Pi-Crust: A Raspberry Pi Cluster Implementation

Eric Wilcox
Department of Computer Science and Engineering
Texas A&M University
ewilcox42@gmail.com

Pooja Jhunjhunwala
Department of Computer Science and Engineering
Texas A&M University
pj861992@gmail.com

Karthik Gopavaram
Department of Computer Science and Engineering
Texas A&M University
karthikgopavaram@gmail.com

Jorge Herrera
Department of Computer Science and Engineering
Texas A&M University
jorgeivan000@gmail.com

***Abstract*—Raspberry Pi is revolutionizing the computing industry. Originally designed to provide low cost computers to schools it quickly expanded far beyond that. With this inexpensive technology you can accomplish tasks previously unexplored. One such set up is a cluster computer to run parallel jobs. Many systems built for parallel computing jobs are either very expensive or unavailable to those outside of academia. Supercomputers are extremely expensive to own, use, power and maintain. Even though average desktop computers have come down in price the expense can still get quite high if you need a larger amount of processing power. In this project, we take ten Raspberry Pi 2 computers and connect them over an ethernet network to build a parallel version of a supercomputer. We show how one can be built inexpensively and compare the performance of our system to both desktop machines and supercomputers in use by academia. We look at a price comparison as well as performance and show the differences between the older versions of the Raspberry Pi cluster computers and our own implementation. Our results show that our Pi cluster implementation significantly outperforms previous Pi cluster projects with the cost being similar. This project strongly made us believe that the Raspberry Pi is a great learning platform for academia and personal use and the processing power at the cost involved is hard to beat.**

I. Introduction

Raspberry Pi is an inexpensive credit card sized single board general purpose computer originally developed in the UK. It was conceptualized to give the schools access to widespread computing at a very low cost. The Raspberry Pi 2 Model B (released in February 2015) comes equipped with a 900 MHz quad-core ARM Cortex-A7 CPU (BCM2836) and 1GB of RAM (LP DDR2 SDRAM). The cost of the new version 2 is the same as the older versions, at just $35. While not an extremely powerful computer it has some significant benefits such as compact size, reduced cost, low power consumption and low heat generation. This makes it ideal not only for the original intent in academia but also for many other projects, such as building portable and energy-efficient cluster computers. Some have even created honeypot traps simulating web servers using the Raspberry Pi cluster computer setup [3]. In this framework not only can the researchers safely simulate a web server to trap SQL injection attacks they can more efficiently handle them in the cluster and through redirection increase the amount of attack data that could be analyzed [3]. With the new explosion of research into IoT (Internet of Things) researchers show how the Raspberry Pi can be used to provide an inexpensive and low power consumption framework for service provisioning [7] which can be extended in many ways including cluster systems or just networking the units. Users today do not have to compromise on computing power to take advantage of multicore benefits.

Raspberry Pi is one of the first low cost and reasonably high performance computers to revolutionize the education industry. It has afforded the current generation of students to interact with computers in a way that was never possible before. It is immensely successful as a single computer platform and we wanted to investigate if it could also be a viable option for building high performance cluster computers. The low cost of the Pi was a critical driving factor for this project and helped in materializing our motivation into a course project. Therefore, the primary goal of the project is to study Raspberry Pis capability to perform in a parallel and cluster computer setup.Through this project we also investigate the cost of building a general purpose parallel programmable machine using low priced components and analyze if it meets the requirements to be used as an instructional aid in a college level parallel programming course.

Additional motivation for this project is to understand the construction and benchmarking process of a computer cluster. We wanted to study and implement the various aspects required to build a fully functional cluster from the networking protocols to MPI configuration. We also wanted to identify and execute the standard performance benchmarks and evaluate how the Pi cluster performance compares to traditional PCs and supercomputers commonly used in academic labs. We also want to compare the Pi cluster with commercial computers and available cluster platforms in terms of cost, power, performance, scalability and functionality. With these results we aim to determine if it the Raspberry Pi cluster computer is a feasible alternative for academic institutions that cannot afford or utilize production clusters for educational purposes.

