Power vs Throughput vs Scalability: Apache Spark/Hadoop on Raspberry Pi Cluster & ThunderX Server

The following report covers the feasibility analysis of deploying a distributed software on lightweight compute nodes, the raspberry pis, and comparing their target performance with a more powerful server, the ThunderX. The target measurements of this experiment were cluster throughput, scalability, and the data to power ratio on MapReduce and Spark benchmarks. The Word Count application was chosen for benchmarking MapReduce and KMeans clustering was chosen for benchmarking Spark. These benchmark results and overall cost estimates suggest clusters constructed purely with raspberry pis is a viable option when deployed in small scales.

Anna Marusarz Ji Tin Justin Li

Table of Contents

1. Background

Distributed frameworks provide capabilities to process big data that could potentially benefit many public sectors and significantly impact ongoing trends. Data mining and machine learning techniques often rely on the abundance of available data to generate models with good predictive power, and distributed file systems serve as the backbone of these applications.
On the economic side, initial costs and maintenance costs are important factors for companies to consider when investing in a new system to carry out these tasks. There are a few natural solutions to this issue: using public cloud solutions provided by vendors such as AWS and Microsoft Azure, or to invest and purchase private hardware, or a combination of both. In regard to private hardware, server machines of hierarchical capabilities are available for consumers to choose from. Here, we explore the feasibility of an alternative solution to standard high-cost server machines: the use of light-weight compute nodes running Apache Hadoop/Spark.

Apache Hadoop and Spark are open-source, distributed software frameworks that facilitate multiple compute nodes to work on vast amounts of data. Hadoop comprises three main components: MapReduce, Yarn, and HDFS. The MapReduce program is Hadoop's big data processing technique. Each MapReduce program is composed of a map and a reduce method. The map function will take the data and assign meaningful key-value pairs, writing any output to temporary storage, verifying there is no redundant data. The shuffle stage will place all the data with one key generated by the map stage on one worker. Finally, each worker will work in parallel to reduce the data on their node. For example, in the application word count, the input data is first split in chunks on each node. For the data on each node, it is mapped to a key, in this case, the number of occurrences of the word. The shuffle stage will combine all the same keys together on one node, in this example, the word name. Finally, the reduce stage will then reduce the data on their node to the word and its number of occurrences. Though word count is a typical, simple application of Map Reduce, there are many real-world applications with extensive amounts of data that can be efficiently applied to MapReduce. The second component is Yarn, which stands for Yet Another Resource Negotiator. Yarn was introduced as a way to reduce the bottleneck in previous versions of Hadoop by separating resource management from the processing side, offering further scalability and cluster utilization. When launching Yarn, a resource manager is launched as the master daemon that will coordinate tasks for its worker nodes. On each worker node, a node manager is launched to monitor resource usage, manage and track application containers, and send "heartbeats" to the resource manager. Finally, Hadoop has its own distributed file system, HDFS standing for Hadoop distributed file system. It uses a master/slave architecture where the master or NameNode manages the file system, while the slaves or DataNodes manage the space on the nodes they run on. HDFS is highly fault-tolerant, robust, and designed to be deployed on commodity hardware. HDFS has a simple coherency model, meaning the files are immutable and if changes must be made to a file, the modified file must be uploaded back to HDFS.

Apache Spark is a cluster computing framework that was developed to offset the limitations of the MapReduce paradigm when running machine learning, iterative algorithms. It is based on immutable, fault-tolerant, and general-purpose Resilient Distributed datasets (RDDs). Spark provides an easy to use API, offering multiple options to deploy, and supports many programming languages. By exploiting in-memory computations, Spark can achieve a 100X speedup over Hadoop. Spark has libraries on top of its processing core for various applications such as SQL, machine learning, graph computation, and stream processing which can all be used together. Similar to MapReduce, Spark requires a cluster manager, such as Yarn and a distributed file system, such as HDFS, or can be run in standalone mode.

