BENCHMARKING THE PERFORMANCE OF A CLUSTER OF HETEROGENEOUS SINGLE-BOARD COMPUTERS TO SCALE

OBAID MUHAMMAD TALHA

(BSc (Hons.))

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF INFORMATION SYSTEMS

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2017

DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in it is entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

[illegible]

____

Obaid Muhammad Talha

27 April 2017

ACKNOWLEDGEMENT

I would like to acknowledge Prof. Tuan Q. Phan for giving me a valuable opportunity to work with him for this dissertation. It was indeed an honor to be part of his team.

Moreover, I would love to acknowledge my parents. It was only because of their immense support and encouragement, I reached this far in life.

Also, I would like to acknowledge my loving wife for keeping me motivated and focused in pursuing this dissertation, while also helping me all the way in raising our three lovely and adorable kids.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

In this dissertation, a cluster of heterogeneous single-board computers running a computing framework, is benchmarked while scaling considerations were addressed. The single-board computers in the cluster have different hardware configurations from different vendors. Permutations of these single-board computers were used, and benchmarked against each other. Four cluster arrangements were setup and benchmarked, i.e. same nodes in a cluster, different nodes in a cluster, a cluster with single powerful node and a cluster where both master and slave is running on a master node.

The cluster contained six nodes in total, having three pairs from three different vendors and model. The types are Banana Pi, Raspberry Pi and Odroid. The Odroid have more than double the compute capability than that of Banana Pi and Raspberry Pi. Apache Spark is used as a cluster computing framework, having version 1.6.1. For benchmarking, an open-source Apache Spark performance benchmark, i.e. spark-perf, is used and end-to-end application running time is logged.

The benchmark tasks executed over the cluster involved generated dataset of size reaching over a million data points. The task types are data aggregation, sorting, counting and scheduling. Also, test runs were executed iteratively over the cluster to measure performance after cache is warmed up. The median time of iterative runs, consumed by every task to complete in seconds, is used as a performance metric.

Multiple comparative studies were conducted over various cluster setup results. It was observed that when scaling horizontally i.e. increasing the nodes in the cluster, of same type, or vertically i.e. adding nodes of higher capability, does not always impart gain in performance of the benchmark task. Also, higher compute capable nodes do not perform much well compared to their lower counterparts. Also, once the cache is warmed up, the performance is much improved and stays consistent.

CHAPTER 1: INTRODUCTION

As the advent of miniaturization progressed, and more silicon is being squeezed into a fraction of a centimeter, it gave birth to Single-board computers. As the name suggest, it is a silicon wafer having all computing components embedded on the same board, hardwired together. Such single-board computers are mostly equipped with all necessary hardware interfaces onboard, such as USBs, Ethernet, and display ports etc.

Since they serve as independent computing and I/O capable hardware, they can be used as a node in a cluster computing settings. With the availability of efficient cluster computing frameworks, it is imperative to use such nodes as a compute node.

For the cluster to perform optimal with ever increasing data and computing requirements, it must scale. Scaling is essential to resolve current computing challenges of big data analytics. The cluster can linearly scale, both horizontally, i.e. replicating existing nodes, and vertically, i.e. adding more powerful nodes to the cluster. Either of such scaling, would results in improved computational performance while lowering the cost and time from build to scale, is the primary motivation for the benchmarking.

In the past, there had been significant attempts in proving the possibility, viability and efficiency of clusters of single-board computers of same type, in both small and large scale deployments in a lab and a production environment. In such attempts, various features of hardware were benchmarked involving Ethernet interface, USB interface, SD Card I/O, CPU Stress and Disk I/O. However, a void is felt in terms of answering the questions of how to build and later how to optimize and scale a heterogeneous cluster of single-board computers with the application running time measured end-to-end.

Herewith the objective is to identify whether horizontally expanding a cluster by adding similar compute nodes, results in any significant gain in performance. In contrast, when trying to scale the cluster vertically, by adding a single compute node, with higher number of cores, whether it would perform any better than the same number of cores split over multiple compute nodes.

Also in a cluster setup, having nodes with double the compute capability, would they result in any significant performance gain, when compared against a cluster with nodes having half of that compute capability, provided both clusters solving the same problem with the same dataset. For a specific problem and an input dataset of certain size, is there a notion of an optimal count of compute nodes for such requirement, and does adding more of similar nodes would always provide any gain in performance. Moreover, would doubling the size of the dataset, causes any degraded performance compared to half the size of data where the earlier is indexed well in contrast to the latter.

A network of nodes is setup, connected over Ethernet while utilizing their capabilities using a cluster computing framework, i.e. Apache Spark. To measure and benchmark the performance, an off-the-shelf benchmark is used, i.e. spark-perf, and ran over the cluster. The performance is logged by running applications and results are deduced.

The performance is measured in the units of seconds, and the end-to-end running time of each task in a series of tests is recorded. Thus, the running time involved any latencies of Storage I/O, network lag, CPU Stress and Cache latency is also accounted for.

Latency is critical in applied domain of cluster computing, such as big data analytics. The model can be developed off the cluster, and later can be deployed on the cluster, primarily involved in lift-and-shift of data. Moreover, since such cluster is low-power and low-cost, and it is quite portable, it can be deployed on any off-the-grid device non-stationary device or vehicle, capable of performing any of intelligent machine learning tasks. From monitoring the inbound and outbound traffic, to making intelligent decisions in real time onboard such devices, even few seconds of latency once avoided provides significant impact.

The known single-board computers or just SBCs, available for the cluster setup are explored to begin with, while considering their specifications individually. Later Apache Spark – the cluster computing framework is explored. The setup of SBC's is discussed, i.e. installing OS over them, and configuring Apache Spark on each SBC and the master node, i.e. the laptop. After that the spark-perf benchmark is configured on the nodes. A deeper dive was taken into the benchmark code, and certain parameters were tweaked to make it work with the cluster, over the provided generated data.

Once the cluster is live, connected and Apache Spark and spark-perf is running over it, the cluster is altered by changing the slave nodes i.e. SBC's, of various types and in various permutations, having laptop as a master node, and the performance of each application is noted.

Various cluster setups were benchmarked, from having same nodes in the cluster, having different nodes in the cluster, to having single powerful node in the cluster. Also, benchmarked a setup where both the master and slaves were run over the same hardware, to serve as a comparison against the cluster performance over a network. The findings are shared with tables and figures, where each choice of worker nodes is compared against the other suitable counterpart. This helped to conclude results based of how one setup of nodes performed against another setup.

The cache warmup was also considered, i.e. populate the cache, and then run the set of tasks over it, a feature provided in the benchmark. For each cache warmup run, it is plotted against each other, to identify once the cache is warmed up, how it impacts performance. This is in addition to each task performing on each permutations of nodes, and each node permutation performing against all other permutations for a given task, both are also benchmarked. It is concluded with elaborating observed results and business value from the contributions, with avenue for future extensions.

CHAPTER 2: PREVIOUS EFFORTS

In the past, there had been attempts to prove the fact that setting up the cluster with SBC's alone, is practical and reasonable for running compute intensive applications. For instance, while discussing the performance of five Cubieboard in a cluster [Fan16], solving classical benchmark problems, it proves that such clusters are viable for real-world problems.

Another effort in which a portable cloud was built using Raspberry Pi [Haj16], to demonstrate that, compared to virtualized environment over the cloud, the cluster of Raspberry Pi performs way better.

From proof of an idea and value, as detailed above, to expansion and deployment in larger number of nodes in a cluster, there had been efforts to build a cluster with tens of nodes. For instance, a 40-node cluster of Raspberry Pi tower is erected [DG14], for an MSCE thesis.

Another effort builds a 32-node Raspberry Pi cluster [JK, JK13].

Furthermore, using Cubieboard, a cluster is setup with 22-nodes [CK14, SUTAC, AIY], where two nodes are used as data nodes.

Once the cluster was expanded, there were critiques as well, to identify whether it is viable for big data processing. Studies [LPHC] shed light on that area.

And, there had been efforts about setting up Apache Spark on a cluster of Odroids, [MK16].

Over Raspberry Pi, Odroid and Orange Pi, there had been benchmarking done [Jeff16]. It measures their MicroSD transfer rate [JeffSD], and performs their Networking Benchmark, Power consumption benchmark and Network interface benchmarks [Jeff15].

However, a void is observed, for a question asking; how would the cluster perform, when built with the available mix of SBC's, running in unison. Also, how to identify whether higher compute nodes, perform any better than their lower counterparts, for a given problem and dataset. Moreover, what is the cluster performance for an end-to-end task execution, including any latency of data flow, when going through the hardware interfaces, like that of networking and MicroSD. Also, do doubling the nodes return any gain in performance. And, what impact does cache warmups have, over the overall cluster performance.

CHAPTER 3: CLUSTER ENVIRONMENT OVERVIEW

A cluster is setup, using a mix of nodes, from varying compute capacity, to difference in cost. A benchmark is executed over the cluster to obtain end-to-end impact for a volume of, generated data points, i.e. in millions. Also, the number of nodes in the cluster are doubled to identify the impact on performance. Not only this, but also the cache warmup runs were measured to observe the performance once the cache is optimal.

3.1 SINGLE-BOARD COMPUTERS

Single board computer is a fully functional computer including microprocessor, memory, I/O, etched on the same board [SBC]. They lack expansion slots for peripheral attachments. They are different from Microcontrollers, as they are equipped with separate storage and memory on the same board.

There are many vendors providing single-board computers. The select few and the arguably most widely used ones are Banana Pi, Raspberry Pi and Odroid. Within the cluster design, these single-board computers are used as slave nodes.

3.1.1 RASPBERRY PI

Raspberry Pi's are the most popular single-board computers [RPI]. They have a very active echo system, and come with multiple add-ons and boxing. They are fan-less and thus having a smaller power consumption foot print. The Raspberry Pi 2 Model B is the second-generation Raspberry Pi. Because it has an ARMv7 processor, it can run the full range of ARM GNU/Linux distributions.

Raspbian OS

Raspberry Pi's have their own Debian-based OS, called Raspbian [RPOS]. Raspbian comes pre-installed with plenty of software for education, programming and general use. Raspbian was installed over 8Gb Micro SD card.

Table 1 are the detailed specifications about the Raspberry Pi used.

TABLE 1: RASPBERRY PI 2 MODEL V1.1 SPECIFICATIONS

Model: Raspberry Pi 2 Model B v1.1	
• ARM v7 architecture • **900MHz quad-core** ARM Cortex-A7 CPU • **1.0 GB RAM** (shared with GPU) • 4 USB ports • 40 GPIO pins • Full HDMI port • Ethernet port 10/100 Mbits • Combined 3.5mm audio jack and composite video • Camera interface (CSI) • Display interface (DSI) • Micro SDHC card slot • VideoCore IV 3D graphics core • HDMI composite video	

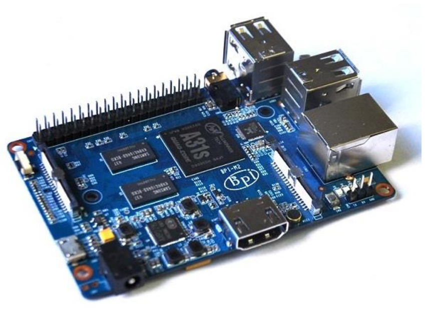
3.1.2 BANANA PI

Banana Pi M2 is a second-generation single board computer with an upgraded SoC to provide even more power for computing tasks [BPI]. It is considered an alternative when compared with Raspberry Pi having 900 MHz, while Banana Pi have a higher clock speed of 1.2 GHz.

Bananian OS

The version of OS installed over Banana Pi is a variant of Raspbian OS, specifically ported over Banana pi [BPOS]. It supports all the features available over Raspberry Pi, and is installed over 16 GB Micro SD card.

TABLE 2: BANANA PI BPI M2 SPECIFICATIONS

Model: Banana Pi BPI M2	
• A31S ARM Cortex-A7™ **Quad-core 1.2 GHz** 256K B L1 cache 1MB L2 cache • PowerVR SGX54MP2 Comply with OpenGL ES 2.0 OpenCL 1x, DX9_3 • **1.0 GB DDR3** (shared with GPU) • MicroSD Card (up to 64GB) • 10/100/1000 Ethernet RJ45 and Wi-Fi 802.11b/g/n • Parallel 8-bit camera interface • HDMI, LVDS/RGB (no composite video) • 3.5 mm Jack and HDMI • 5V DC @ 2A (4.0mm/1.7mm barrel plug - center positive) or USB OTG • 4x USB 2.0 • Power/Reset: next to Camera Connector • Dimensions 92mm x 60mm • Weight 52g	

3.1.3 ODROID

ODROID-XU4 is a new generation of computing device with more powerful, more energy-efficient hardware and a smaller form factor [OD].

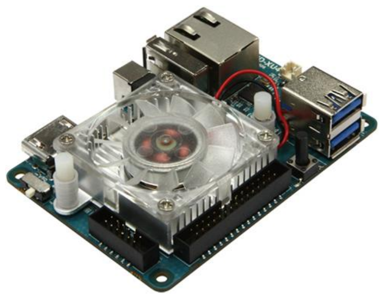
Offering open source support, the board can run various flavours of Linux, including Ubuntu 15.10 and Android 4.4 KitKat and 5.0 Lollipop.

By implementing the eMMC 5.0, USB 3.0 and Gigabit Ethernet interfaces, the ODROID-XU4 boasts amazing data transfer speeds, a feature that is increasingly required to support advanced processing power on ARM devices.

Ubuntu 15.10 mate

Odroid is strong and capable enough to run full version of Ubuntu 15.10 mate. It is equipped with Kernel 3.10.92. Since it is Debian based, it supports all APT software install and upgrade. It is installed over a 16 Gb Micro SD card.

TABLE 3: ODROID-XU4 SPECIFICATIONS

Model: Odroid-XU4	
• Samsung Exynos5422 Cortex™-A15 **Quad 2.0 GHz** and Cortex™-A7 **Quad 1.4 GHz** – Octa core CPUs • Mali-T628 MP6(OpenGL ES 3.0/2.0/1.1 and OpenCL 1.1 Full profile) • **2.0 GB** LPDDR3 RAM PoP stacked • eMMC5.0 HS400 Flash Storage (Switch for that – wow) • 2 x USB 3.0 Host, 1 x USB 2.0 Host • Gigabit Ethernet port 10/100/1000 • HDMI 1.4a for display • Size: 82 x 58 x 22 mm approx. (including cooling fan)	

3.2 CLUSTER MASTER

The laptop is used as a master node in the Apache Spark cluster computing settings. A master node is responsible to assign tasks to slaves nodes, i.e. the SBC's mentioned above. It is equipped with Mac OS 10+. Table 4 are the detailed specifications of the master node.

TABLE 4: LAPTOP MASTER MACBOOK PRO MID-2012 SPECIFICATIONS

Model: MacBook Pro Mid-2012	
• **2.6GHz Intel Core i7** processor (Turbo Boost up to 3.6GHz) with 4MB L3 cache • Processor: **8 Cores** • **16GB** of 1600MHz DDR3 memory • 2.5GHz 750GB 5400-rpm hard drive • Intel HD Graphics 4000 • Gigabit Ethernet port • Two USB 3 ports (up to 5 Gbps) • 802.11n Wi-Fi wireless networking;3 IEEE 802.11a/b/g compatible	

3.3 CLUSTER COMPUTING FRAMEWORK

The availability of the single-board computers, opened avenue of having a bunch of them, i.e. a cost-effective way to setup a cluster. Thus, it opened an opportunity to run multiple jobs in parallel – a distributed execution.

For a distributed system [DS], a problem is divided into tasks and the tasks are solved by one or more compute nodes. To achieve the task distribution, and later result collation, a framework comes to the rescue. One such framework is Apache Spark, from Apache foundation, explained next.

3.3.1 APACHE SPARK

Apache Spark is an open source cluster computing framework, originally developed by AMPLab at UC Berkley [SPRK]. Later the codebase was donated to Apache Software Foundation.

The strength of Apache Spark lies in RDD – resilient distributed dataset. Apache Spark also referred as just Spark, follows a Master and Slave or Worker execution model. It provides different set of API functions to build applications on top of them.

3.3.2 CLUSTER SETUP

The cluster is setup, including the Master node, i.e. the laptop and, all Worker/Slave nodes.

The detail steps of how to setup the cluster is explained in Appendix 1.

The setup will ensure that the Apache Spark is configured properly and is running over the cluster of Single Board Computers. When tried to submit the job, it should give results of the sample job.

