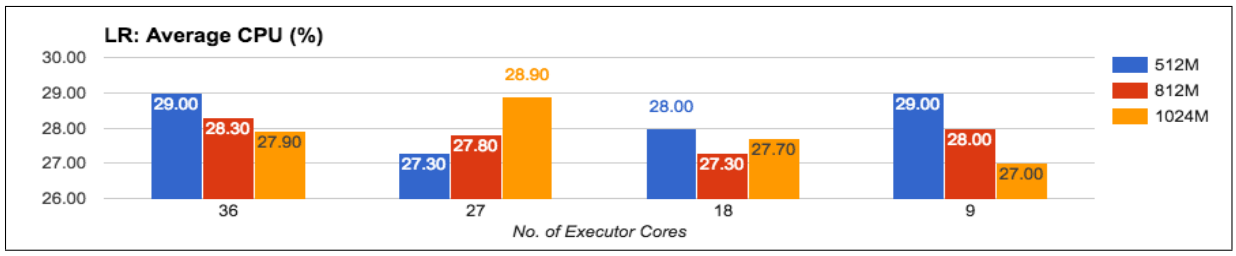

Fig. 6: Average percentage of CPU consumed while running the logistic regression job with no network latency

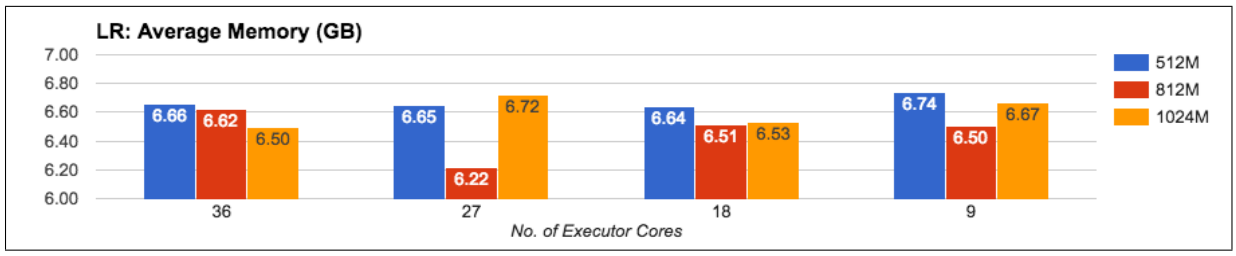

Fig. 7: Average memory consumed while running the logistic regression job with no network latency

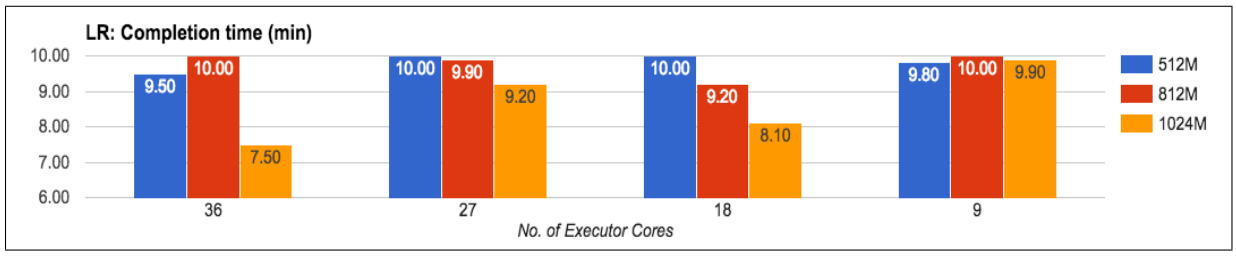

Fig. 8: Average time to complete the logistic regression job running with a uniform network latency of 100ms

