International Engineering Journal For Research & Development
Vol.5
Issue 5

E-ISSN NO:-2349-0721

Impact factor : 6.549

A STUDY ON CLUSTERING OF RASPBERRY PI

[1]Ms. Aditi Satish, [2]Mr Jasan Sandeep, [3]Mr. Jeet Barot, [4]Prof. Sonal Hutke, [5]Mr. Darshan Choudhary
Department of Electronics and Telecommunication Engineering, SIES Graduate School of Technology, Navi Mumbai, India[1,2,3,5], Assistant Professor Department of Electronics and Telecommunication Engineering, SIES Graduate School of Technology, Navi Mumbai, India[4]
satish.aditi16@siesgst.ac.in[1], sandeep.jasan16@siesgst.ac.in[2], jeet.barot16@siesgst.ac.in[3] , Sonal.jatkar@siesgst.ac.in[4], darshan.choudhary16@siesgst.ac.in[5]

ABSTRACT

In today's era, most of our work is dependent on computers and the processing done by computers. There is a demand for faster, smarter, and efficient computers. By using our proposed project, we intent to study and prototype load balancing techniques used in computing. This demonstration is being done by using Raspberry Pis as the primary component. Our demonstration depicts how Raspberry Pi nodes communicate with each other in a wireless manner and help in efficient computing.

Keywords—Clustering, computers, cores, processing, Raspberry Pi.

INTRODUCTION

Raspberry Pi is a small sized single board computer. It is a powerful little device that allows people to discover computing, and to learn how to program in languages like Scratch and Python. It has the ability to do everything you'd expect a desktop computer to do, from browsing the internet and playing high-definition video. A raspberry along with a mouse, keyboard and memory card behaves like a traditional desktop computer. In this project we are using Raspberry Pi 3 model B. It runs on a frequency of 1.2 GHz with a 64-bit CPU and has a Broadcom BCM2837 quad core processor.

In a computer system, a cluster is a group of servers and other resources that act like a single system and enable high availability, load balancing and parallel processing.[1] Our focus here is to study the load balancing in a tetra-pi cluster setup. We would be clustering four Raspberry Pis and set them up as a server and use another fifth Raspberry Pi, which would act like a client. We perform cluster computing. Unlike grid computing which have non interactive workload and have each node set to perform a different task/application, cluster computers have each node set to perform the same task, controlled and scheduled by software. Clusters are normally conveyed to improve execution and accessibility over that of a solitary PC, while ordinarily being significantly more cost-effective than single PCs of practically identical speed or accessibility.

These clusters are basically a personalized version of the existing large server networks. We intend to study how a program is actually executed in this tetra-pi setup and study various load balancing methods. Besides studying the load balancing algorithms, we will also study the performance of the cluster vs a single raspberry pi computer. This cluster can be further used in image processing, big data and bioinformatics.

LITERATURE SURVEY

Current commodity Single Board Computers (SBCs) are sufficiently powerful to run mainstream operating systems and workloads. Many of these boards may be linked together, to create small, low-cost clusters that replicate some features of large data center clusters. The Raspberry Pi Foundation produces a series of SBCs with a price/performance ratio that makes SBC clusters viable, perhaps even expendable. These clusters are an enabler for Edge/Fog Compute, where processing is pushed out towards data sources, reducing bandwidth requirements and decentralizing the architecture. In this paper we investigate use cases driving the growth of SBC clusters, we examine the trends in future hardware developments, and discuss the potential of SBC clusters as a disruptive technology. Compared to traditional clusters, SBC clusters have a reduced footprint, are low-cost, and have low power requirements. This enables different models of deployment—particularly outside traditional data center environments. We discuss the applicability of existing software and management infrastructure to support exotic deployment scenarios and anticipate the next generation of SBC.[2]

We conclude that the SBC cluster is a new and distinct computational deployment paradigm, which is applicable to a wider range of scenarios than current clusters. It facilitates Internet of Things and Smart City systems and is potentially a game changer in pushing application logic out towards the network edge.