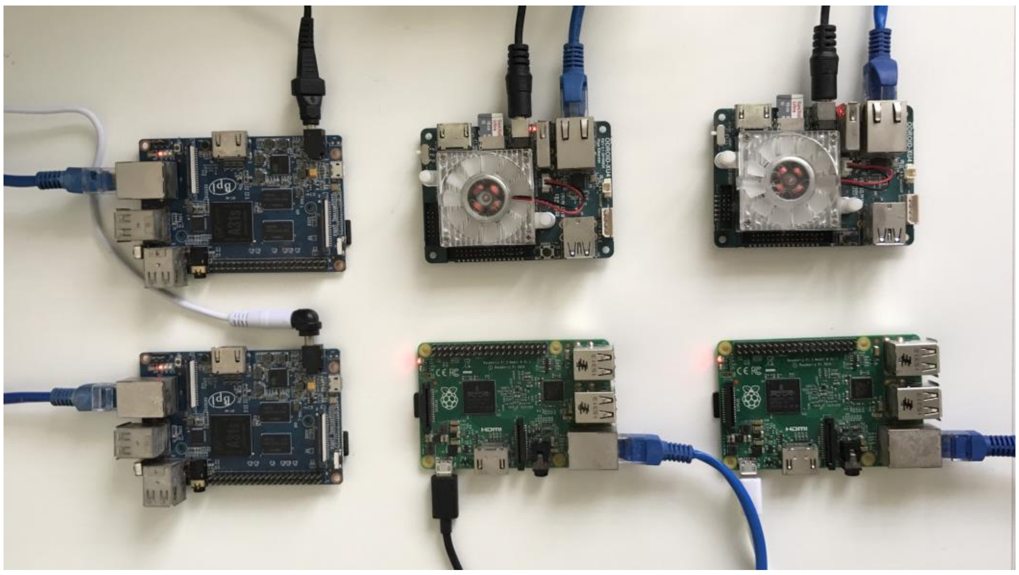


FIGURE 1: CLUSTER NETWORK

3.4 PERFORMACE BENCHMARKING

To realize how the cluster will perform, benchmarking is done by running different applications on the same cluster, with different permutations of single-board computer nodes. Benchmarking provides the metrics in terms of time taken, i.e. in seconds to complete the job on the cluster.

The choice of benchmark can be either a custom program, or off-the-shelf benchmark. The latter was opted, i.e. spark-perf [PERF] benchmark from Databricks; the same people behind Apache Spark. It is equipped with a comprehensive suite of tasks and test runs, and it is suitable for the cluster at hand. Another choice of benchmark is SparkBench, however it serves more to augment what is tested by spark-perf [DA16].

3.4.1 Spark-Perf

The benchmark provides eight set of applications. The list of applications from the benchmark are:

```
val test: PerfTest = testName match {
  case "aggregate-by-key" => new AggregateByKey(sc)
  case "aggregate-by-key-int" => new AggregateByKeyInt(sc)
  case "aggregate-by-key-naive" => new AggregateByKeyNaive(sc)
  case "sort-by-key" => new SortByKey(sc)
  case "sort-by-key-int" => new SortByKeyInt(sc)
  case "count" => new Count(sc)
  case "count-with-filter" => new CountWithFilter(sc)
  case "scheduling-throughput" => new SchedulerThroughputTest(sc)
}
```

Figure 2: Spark-Perf Tests Names

On top of these applications, Spark-Perf also does Disk-Warmups, before executing these tests.

These tests are part of Shuffling operation done by Spark. The tests include, aggregating data using a key, which is an integer in the second test. Similarly, the sort operation is also performed, exclusively with integer as the key, as well. Moreover, the count operation is also performed, exclusively with the filter.

All, of the tests are run by number of trial times. The median time in seconds is used as the time taken for the task to complete.

3.4.1.1 Disk Warmups

```
 # The following options value sets are shared among all tests.

 COMMON_OPTS = [

     # How many times to run each experiment - used to warm up system caches.

     # This OptionSet should probably only have a single value (i.e., length 1)

     # since it doesn't make sense to have multiple values here.

     OptionSet("num-trials", [10]),

     # Extra pause added between trials, in seconds. For runs with large amounts

     # of shuffle data, this gives time for buffer cache write-back.

     OptionSet("inter-trial-wait", [3])
```

FIGURE 3: SPARK-PERF CACHE WARMUPS

The disk warmup is explained further.

Setup Configuration

The performance benchmark setup steps are explained in detail in Appendix 2.

Caveats

There are few caveats to look out for while executing the benchmark and keeping the cluster up and running. They are discussed in detail in Appendix 4.

Benchmark Load details

Test Runner Tasks

The tasks have specific number of data points generated, to execute on. There are set of unique keys, with length of each key, along with how many values are unique in the benchmark data. Also, what is the length of each data point is also specified. Table 5 shows these details of generated dataset.

TABLE 5: BENCHMARK DATA DETAILS – TEST RUNNER TASKS

	Number of Records	Unique Keys	Key Length	Unique Values	Value Length
aggregate-by-key	**600,000**	60	10	3,000	10
aggregate-by-key-int	**1,200,000**	120	10	6,000	10
aggregate-by-key-naive	**600,000**	60	10	3,000	10
sort-by-key	**60,000**	6	10	300	10
sort-by-key-int	**120,000**	12	10	600	10
count	**600,000**	60	10	3,000	10
count-with-filter	**600,000**	60	10	3,000	10

As observed in Table 5, the tasks dealing with integer keys, are doubled in number of records than that of non-integers. This will help to identify how much the performance is impacted when the data is keyed optimally.

Test Runners

For each of the task, specified in Table 5, and for the Cache Warmup, the details of these trials are mentioned in Table 6.

The number of trial is kept consistent, and the wait between two trials. Number of partitions to spread the data over, is also specified for each task. Since the data is generated, executing in multiple trials ensured that the cluster exposed to varying data each time.

TABLE 6: BENCHMARK TRIALS DETAILS – TEST RUNNER

	Number Trials	**Inter Trial Wait**	**Number of Partitions**	**Reduce Tasks**	**Random Seed**	**Persistent Type**
aggregate-by-key	10	3	1	1	5	memory
aggregate-by-key-int	10	3	2	2	5	memory
aggregate-by-key-naive	10	3	1	1	5	memory
sort-by-key	10	3	1	1	5	memory
sort-by-key-int	10	3	1	1	5	memory
count	10	3	1	1	5	memory
count-with-filter	10	3	1	1	5	memory
			Number of Tasks	**Number of Jobs**		**Closure size**
scheduling-throughput	10	3	10000	1	5	0

The scheduling task distributes number of tasks in a job, across the cluster. It is interesting to observe the cluster performance while executing such task. Moreover, all the tasks are kept within in-memory, for allowing warming up of the cache.

CHAPTER 4: EXPERIMENT SETUP & RESULTS

Different selection of nodes was used in setting up clusters. The benchmark was run on clusters having various permutations of same nodes, different nodes, single node, where their performance is differentiated against each other.

Primarily, the laptop is used as a master, and nodes as slaves or worker. Later in the benchmark laptop alone is also tried, as both master and slave running on same hardware.

Each benchmark run involved two sets of measurements. First one measured different tasks running back to back each other on the cluster. The second involving running the same task in iteration, thus measuring the impact of cache-warmup on performance.

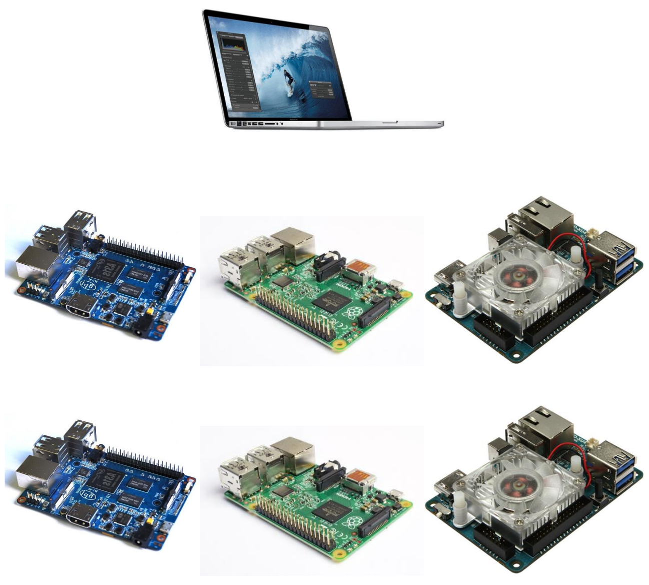


FIGURE 4: CLUSTER MASTER/SLAVE SETUP

4.1 SETUP I – SAME SLAVE NODES IN THE CLUSTER

In this setup, exactly similar slave nodes in a cluster are benchmarked, having laptop as a master node. Thus, cluster having two nodes, of both Raspberry Pi's are executed. Similarly, the cluster of Banana Pi's only, and a cluster of Odroids are benchmarked.

Because of Setup 1, the following results were obtained.

4.1.1 2 Raspberry Pi – 8 (4+4) cores, 1.0 (512 + 512) GB

Results of different tasks running across 2 Raspberry Pi cluster are presented in Table 7. The tasks running first are first in the table row.

Table 7: 2 RP - Test Runner (Different Tasks)

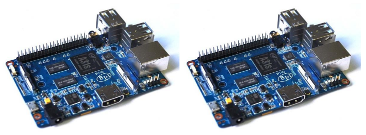
Name	Cores	Memory per Node	Secs
TR: scheduling-throughput	8	512.0 MB	300
TR: aggregate-by-key	8	512.0 MB	210
TR: aggregate-by-key-int	8	512.0 MB	96
TR: aggregate-by-key-naive	8	512.0 MB	510
TR: sort-by-key	8	512.0 MB	108
TR: sort-by-key-int	8	512.0 MB	126
TR: count	8	512.0 MB	156
TR: count-with-filter	8	512.0 MB	168

Test runner repeated 10 times to observe impact of caching. The first row is run first and last is run last.

Table 8: 2 RP - Test Runner (10 Repeats)

Name	Cores	Memory per Node	Secs
TR - 1	8	512.0 MB	480
TR - 2	8	512.0 MB	204
TR - 3	8	512.0 MB	126
TR - 4	8	512.0 MB	228
TR - 5	8	512.0 MB	60
TR - 6	8	512.0 MB	66
TR - 7	8	512.0 MB	114
TR - 8	8	512.0 MB	162
TR - 9	8	512.0 MB	72
TR - 10	8	512.0 MB	72

4.1.2 2 Banana Pi – 8 (4+4) cores, 1.0 (512 + 512) GB



Different tasks running on Banana Pi. The tasks which are first in row, are run first, then second row and so on to last row.

Table 9: 2 BP - Test Runner (Different Tasks)

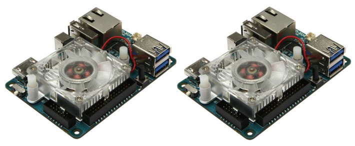
Name	Cores	Memory per Node	Secs
TR: scheduling-throughput	8	512.0 MB	234
TR: aggregate-by-key	8	512.0 MB	186
TR: aggregate-by-key-int	8	512.0 MB	84
TR: aggregate-by-key-naive	8	512.0 MB	426
TR: sort-by-key	8	512.0 MB	96
TR: sort-by-key-int	8	512.0 MB	108
TR: count	8	512.0 MB	138
TR: count-with-filter	8	512.0 MB	150

Cache warmup time measured for Banana Pi. It is observed that the time improved with iterations running from top to down.

Table 10: 2 BP - Test Runner (10 Repeats)

Name	Cores	Memory per Node	Secs
TR - 1	8	512.0 MB	510
TR - 2	8	512.0 MB	174
TR - 3	8	512.0 MB	114
TR - 4	8	512.0 MB	192
TR - 5	8	512.0 MB	58
TR - 6	8	512.0 MB	60
TR - 7	8	512.0 MB	96
TR - 8	8	512.0 MB	138
TR - 9	8	512.0 MB	60
TR - 10	8	512.0 MB	60

4.1.3 2 ODROID – 16 (8+8) CORES, 1.0 (512 + 512) GB



Cluster of Odroid running different tasks. Odroids builds up of 16 cores, with 1 Gb of memory.

TABLE 11: 2 OD - TEST RUNNER (DIFFERENT TASKS)

Name	Cores	Memory per Node	Secs
TR: scheduling-throughput	16	512.0 MB	1080
TR: aggregate-by-key	16	512.0 MB	2340
TR: aggregate-by-key-int	16	512.0 MB	900
TR: aggregate-by-key-naive	16	512.0 MB	4680
TR: sort-by-key	16	512.0 MB	780
TR: sort-by-key-int	16	512.0 MB	960
TR: count	16	512.0 MB	1440
TR: count-with-filter	16	512.0 MB	1800

The cache warmup times did not start that good, but improved from second iteration onwards.

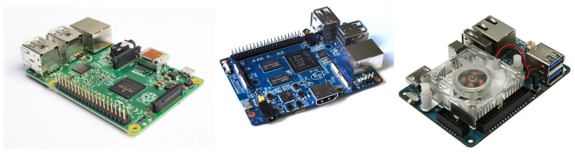
TABLE 12: 2 OD - TEST RUNNER (10 REPEATS)

Name	Cores	Memory per Node	Secs
TR - 1	16	512.0 MB	720
TR - 2	16	512.0 MB	96
TR - 3	16	512.0 MB	78
TR - 4	16	512.0 MB	96
TR - 5	16	512.0 MB	57
TR - 6	16	512.0 MB	59
TR - 7	16	512.0 MB	66
TR - 8	16	512.0 MB	78
TR - 9	16	512.0 MB	72
TR - 10	16	512.0 MB	72

4.2 SETUP II – DIFFERENT SLAVE NODES IN THE CLUSTER

The benchmark was run on a mix of slave nodes, and the laptop as a master node in the cluster. First, one of each of the nodes in the cluster, i.e. three nodes - a Raspberry Pi, a Banana Pi and an Odroid each. Later both nodes of each type were run in the cluster, in total six nodes, i.e. two Raspberry Pi, two Banana Pi and two Odroids.

4.2.1 *1 BP & 1 RP & 1 OD – 16 (4+4+8) CORES, 1.5 (512 + 512 + 512) GB: 3 REPEATS*



In this setup, each of the three types of nodes were used. The results were logged thrice – only for this setup, to ensure that results are consistent. Below is the result of different tasks running on the cluster. In total, they sum up to 16 cores, and 1.5 Gb of memory.

TABLE 13: 1 BP & 1 RP & 1 OD - TEST RUNNER (DIFFERENT TASKS)

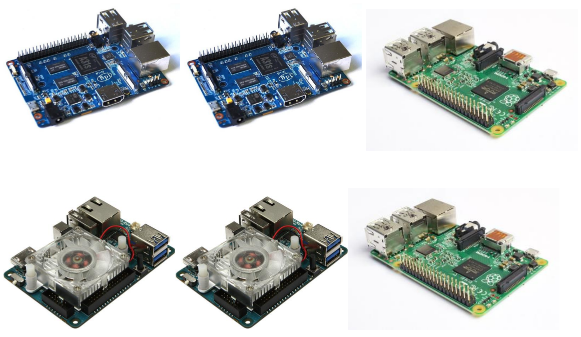
Name	Cores	Memory per Node	Secs 1st - iter	Secs 2nd - iter	Secs 3rd - iter
TR: scheduling-throughput	16	512.0 MB	228	210	222
TR: aggregate-by-key	16	512.0 MB	186	186	186
TR: aggregate-by-key-int	16	512.0 MB	84	84	84
TR: aggregate-by-key-naive	16	512.0 MB	426	432	432
TR: sort-by-key	16	512.0 MB	96	96	96
TR: sort-by-key-int	16	512.0 MB	108	108	108
TR: count	16	512.0 MB	144	138	138
TR: count-with-filter	16	512.0 MB	150	150	150

Later the test runners were executed on the cluster. Again, the test runs were run thrice – only in this setup, to validate that the result are consistent. It is observed that the time is improved.

TABLE 14: 1 BP & 1 RP &1 OD - TEST RUNNER (10 REPEATS)

Name	Cores	Memory per Node	Secs 1st - iter	Secs 2nd - iter	Secs 3rd - iter
TR - 1	16	512.0 MB	366	504	372
TR - 2	16	512.0 MB	174	174	174
TR - 3	16	512.0 MB	114	114	114
TR - 4	16	512.0 MB	186	192	192
TR - 5	16	512.0 MB	57	57	57
TR - 6	16	512.0 MB	60	60	60
TR - 7	16	512.0 MB	96	96	96
TR - 8	16	512.0 MB	144	138	138
TR - 9	16	512.0 MB	90	84	66
TR - 10	16	512.0 MB	72	138	66

4.2.2 2 BP & 2 RP & 2 OD – 32 (4+4+4+4+8+8) CORES, 3 (512 + 512 + 512 + 512 + 512 +512) GB



In this setup, all six nodes were used for the first time now. It sums up to 32 cores, and 3 Gb of memory. Later, it would give a valuable insight into whether doubling the nodes provide any performance gain.

TABLE 15: 2 BP & 2 RP & 2 OD - TEST RUNNER (DIFFERENT TASKS)

Name	Cores	Memory per Node	Secs
TR: scheduling-throughput	32	512.0 MB	138
TR: aggregate-by-key	32	512.0 MB	186
TR: aggregate-by-key-int	32	512.0 MB	84
TR: aggregate-by-key-naive	32	512.0 MB	432
TR: sort-by-key	32	512.0 MB	96
TR: sort-by-key-int	32	512.0 MB	108
TR: count	32	512.0 MB	138
TR: count-with-filter	32	512.0 MB	150

Cache warmup times are also logged for all six nodes.