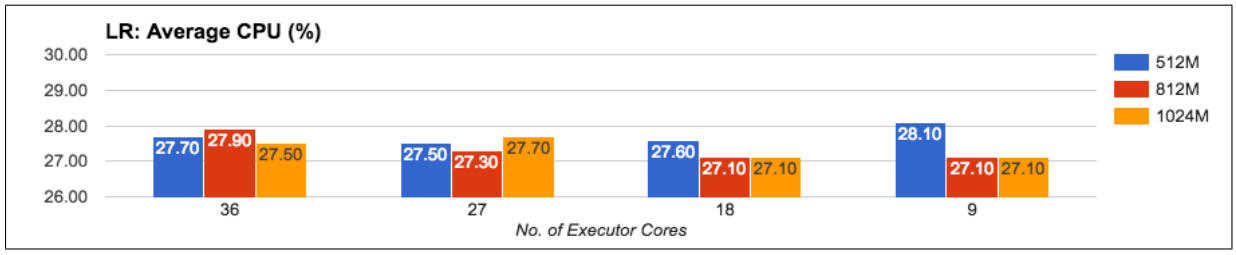

Fig. 9: Average percentage of CPU consumed while running the logistic regression job with a uniform network latency of 100ms

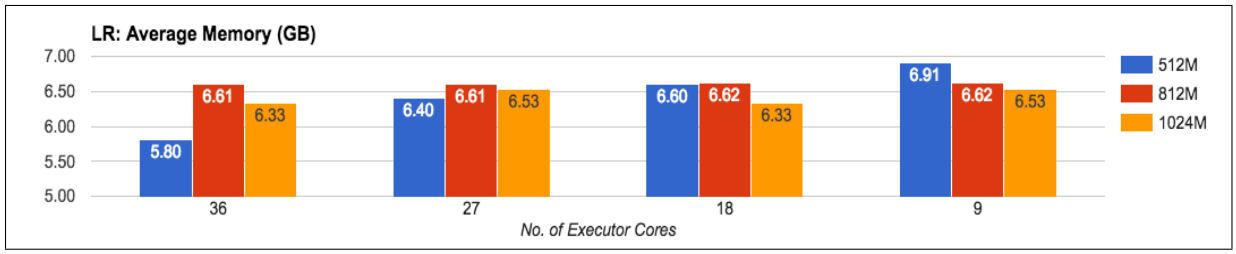

Fig. 10: Average memory consumed while running the logistic regression job with a uniform network latency of 100ms

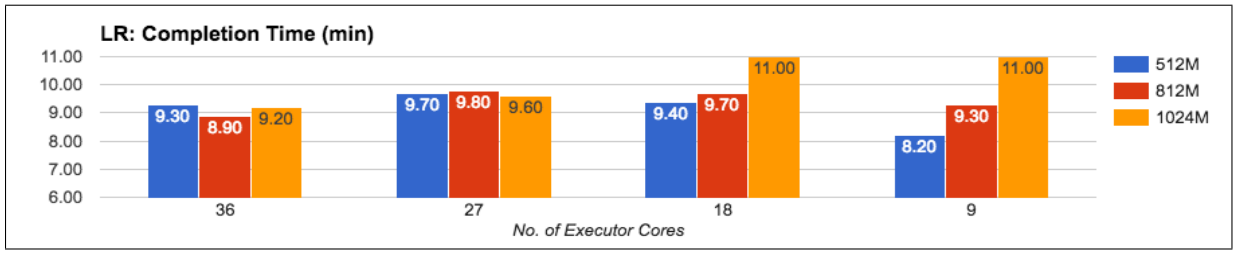

Fig. 11: Average time to complete the logistic regression job running with a variable network latency