2. Proposed Work

2.1 Objectives, Experiment types, and Measured metrics

We aim to study the feasibility of deploying distributed software on light-weight compute nodes composed of Raspberry Pi 4s (referred to as Pi cluster). We compare the acquired performance of the homogenous Pi cluster with a typical server machine, Cavium ThunderX, available in the Mystic cluster. Hadoop and Spark shall be deployed on the Pi cluster and ThunderX. A wordcount benchmark will be run on Hadoop using the MapReduce framework. K-Means clustering benchmark will be run on Spark run on top of HDFS and yarn. Throughput, cluster scalability, and data processed to power consumption ratio are measured and a comparison is drawn.

2.2 Tasks performed

The preliminary tasks in this project were to properly set up the clusters, set up NFS, and configure communication between the nodes. SSH was installed on each node and public/private keys were distributed to enable communication of the name nodes with the data node and the pi server. Cluster security was not considered for this experiment. Commands sourced in the .bashrc file of the server allowed copying files and executing commands on each of the pis from the server terminal, allowing for easy control of the cluster. Java 8 was used as it was recommended for the versions of Hadoop and Spark that were run. Both Hadoop and Spark were downloaded and extracted into the NFS exports folder and then copied to a separate folder on each node. Configurations were added based on hardware capabilities and trial/error (further elaborations in section 3). To add power measuring capabilities, linux commands were explored as well as using provided utilities from outside sources such as the UPS and BMCs. IOzone was used to benchmark the disk performance for a single raspberry Pi. For further optimizations of Hadoop and Spark configs, resource usage logs and plots were generated to understand which resources were used when running benchmarks. Python scripting was used to generate data sets of varying size for both Word Count and KMeans benchmarks and scalability testing. Bash scripting was used to automate the word count and KMeans clustering experiments.

Significantly more time was allocated towards setting up a fully distributed cluster with Apache Hadoop/Spark on the Pi cluster as opposed to the pseudo-distributed version on the Cavium ThunderX. This was expected as there was only one node as compared to the 17 nodes to configure on the Pi cluster. However, once the Pi cluster was set up, both the ThunderX and Pi cluster had comparable usability.

3. Hardware and Cluster Specifications

3.1 Raspberry Pi

The raspberry pi model used for the project was the Raspberry Pi 4 Model B Rev 1.1. This model features a quad-core Arm Cortex-A72 CPU clocked at 1.5GHz with the 2GB RAM configuration. Ubuntu 19.10 was flashed on the SanDisk Extreme 90MB/s U3 64GB SD disk. Performance issues may result from the use of SD disks on Raspberry Pi as opposed to regular compute nodes that use HDDs and SSDs. A promising feature for the Raspberry Pi is its low power consumption, where each Pi can consume around 3.4 watts when idle.

Many of the Hadoop configurations have been based on the hardware of the compute nodes such as its RAM and disk size. The main Hadoop configuration files that were edited for the Pi cluster are hadoop-env.sh, core-site.xml, hdfs-site.xml, yarn-site.xml, mapred-site.xml, master, and workers. The master file contains the name of the master node, pi1 that will represent the NameNode, SecondaryNameNode, and the ResourceManager daemons. The worker file will contain the pis which will be used as workers. (DataNode and NodeManager daemons) This file was modified when running scalability testing. Hadoop-env.sh is edited with the path to java 8. The configurations changed on the hdfs-site.xml specified how much disk space on the SD will be left for the OS and other applications, allowing HDFS 48GB of space. Another configuration that varied throughout the project was the HDFS block size. The block size governs what size the files on HDFS are sliced, and can have many effects on the final benchmarks. The HDFS replication factor was set as 1. Finally, paths to the datanode and node directories were added. The yarn-site and the mapred-site files contained configurations that were based on the hardware capabilities. The yarn-site max allocation for each raspberry pi was set as 1536MB with the min allocation at 128MB. Based on the hardware capabilities, each raspberry pi was configured with 4 cores. The mapred-site.xml will hold memory configurations for MapReduce containers. Based on available ram, 512MB is allocated for each MapReduce container. Spark configurations were also based on the available ram and the Spark overhead when allocating containers .

3.2 ThunderX

The Cavium ThunderX features 2 sockets with 48 64 bit Armv8 cores clocked at 2GHz. It has 64GB of ram. This server machine is optimized for efficient data analytic and web services workloads, with multiple 10/40 GbE ports, and 650W redundant power supplies to support high uptime. It has a total of 16 DDR4 DIMM slots, which gives high flexibility to satisfy memory intensive workloads.