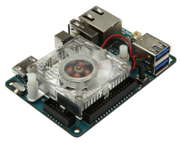
TABLE 16: 2 BP & 2 RP & 2 OD - TEST RUNNER (10 REPEATS)

Name	Cores	Memory per Node	Secs
TR - 1	32	512.0 MB	198
TR - 2	32	512.0 MB	174
TR - 3	32	512.0 MB	114
TR - 4	32	512.0 MB	192
TR - 5	32	512.0 MB	57
TR - 6	32	512.0 MB	60
TR - 7	32	512.0 MB	96
TR - 8	32	512.0 MB	138
TR - 9	32	512.0 MB	84
TR - 10	32	512.0 MB	66

4.3 SETUP III – CLUSTER OF A SINGLE POWERFUL SLAVE NODE

This setup involved a single slave node in the cluster. The slave node is the most powerful node, i.e. the Odroid, and the master node is the laptop.

4.3.1 1 ODROID –8 CORES, 512 MB



Single node is used in this cluster run alone. Off course the impact was observed in run times. The memory is kept at 512Mb for slave.

TABLE 17: 1 OD (512MB) - TEST RUNNER (DIFFERENT TASKS)

Name	Cores	Memory per Node	Secs 1 OD
TR: scheduling-throughput	8	512.0 MB	2040
TR: aggregate-by-key	8	512.0 MB	2400
TR: aggregate-by-key-int	8	512.0 MB	900
TR: aggregate-by-key-naive	8	512.0 MB	4680
TR: sort-by-key	8	512.0 MB	780
TR: sort-by-key-int	8	512.0 MB	960
TR: count	8	512.0 MB	1380
TR: count-with-filter	8	512.0 MB	1800

Cache warmup times were also logged.

TABLE 18: 1 OD (512MB) - TEST RUNNER (DIFFERENT TASKS)

Name	Cores	Memory per Node	Secs 1 OD
TR - 1	8	512.0 MB	1440
TR - 2	8	512.0 MB	90
TR - 3	8	512.0 MB	84
TR - 4	8	512.0 MB	96
TR - 5	8	512.0 MB	58
TR - 6	8	512.0 MB	60
TR - 7	8	512.0 MB	72
TR - 8	8	512.0 MB	84
TR - 9	8	512.0 MB	84
TR - 10	8	512.0 MB	84

4.3.2 1 ODROID –8 CORES, 1024 MB

The memory is doubled in this run, compared to the last run. Its observed that performance improved only slightly.

TABLE 19: 1 OD (1024MB) - TEST RUNNER (DIFFERENT TASKS)

Name	Cores	Memory per Node	Secs 1 OD
TR: scheduling-throughput	8	1024.0 MB	2040
TR: aggregate-by-key	8	1024.0 MB	2340
TR: aggregate-by-key-int	8	1024.0 MB	900
TR: aggregate-by-key-naive	8	1024.0 MB	4680
TR: sort-by-key	8	1024.0 MB	780
TR: sort-by-key-int	8	1024.0 MB	960
TR: count	8	1024.0 MB	1380
TR: count-with-filter	8	1024.0 MB	1740

Cache Warmups were also tested, and almost similar results were obtained with half the memory.

TABLE 20: 1 OD (1024MB) - TEST RUNNER (10 REPEATS)

Name	Cores	Memory per Node	Secs 1 OD
TR - 1	8	1024.0 MB	1440
TR - 2	8	1024.0 MB	90
TR - 3	8	1024.0 MB	78
TR - 4	8	1024.0 MB	96
TR - 5	8	1024.0 MB	59
TR - 6	8	1024.0 MB	59
TR - 7	8	1024.0 MB	66
TR - 8	8	1024.0 MB	78
TR - 9	8	1024.0 MB	84
TR - 10	8	1024.0 MB	84

4.4 SETUP IV – HAVING BOTH SLAVE AND MASTER ON SAME HARDWARE

This setup involved having both Master and the Slave running on the same laptop. This setup provided the performance measurements like that of virtualized-like environment. Initially the slave memory was kept as that of the physical node. Later the slave memory was doubled to observe any difference in performance.

The results from Setup IV resulted in the following studies.

4.4.1 LOCALHOST SLAVE 8-CORE & 512MB

The slave on local host shared the 8 cores with master. Results of test runs of different task are quite improved. The slave is configured to use only 512Mb.

In the next run in same setup, this memory will be doubled.

TABLE 20: LOCALHOST SLAVE (512MB) - TEST RUNNER (DIFFERENT TASKS)

Name	Cores	Memory per Node	Secs
TR: scheduling-throughput	8	512.0 MB	96
TR: aggregate-by-key	8	512.0 MB	43
TR: aggregate-by-key-int	8	512.0 MB	38
TR: aggregate-by-key-naive	8	512.0 MB	66
TR: sort-by-key	8	512.0 MB	40
TR: sort-by-key-int	8	512.0 MB	40
TR: count	8	512.0 MB	39
TR: count-with-filter	8	512.0 MB	39

Cache warmup times were also observed

TABLE 21: LOCALHOST SLAVE (512MB) - TEST RUNNER (10 REPEATS)

Name	Cores	Memory per Node	Secs
TR - 1	8	512.0 MB	192
TR - 2	8	512.0 MB	52
TR - 3	8	512.0 MB	45
TR - 4	8	512.0 MB	59
TR - 5	8	512.0 MB	37
TR - 6	8	512.0 MB	37
TR - 7	8	512.0 MB	40
TR - 8	8	512.0 MB	46
TR - 9	8	512.0 MB	42
TR - 10	8	512.0 MB	40

4.4.2 LOCALHOST SLAVE 8-CORE & 1024MB

The memory is doubled from the previous test runs in the same setup. Though the results are not much different substantially.

TABLE 22: LOCALHOST SLAVE (1024MB)- TEST RUNNER (DIFFERENT TASKS)

Name	Cores	Memory per Node	Secs
TR: scheduling-throughput	8	1024.0 MB	120
TR: aggregate-by-key	8	1024.0 MB	43
TR: aggregate-by-key-int	8	1024.0 MB	38
TR: aggregate-by-key-naive	8	1024.0 MB	60
TR: sort-by-key	8	1024.0 MB	39
TR: sort-by-key-int	8	1024.0 MB	42
TR: count	8	1024.0 MB	39
TR: count-with-filter	8	1024.0 MB	39

The cache warmup time were also logged. They also were close to previous run in the same setup.

TABLE 23: LOCALHOST SLAVE (1024MB) - TEST RUNNER (10 REPEATS)

Name	Cores	Memory per Node	Secs
TR - 1	8	1024.0 MB	186
TR - 2	8	1024.0 MB	50
TR - 3	8	1024.0 MB	43
TR - 4	8	1024.0 MB	53
TR - 5	8	1024.0 MB	36
TR - 6	8	1024.0 MB	36
TR - 7	8	1024.0 MB	41
TR - 8	8	1024.0 MB	46
TR - 9	8	1024.0 MB	40
TR - 10	8	1024.0 MB	41

This concludes benchmarking of all physical arrangements and permutations of nodes across different clusters. Next the comparative study will be conducted among all suitable benchmark result counterparts.

CHAPTER 5: RESULTS STUDY

In this chapter, a comparative study of results is done from the above experimental results.

After running the benchmark in varying setup configurations, the following results are obtained. It is ensured that all meaningful combinations of SBC's are tried, and compared against nodes having similar specifications and are suitable counterparts.

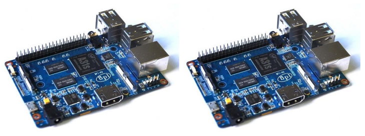
Also, the following tests are done having laptop as a master node, and SBC's as slave nodes. Later the tests are run over laptop, as being both master and slave.

5.1 STUDY I – PERFORMANCE COMPARISONS OF DIFFERENT CLUSTERS HAVING SAME SPECIFICATIONS NODES (CORES AND MEMORY)

This study involved comparisons among various clusters where the specifications are same, but the cluster is different. Same specifications mean that the number of cores and memory is same in across the compared clusters.

5.1.1 2 BP VS 2 RP – SAME SPECIFICATIONS: 8 (4+4) CORE, 1.0 (512+512) GB

From Setup I, the study of how same nodes in a cluster compare against each other.

VS


It is observed below that both Raspberry Pi and Banana Pi perform quite close to each other.

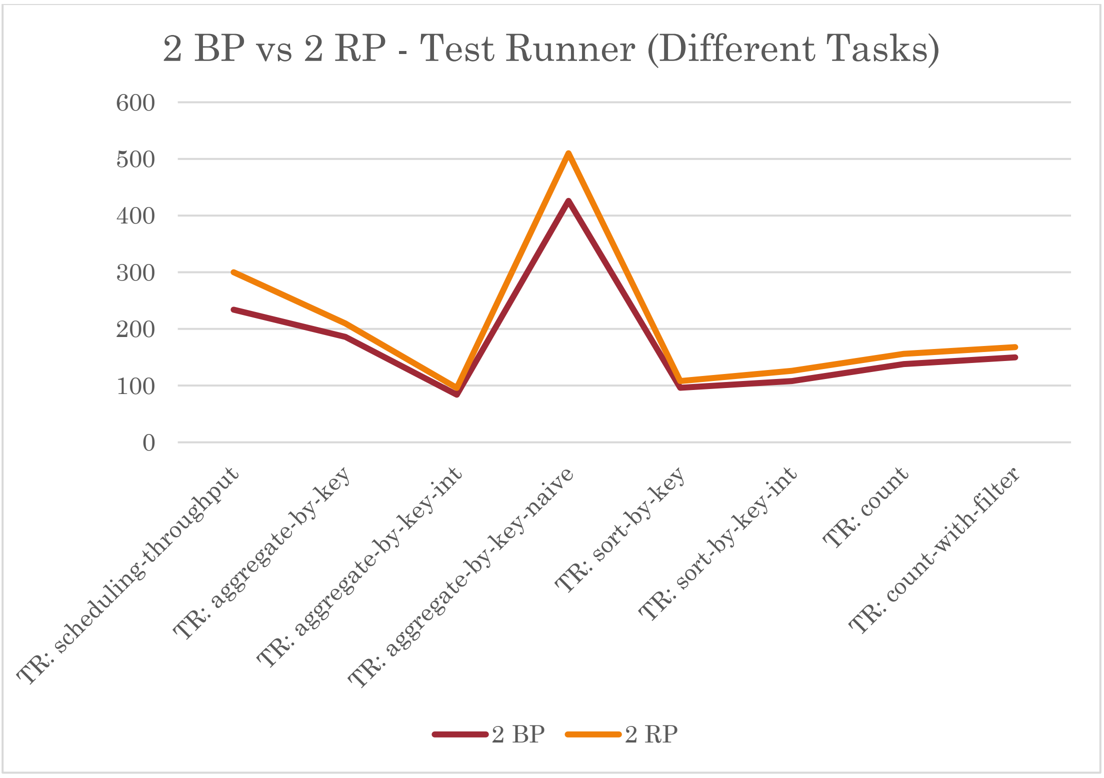


FIGURE 5: 2 BP VS 2 RP - TEST RUNNER (DIFFERENT TASKS)

Similarly, in the cache warmup runs, both demonstrate close enough results.

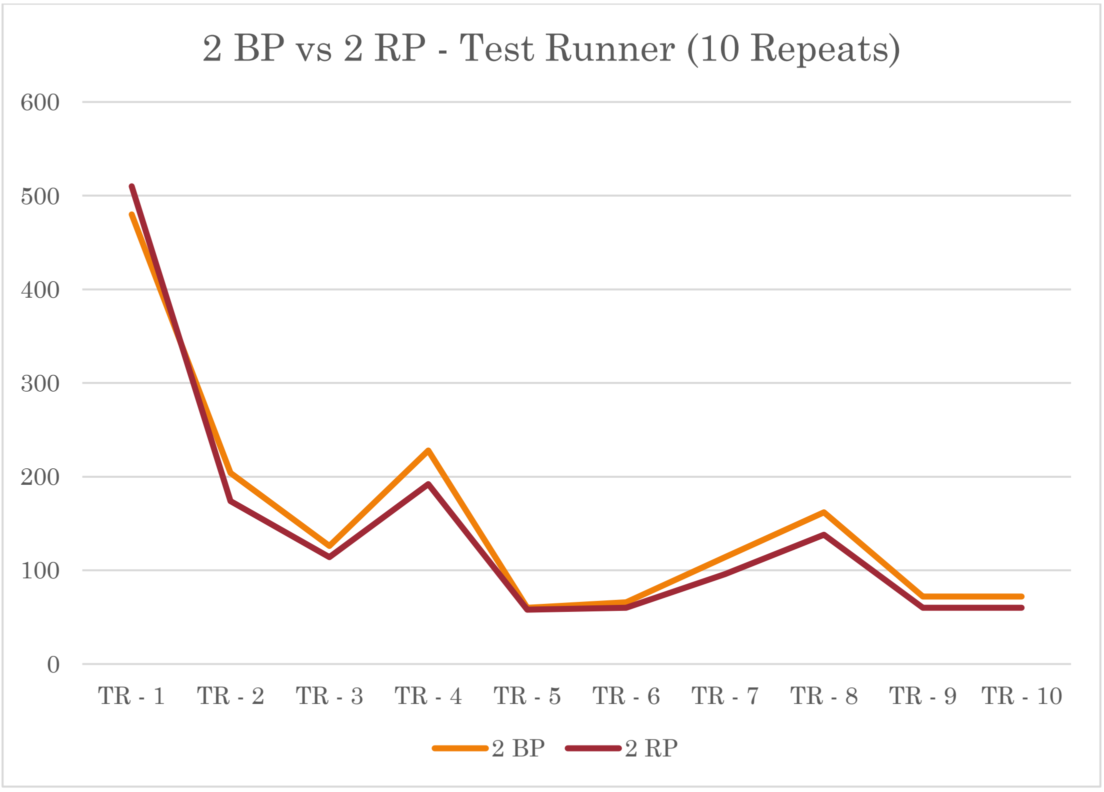
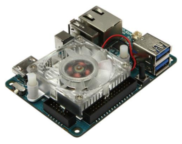


FIGURE 6: 2 BP VS 2 RP - TEST RUNNER (10 REPEATS)

5.1.2 2 RP VS 1 OD – SAME SPECIFICATIONS: 8 (4+4) CORE, 1.0 (512+512) GB

From Setup I, and Setup III, the study of how same nodes in a cluster compare against a cluster having single powerful node. In this case, it is two Raspberry Pi against one Odroid.

VS



As seen below that there is substantial difference in the way single Odroid performed compared to two Raspberry Pi.

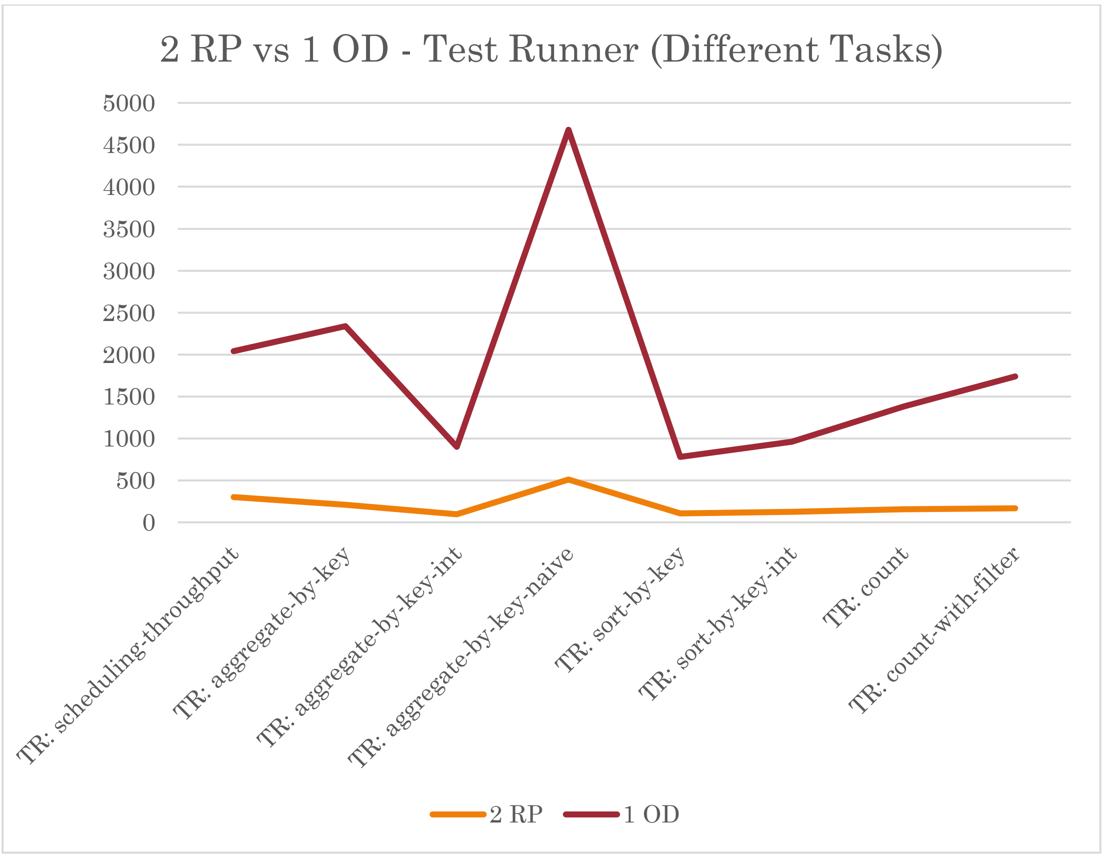


FIGURE 7: 2 RP VS 1 OD - TEST RUNNER (DIFFERENT TASKS)

Also, the cache for Odroid started at very high, but later catchup with the Raspberry Pi.