In this project, we build a cluster computer out of 10 Raspberry Pi 2 Model B units. We will discuss the required equipment, the operating system on each one, how they are connected, configured and tested. We will cover the challenges and issues faced during this project and the solutions we came up with. We will present the results of benchmark testing the system and show comparisons between older version Raspberry Pi clusters in addition to the supercomputer available at Texas A&M. We will discuss the actual price comparisons of this system versus some other

systems. Finally we will conclude with what was learned from this project and possible avenues of future work.

II. Prior Work

There has already been some prior work involving the Raspberry Pi clusters. However, none of these previous Pi clusters utilize the new Pi version 2 which we use and benchmark in this project. The Bolzano Raspberry Pi cloud cluster experiment implemented a 300 node Pi cluster [1]. The main goal of this project was to study the process and challenges of building a Pi cluster on such a large scale . Their work demonstrates how to setup and configure the hardware, the system and the software. It also presents how to monitor and maintain the system and utilize it as a cloud cluster. However, the Bolzano project does not benchmark the performance of their cluster which we do in this project. The Iridis-Pi project implemented a 64 node Pi cluster [2]. Their work benchmarks the cluster performance using the HPL Linpack benchmark and makes a strong argument for utilizing Pi clusters in academia. However, their work does compare the Pi cluster performance against desktop PCs or a supercomputer like we do in this project. Lastly, the Raspberry Pi based Beowulf cluster implemented a 32 node Pi cluster [4]. This work documents the cluster construction process and provides information on the clusters performance and power consumption. But, this work also does not use the HPL Linpack to benchmark cluster performance.

III. Building the Cluster

A. Equipment

In this section, we present all the various components used to construct our ten node cluster computer. We also report the cost of the components as we wish to demonstrate that building a high performance Raspberry Pi cluster is very economical. The components of the cluster include:

- Ten Raspberry Pi 2 Model B. The total cost for the Pis is $350 ($35 each).
- Each Pi requires a microSD card as the Pi can boot only from an SD-card. The Pi has been tested to support microSD cards with sizes up to 32GB. We decided to procure the 16GB SanDisk MicroSD cards. The ten SD-cards cost us $87.90 total.
- We required ten micro USB cords to power the individual Pis. The USB cords cost us $7.98 total.
- One ten port USB power strip costing $46.99.
- Two switches of eight ports each costing $59.98 total. We decided to get two 8 port switches instead of a single 10 port switch as the cost of two 8 port switches was slightly cheaper than one 10 port switch and had a better performance rating.
- Ten ethernet cables costing $14.99 total. The cables are used to connect the individual Pis to the switches. We later realized that we required one extra ethernet cable to connect the switches together.
- A lego block set costing $39.95. We used the legos to build a structure to hold the cluster together in one place. The lego structure is shown in Figure 2 and the complete setup in Figure 3. The legos helped to make the cluster portable, sturdy and added some aesthetic value to the project. For a more permanent solution we recommend super glue if you are going to use legos, that or internal desktop construction parts.

This list does not include several other items we used in the project. The Raspberry Pi by default needs a monitor and keyboard at a minimum usually. We had these available and did not have to purchase them. You can remote into the system with another computer though and then those would not always be needed. They do help considerably with the initial setup though and we recommend having them available, especially if you want easy quick access to the system on a consistent basis without having to ssh into it from another machine. For a regular computer monitor to work you will need an hdmi to digital video adapter, assuming the monitor has a digital input. Alternatively you can use any hdmi enabled device. We started off with a television for the setup at first. An hdmi cable is needed in that case but if you have a television you can borrow the cable to get through the setup. With shipping and handling and all the components we did buy the total cost of the system was under $650 which is similar to the cost of a single multi-core PC or laptop. Without the legos it was under $600 which should be easily duplicated. Computing power of up to 40 cores is possible with this setup compared to the 4 cores available in a standard multi-core PC or laptop.

B. Design and Setup

The Cluster design consists of 10 Raspberry Pis wired by switches in order to establish connection between them. One raspberry pi is the master or head of the cluster. The other nine are slaves or workers. You can see the configuration setup in 1. The network configuration was established by setting static IP addresses to each node. The master has the address of every other node, and it is the only one that can talk to every node. The slaves can only communicate with the master node. We will detail that more below.