The main Hadoop configuration files that have been updated for the ThunderX are hadoop-env.sh, hdfs-site.xml, yarn-site.xml, and mapred-site.xml. The hdfs-site.xml holds configurations specific to HDFS and the configurations added here specify allocating 384GB of disk space to HDFS and a replication factor of 1. The HDFS block sizes were varied when running each benchmark. Inside the yarn-site.xml file, 64 v-cores were allocated as comparable to the raspberry pi cluster. The maximum yarn allocation was 48GB to leave space for the OS, etc. The yarn min allocation was set as 768MB. In the mapred-site.xml file, 768MB was allocated for the map-reduce container in order for Hadoop to use all its 64 cores.

3.3 Iozone Disk Benchmark

Since Hadoop and Spark are optimized for HDDs and SSDs, and the raspberry pis are running on SD cards and SD card controllers, we could not be sure they would behave as expected. Using IOzone, we benchmarked individual pis to ensure performance “dark spots” are avoided when configuring Hadoop and Spark. Fortunately, the disk behaviour for the Pis are similar to normal HDDs and SSDs. To sum up the performance achieved by the cards, they are able to reach up to 35MB/s for sequential and random reads, and 30MB/s for sequential and random writes, when benchmarked using 16-64MB file sizes and 512kB+ record lengths.
The detailed IOzone result log file is included with this report.