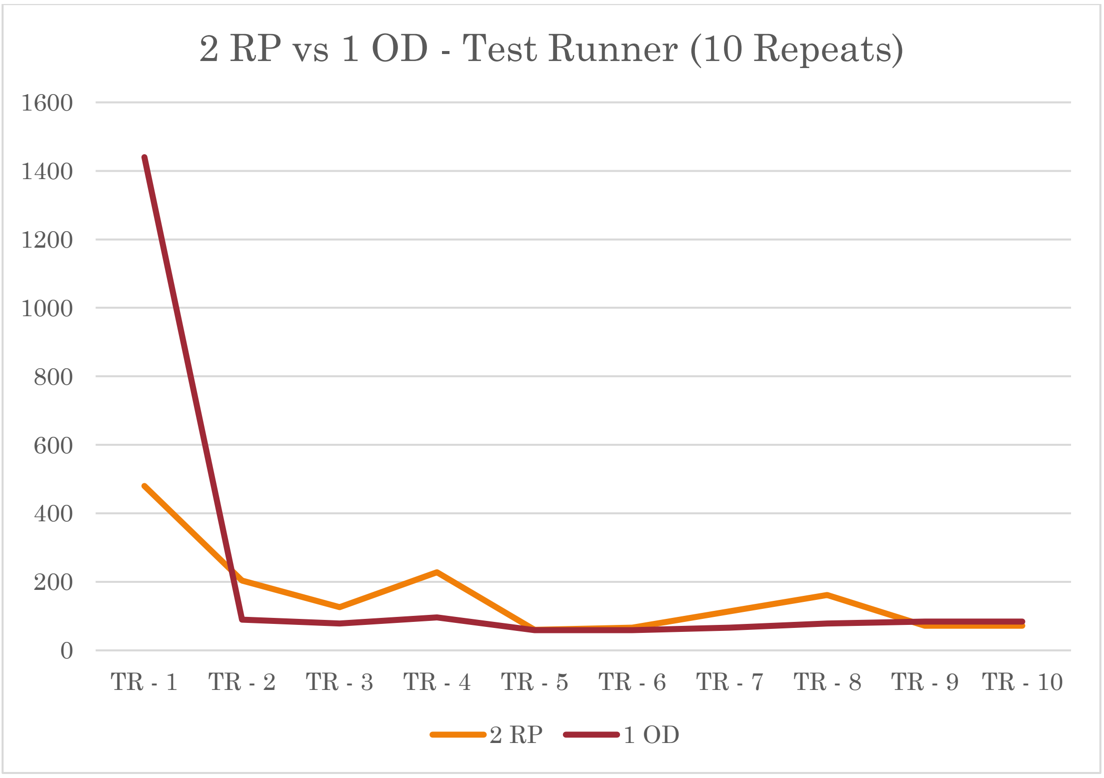
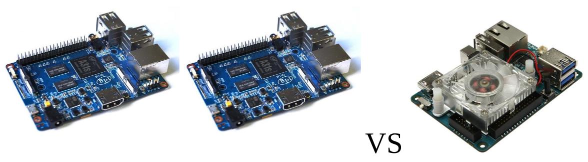


FIGURE 8: 2 RP VS 1 OD - TEST RUNNER (10 REPEATS)

5.1.3 2 BP VS 1 OD – SAME SPECIFICATIONS: 8 (4+4) CORE, 1.0 (512+512) GB

From Setup I, and Setup III, the study of how same nodes in a cluster compare against a cluster having single powerful node. In this case, it is two Banana Pi against one Odroid.



It is observed that the results are quite like what was observed in the same study earlier.

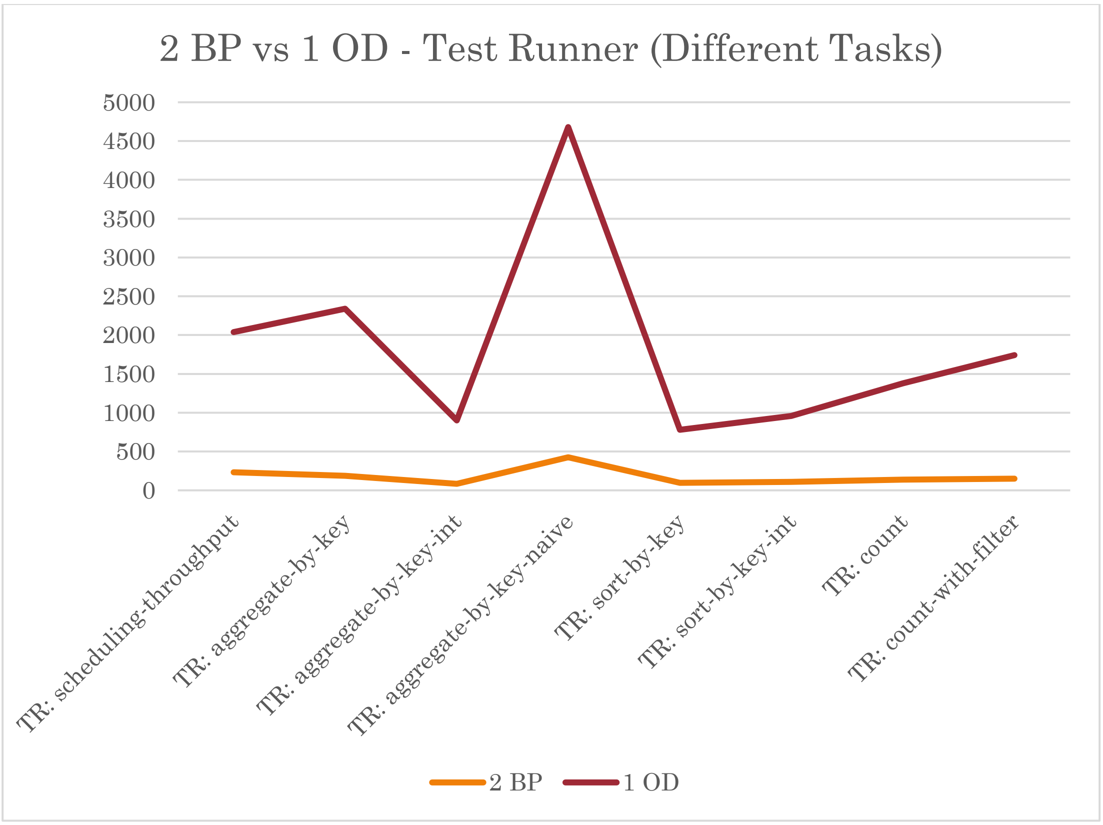


FIGURE 9: 2 BP VS 1 OD - TEST RUNNER (DIFFERENT TASKS)

Also, the cache runs times for Odroid started like previous run, but later did a good catchup.

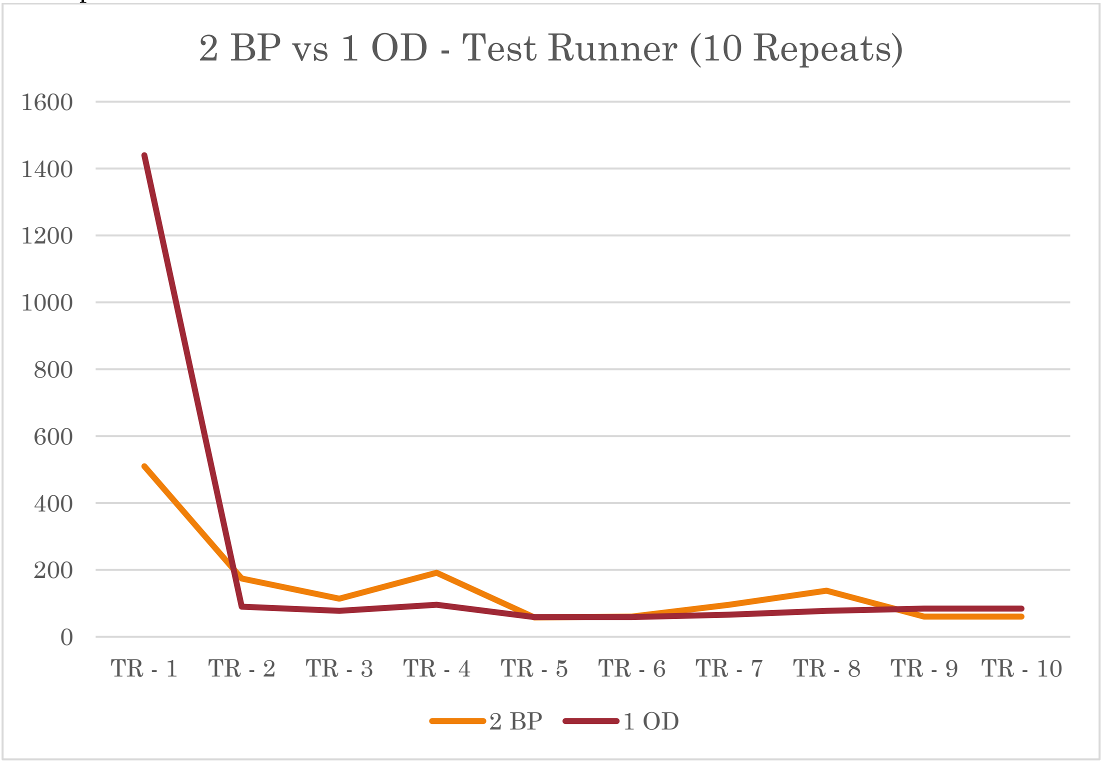


FIGURE 10: 2 BP VS 1 OD - TEST RUNNER (10 REPEATS)

5.2 STUDY II – PERFORMANCE COMPARISONS OF CLUSTERS WITH SAME NUMBER OF CORES

5.2.1 1 BP & 1 RP & 1 OD vs 2 OD – same cores 16 (4+4+8 vs 8+8) cores, 1.5 vs 1.0 (512+512+512 vs 512+512) Gb

From Setup I and Setup II, the performance of both clusters setups is measured, keeping the number of cores same.

In the following test, there are 16 cores in both clusters, whereas the memory is different by half Gb. Though on Odroid side, there is a powerhouse with both together, however its observed that when the Odroid used with other low compute power nodes, the cluster performed better significantly.

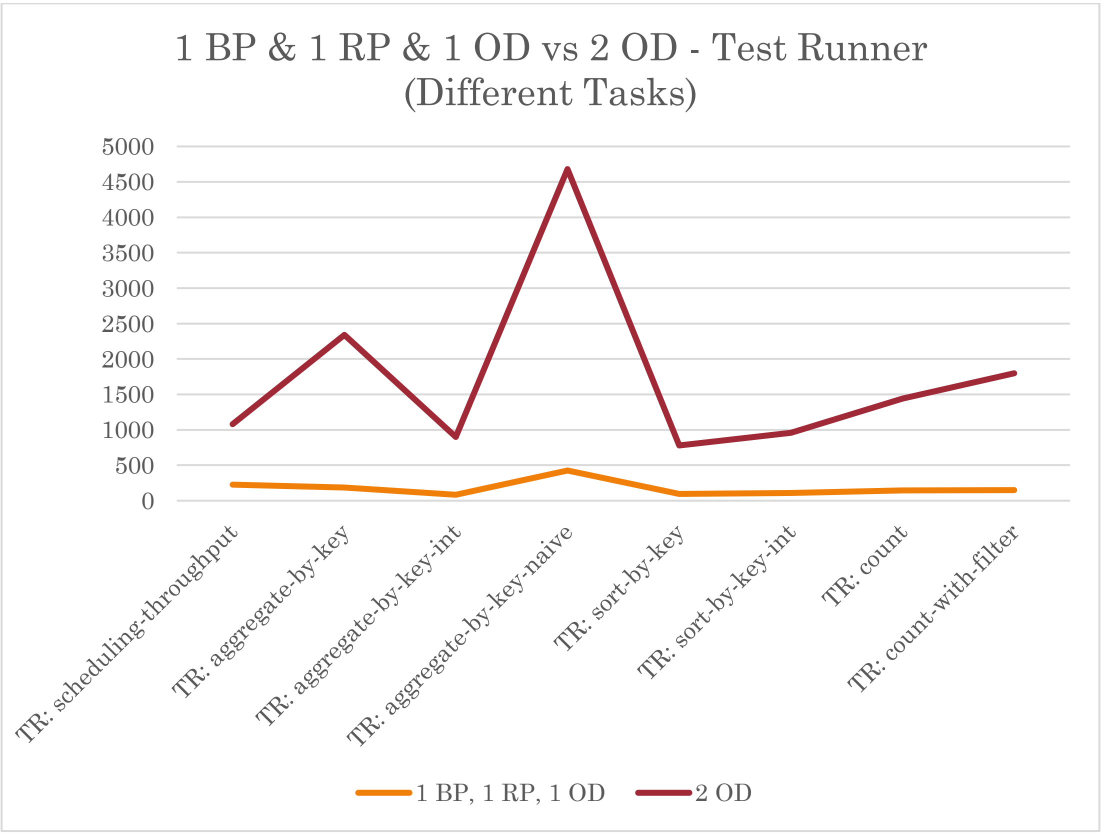


Figure 11: 1 BP & 1 RP & 1 OD vs 2 OD - Test Runner (Different Tasks)

Cache warmup times started differently, but later reached close to each other.

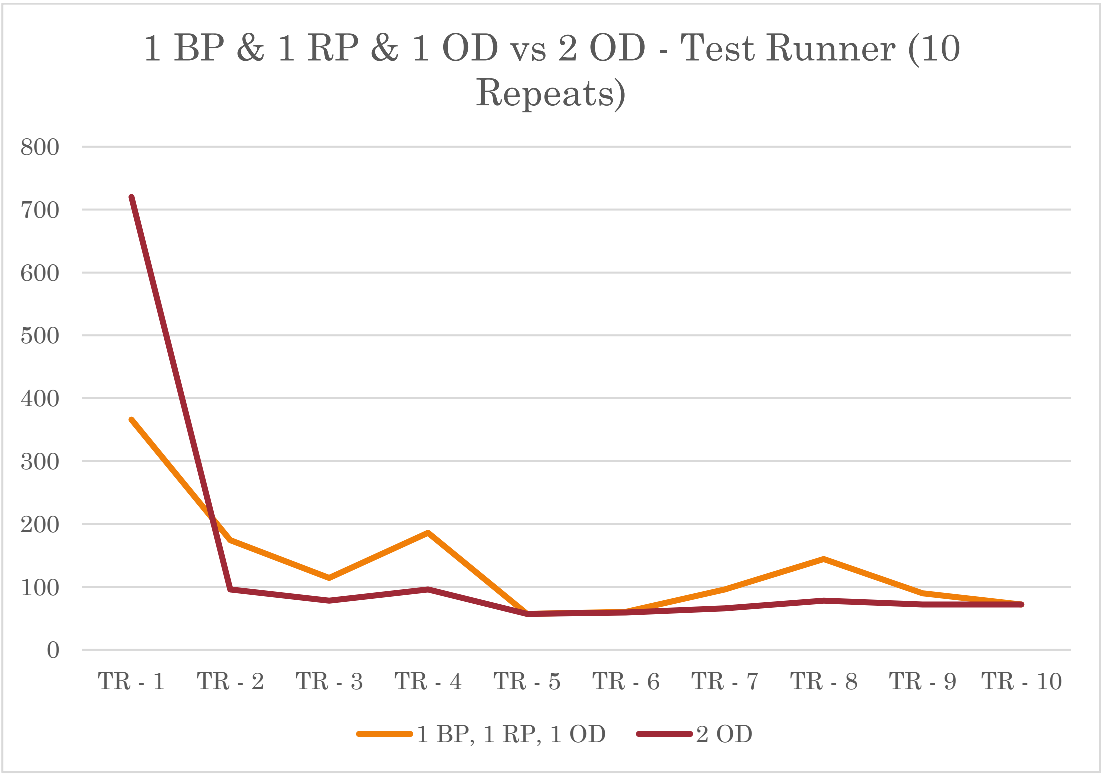


FIGURE 12: 1 BP & 1 RP & 1 OD VS 2 OD - TEST RUNNER (10 REPEATS)

5.2.2 *2 BP VS 2 RP VS 1 OD – SAME CORES 8 (4+4 VS 8) CORES, 1.0 VS 0.5 (512+512 VS 512) GB*

Setup 1 and Setup III provided the study of three-way comparison of cluster performance.

Three-way comparison is made, where both Raspberry Pi are in one cluster, both Banana Pi in another cluster, and Odroid is alone in yet another cluster, with a difference of half a Gb more with Odroid.

The results show that Raspberry Pi and Banana Pi cluster showed consistent results. Whereas Odroid performed poorly.