Simon J. Cox, James T. Cox, Richard P. Boardman, Steven J. Johnston, Mark Scott, Neil S. O'Brien presented a paper on Idris-Pi a 64 node raspberry-pi cluster which consisted of which consists of 64 Raspberry Pi Model B nodes each equipped with a 700 MHz ARM processor, 256 Mbit of RAM and a 16 GiB SD card for local storage. The cluster has a number of advantages which are not shared with conventional datacenter-based cluster, including its low total power consumption, easy portability due to its small size and weight, affordability, and passive, ambient cooling. We propose that these attributes make Idris -Pi ideally suited to educational applications, where it provides a low-cost starting point to inspire and enable students to understand and apply high-performance computing and data handling to tackle complex engineering and scientific challenges.[3]

OBJECTIVES

The main aim of this project is to study and discover ways to improve the overall computing experience with the help of a raspberry pi cluster, which acts like a personalized computer cluster. The other objectives of this project are mentioned below

1. Networking and communication between the master and the slave nodes.
2. Cost cutting (in running a server)
3. Power saving

SIGNIFICANCE

In computing, a server is a computer program or a device that provides functionality for other programs or devices, called "clients". This architecture is called the client–server model, and a single overall computation is distributed across multiple processes or devices. Servers can provide various functionalities, often called "services", such as sharing data or resources among multiple clients, or performing computation for a client. A single server can serve multiple clients, and a single client can use multiple servers. Hence, servers typically consume a lot of power and are also very expensive. To overcome these problems, we use cluster setup which

computes the given task with much less power consumption rate and also are comparatively cheaper than traditional servers.

METHODOLOGY

This study involves hardware as well as software procedures. Four nodes of Raspberry Pi 3b are used. They are clustered together to form a prototype of a supercomputer. This structure is named as “Tetra-Pi” as this study is done using four nodes and Tetra stands for four in Greek.

Following steps need to be completed to conduct this study.

a) Set up the Raspberry Pi Cluster [Tetra-pi]

1. To begin this study first we need to setup a fully running cluster made of four nodes of Raspberry Pi and an additional node which will act as the master node, so in total we have used 5 nodes in this study.
2. Make sure every node receives exactly 2A current and also make sure that the SD card used in every node is at least of class 10.
3. Install Raspbian on all five nodes and make sure that the installation is done identically on every node including the master node. This is done because when nodes are being configured as slave/server nodes, they need to be identical in the way their data is distributed on Raspberry Pi, i.e. they need to look the same.
4. After installation of Raspbian on every node, download the necessary packages and software.
5. Create a dedicated Wi-fi network for the cluster using a separate router.
6. Create a Client/ Master Node.
7. Create Server/Slaves using four Raspberry Pis.

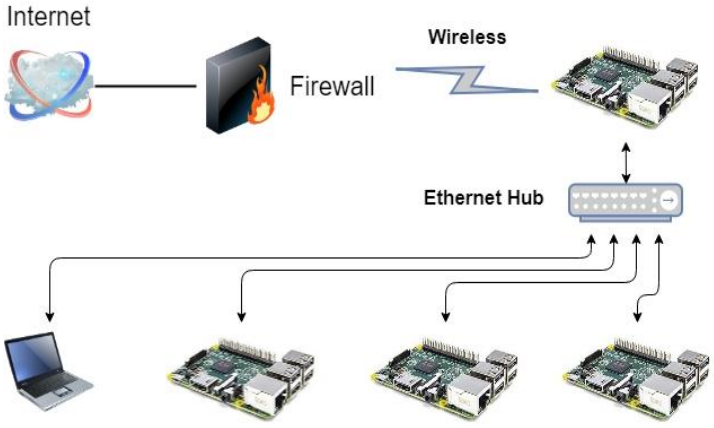

Fig 4.1

Figure 4.1 depicts a basic skeleton of a Raspberry Pi cluster, where the components include multiple Raspberry Pi nodes and an ethernet box / router which helps the master node connect with the slave nodes wirelessly.

a) Test the cluster

You can test your cluster by installing the *dispy* module in python. Dispy framework is normally used in distributed network scenarios where parallel computation is expected.

b) Observe the result

Observe the results obtained after computation is done using the cluster and compare it with the results that you would obtain if only one Raspberry pi node was used for computational purposes. Compare the speed, the energy efficiency and you can also compare the cluster's performance to the that of a proper working server (this would help in concluding about cost efficiency).