The first phase of the project was to setup the individual Raspberry Pis to be able to operate in a cluster computer environment. Below is an overview of the procedure involved in setting up the Pis:

- First, we needed to install and configure an operating system for the Raspberry Pi. We choose to use the Raspbian OS since it is a well documented and widely used operating system for Raspberry Pi. Another reason for using Raspbian is that it comes with many programs already installed and there is a lot of support and documentation available. The Raspbian operating system came bundled with Python version 2.7.6.
- The Raspberry Pi can boot only from the microSD card. Therefore, we formatted the microSD card to include the

bootloader and the Raspbian operating system.

- Once we were able to boot up a single Raspberry Pi, we further configured the operating system and installed all the required dependencies.
- The next important step was to add some parallel programming framework on to the Pis for them to be able to communicate with each other and behave like one individual cluster. We chose to equip the Pis with MPI (Message Passing Interface) by installing MPICH and MPI4Py. The libraries were needed for coding in python for the parallel algorithms and they included some testing utilities we wanted to take advantage of as well.
- Once the operating system and parallel programming framework was configured completely for a single Pi, the operating system image was just burned onto the SD cards for the other Raspberry Pis so as to avoid the repetition of the work for each Pi. The simplest and quickest way to accomplish this is to make a disk image of the SD Card on another system and then image the other SD Cards from that. This process can be very time consuming so patience is key.

This should leave you with all the Pis as an exact duplicate of the first. The next step was the network configuration. The Pis were set up as master and slave to enable them to function as a cluster computer. One Pi was designated the head node or master of the cluster, and the nine other Pis were configured as the slaves. First we accessed each Pi individually and gave it a static IP address. We chose the 192.168.4.0 domain for this project but if you have a network you are setting the system up with you should choose a range of addresses which would not be normally assigned by your router. We had our head node labeled Pi10-Head at 192.168.42.10 and the other nodes as Pi01 at 192.168.42.1 then 2 and so on. This was also a good time to change the label of each Pi in the configuration. We want 192.168.42.1 to be labeled Pi01 and so forth. This was important so you could see at the time of any login which Pi you were in and know the corresponding IP. The IP scheme would need changed for systems greater the 255 nodes, but we recommend similar logic to naming and addressing them.

After the static addressing we could then SSH into each node from the master with them all connected to the switches, but the default password of raspberry was needed each time. The master node needed passwordless access into the slave Pis over SSH for the cluster to function correctly and this was achieved by generating SSH keys for each Pi and sharing the keys of each slave node with the master node. The master node's key was added to the list of authorized keys for each of the slave nodes and the keys of the slave nodes was added to the authorized keys list of the master node. This ensured that the slave nodes could talk to the master node at will and vice versa. It is not necessary to have all the slave nodes keys on the slaves as they only communicate in this setup with the master node.

One of the issues we faced while configuring and benchmarking the individual Raspberry Pis was that some of the files needed to be copied to every node. This task is repetitive and can get exhaustive when working with a large cluster computer. Therefore, in order to optimize the time invested in this task, we developed a simple script to automate the process of copying files to every Raspberry Pi. Our script was rather simplistic just taking a file or directory to copy to all nodes in a loop. However, for a more permanent design we recommend a more developed script for both copies and removes from all nodes. These operations on files can get repetitive if you have to SSH into each node individually and for anything larger than a ten node cluster we imagine it would be both extensive and unsafe to do manually.

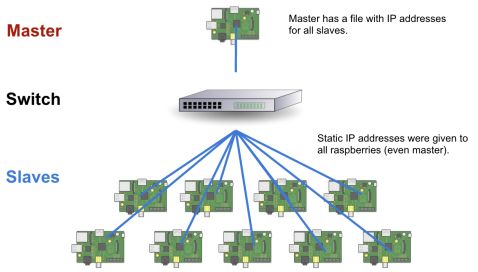

Fig. 1. Architectural diagram of setup.