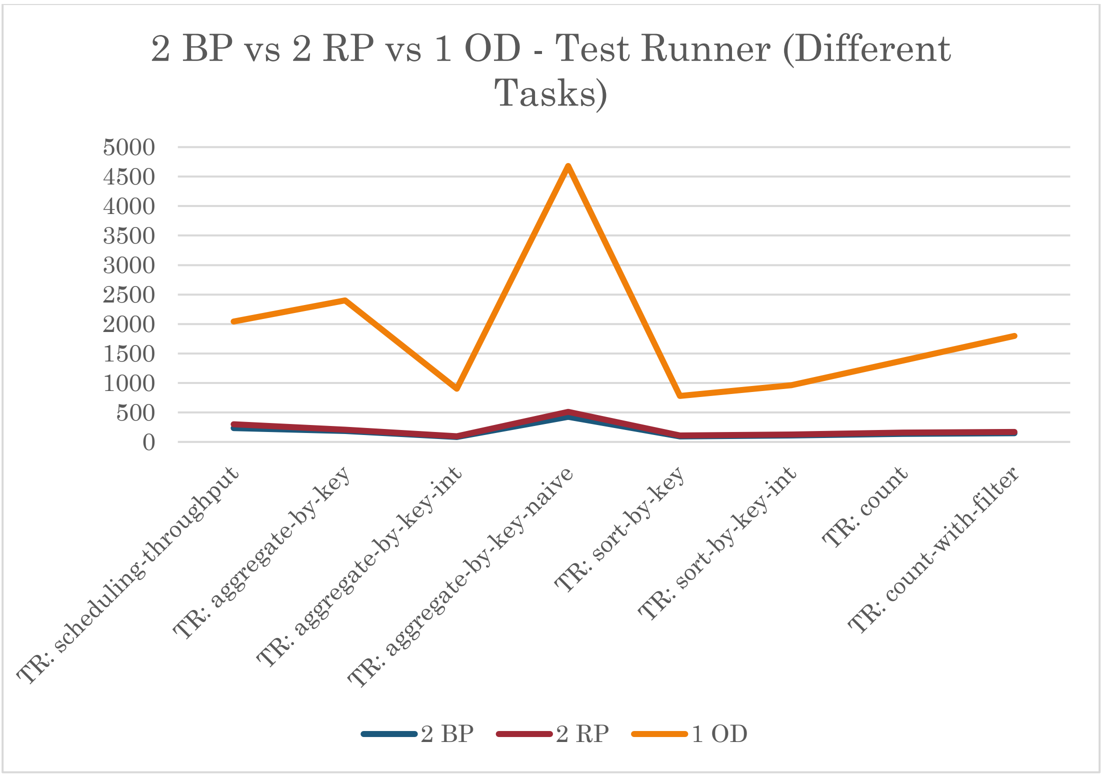


FIGURE 13: 2 BP VS 2 RP VS 1 OD VS - TEST RUNNER (DIFFERENT TASKS)

Odroid was consistent immediately after first run, others were close to each other.

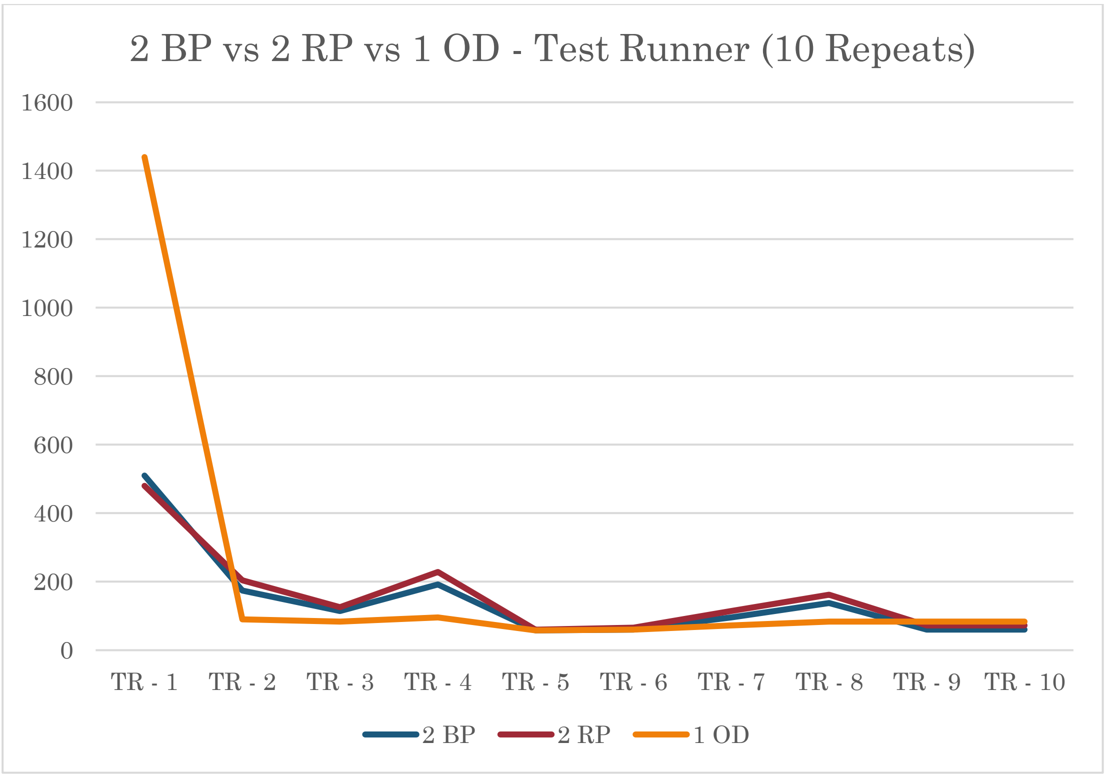


FIGURE 14: 2 BP VS 2 RP VS 1 OD - TEST RUNNER (10 REPEATS)

5.2.3 2 BP & 2 RP & 2 OD vs 1 BP & 1 RP & 1 OD

Results from within Setup II provided the following study. First the nodes were kept single each. Later the nodes are doubled. To achieve accuracy, three iterations of results were obtained for the one each.

VS

In this study, a full-size cluster of six nodes performance is compared against half the size, three nodes cluster. For half the size, again three iterations were used. Very surprisingly, both the results are too close to each other.

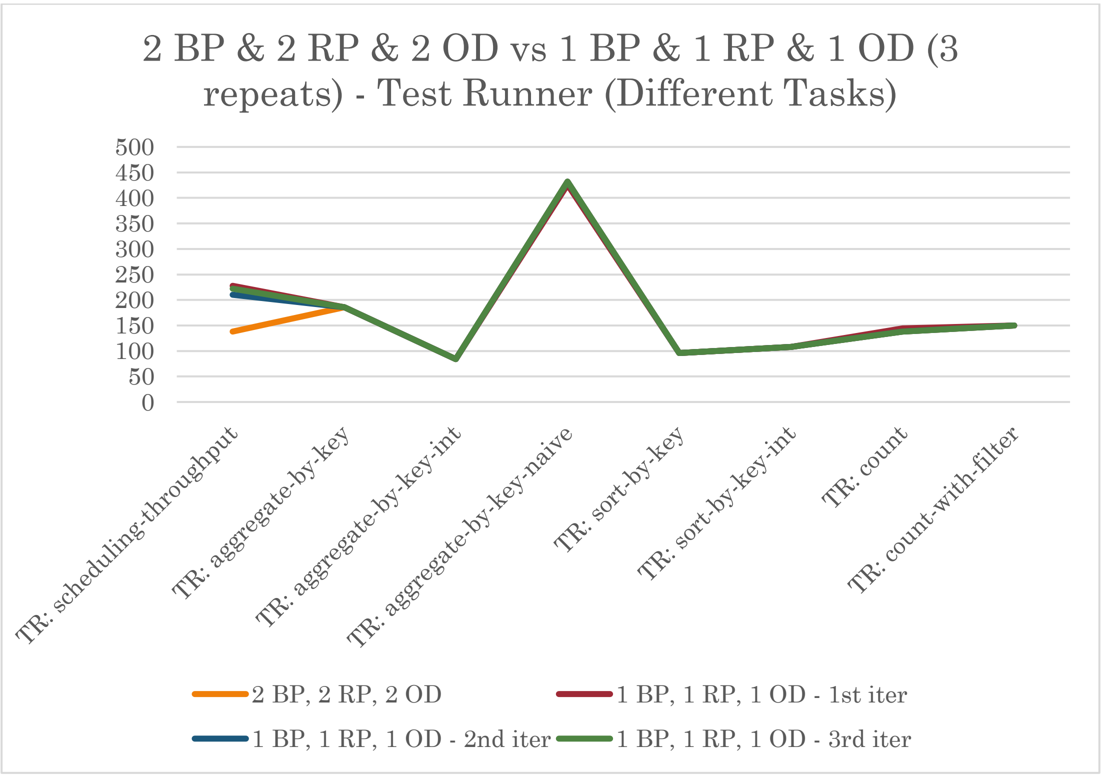


FIGURE 15: 2 BP & 2 RP & 2 OD VS 1 BP & 1 RP & 1 OD (3 REPEATS) - TEST RUNNER (DIFFERENT TASKS)

The cache warmups started on a different note, however they became close later.

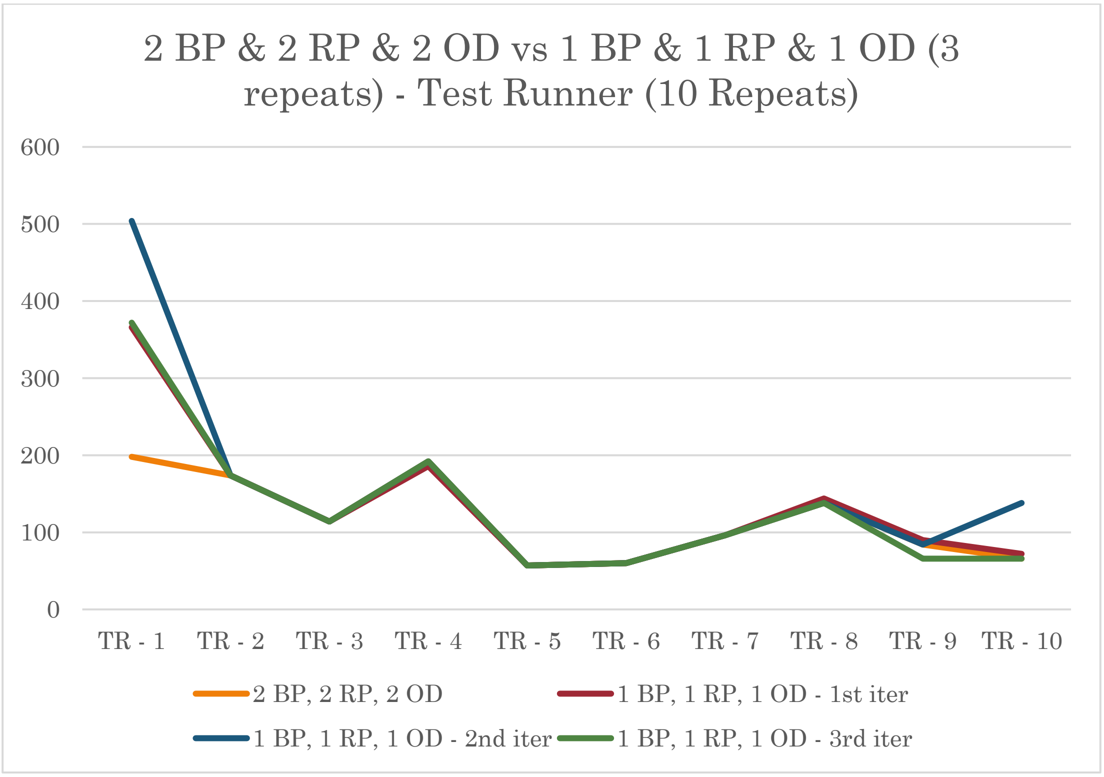


FIGURE 16: 2 BP & 2 RP & 2 OD VS 1 BP & 1 RP & 1 OD (3 REPEATS) - TEST RUNNER (10 REPEATS)

5.3 STUDY III – HAVING BOTH MASTER AND SLAVE NODE ON SAME HOST HARWARE

5.3.1 LOCALHOST SLAVE (1024MB) VS LOCALHOST SLAVE (512MB)

For Setup IV, the memory is doubled and performance is measured.

Running on the same hardware provided liberty from latencies like network and bandwidth. It provided a virtualized-like comparison, where both the slave and master are running independently on the same hardware, and communicating together with least latencies.

Interesting to note that for the same task and same dataset, both runs; with different memory size for slave, performed quite similar.

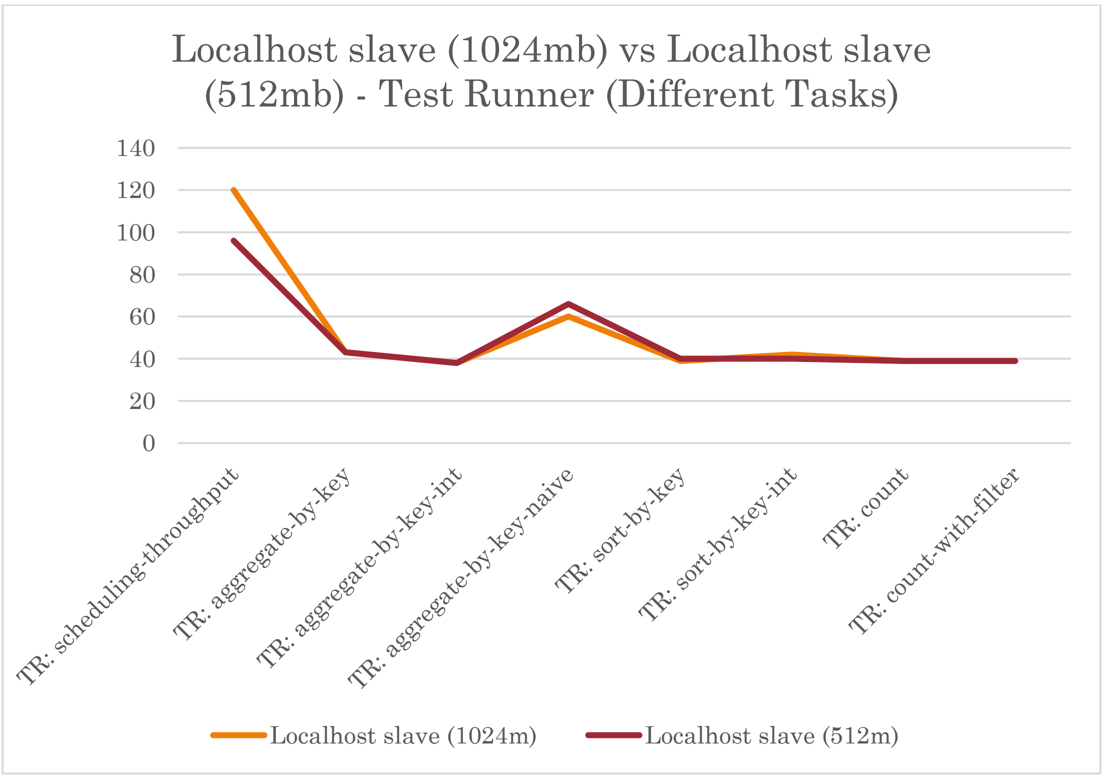


FIGURE 17: LOCALHOST SLAVE (1024MB) VS LOCALHOST SLAVE (512MB) - TEST RUNNER (DIFFERENT TASKS)

Also, looking at the cache warmup times, they are also again very close.

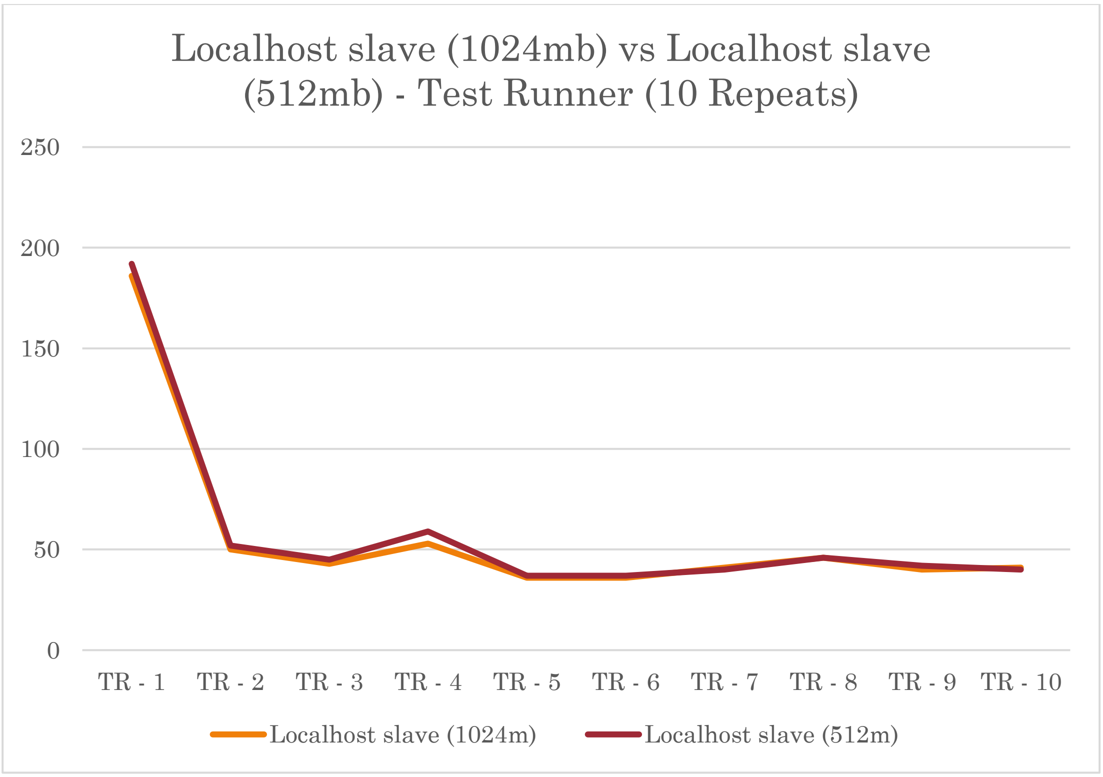


FIGURE 18: LOCALHOST SLAVE (1024MB) VS LOCALHOST SLAVE (512MB) - TEST RUNNER (10 REPEATS)

5.4 STUDY IV – PERFORMANCE OF RUNNING ON SAME HOST HARWARE COMPARED WITH RUNNING ON CLUSTER

After having results from Setup III and Setup IV, the following study explains how the difference in performance is compared against each other.