SIMULATION AND PRACTICAL RESULTS

For simulation purposes and for testing the cluster, the dispy library in python was downloaded and installed. This library is mainly used in scenarios involving distributed computing and remote computing.

The dispy library is an easy to use framework employed to create and use compute clusters, which are basically a form of loosely connected computers that enables users to run medium-to-large scale operations by distributing multiple, independent tasks across many nodes (computers).

Dispy can be used to distribute a single process in all of the nodes or it can be used to execute multiple processes simultaneously sharing the nodes.

For simulation purposes, we took a scenario where we had to execute 1 lakh random jobs. We used the dispy library along with the compute.py code which assigns 1 lakh random jobs to all of the nodes. But before we run this simulation, the cluster should be well defined and full-proof. Basically, there should be a dedicated network through which only the master and slaves communicate with each other and no other random device interrupts this communication.

For the first part of the simulation, the server nodes are switched off and only the master node is on. Then a code is written to the master node "sudo python3 compute.py". The result of this command is show in Fig 5.1 and it displays how much time the node takes to execute a particular number of commands.

The compute.py Python code runs 15 jobs on the master node. They are all just random delays before returning. After the execution of the code is complete, a table similar to this will be displayed which depicts the approximate time taken to execute the no. of jobs assigned to "n" nodes.

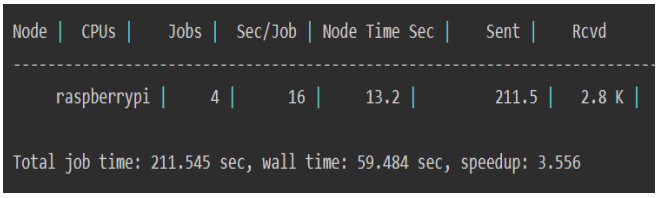

Fig 5.1

Now, connect the nodes and make sure every node receives proper power supply and is connected to the master node.

Again execute "sudo python3 compute.py" code and now, the same code will execute the same jobs in a distributed manner and the result would reflect that the execution was much faster and coordinated.Now, run the code "sudo python3 compute_pi_efficient.py 1000 100000". This code will use the dispy library and execute 1 lakh random jobs across all the nodes in a distributed manner. Every node receives 1000 jobs at once. The jobs

received by every node determines the speed of the cluster execution. This magnitude can be scaled by adjusting the jobs received by the nodes.

When this code was being executed, we tested a few things. We observed that if a node shuts down or stops communicating with the master, the jobs assigned to that node is automatically assigned to other nodes. This adjustment is automatic and if the node which stopped communicating comes back online, then the node is assigned back its jobs and the other nodes ignore the extra assigned jobs.

The following Fig 5.2 shows the results of the simulation:

2019-10-13 21:38:41 dispy - job ”70000” returned (47882, 766), 70jobs pending2019-10-13 21:38:48 dispy - job ”71000” returned (42857, 788), 77jobs pending2019-10-13 21:38:56 dispy - job ”72000” returned (59652, 786), 50jobs pending2019-10-13 21:39:03 dispy - job ”73000” returned (51051, 757), 71jobs pending2019-10-13 21:39:09 dispy - job ”74000” returned (51554, 786), 85jobs pending2019-10-13 21:39:17 dispy - job ”75000” returned (7624, 803), 66 jobspending2019-10-13 21:39:26 dispy - job ”76000” returned (10596, 780), 52jobs pending2019-10-13 21:39:34 dispy - Could not connect to 192.168.1.230:61591,Traceback (most recent call last):File ”/usr/local/lib/python3.7/dist-packages/dispy/init.py”,line669,insendyieldsock.connect((self.ipaddr,self.port))socket.timeout:timedout2019-10-13 21:39:34 dispy - Could not connect to 192.168.1.230:61591,Traceback (most recent call last):File ”/usr/local/lib/python3.7/dist-packages/dispy/init.py”,line669,insendyieldsock.connect((self.ipaddr,self.port))socket.timeout:timedout18