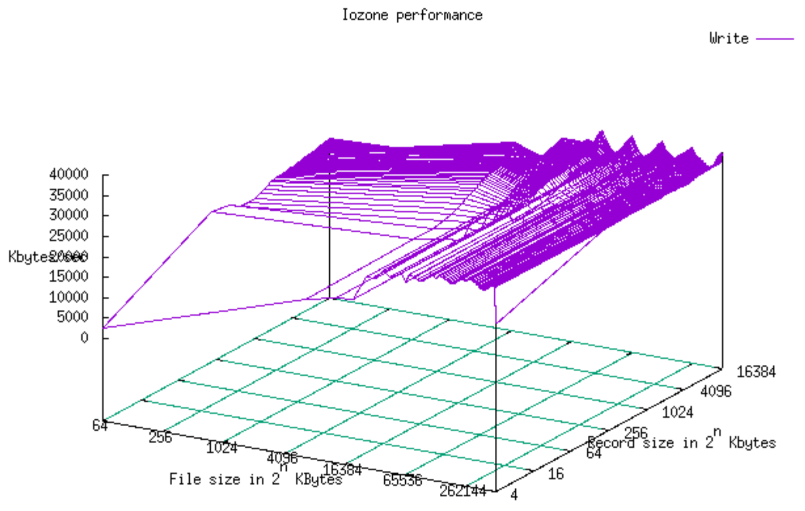


Fig 3.3.1 IOzone write results plot, ran on a single raspberry Pi

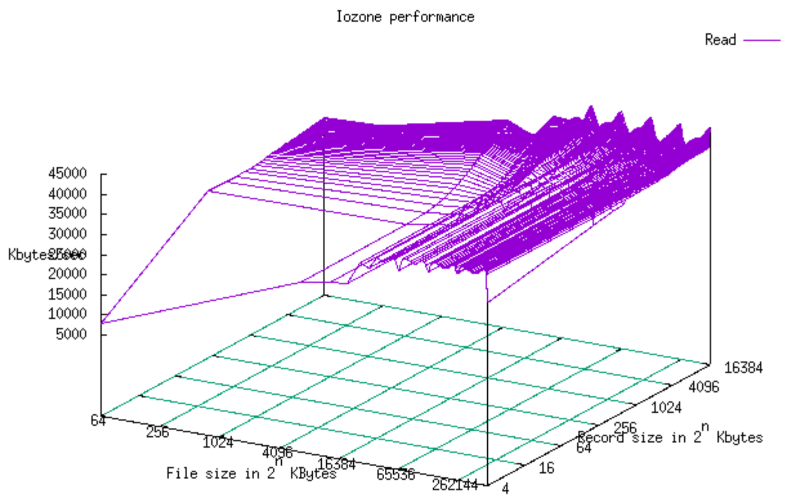


Fig 3.3.2 IOzone read results plot, ran on a single raspberry Pi

4. Benchmark Results

4.1 Word Count on MapReduce

The MapReduce benchmark that was chosen was Word Count as it applies well to the MapReduce programming model. Benchmarks were tested on HDFS block sizes ranging from 4MB to 256MB. As larger files were used, it was more efficient to use larger block sizes and 256MB proved to get the most optimized result. For the Pi cluster, based on the limited disk space, each node can handle around 16GB of data considering temporary storage. Using a weak scaling approach, 16GB of data is added to the workload when joining additional nodes to the experiments. Thus, for the final benchmark, 16 worker nodes handle 256GB of data.

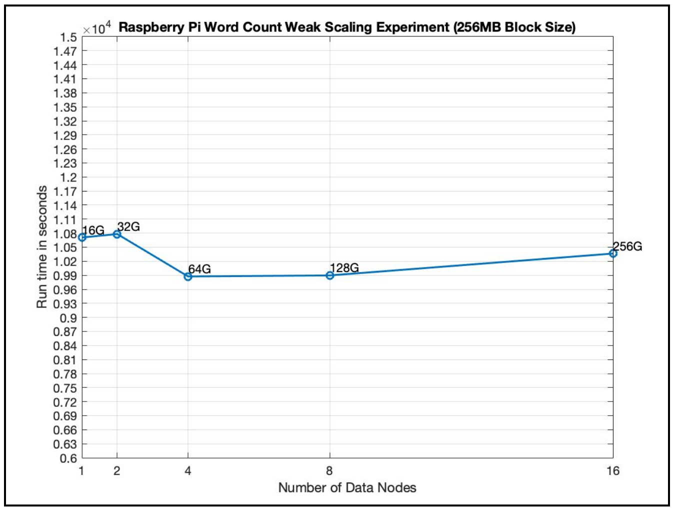


Figure 4.1.1: Raspberry Pi Weak Scaling Word Count Results

Figure 4.1 confirms that when running MapReduce on the Pi cluster, scalability is achieved, showing that the distributed framework is able to run well even with the performance drawbacks of the raspberry pi nodes.

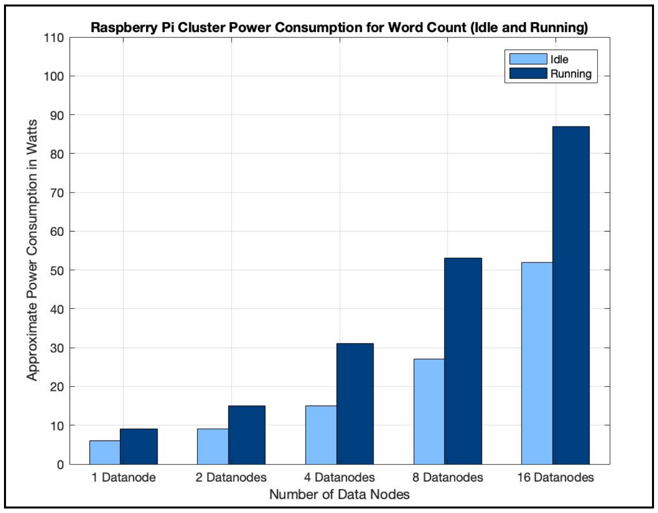


Figure 4.1.2: Raspberry Pi Power Consumption when running Word Count

To measure power consumption for the Pi cluster, an external UPS with a 10-watt granularity was used. Figure 4.2 displays the power readings as they change with the additional workers. For the 16 data node scale, power is measured at 87 watts, meaning each Pi was roughly consuming 5 watts of power when the wordcount benchmark was running. For the Raspberry Pi cluster considering a 16 data node scale, 256GB of data was processed in 10364 seconds, meaning 25.3 MB of data is processed per second. This translates to 87W/ 25.3 MB/s and 3.44 Watts of data processed per 1 MB/s.