5.4.1 LOCALHOST SLAVE (512MB) VS 2 BP & 2 RP & 2 OD

vs

A comparison is made against running on same hardware, providing a virtualized-like environment versus, running over a network of physical nodes.

It shows what is obvious that running over the network does lags a lot in performance

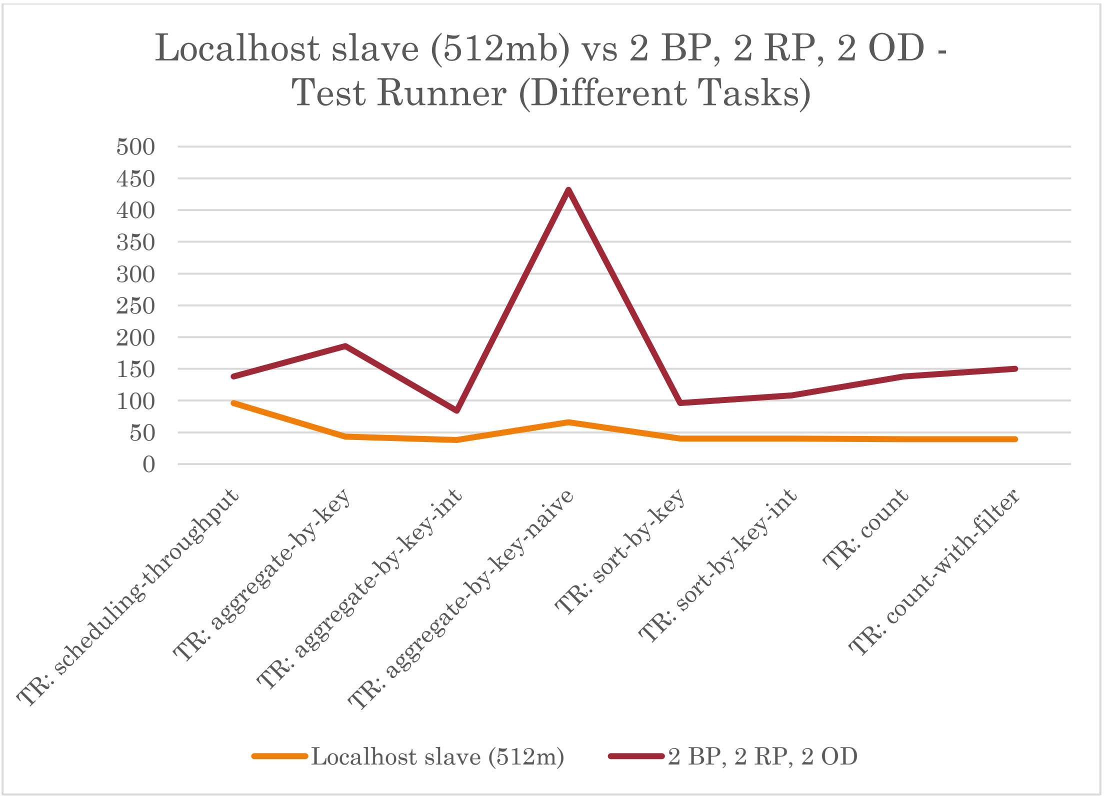


FIGURE 19: LOCALHOST SLAVE (512MB) VS 2 BP & 2 RP & 2 OD - TEST RUNNER (DIFFERENT TASKS)

Similarly, when cache warmup times are considered, the cluster over the network showed degraded performance.

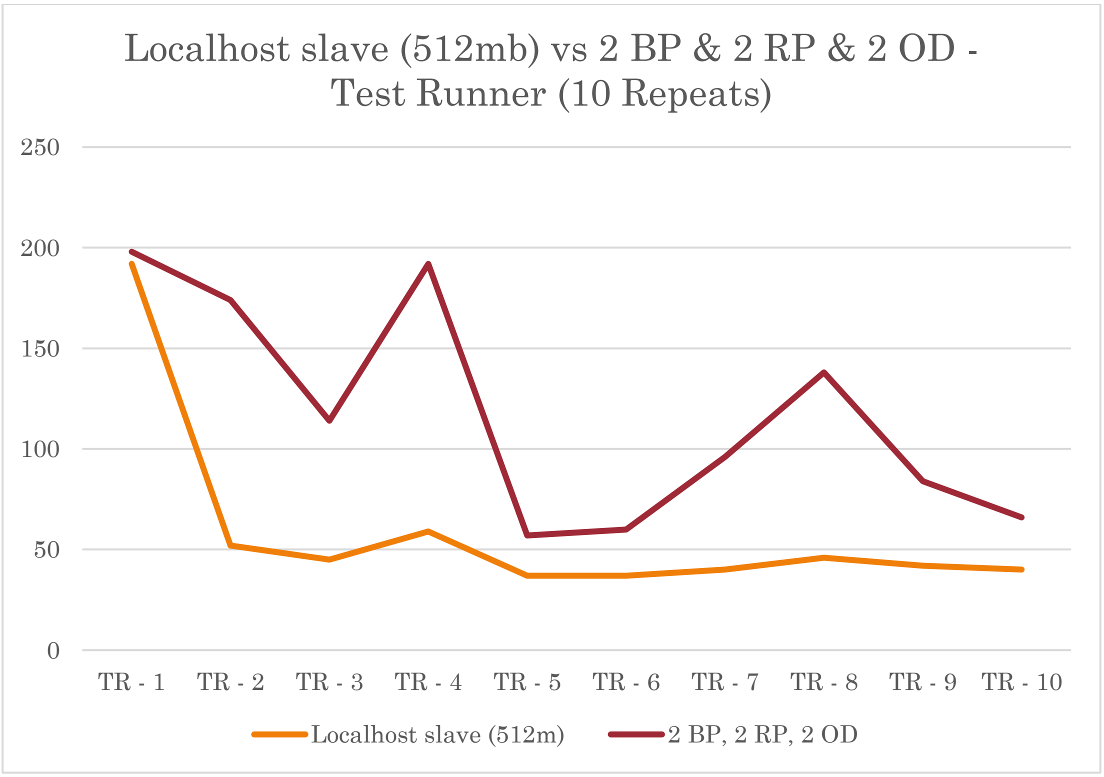


FIGURE 20: LOCALHOST SLAVE (512MB) VS 2 BP & 2 RP & 2 OD - TEST RUNNER (10 REPEATS)

5.4.2 *LOCALHOST SLAVE (1024MB) VS 2 BP & 2 RP & 2 OD*

Setup III and Setup IV, with double memory, provided results for the comparison against each other.

VS

The memory from the previous run is doubled in here for the slave to run over the laptop.

Again, the performance of the networked cluster is lesser than that of running over the same hardware. However, near the end, they came close to each other. Also for integer indexed tasks, the results are very close.

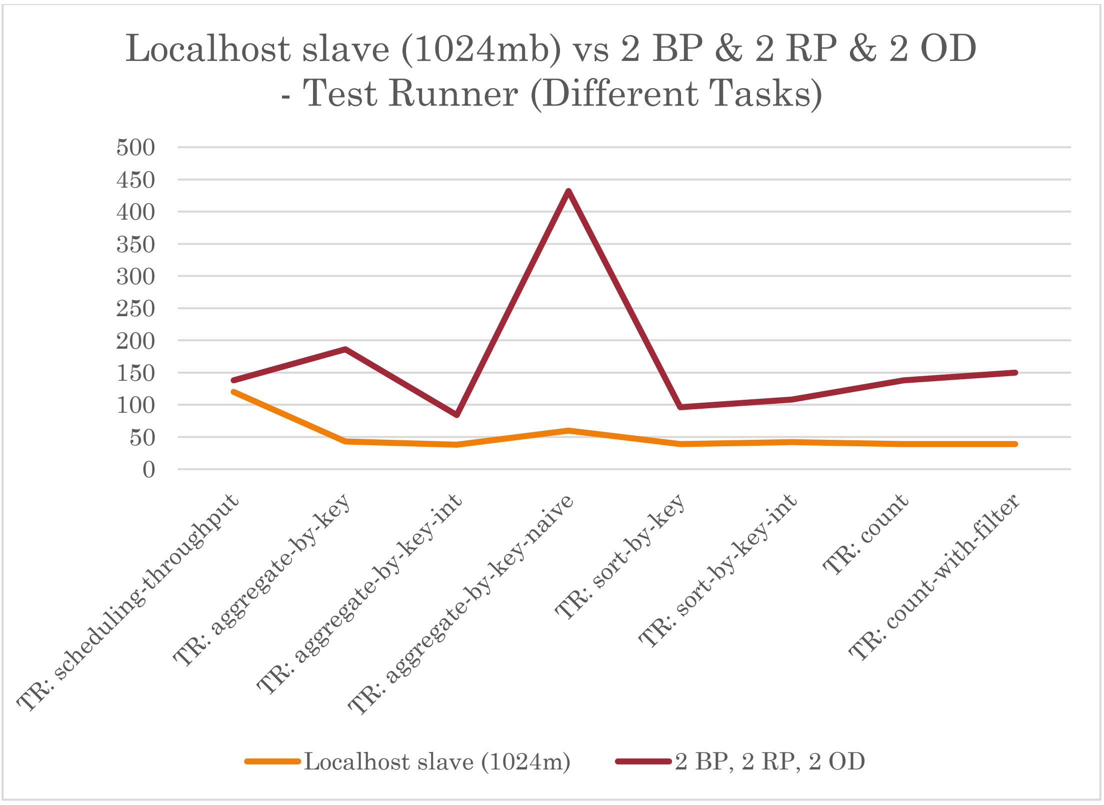


FIGURE 21: LOCALHOST SLAVE (1024MB) VS 2 BP, 2 RP, 2 OD - TEST RUNNER (DIFFERENT TASKS)

Moreover, the cache warmup times also showed similar behavior. The performance over network was degraded compared to the same hardware. However, nearing the end of the run, the warmup times came closer to each other.

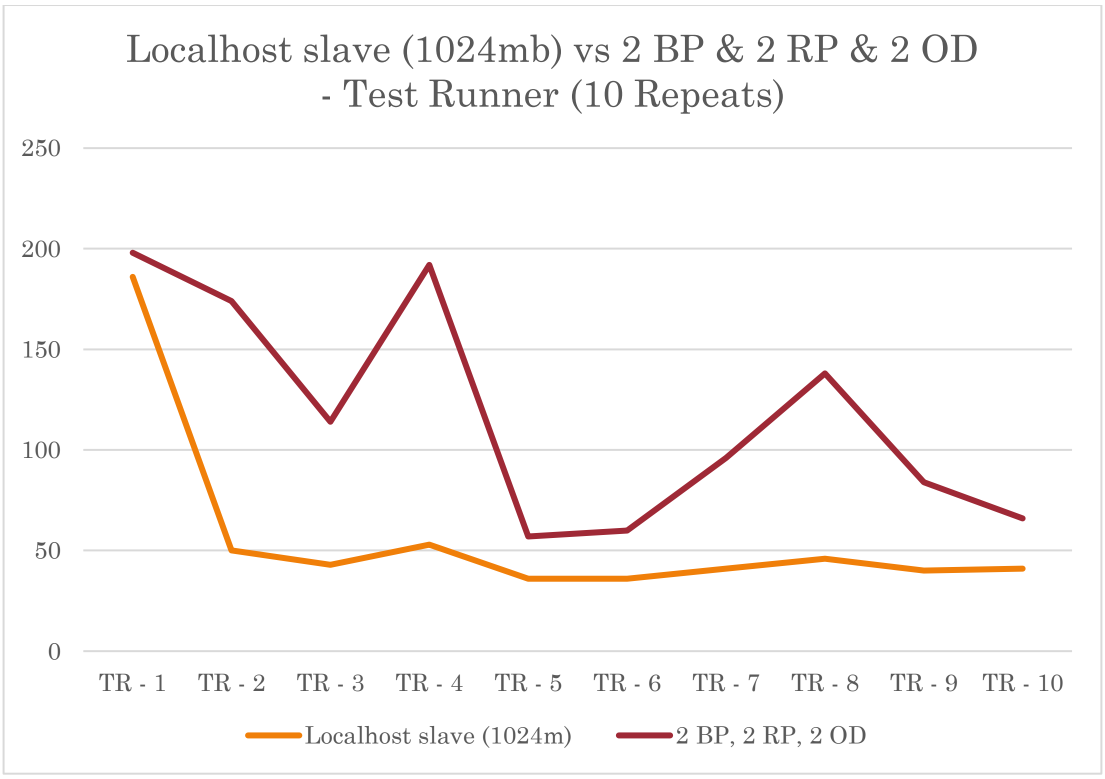


FIGURE 22: LOCALHOST SLAVE (1024MB) VS 2 BP & 2 RP & 2 OD - TEST RUNNER (10 REPEATS)

This concludes comparative review among suitable benchmark results only. Further ahead a comparative review across the board, i.e. across all node permutations, for all applications is considered. The application comparisons are split across two categories, i.e. different application test runs and iterative cache warmup runs.

CHAPTER 6: RESULTS REVIEW

Once the comparative studies conducted across suitable counterparts, it is required to look further into overall results across all permutations of nodes, primarily over two classes of benchmark runs, the test runs for tasks, and iterative test runs for cache repeats.

6.1 SBCs PERMUTATIONS – TEST RUNNER TASKS

It explains how the same task performed across different SBC's permutations and over the laptop.

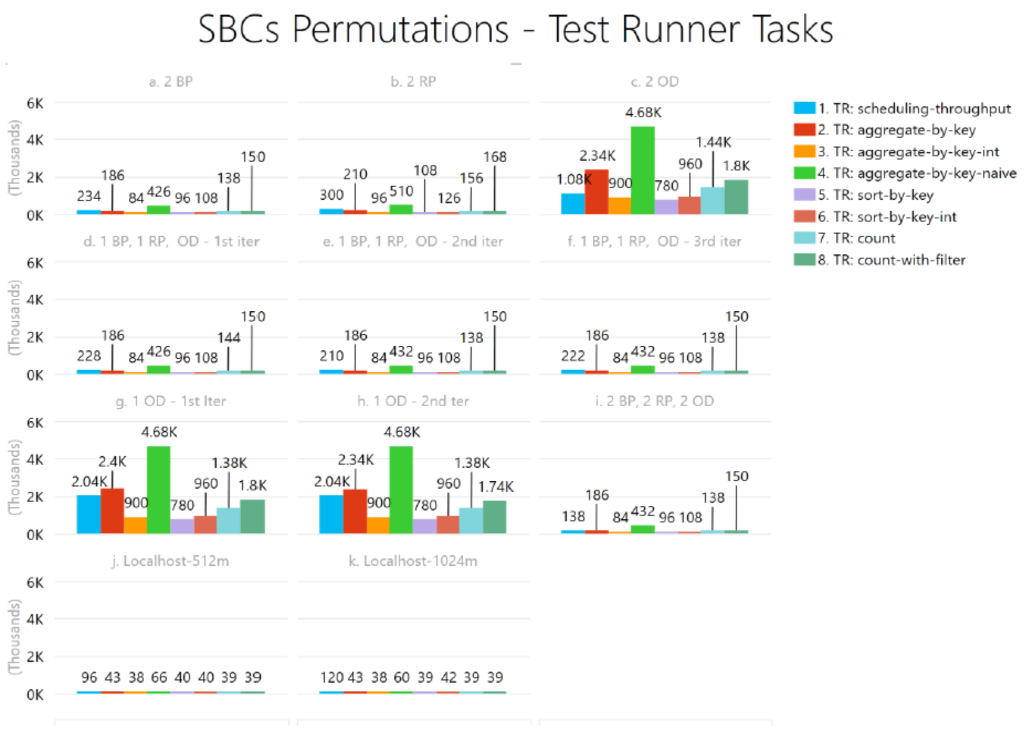


FIGURE 23: SBCS PERMUTATIONS – TEST RUNNER TASKS (THE LOWER THE BETTER) – SEE APPENDIX 3 FOR DATA TABLE

The tasks are kept in order in which they are executed on the cluster. Total of 8 tasks are executed.