2019-10-13 21:39:34 dispy - Failed to run job 1969490224 on 192.168.1.230for computation compute; removing this node2019-10-13 21:39:34 dispy - Failed to run job 1969490352 on 192.168.1.230for computation compute; removing this node2019-10-13 21:39:34 dispy - Failed to run job 1969489072 on 192.168.1.230for computation compute; removing this node2019-10-13 21:39:34 dispy - job ”77000” returned (30254, 771), 58jobs pending2019-10-13 21:39:43 dispy - job ”78000” returned (1689, 788), 34 jobspending2019-10-13 21:39:51 dispy - job ”79000” returned (54152, 785), 60jobs pending2019-10-13 21:39:59 dispy - job ”80000” returned (4092, 762), 32 jobspending2019-10-13 21:40:07 dispy - job ”81000” returned (61759, 792), 52jobs pending2019-10-13 21:40:16 dispy - job ”82000” returned (24561, 806), 68jobs pending2019-10-13 21:40:26 dispy - job ”83000” returned (33548, 809), 44jobs pending2019-10-13 21:40:35 dispy - job ”84000” returned (50644, 808), 60jobs pending2019-10-13 21:40:44 dispy - job ”85000” returned (22408, 777), 95jobs pending2019-10-13 21:41:00 dispy - job ”86000” returned (278, 773), 51 jobspending2019-10-13 21:41:06 dispy –

Ignoring invalid reply for job 1969489328from 192.168.1.922019-10-13 21:41:06 dispy - Ignoring invalid reply for job 1969489328from 192.168.1.922019-10-13 21:41:06 dispy - Ignoring invalid reply for job 1969489328from 192.168.1.922019-10-13 21:41:06 dispy - Ignoring invalid reply for job 196948932819

from 192.168.1.922019-10-13 21:41:06 dispy - Ignoring invalid reply for job 1969489328from 192.168.1.922019-10-13 21:41:07 dispy - Ignoring invalid reply for job 1969489328from 192.168.1.922019-10-13 21:41:07 dispy - Ignoring invalid reply for job 1969489328from 192.168.1.922019-10-13 21:41:08 dispy - Ignoring invalid reply for job 1969489328from 192.168.1.922019-10-13 21:41:08 dispy - Ignoring invalid reply for job 1969489328from 192.168.1.922019-10-13 21:41:09 dispy - Ignoring invalid reply for job 1969489328from

```
192.168.1.922019-10-13 21:41:09 dispy - Ignoring invalid reply for job 1969489328from
192.168.1.922019-10-13 21:41:09 dispy - Ignoring invalid reply for job 1969489328from
192.168.1.922019-10-13 21:41:09 dispy - Ignoring invalid reply for job 1969489328from
192.168.1.922019-10-13 21:41:09 dispy - Ignoring invalid reply for job 1969489328from
192.168.1.922019-10-13 21:41:10 dispy - Ignoring invalid reply for job 1969489328from
192.168.1.922019-10-13 21:41:10 dispy - Ignoring invalid reply for job 1969489328from
192.168.1.922019-10-13 21:41:11 dispy - Ignoring invalid reply for job 1969489328from
192.168.1.922019-10-13 21:41:11 dispy - Ignoring invalid reply for job 1969489328from
192.168.1.922019-10-13 21:41:11 dispy - Ignoring invalid reply for job 1969489328from
192.168.1.922019-10-13 21:41:11 dispy - Ignoring invalid reply for job 1969489328from
192.168.1.9220

2019-10-13 21:41:12 dispy - Ignoring invalid reply for job 1969489328from 192.168.1.922019-10-13
21:41:12 dispy - Ignoring invalid reply for job 1969489328from 192.168.1.922019-10-13 21:41:12
dispy - Ignoring invalid reply for job 1969489328from 192.168.1.922019-10-13 21:41:12 dispy -
Ignoring invalid reply for job 1969489328from 192.168.1.922019-10-13 21:41:12 dispy - job
”87000” returned (45235, 779), 91jobs pending
```