When running word count on the ThunderX server, a 128GB workload was used based on the 384GB disk space allocated for HDFS. To measure the power consumption during this benchmark, a BMC is used that hosts a web server and displays the power measurement every second. When running the benchmark, the average of the power measurements is taken as 342 Watts. This is around 100 watts higher than idle power for the ThunderX.

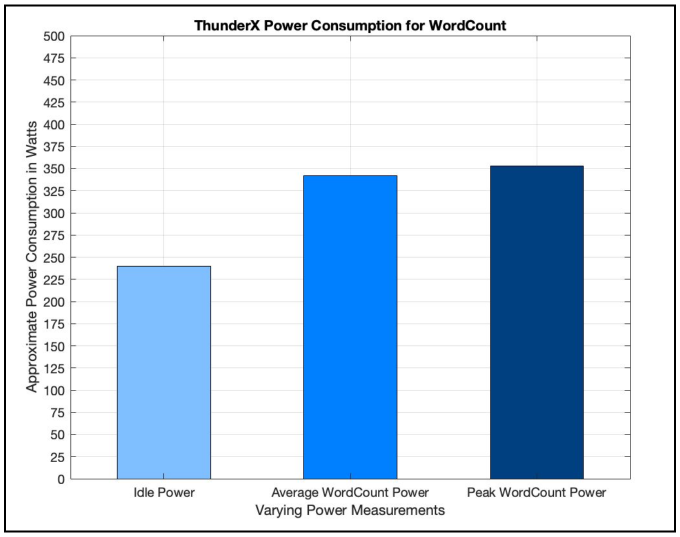


Figure 4.1.3: ThunderX Power Consumption when running Word Count

This workload finished in 2978s when running with 64 cores and 48GB of RAM, processing 44 MB of data per second. This translates to 342W/44 MB/s and 7.77 Watts of data processed per 1 MB/s. This measurement is double the raspberry pi, translating that the ThunderX takes up twice the amount of power for the same amount of data processed per second.

4.2 KMeans on Spark

The KMeans clustering is an unsupervised machine learning algorithm. Initially, K centroids are randomly selected, and then the following iterations are performed on input data to allocate the data point to the closest centroid. The algorithm keeps iterating until the centroid values no longer change and the optimal centroid position is located. This algorithm was chosen for the Spark benchmark because Spark was optimized for iterative algorithms that access the same data set repeatedly. The algorithm used for benchmarking has 5 K centroids and is limited to 20 iterations. The input data contains 3D points ranging from -10 to 10 with 3 decimal places. Because of the use of small files in this benchmark, the HDFS block size used was 4MB, which proved to attain the ideal result.

When running Spark on the raspberry pi cluster, the max yarn allocation was changed to 1.8GB to make the 1 data node experiment possible, giving space for both an application master container and an executor container. The application master size was left at the default of 512m while the executor memory was given 465m. Multiple benchmarks were run to tune Spark to use all the cluster resources. Figure 4.2.1 and 4.2.2 both show weak scaling results from KMeans on the Pi cluster. The tuned KMeans showed a speedup of 2x as compared to the original benchmark. This was expected as more CPU resources were allocated to the job by increasing the number of executor-cores.