The 2nd task, runs over the dataset which is half the size of the 3rd task, where the latter is utilizing double partition as well. However, the performance of the 3rd task is quite improved than the 2nd one.

Here the first row is about pairs of SBC's. The second row is about one-each SBC, running all tasks, in 3 iterations. The third row first two columns explain how the Odroid's as a single node ran the tasks, which is comparable to first two columns of the first row, because they have similar specifications. The last column of second row shows how all the six nodes executed the tasks. The last column of both second row and third row, explains whether doubling the nodes have any significant impact on the cluster performance. The last row explains how the same tasks were executed over the laptop being both the slave and the master, where the slave is having 512mb and 1024mb memory. It shows that even when the tasks are run over the laptop, which is the most powerful in the cluster, the performance gain is not in by-folds of double digits, even when both master and slave are running on the same node.

6.2 SBCs PERMUTATIONS – TEST RUNNER REPEATS

It explains the performance of Cache Warmups on the SBC's permutations.

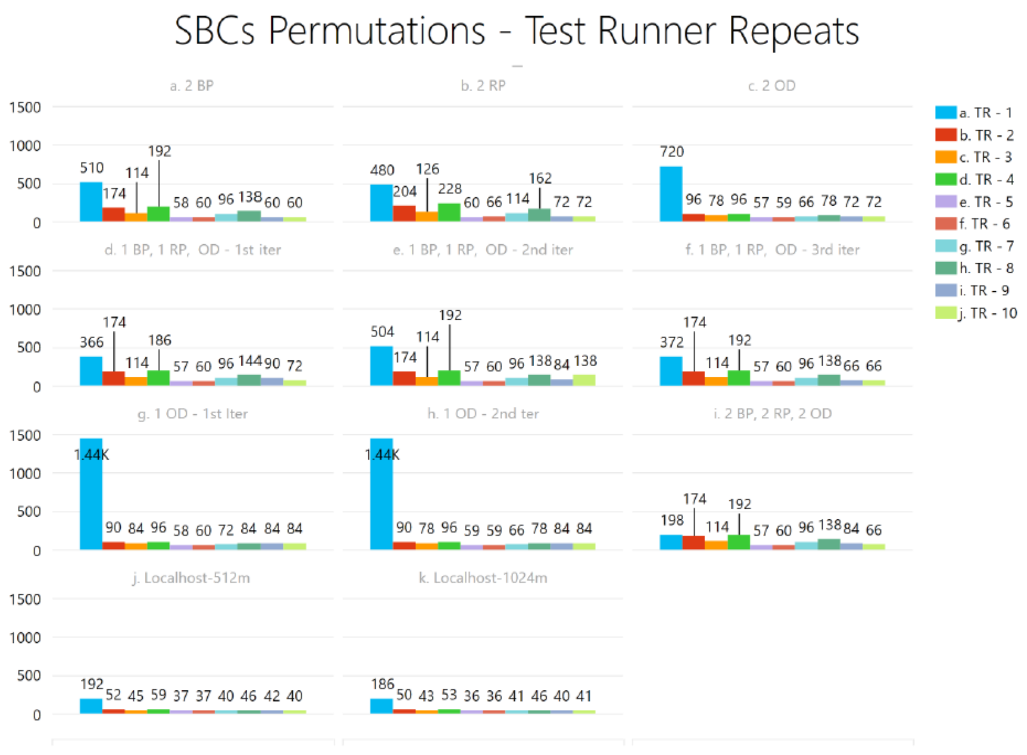


FIGURE 24: SBCS PERMUTATIONS – TEST RUNNER REPEATS (THE LOWER THE BETTER) – SEE APPENDIX 3 FOR DATA TABLE

The Cache warmups are seen here, show significant performance gain, when Caches are warmed up.

The 1st warmup attempt is peaked as expected. However, it gets better starting from the 2nd attempt. And later with each iteration it tries to stay consistent.

The first row here, each of the pair of SBC's demonstrated different than that of individual tasks. It is observed that the Odroid pair, though least performing in the individual tasks; however, in cache warmup, quickly showed improved execution times than the other pairs, relatively.

Again, once the caches are warmed up, doubling the nodes did not show significant performance gain. It is observable from the second and third row of the last column. Also, performance of single Odroid, compared against a pair of Odroid, doesn't show significant performance gains.

Moreover, in a mix of nodes, the cache warmup performance fluctuates little bit, compared to the same pair of nodes. The last row, highlights the performance times when there is both master and slave on the same node, i.e. the laptop.

6.3 TEST RUNNER TASKS – SBCS PERMUTATIONS

It explains how the individual task, performed alongside each other on different permutations of SBC.

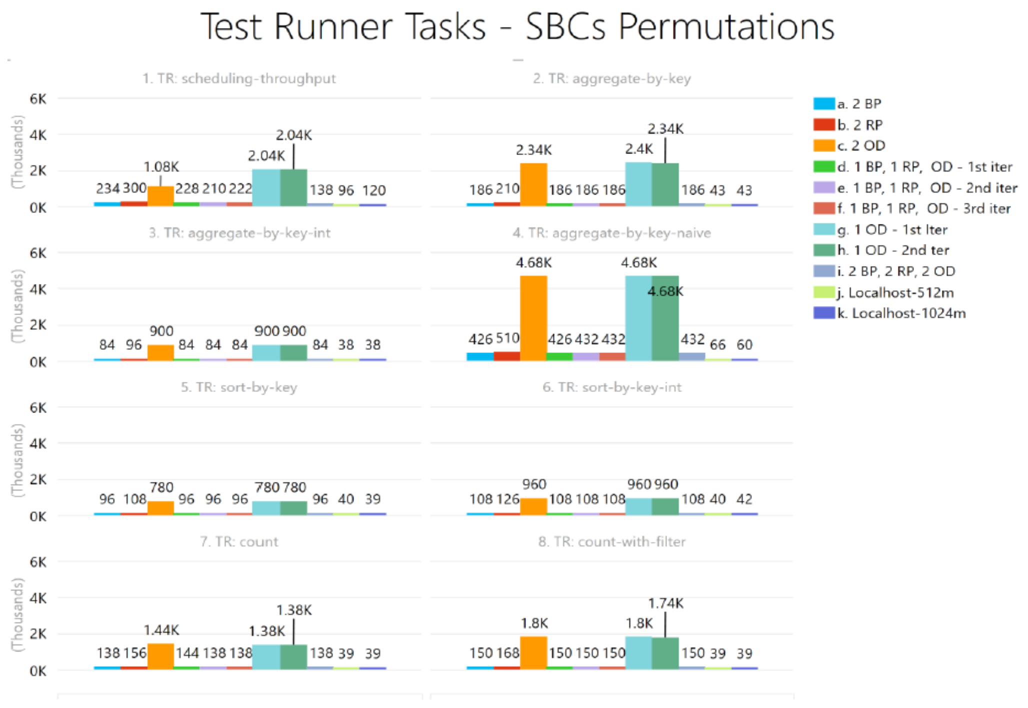


FIGURE 25: TEST RUNNER TASKS – SBCS PERMUTATIONS (THE LOWER THE BETTER) – SEE APPENDIX 3 FOR DATA TABLE

The costliest tasks across all tasks is the first task. The last two bars on each plot is about the laptop serving as both master and slave. The first three bars are from pair of each SBC type. The bars showing Odroid’s are the least performing on all tasks, i.e. the third, seventh and eighth, where third is the pair of Odroid’s and seventh and eights are single Odroid. The ninth bar is the execution of all the six nodes in unison.

The 2nd task, and the 3rd task are different in size of data points, where the 3rd is double the size, and partitions, compared to 2nd task. However, the performance of 3rd is relatively better than 2nd task across the board.

From fourth, fifth and sixth bar from the left, i.e. each of single SBC together, when compared against the ninth bar, which is double the nodes, it is obvious from these bars that performance is either close to each other, as in 1st task, or, almost same in latter tasks, thus doubling the nodes, doesn’t result in performance gain.

6.4 TEST RUNNER REPEATS – SBCS PERMUTATIONS

It explains how the Cache Warmup performed for each SBC permutation.

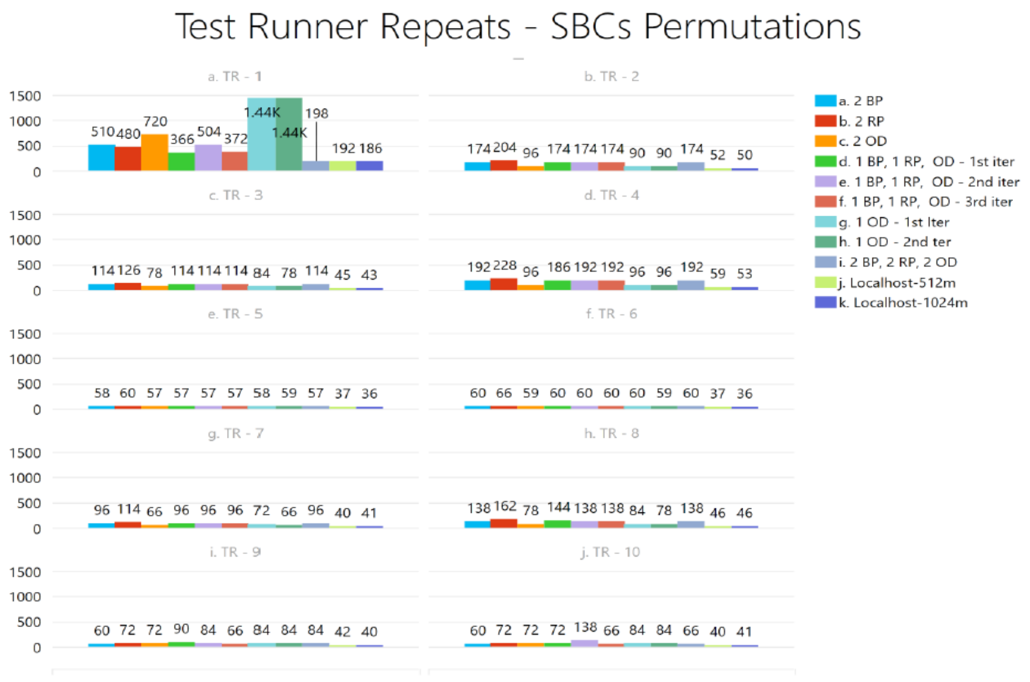


FIGURE 26: TEST RUNNER REPEATS – SBCS PERMUTATIONS (THE LOWER THE BETTER) – SEE APPENDIX 3 FOR DATA TABLE

The first plot, corresponds to the first attempt for warming up the Cache. As it is evident that the first test run did not performed well against the latter test runs.

Moreover, it is also observed that after the 2nd and 3rd run, the test runs, tries to stay consistent. Later, for the 5th and 6th run, all times stays within double digits.

It is also observed that once the cache is warmed up good, i.e. the 5th and 6th run, most of the SBC’s depict quite similar time consumptions. This is interesting, because the SBC’s permutations are quite different from each other, including, the last two bars belonging to the laptop acting both as a master and a slave node.

The sixth bar from the left of each plot, have times from each type of SBC, and the ninth bar have times from double the nodes, from that of sixth. However, the time taken by both is quite the same. Thus, doubling the number of nodes here, did not impart that much of performance gain.

The last two bars of each plot, belongs to the laptop serving as both master and slave, and is quite powerful comparatively. However, it is observed that both the last two started well, but when the cache is warmed up, the performance of SBC's came closer to that of the laptops.

That concludes comparative review of all benchmark results, across every node permutations in all clusters and applications running over them.

CHAPTER 7: CONTRIBUTION

When setting up a cluster in commercial settings for a specific problem, specially, where both cost and power usage is of significance, the cluster of SBC’s can be utilized for the compute intensive problem. However, adding nodes into the cluster, may not always result in performance improvement, though increasing power consumption and cost.

The obvious choice for setting up such cluster, is to procure virtualized nodes available over the cloud and consume them as worker nodes. However, that arguably proves to be not a cost-effective solution, provided a cluster of SBC’s can be setup. Thus, fulfilling the requirement at hand, involving a specific input dataset and for running the tasks. Over the cloud, any idle-time for even a low-tier node, also incurs cost with it, whereas a low-cost SBC may incur cost only once, arguably, and even that is not more than double digits of dollars. As observed above as well, that the better performing SBC’s are fan-less, therefore having such cluster would not only require lesser power, but also lesser cooling requirements.

A Data Analytics problem, involving training and re-tuning of a specific machine learning model, intermittently, is no doubt a compute intensive task. It is viable to execute it off-the-cluster, as it would enormous computational resources to build the model. However, when the model is ready, and the model can start churning the data to provide predictions, discussed cluster of SBC’s can be used. In this case, the cluster will be occupied mostly with distributing the job and data, and later collating results; in other words, lift-and-shift of data, it is imperative to use a cluster of SBC’s for the prediction.

The volume of data and consumed in the explained cluster setup, reached over a million data points. However, even for such a large volume, no external Disk Storage was attached to any of the node in the above cluster. Since the nodes can be equipped with Gigabytes of storage onboard, the cluster can handle up-to a Terabyte of data, with carefully increasing the number of nodes. Though, when the data requirements exceed the storage capacity, additional external storage can be attached, at the cost of performance, slightly, because once the cache is warmed up, it should perform optimal.

Moreover, during procurement, when faced with a choice of infrastructure for setting up a cluster, the more powerful and expensive nodes are not always the optimal choice for the dataset and problem at hand, as observed. In fact, nodes with lesser power, but a few more of them, are a better choice for the cluster running distributed application.

SBC's benchmarking had been done on individual components of the SBC, i.e. I/O, Network and CPU Stress. Though such benchmarks do elaborate on how a component of hardware would perform, however they lack in explaining the end-to-end task execution performance while running under a cluster computing framework, like Apache Spark.

When the above benchmark was executed over the cluster of SBC's, performance is measured for a complete lifecycle of the task. It involved task and data distribution, and later collation, involving the onboard components of the SBC's and their contributions resulting in a compounded impact on performance. This explained the impact of Network latency to Storage I/O, from CPU Stress to Cache Latency. Thus, it gives a wider picture in to what to expect from the cluster.

CHAPTER 8: FUTURE WORK

The cluster setup earlier, involved nodes from multiple vendors and of different types, to obtain signature impact of each of them, compounded together in the performance benchmark. It would be imperative to have more variety in the cluster. By variety, it can include SBC's from other vendors, having the closest attributes to the best performing SBC's as per the above conclusions. This would help build insights into other SBC's impact in a cluster.

Both Apache Spark, the cluster computing framework, and spark-perf, the performance benchmark is open source, with the earlier being very actively enhanced. Therefore, it would be better to upgrade the Apache Spark and the spark-perf on the cluster to the newest most stable version. The current setup discussed, is only making use of the functions such as shuffling, sorting, counting and scheduling, which is at the core of Apache Spark. However, any future enhancements such as code optimization at the core; upgrading the framework and the benchmark may result in better cluster performance. The dataset consumed in the performance benchmark is a generated one, therefore it would be interesting to consume other publicly available data corpuses, and benchmark over them.

Such cluster can be enhanced to measure exact watts consumed by each node for an end-to-end task cycle, which would give a valuable insight into the power consumed by the cluster. This would provide detailed power profile of the nodes and the cluster, and help in reducing the consumption by altering SBC's choice and arrangements within the cluster.

Though, arguably the realistic and minimalistic use of the cluster probed, involved only the lift-and-shift of data, while deliberately shying away from running machine learning training and re-tuning algorithms over the cluster. However, it can be executed and benchmarked as well over the cluster. For this purpose, both enabling

MLLib of spark-perf or using spark-bench performance benchmark would provide valuable insight into such compute intensive data analytics tasks.

The dataset generated, reached the size of millions in the benchmark. It can be enhanced to reach billions or trillions of data points, and observe what is the impact over the cluster performance. To facilitate this, it can be achieved either by increasing the size of onboard storage card, or attaching an external storage via USB. Both approaches can be taken, and benchmarked against in the future.

As SBC's are becoming more and more powerful, and getting equipped with GPUs as well, this opens an avenue in evaluating Deep Learning algorithms running over a cluster of such SBCs in the future. As most of the SBCs are fan-less and require low power, therefore their performance would provide interesting insights into viability of such clusters.

Gradually increasing the data, to observe what's the optimal number of nodes for a volume of data, would highlight any tipping point in the performance of the better performing SBC's in a cluster. This would trigger adding more nodes and re-trying the tasks again. It would provide valuable insight into what volumes of data, demands increasing the number of nodes in a cluster for a given problem.

CHAPTER 9: CONCLUSION

Running the benchmark in different experiment setups of cluster, having varying node permutations, and conducting comparative studies of suitable counterparts, provided interesting finds. It is realized that for both building the cluster and later scaling linearly, having higher number of cores on a single SBC’s doesn't perform better or similar, to that of having same number of cores split across multiple SBC’s. In fact, the latter performs better. Thus, given a choice to procure or expand the cluster, either by getting a very powerful and expensive SBC nodes, compared to getting roughly half the cycle-power, but double the quantity and relatively cheaper nodes, it is better to go for the latter. Having more similar low power nodes perform better than having powerful nodes alone. Conclusively, scaling horizontally provides more value in terms of both performance and cost.