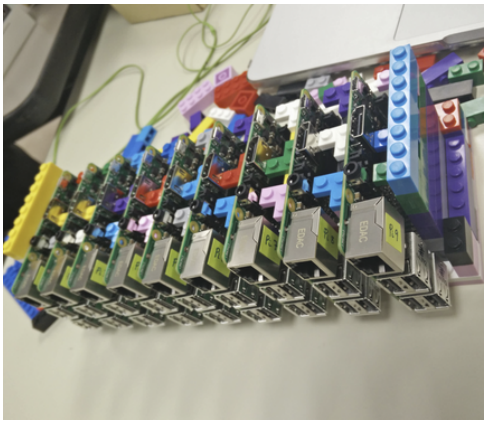

Fig. 2. Nine Pi cluster setup.

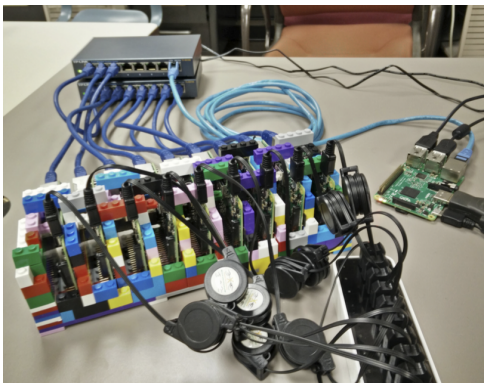

Fig. 3. Complete cluster setup while running.

IV. Evaluation

A. Benchmarks

The tests run on the raspberry cluster aimed to determine if the cluster was scalable, i.e. if the inclusion of more cores added to the processing capacity of the cluster. Another parameter that the tests aimed to measure was to determine if the cluster displayed a performance comparable to a conventional, commercial, supercomputing cluster in terms of computing power per unit cost and per unit power. The problem chosen to test the first parameter was a matrix multiplication problem. Matrix multiplication is among the most common and frequently used computation and is very computationally intensive. It is also a highly parallelizable problem and, therefore, was chosen as the problem to be analyzed for the performance of the cluster. The problem executed on the p node cluster was the multiplication of two square matrices, $A * B$, with dimensions nxn each. A replica of matrix B was provided to every process, and n/p rows of matrix a were provided to each processor. Each processor calculated n/p rows of the output matrix in parallel. These rows are then transmitted to the head node and combined into one final matrix. This test was carried on varying number of processors, starting from one processor and increasing the number of processors by one till we reached ten. The same experiment was run on the Eos cluster of Supercomputing facility at Texas A&M University on the Nehalem nodes in the same procedure as was done for the Pi cluster. The results for this experiment are displayed in Figures 4 and 5.

Another test run on the Raspberry Pi cluster was the High Performance Linpack (HPL) Benchmark test. HPL solves a (random) dense linear network in double precision (64 bits) arithmetic on distributed memory computers. It is a widely recognized performance benchmark for systems like the cluster and therefore was chosen to test the performance of the cluster. The HPL Benchmark was tested for various configurations. To determine the scalability of the system, the problem size was kept constant and the number of processors was gradually varied from one to ten. The results obtained are displayed in Figure 6. The results obtained from the HPL benchmark test were compared against an older version Raspberry Pi cluster setup and against a Yellowstone supercomputer as seen in Figures 8 and 9.

In order to get the best possible results out of HPL benchmark, we had to fine tune several configuration parameters in the input file HPL.dat. The values for these tunable parameters depended on several factors such as the processing power, memory of the available cluster, cluster size and layout etc. [8]. Some of these parameters could be deduced from the system characteristics while the others required a trial and error approach. There is no formula mentioned anywhere in the guidelines to get the best performance out of the cluster. Therefore, we ran the HPL benchmark multiple times with different parameter values to extract the best possible performance from the Pi cluster. The most important parameters that we had to configure are:

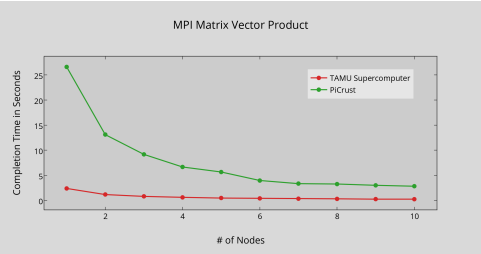

Fig. 4. MPI matrix vector product completion time between TAMU supercomputer and Pi-Crust cluster.