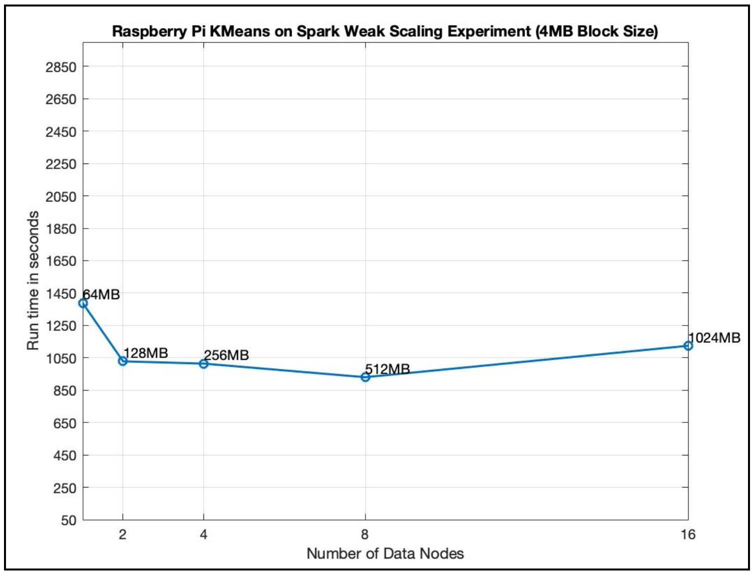


Figure 4.2.1: Raspberry Pi KMeans on Spark

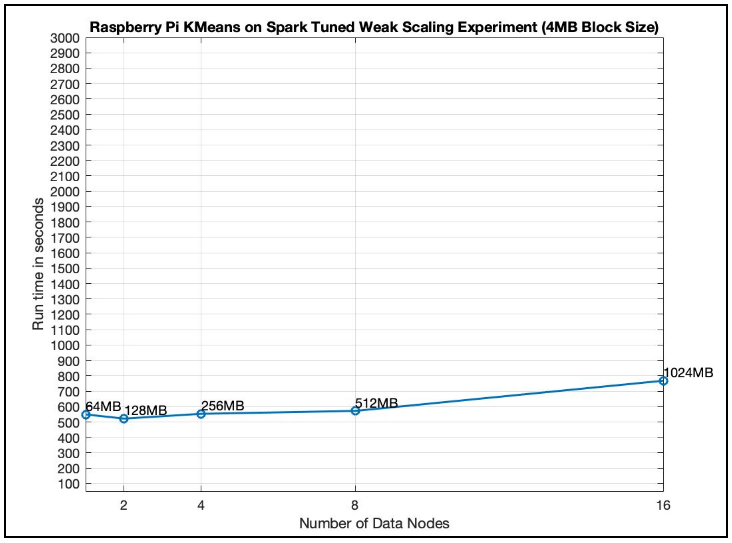


Figure 4.2.2: Raspberry Pi KMeans on Tuned Spark

When measuring power for the KMeans clustering, the UPS showed peaks and troughs of power consumption as the algorithm ran through its iterations. For power analysis calculation, the average power measurement was used.

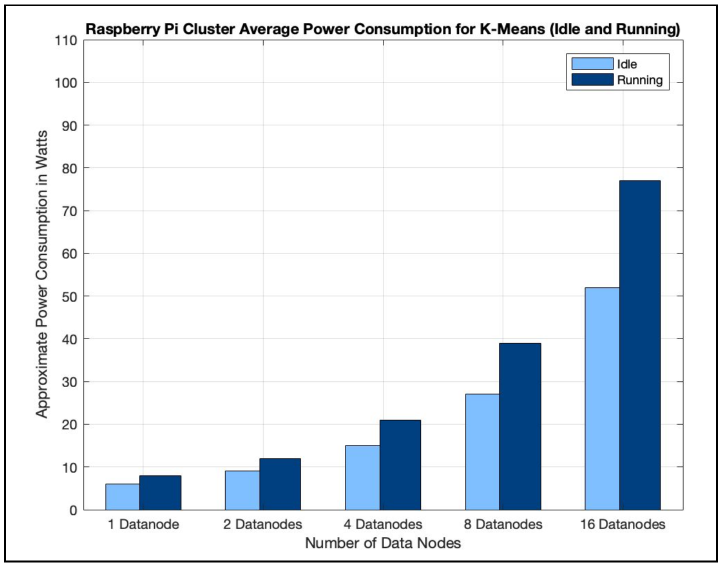


Figure 4.2.3: Raspberry Pi KMeans Power on Tuned Spark

As shown in figure 4.2.2, when considering 16 data nodes, the average power consumption was 77 watts. 1024MB/768s translates to 1.33 MB of data processed per second and 57.89 watts consumed per 1 MB of data processed per second.

KMeans clustering was repeated on the ThunderX server. A similar size workload of 3072MB was processed on the ThunderX in 681 seconds, translating to 4.5 MB/s.