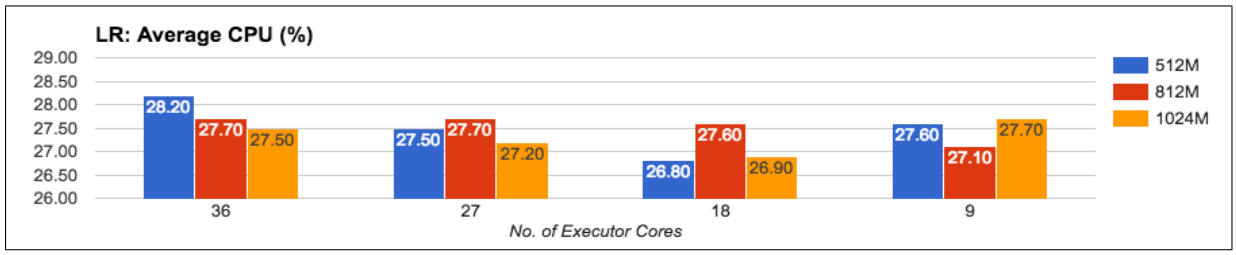

Fig. 12: Average percentage of CPU consumed while running the logistic regression job with a variable network latency

Fig. 13: Average memory consumed while running the logistic regression job with a variable network latency

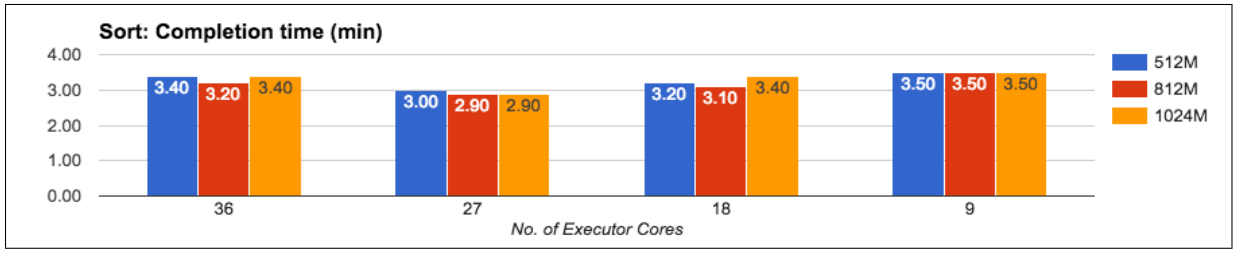

Fig. 14: Average time to complete the sorting job running with no network latency

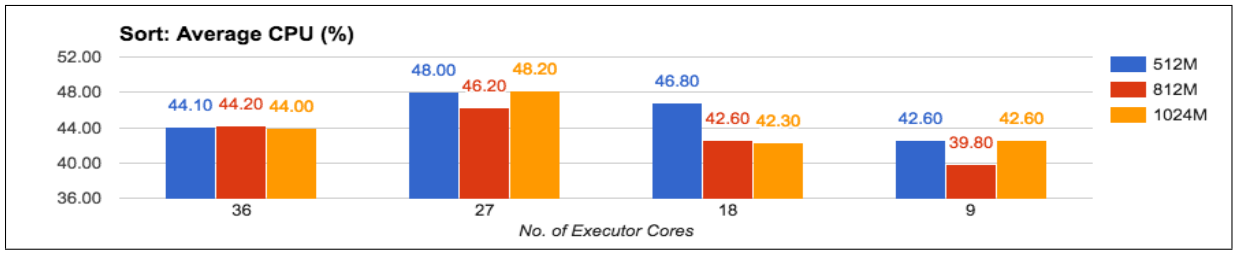

Fig. 15: Average percentage of CPU consumed while running the sorting job with no network latency

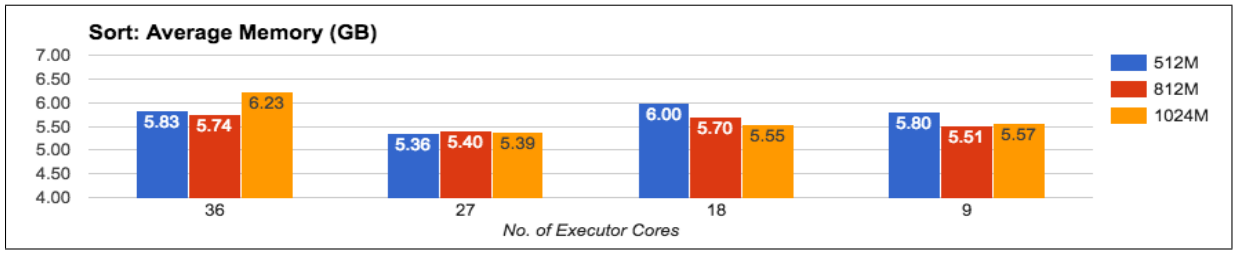

Fig. 16: Average memory consumed while running the sorting job with no network latency