Fig 5.2

The final result of the simulation after proper execution is displayed in the Fig 5.3 which shows how many jobs each node executed at a certain time along with wall time and speed.

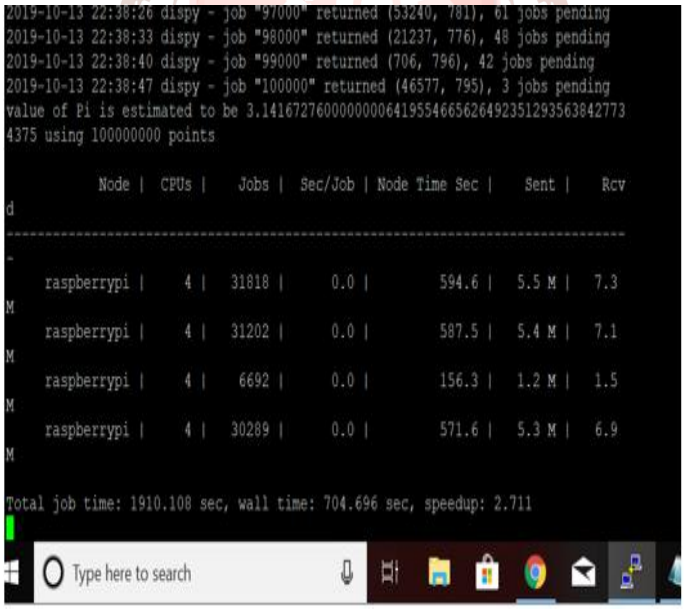

Fig 5.3

CONCLUSION

This study of clusters of Raspberry Pi helped us to understand and learn about scalability and working of clusters instead of expensive servers to compute large chunks of data. Clustering of Raspberry Pis, if done correctly would help us achieve the results which would normally require the use of expensive server machines.

The clusters communicated with each other wirelessly and coordinated the execution of the given jobs. This task was achieved by efficient networking and communication between the cluster nodes and the master.

The clusters can be used in various applications like big data analytics using apache Hadoop, in machine learning using TensorFlow software or in cybersecurity domain which usually require pcs with high

computational power. The use of clusters helps in efficient computing and cost cutting by a very significant margin.

REFERENCES

[1] ("What is cluster? - Definition from WhatIs.com", 2020)

[2] Johnston, Steven J., et al. "Commodity single board computer clusters and their applications." Future Generation Computer Systems 89 (2018): 201-212.

[3] Cox, Simon J., et al. "Iridis-pi: a low-cost, compact demonstration cluster." Cluster Computing 17.2 (2014): 349-358.

[4] ("SSH Protocol – Secure Remote Login and File Transfer", 2020)

[5] Krauss, Ryan. "Combining Raspberry Pi and Arduino to form a low-cost, real-time autonomous vehicle platform." In 2016 American Control Conference (ACC), pp. 6628-6633. IEEE, 2016.

[6] Abrahamsson, Pekka, et al. "Affordable and energy-efficient cloud computing clusters: The bolzano raspberry pi cloud cluster experiment." 2013 IEEE 5th International Conference on Cloud Computing Technology and Science. Vol. 2. IEEE, 2013.

[7] Bhave, S., Tolentino, M., Zhu, H. and Sheng, J., 2017, July. Embedded middleware for distributed raspberry pi device to enable big data applications. In 2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC) (Vol. 2, pp. 103-108). IEEE.