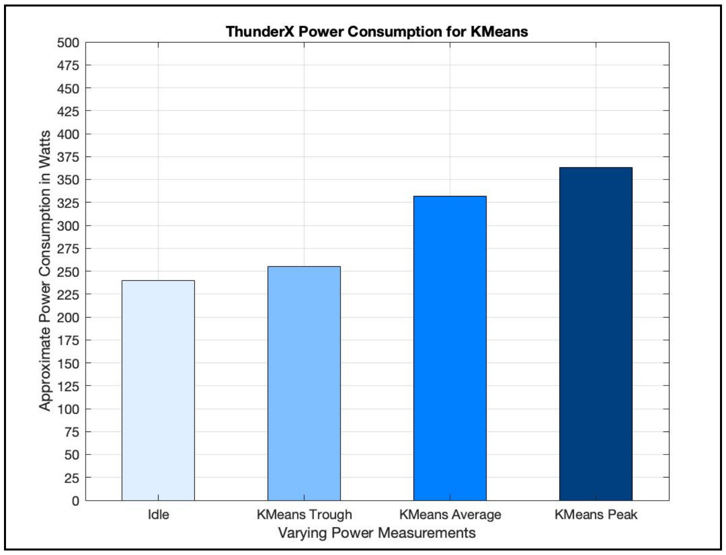


Figure 4.2.4: Raspberry Pi KMeans Power on Tuned Spark

Similar to the power consumption on the pi cluster, the power consumption when running KMeans has both peaks and troughs. The average was calculated at 332 watts, translating to 332/4.5 MB/s or 73.77 Watts consumed per 1 MB of data processed per second.

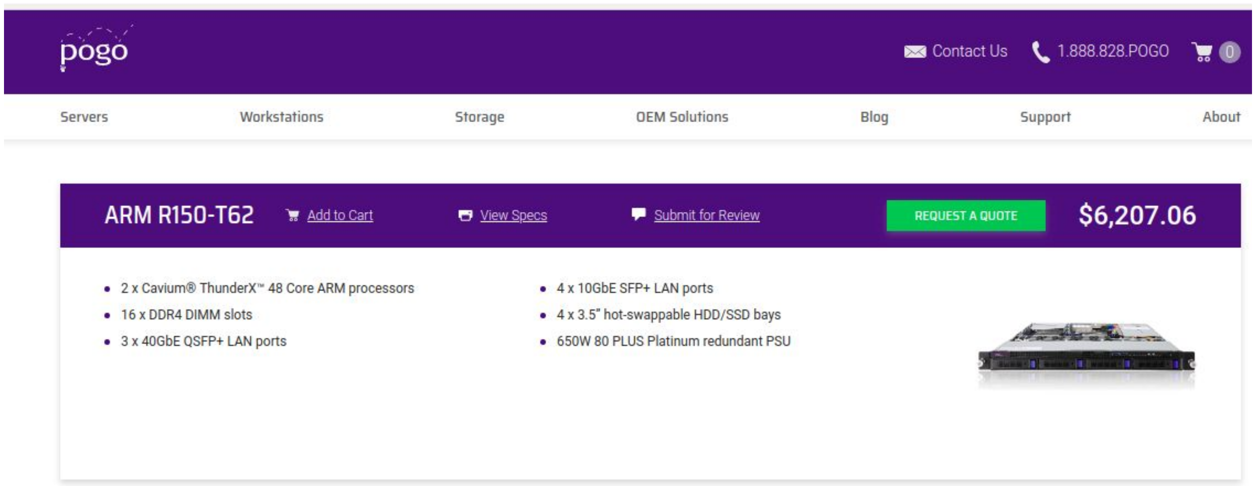
5. Cost and Performance Evaluation

To estimate the practical cost when deployed in real life scenarios, we study the amortized cost of deploying a pi cluster versus a ThunderX server machine. Here we focuses on two major contributing costs: Hardware costs and power costs.