- Number of problems sizes (N).- In order to extract the best performance out of the system, we need to have the largest problem size that can fit in the memory. The amount of memory used by HPL is essentially the size of the coefficient matrix. For our cluster, the maximum possible value for N is 32768. However, we were unsuccessful in reaching this maximum value. The cluster was able to produce results for N at 17400 with 10 processors (one core each). And, when the cluster utilized all 40 cores, we could only reach a value of 10000 for N. For all the values over 10000, we were unable to produce results as the cluster kept crashing with errors.
- Number of NBs.- HPL uses the block size NB for the data distribution as well as for the computational granularity. From a data distribution point of view, the smaller the NB, the better the load balance. It is prefered not to have very large values of NB. From a computation point of view, a too small value of NB may limit the computational performance by a large factor because almost no data reuse will occur in the highest level of the memory hierarchy. Therefore, we were unsure what the optimal value of NB was for our cluster. We took the trail and error approach and measured cluster performance with NB values at 32, 64 and 128.
- Number of process grids (PxQ).- The values of P and Q depend on the physical interconnection network used. Assuming a mesh or a switch, HPL prefers a $1 : k$ ratio with k in $[1..3]$. In other words, P and Q should be approximately equal, with Q slightly larger than P. The best values for our cluster are 5 and 8 for P and Q.

Our best performance result was generated with the values $N = 10000$, $NB = 128$, $P = 5$ and $Q = 8$. We strongly believe that we could have extracted higher performance had been able to run the HPL benchmark with larger problem sizes N.

B. Results

The Raspberry Pi cluster computer displayed very promising results with the tests conducted on them. The aim of the project was to test the feasibility of the Raspberry Pi cluster as a teaching aid to teach parallel programming. Programming on the cluster was done in Python using MPI4Py and the programming experience was quite flexible. The parameters

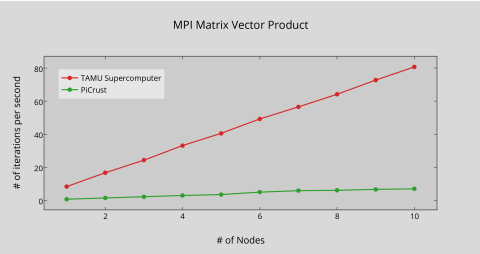

Fig. 5. MPI matrix vector product iterations per second between TAMU supercomputer and Pi-Crust cluster.

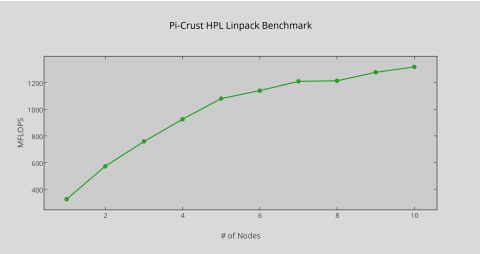

Fig. 6. Pi-Crust HPL Linpack Benchmark test from 1 to 10 nodes.

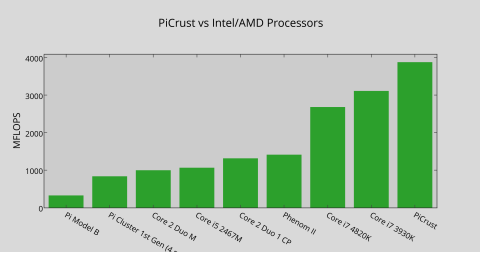

Fig. 7. Pi-Crust Mflops performance chart vs Intel and AMD processors [5].

could be tuned easily as the whole cluster was at the users disposal.

The result of the matrix multiplication showed that the cluster scales almost linearly when tested with a low effort parallelization program like the matrix multiplication problem. The cluster performs considerably slower than the Texas A&M supercomputer, but the power consumed and the cost of the Pis is also significantly lower than that of the supercomputer node. This justifies the performance gap between the two systems. However, in order to used the cluster as a teaching aid, functionality is more important than the raw performance, and the tests demonstrate that the cluster is a good abstraction of a high cost cluster computer, albeit at a smaller scale. The similar scaling patterns of the supercomputer and the Pi cluster attest to this observation.