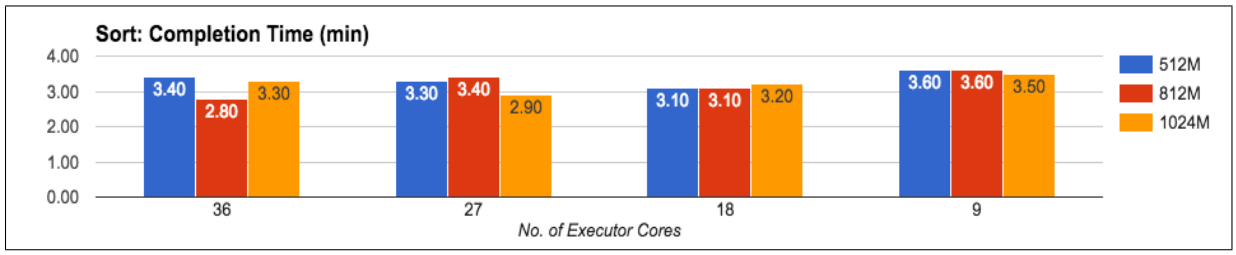

Fig. 17: Average time to complete the sorting job running with a uniform network latency of 100ms

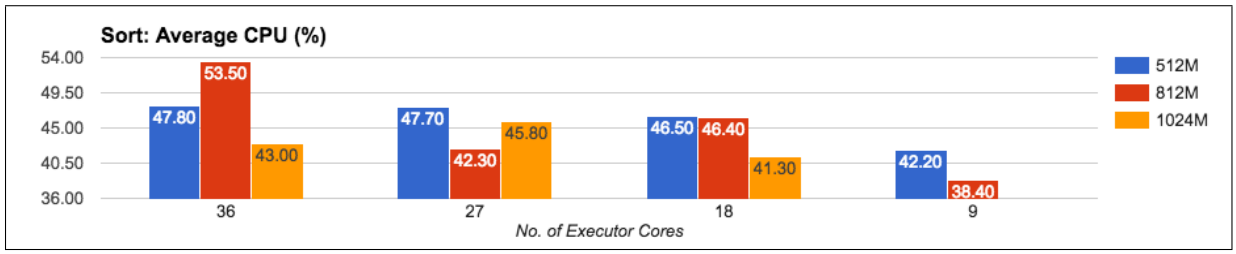

Fig. 18: Average percentage of CPU consumed while running the sorting job with a uniform network latency of 100ms

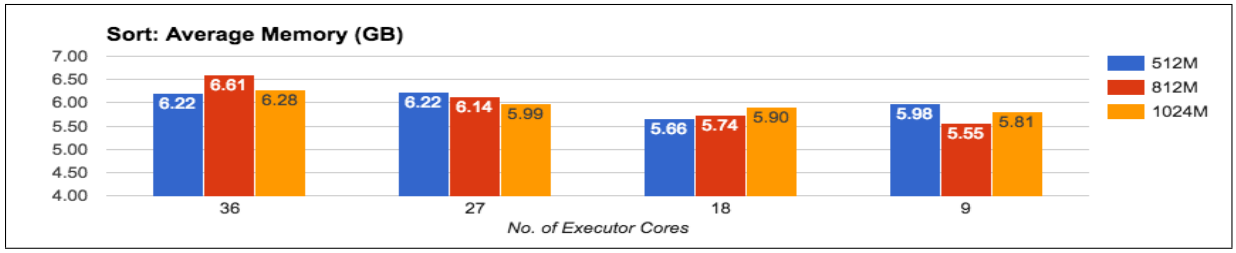

Fig. 19: Average memory consumed while running the sorting job with a uniform network latency of 100ms