For hardware costs, we obtained pricing quotes from online sources. Below is an estimate cost of the total hardware costs for the raspberry Pi cluster, consisting of 17 pis:
Raspberry Pi 4 2GB - $55.00 per Pi
SanDisk 64GB - $16.43 per card
TP-Link 8 Port Gigabit Ethernet Network Switch - $18.99 per switch
The total estimated hardware costs for the Pi cluster:
17*($55.00+$16.43) + 3*$18.99 = $1271.28

Pi: https://www.canakit.com/raspberry-pi-4-2gb.html
SD cards: https://www.amazon.com/SanDisk-64GB-Extreme-UHS-I-SDSDXV6-064G-GNCIN/dp/B07H42L4TJ/ref=dp_ob_title_ce
Switches:https://www.amazon.com/Ethernet-Splitter-Optimization-Unmanaged-TL-SG108/dp/B00A121WN6/ref=sr_1_4?dchild=1&keywords=network+switches&qid=1588882773&s=electronics&sr=1-4

For ThunderX machine, we are able to get a quote from Pogolinux:



ARM R150-T62 $6,207.06

- 2 x Cavium® ThunderX™ 48 Core ARM processors
- 16 x DDR4 DIMM slots
- 3 x 40GbE QSFP+ LAN ports
- 4 x 10GbE SFP+ LAN ports
- 4 x 3.5" hot-swappable HDD/SSD bays
- 650W 80 PLUS Platinum redundant PSU

Entering the amount of disk and ram we have on the current thunderX machine, gives an estimated hardware cost of $6,207.06 (excluding the raid controller that is installed in the machine).

We shall compare the estimated cost of ownership amortized over a 5 year period. Here are the parameters for cost evaluation:
Power cost: 13.19 cents per kilowatt hour (average power cost at April 2020)
Assuming both machines are fully utilized and running 24/7 -
Wordcount on Mapreduce -

	Data process rate(MB/s)	Total data processed over 5 years (PB)	Power consumption rate (W)	Total power consumed (kW h)
Pi cluster	25.6	4.036608	87	87/1000 * 5*365*24 = 3810.6
Thunderx	44	6.93792	342	342/1000 * 5*365*24 = 14979.6

=> Power cost -
Pi cluster = 3810.6*0.1319 = $502.61814
ThunderX = 14979.6*0.1319 = $1975.80924

Assuming the Pi cluster can be scaled up to reach the performance of the ThunderX machine, as supported by our scalability test of the Pi cluster, then the hardware cost for the theoretical pi cluster will be:
(Theoretical) Pi cluster, scale factor = 44/25.6 = 1.72
Hardware cost = 1271.28*1.72 = $2185.01
Power cost = 3810.6*1.72 = $864.00

Similar calculations can be done for K-means on Spark -

	Data process rate(MB/s)	Total data processed over 5 years (PB)	Power consumption rate (W)	Total power consumed (kW h)
Pi cluster	1.33	0.2097144	77	77/1000 * 5*365*24 = 3372.6
Thunderx	4.5	0.70956	332	332/1000 * 5*365*24 = 14541.6

=> Power cost -
Pi cluster = 3372.6*0.1319 = $444.84594
ThunderX = 14541.6*0.1319 = $1918.03704
(Theoretical) Pi cluster, scale factor = 1.33/4.5 = 3.38
Hardware cost = 1271.28*3.38 = $4301.32
Power cost = 444.85*3.38 = $1503.59

Comparing the cost of ownership of the theoretical pi cluster and ThunderX server side by side:
Amortized cost of ownership over a 5 year period:

	(Theoretical) Pi Cluster	ThunderX
Disk intensive workloads (Wordcount)	$3049.01	$8183.87
CPU intensive workloads (K-means)	$5804.91	$8126.10

6. Conclusion

The raspberry pis are cheap, compact, and consume low power. Apache Hadoop/Spark works well on this commodity hardware as each benchmark achieved great scalability and throughput. Investing on a cluster of raspberry pis as opposed to 1 ThunderX showed greater flexibility, fault tolerance and power saving capabilities. From our cost of ownership study, a comparison of the two solutions shows the Pi cluster is an economically feasible and perhaps performance-wise superior alternative to typical server machines when deployed on small scales.

However, there are some downsides for the Pi cluster that should be considered. The maintenance effort is higher than traditional server machines. Typical server machines come packaged with monitoring hardwares (BMCs) and tools, where raspberry pis generally lack. Pi cluster also does not come with server-rack compatible mounts, which also negatively affects maintainability.

7. References

[1] Apache Hadoop. https://hadoop.apache.org.
[2] Apache Spark. https://spark.apache.org.
[3] server-world.info. https://www.server-world.info/en/note?os=Ubuntu_18.04&p=nfs&f=2