The High Performance Linpack benchmark tests performed very well on the Pi, cluster, too. As shown in 7, the Pi was able to achieve a performance of 3881 MFLOPs while using 10 CPUs which puts its performance significantly better than the Intel i7 3930K processor [5], which is a Hexa core processor, with each core clocked at 3.2 GHz, for a comparable price range at the time of writing this paper. The

	Pi Cluster 1st gen (4 nodes)	**Pi-Crust** (10 nodes)
Power	15.6 Watts	20/27 Watts
Performance	836 MFLOPS	3881 MFLOPS
Price	$250.00	$600.00
MFLOPS/W	54.8	165.2
MFLOPS/$	3.4	6.5

Fig. 8. Pi-Crust analysis verses version 1 Raspberry Pi clusters as found in [6].

	Pi-Crust ARM11, 32bit, Double Precision	**Yellowstone** Sandy Bridge Xeon, 64 bit, Double Precision
Power	20/27 Watts	1.4 MWatts
Performance	3881 MFLOPS	1.2 PFLOPS
Price	$600.00	$22,500,000.00
MFLOPS/W	165.2	875.3
MFLOPS/$	6.5	57.2

Fig. 9. Pi-Crust analysis Yellowstone supercomputer, as found in [6].

advantage of using a Pi cluster over a general purpose Intel processor is the customizability that comes with being able to control the hardware at will.

V. Challenges

In this section, we present some of the problems and challenges that we encountered during different phases of the project:

- Code corruption: While testing the raspberry pi cluster computer, some nodes did not perform any computation. After troubleshooting and debugging the issue, we found that the problem was caused due to corrupted code files inside the SD-card of those particular nodes. Since the cluster is running the same code inside every node, it is required that every node have the same non-corrupted copy of that file for the cluster to function correctly.
- Supercomputer configuration: The performance of the Pi cluster was to be compared against the performance of the Texas A&M supercomputer and it was essential to have the same configurations on both machines. The same versions of Python and MPI4Py were installed in a virtual environment in the supercomputing Eos cluster. During the execution of the programs on the supercomputing cluster, we observed that the programs ran correctly in the interactive mode, but when a batch job was submitted, the cluster ran p individual instances of the code, rather than running one process with p threads. This happened consistently for some time and it was difficult to figure

out how to debug the issue. We finally got help from the supercomputing facility staff and realized that we were loading OpenMPI for the batch jobs, and Intel MPI for the interactive jobs and the two versions of MPI were causing the difference in performance.

- Supercomputer HPL configuration: We ran the High Performance Linpack (HPL) benchmark on the Pi cluster and wished to run the same benchmark on the Eos supercomputing cluster. However, the configurations for the Linpack benchmark on the Eos cluster (LinpackX) were very different from the HPL configuration and it was difficult to know if they were giving results for the same set of parameters. This was the reason that the Linpack benchmarks for the Eos cluster were not reported in the results.
- Raspberry Pi HPL guidelines inaccurate: It took us considerable time and effort to setup and run the HPL Linpack on the Pi cluster. The documentation available to setup the HPL Linpack was incomplete and inconsistent. It took us multiple tries with the setup to troubleshoot and fix all the issues.
- Random reboots: This problem happened occasionally while performing long benchmark runs. It may have to do with processor overheating. However, the reboots were not that frequent and did not recur after the first night. If heat ever did become a problem it would be a simple thing to attach small fans (even USB) or even heat sinks to the processors.
- HPL configuration issues: In order to get the best possible results out of HPL benchmark, we had to fine tune several configuration parameters. There is no formula mentioned anywhere in the guidelines to get the best performance out of the cluster. Through extensive testing we found good values but challenges arise with not knowing all the details of the HPL testing benchmark. The goal is giving the processors as much as they can handle and recording what performance they achieve under a maximum load. Sometimes trial and error is the best plan to try for the best result. The guidelines tell you to try changing them until better results are achieved. With a better understanding of the parameters and how data is being calculated and swapped around on the processors achieving the ideal result becomes guesswork to some extent.
- Benchmarking took considerable time: The HPL benchmark configurations to achieve the best results (Higher Mflops) took notoriously long to complete each run. The better the results we wanted to produce, the longer the benchmark would have to be run. A lot of time was invested in benchmarking due to this situation.

VI. Future Work

The Raspberry Pi cluster has great potential for future work.