While running the master and slave, both on single hardware, i.e. the laptop, not much improvement in performance is gained, even when the memory for slave is doubled and the laptop as a master have way more powerful cores than that of the SBC’s. Though once compared to running on the SBC’s as slaves over the network, the time taken for each application is not significantly different, compared to running both master and slave on the same node, i.e. the laptop. This is significant because there is negligible network latency while running both master and slave on the same hardware, i.e. the laptop alone.

Caches once warmed-up, only then resulted in much improved and consistent performance. Hence, there is a latency in reaching optimal performance for cache warmup. However, it was found that after couple of first few iterations, depending on the size of dataset and after cache warmed-up, the times of tasks completion remains relatively consistent across all SBC’s, even for better performing permutations. Therefore, for a start of a cluster, the cache should be allowed to warmup with the dataset and performance should be measured once time consumed is consistent.

The tasks executed while running the benchmark differed in the size of the dataset. The aggregation and sorting tasks, were done by both integer key and a mixed-type key. The dataset size was doubled for integer key compared to the mixed-type key data. However, it was observed that there wasn't much decrease in performance, even when the input dataset size was doubled indexed with integer key. This elaborates that even when the data is doubled, if it is indexed properly, it gives much improved performance over the cluster.

Banana Pi(s) and Raspberry Pi(s) demonstrates similar performance, though they are of lesser specifications compared to Odroid(s) which demonstrates degraded performance. Surprisingly, Odroid performed better when co-existed with other low compute SBCs. In fact, in the cluster setup, Raspberry Pi(s) have the least clock power, but it showed consistently better results relatively to Odroid(s) and even to Banana Pi(s).

Replicating nodes, may not always result in enhanced performance, i.e. against the obvious. When tried with doubling the nodes, the time taken by each task wasn't reduced much as anticipated. Though it varies from problem to problem, and from dataset to dataset, however doubling the nodes should demonstrate at least slightly significant performance improvement, which was not the case in the results obtained.

APPENDIX 1: CLUSTER SETUP

Setting up nodes – OS

- Image the SD Cards with the images of relevant OSes over them
  - Raspberry – Raspbian
  - Banana Pi – Raspbian for BPI
  - Odroid – Ubuntu 15.10 mate
- Boot the nodes
  - Set the hostname
    - vim /etc/hostname
  - Resize the SD card
    - raspi-config (Banana Pi /Raspberry Pi)
    - resize2fs (Odroid)
  - Update the nodes
    - rpi-update (Banana Pi /Raspberry Pi)
    - apt-update
  - Set the static IP for both Master and Slave nodes. The IP scheme is

Master	192.168.2.222
Odroid 1	192.168.2.111
Raspberry Pi 1	192.168.2.112
Banana Pi 1	192.168.2.113
Odroid 2	192.168.2.211
Raspberry Pi 2	192.168.2.212
Banana Pi 2	192.168.2.213

    - auto eth0
    - iface eth0 inet static
    -    address 192.168.2.222

        - netmask 255.255.255.0
        - broadcast 192.168.2.255
        - gateway 192.168.2.1
        - network 192.168.2.0
    - Raspberry pi need to have the IP changed in the following file instead
        - vim /cmdline.txt

Setting up nodes – Apache Spark

- Download the latest from Apache Spark download page (Master & Slaves)
        - wget http://mirror.nus.edu.sg/apache/spark/spark-1.6.1/spark-1.6.1-bin-hadoop2.6.tgz
    - Unzip it in /opt directory (Master & Slaves)
        - tar –zxvf spark-1.6.1-bin-hadoop2.6.tgz –C /opt/
- Add spark user (Master & Slaves) (On Mac OS, create user/group from System Preferences)
    - sudo groupadd -g 5000 spark
    - sudo useradd -g 5000 -u 5000 -m spark
    - sudo passwd spark
- Change the ownership of the Spark folder to spark user and spark group
    - sudo chown -R spark:spark /opt/spark-1.6.1-bin-hadoop2.6
- Create symbolic link to /opt/spark (Master & Slaves)
    - ln –s /opt/spark-1.6.1-bin-hadoop2.6 /opt/spark
- Change the hostname of master (/etc/hostname) as well
    - master (only if Linux is a master node)
    - sudo scutil --set HostName master (do this on MacOSX)
- Login on master as spark user
    - sudo su – spark # don’t miss the ‘-’

- Change the hostnames on *all* slaves (/etc/hostname) as needed. Banana Pi is given ‘bp#’, Raspberry Pi is ‘rp#’, and Odroid is ‘od#’ for easy identification. The ‘#’ is the number of the node starting with 1
  - vim /etc/hostname
- Nodes will have IP hosts defined as below – (All nodes must have all slave nodes and master node IP addresses in /etc/hosts):
  - 192.168.2.222 master
  - 192.168.2.111 od1
  - 192.168.2.112 rp1
  - 192.168.2.113 bp1
  - 192.168.2.211 od2
  - 192.168.2.212 rp2
  - 192.168.2.213 bp2
- SSH key (Master). This is to enable login without prompting for password by the benchmark application from Master to Slave nodes. The following will copy public key to all slaves
  - ssh-keygen -C "spark@master" -b 2048 -t rsa
  - ssh-copy-id spark@od1
  - ssh-copy-id spark@rp1
  - ssh-copy-id spark@bp1
  - ssh-copy-id spark@od2
  - ssh-copy-id spark@rp2
  - ssh-copy-id spark@bp2
- Making sure it is possible to ssh *without* giving password to all above slaves and even on master
  - ssh spark@master
  - ssh spark@rp1
  - ssh spark@bp1

  - ssh spark@od1
  - ssh spark@rp2
  - ssh spark@bp2
  - ssh spark@od2
- Add IP addresses of worker nodes in the file (Master & Slaves)
- Copy the Spark environment definition template file, and remove the .template extension
  - cp /opt/spark/conf/spark-env.sh.template /opt/spark/conf/spark-env.sh
- Append the SPARK_MASTER_IP in all Workers and set memory as well. It is 768mb by default. Set it to 512mb (Master& Slaves)
  - vim /opt/spark/conf/spark-env.sh
    - SPARK_MASTER_IP=192.168.2.222
    - SPARK_WORKER_MEMORY=512mb
- Copy the slaves' definition template file on the Master node, and remove the .template extension.
  - cp /opt/spark/conf/slaves.template /opt/spark/conf/slaves
- Add slaves IP addresses in /opt/spark/conf/slaves file (Master)
  - vim /opt/spark/conf/slaves
    - od1
    - rp1
    - bp1
    - od2
    - rp2
    - bp2

- Add the JAVA_HOME variable leading to /bin/java in (java 1.7+) (Master)

        - vim ~/.bash_profile
            - export JAVA_HOME='/usr'

- Make sure there is java on every worker (Slaves)
    - java –version
        - sudo apt-get install openjdk-8-jre
- Copy the logging configuration template file, and remove the .template extension.
    - cp /opt/spark/conf/log4j.properties.template /opt/spark/conf/log4j.properties
- Change the log level to error to reduce clutter on console
    - vim /opt/spark/conf/log4j.properties
        - log4j.rootCategory=ERROR, console
- Test starting the master
    - /opt/spark/sbin/start-master.sh
- Test starting the slaves
    - /opt/spark/sbin/start-slaves.sh
- Try submitting any job from /opt/spark/examples to Spark (Have it under /opt/spark/work/* to avoid path issues)
    - /opt/spark/bin/spark-submit work/SVM/svm.py
- At the end, stop both master and slaves
    - /opt/spark/sbin/stop-all.sh

APPENDIX 2: PERFORMANCE BENCHMARKING SETUP

After downloading the benchmark from the GitHub repo i.e. https://github.com/databricks/spark-perf, copy it to /opt directory of the master node. In this case, the master node is the MacBook Pro.

Once copied, please perform the following steps:

- Have spark-perf directory under /opt/ and change it is ownership
  - sudo chown -R spark:spark /opt/spark-perf
- To configure spark-perf, copy config/config.py.template to config/config.py by removing .template extension, and edit that file. See config.py.template for detailed configuration instructions.
  - cp /opt/spark-perf/config/config.py.template /opt/spark-perf/config/config.py
- Running on an existing Spark cluster
  - SSH into the machine hosting the standalone master
  - Set config.py options:
    - SPARK_HOME_DIR = /opt/spark
    - SPARK_CLUSTER_URL = "spark://master:7077"
    - SCALE_FACTOR = .003
    - SPARK_DRIVER_MEMORY = “512mb”
    - spark.executor.memory = “512mb”
    - PROMPT_FOR_DELETES = True

- Uncomment at least one SPARK_TESTS entry and set to True
  - RUN_SPARK_TESTS = True
  - RUN_PYSPARK_TESTS = True
  - RUN_STREAMING_TESTS = False
  - RUN_MLLIB_TESTS = False
  - RUN_PYTHON_MLLIB_TESTS = False
  - PREP_SPARK_TESTS = True
  - PREP_PYSPARK_TESTS = True
  - PREP_STREAMING_TESTS = False
  - PREP_MLLIB_TESTS = False
- http://master:8081, or http://master:8080 for accessing master portal

- An important comment about the scale factor from the /opt/spark-perf/config/config.py

```
# The default values configured below are appropriate for approximately 20 m1.xlarge nodes,
# in which each node has 15 GB of memory. Use this variable to scale the values (e.g.
# number of records in a generated dataset) if you are running the tests with more
# or fewer nodes. When developing new test suites, you might want to set this to a small
# value suitable for a single machine, such as 0.001.
#SCALE_FACTOR = 1.0
SCALE_FACTOR = 0.003
```

APPENDIX 3: COLLATED RESULTS

Tasks	**a. 2 BP**	**b. 2 RP**	**c. 2 OD**	**d. 1 BP 1 RP 1 OD - 1st iter**	**e. 1 BP 1 RP 1 OD - 2nd iter**	**f. 1 BP 1 RP 1 OD - 3rd iter**	**g. 1 OD - 1st Iter**	**h. 1 OD - 2nd Iter**	**i. 2 BP 2 RP 2 OD**	**j. Localhost-512mb**	**k. Localhost -1024mb**
1. TR: scheduling-throughput	234	300	1080	228	210	222	2040	2040	138	96	120
2. TR: aggregate-by-key	186	210	2340	186	186	186	2400	2340	186	43	43
3. TR: aggregate-by-key-int	84	96	900	84	84	84	900	900	84	38	38
4. TR: aggregate-by-key-naive	426	510	4680	426	432	432	4680	4680	432	66	60
5. TR: sort-by-key	96	108	780	96	96	96	780	780	96	40	39
6. TR: sort-by-key-int	108	126	960	108	108	108	960	960	108	40	42
7. TR: count	138	156	1440	144	138	138	1380	1380	138	39	39
8. TR: count-with-filter	150	168	1800	150	150	150	1800	1740	150	39	39

Test Runs	**a. 2 BP**	**b. 2 RP**	**c. 2 OD**	**d. 1 BP 1 RP 1 OD - 1st iter**	**e. 1 BP 1 RP 1 OD - 2nd iter**	**f. 1 BP 1 RP 1 OD - 3rd iter**	**g. 1 OD - 1st Iter**	**h. 1 OD - 2nd Iter**	**i. 2 BP 2 RP 2 OD**	**j. Localhost-512mb**	**k. Localhost-1024mb**
a. TR - 1	510	480	720	366	504	372	1440	1440	198	192	186
b. TR - 2	174	204	96	174	174	174	90	90	174	52	50
c. TR - 3	114	126	78	114	114	114	84	78	114	45	43
d. TR - 4	192	228	96	186	192	192	96	96	192	59	53
e. TR - 5	58	60	57	57	57	57	58	59	57	37	36
f. TR - 6	60	66	59	60	60	60	60	59	60	37	36
g. TR - 7	96	114	66	96	96	96	72	66	96	40	41
h. TR - 8	138	162	78	144	138	138	84	78	138	46	46
i. TR - 9	60	72	72	90	84	66	84	84	84	42	40
j. TR - 10	60	72	72	72	138	66	84	84	66	40	41

SBC	**1. TR: scheduling-throughput**	**2. TR: aggregate-by-key**	**3. TR: aggregate-by-key-int**	**4. TR: aggregate-by-key-naive**	**5. TR: sort-by-key**	**6. TR: sort-by-key-int**	**7. TR: count**	**8. TR: count-with-filter**
a. 2 BP	234	186	84	426	96	108	138	150
b. 2 RP	300	210	96	510	108	126	156	168
c. 2 OD	1080	2340	900	4680	780	960	1440	1800
d. 1 BP, 1 RP, 1 OD - 1st iter	228	186	84	426	96	108	144	150
e. 1 BP, 1 RP, 1 OD - 2nd iter	210	186	84	432	96	108	138	150
f. 1 BP, 1 RP, 1 OD - 3rd iter	222	186	84	432	96	108	138	150
g. 1 OD - 1st iter	2040	2400	900	4680	780	960	1380	1800
h. 1 OD - 2nd iter	2040	2340	900	4680	780	960	1380	1740
i. 2 BP, 2 RP, 2 OD	138	186	84	432	96	108	138	150
j. Localhost-512mb	96	43	38	66	40	40	39	39
k. Localhost-1024mb	120	43	38	60	39	42	39	39

SBC	**a. TR - 1**	**b. TR - 2**	**c. TR - 3**	**d. TR - 4**	**e. TR - 5**	**f. TR - 6**	**g. TR - 7**	**h. TR - 8**	**i. TR - 9**	**j. TR - 10**
a. 2 BP	510	174	114	192	58	60	96	138	60	60
b. 2 RP	480	204	126	228	60	66	114	162	72	72
c. 2 OD	720	96	78	96	57	59	66	78	72	72
d. 1 BP, 1 RP, 1 OD - 1st iter	366	174	114	186	57	60	96	144	90	72
e. 1 BP, 1 RP, 1 OD - 2nd iter	504	174	114	192	57	60	96	138	84	138
f. 1 BP, 1 RP, 1 OD - 3rd iter	372	174	114	192	57	60	96	138	66	66
g. 1 OD - 1st iter	1440	90	84	96	58	60	72	84	84	84
h. 1 OD - 2nd iter	1440	90	78	96	59	59	66	78	84	84
i. 2 BP, 2 RP, 2 OD	198	174	114	192	57	60	96	138	84	66
j. Localhost-512mb	192	52	45	59	37	37	40	46	42	40
k. Localhost-1024mb	186	50	43	53	36	36	41	46	40	41

APPENDIX 4: CAVEATS

While executing the benchmark, be considerate of the following actions performed by the benchmark:

- Running bin/run, will copy the master /opt/spark directory to all slaves
- The folder /opt/spark/work on master will be asked to delete content of it
- STREAMING_TEST, MLLIB_TEST, PYTHON_MLLIB_TEST; are not what this benchmark is going to test running on cluster
- Add the following line before cluster stop so can save results before exit
  - vim /opt/spark-perf/lib/sparkperf/main.py
    - raw_input = raw_input("Save the http://master:8080, and press enter to exit!")
- Banana Pi, once powered with USB, results in USB and Ethernet not functional. Use direct power using AC adapter.
- Raspberry Pi, once powered with an USB extension cable, results in Ethernet not functioning. Use direct power without any extension.
- Halting/Freezing of one of the Odroid frequently over SSH. Turns out it is a known problem, and the hardware had to be replaced.
- Keep the master node awake all the time, i.e. no standby or sleep or hibernate mode.

BIBLIOGRAPHY

[Fan16] An ARM-Based Hadoop Performance Evaluation Platform: Design and Implementation By Xiaohu Fan et. al.

[Haj16] Understanding the Performance of Low Power Raspberry Pi Cloud for Big Data By Wajdi Hajji and Fung Po Tso

[BPC14] Clusters of single-board computers

[DG14] Raspberry Pi Cluster By David Guill

[JK] Raspberry Pi By Joshua Keipert

[JK13] Creating a Raspberry Pi-Based Beowulf Cluster By Joshua Keipert

[CK14] Big Data Processing on an ARM Cluster By Chanwit Kaewkasi

[MK16] Installing Spark onto the ODROID XU4 Cluster By Michael Kamprath

[LPHC] A study of big data processing constraints on a low-power Hadoop cluster By Chanwit Kaekasi and Wichai Srisuruk

[Jeff16] Review: ODROID-C2, compared to Raspberry Pi 3 and Orange Pi Plus By Jeff Geerling

[Li15] SparkBench: a comprehensive benchmarking suite for in memory data analytic platform Spark By Min Li et al.

[DA16] SparkBench – A Spark Performance Testing Suite By Dakshi et. al.

[SPRK] Apache Spark

[PERF] Spark Performance test

[SBC] Single-board computers

[DS] Distributed Systems

[RPOS] Raspbian

[BPOS]	Bananian
[BPI]	Banana Pi
[RPI]	Raspberry Pi
[OD]	Odroid OS
[Jeff15]	Getting Gigabit Networking on a Raspberry Pi 2, 3 and B+ By Jeff Geerling
[AIY]	Aiyara cluster
[SUTAC]	SUT Aiyara Cluster
[JeffRPi]	Review: Raspberry Pi model 3 B, with Benchmarks vs Pi 2
[JeffSD]	Raspberry Pi microSD card performance comparison - 2015