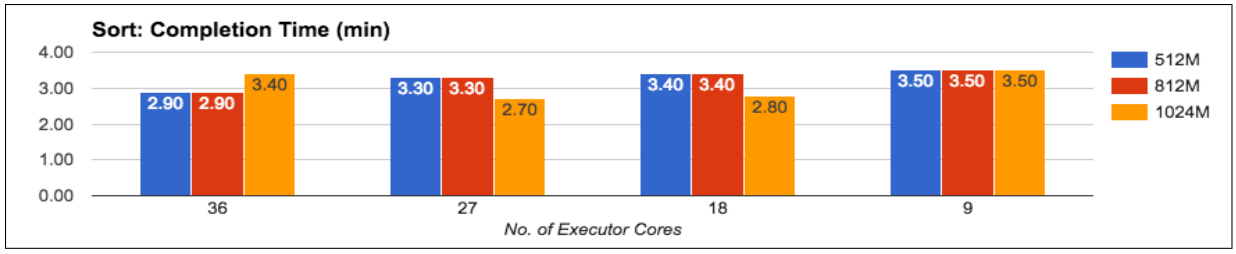

Fig. 20: Average time to complete the sorting job running with a variable network latency

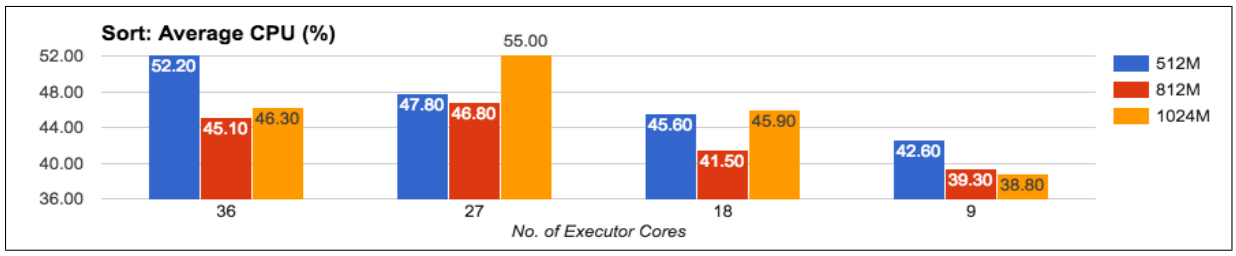

Fig. 21: Average percentage of CPU consumed while running the sorting job with a variable network latency

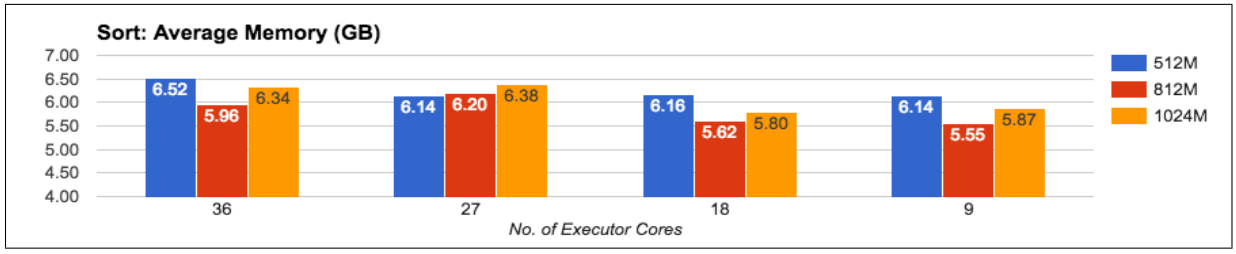

Fig. 22: Average memory consumed while running the sorting job with a variable network latency

Fig. 23: Average time to complete the logistic regression job running with no network latency in a Docker container

Fig. 24: Average percentage of CPU consumed while running the logistic regression job running with no network latency in a Docker container

Fig. 25: Average memory consumed while running the logistic regression job running with no network latency in a Docker container