- More functionality can be added to the Pi in terms of additional programming languages supported by the Pi for programming using MPI.
- An additional Solid State Disk could be added to the cluster which could behave as a shared memory for the cluster opening up avenues for other paradigms of parallel programming, like OpenMP.
- The cluster can be further parallelized by running one thread of the process on each of the four cores of every Raspberry Pi. In this project, we go up to the granularity of individual Pis as the programming was done in Python.
- The Pi cluster needs to be tested for non-trivial parallel programs to truly understand the communication overheads incurred due to the distributed nature of the cluster (no shared memory).
- Some mechanism for fault tolerance needs to be built into the system, either in the form of redundant nodes, or as some backup mechanism with a rollback feature to a consistent state. There were instances when Pis lost power due to faulty connections, resulting in reboots and erroneous test results.
- The entire cluster could be made more portable by building a case for the entire setup which could house the switches and the power strip as well.
- The Pis on the cluster were assigned static IPs for them to be able to communicate with each other. However, this disabled them from having internet access. In the future, the Pis could be connected over a router, enabling dynamic IPs, and subsequently, internet access.

VII. Conclusion

The main goal of this project was to build a Raspberry Pi cluster computer and compare its performance against a more robust supercomputer. We accomplished this goal and successfully built a fully functional ten node Raspberry Pi 2 Model B cluster computer. The results using the standard performance and included MPI python benchmarks demonstrate that the Raspberry Pi cluster is a very promising alternative to multi-core PCs and cluster computers commonly used in academia. Throughout this project we have also demonstrated that the Pi cluster can be constructed at a very economical cost. It would be beneficial to use these low cost cluster computers in academia as well as many other fields. Universities or research institutes that do not have the space or budget to acquire a large scale supercomputer could opt to build their own cluster computer made of Raspberry Pis. Individuals, hobbyists and even corporations should be taking advantage of these systems, either at the individual level or by putting cluster systems together. This project has been both fun and extremely educational for us and we know it could be for others as well.

Acknowledgment

We would like the thank Dr. Daniel A. Jimnez at Texas A&M University for his funding and support of this project.

REFERENCES

[1] P. Abrahamsson, S. Helmer, N. Phaphoom, L. Nicolodi, N. Preda, L. Miori, M. Angriman, J. Rikkila, Xiaofeng Wang, K. Hamily, and S. Bugoloni. Affordable and energy-efficient cloud computing clusters: The bolzano raspberry pi cloud cluster experiment. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 2, pages 170–175, Dec 2013.

[2] Simon J. Cox, James T. Cox, Richard P. Boardman, Steven J. Johnston, Mark Scott, and Neil S. O'brien. Iridis-pi: A low-cost, compact demonstration cluster. *Cluster Computing*, 17(2):349–358, June 2014.

[3] S. Djanali, F.X. Arunanto, B.A. Pratomo, H. Studiawan, and S.G. Nugraha. Sql injection detection and prevention system with raspberry pi honeypot cluster for trapping attacker. In *Technology Management and Emerging Technologies (ISTMET), 2014 International Symposium on*, pages 163–166, May 2014.

[4] Joshua Kiepert. Creating a raspberry pi-based beowulf cluster. *Boise State University*, pages 1–17, 2013.

[5] Roy Longbottom. Linpack benchmark results on pcs. http://www.roylongbottom.org.uk/linpack%20results.htm, May 2015.

[6] Justin Moore. Performance benchmarking a raspberry pi cluster. https://www2.cisl.ucar.edu/siparcs/calendar/raspberry-pi-benchmarking-performance, 2014.

[7] L.H. Nunes, L.H. Vasconcelos Nakamura, H. De F Vieira, R.M. De O Libardi, E.M. de Oliveira, L. Junqueira Adami, J.C. Estrella, and S. Reiff-Marganiec. A study case of restful frameworks in raspberry pi: A performance and energy overview. In *Web Services (ICWS), 2014 IEEE International Conference on*, pages 722–724, June 2014.

[8] A. Petitet, J. Whaley, R.C.and Dongarra, and A. Cleary. Hpl faq. http://www.netlib.org/benchmark/hpl/faqs.html, September 2008.