EXPLOITING RASPBERRY PI CLUSTERS AND CAMPUS LAB COMPUTERS FOR DISTRIBUTED COMPUTING

Jacob Bushur and Chao Chen

Department of Electrical and Computer Engineering,
Purdue University Fort Wayne Fort Wayne, Indiana, USA

ABSTRACT

Distributed computing networks harness the power of existing computing resources and grant access to significant computing power while averting the costs of a supercomputer. This work aims to configure distributed computing networks using different computer devices and explore the benefits of the computing power of such networks. First, an HTCondor pool consisting of sixteen Raspberry Pi single-board computers and one laptop is created. The second distributed computing network is set up with Windows computers in university campus labs. With the HTCondor setup, researchers inside the university can utilize the lab computers as computing resources. In addition, the HTCondor pool is configured alongside the BOINC installation on both computer clusters, allowing them to contribute to high-throughput scientific computing projects in the research community when the computers would otherwise sit idle. The scalability of these two distributed computing networks is investigated through a matrix multiplication program and the performance of the HTCondor pool is also quantified using its built-in benchmark tool. With such a setup, the limits of the distributed computing network architecture in computationally intensive problems are explored.

KEYWORDS

Distributed Computing, Single-Board Computers, Raspberry Pi.

1. INTRODUCTION

Biology, chemistry, computer science, engineering, meteorology, and additional fields of study share the need to process large amounts of data. From protein folding simulations to sub-atomic chemical reaction simulations to analyzing large data sets, many fields of research require an increasing amount of computational power. There are currently two ways to meet this rising demand - supercomputers and distributed computing networks. Supercomputers are essentially large computers composed of many processors, typically housed compactly within an array of server cabinets. The compact arrangement of processors introduces challenges such as hardware, cooling, space, and electricity costs. These costs can be illustrated using an existing supercomputer, Summit, at the Oak Ridge National Laboratory. The Summit supercomputer is composed of 4,608 compute servers, each with two IBM POWER9 22-core processors and six NVIDIA Tesla V100 graphics processing unit (GPU) accelerators [1,2]. Albeit being a 200-pertaflop machine, representing the capability to perform 200 quadrillion floating-point operations per second, Summit comes with a total cost of $325 million, covering its processors, memory, networking equipment, and other necessary hardware. The supercomputer also requires approximately 13 MW of power to operate, including both the power consumption of the hardware and the power usage of the cooling solution. Additionally, the supercomputer occupies

5,600 square feet. Therefore, in order to build and operate a supercomputer, an institution must invest in the hardware, find a space large enough to house the supercomputer, and pay the recurring electricity costs. Thus, operating a supercomputer is cost-prohibitive for most institutions. As an alternative, researchers may rent time on supercomputers, but the cost can also be excessive. For example, one-month usage of Hewlett Packard Enterprise's smallest GreenLake cloud supercomputer platform costs $35,883 [3].

Distributed computing networks offer a potentially cheaper alternative without sacrificing computing power. Distributed computing is based on the premise that many smaller computers can be networked together, forming a virtual supercomputer. To run a program on a distributed computing network, the network manager must first split the program into a collection of jobs. Each job is sent to a computer connected to the network. The results are then returned to the central manager and presented to the user. These networks are typically configured using specially developed middleware such as BOINC [4], Folding@Home [5], or HTCondor [6], and are much cheaper to maintain because of their distributed architecture. Since the middleware takes care of connecting and managing computers over a network, operators almost entirely circumvent the hardware, electricity, space, and cooling costs. Additionally, computers on the network can perform valuable work while otherwise sitting idle. Although they may be cheaper to operate, distributed computing networks are not necessarily less powerful than supercomputers. For example, the statistics from Folding@Home project shows ever growing collective computing power from contributors around the world, with a total processing power of about 2.4 exaFLOPS in 2020, making it more powerful than the top 500 supercomputers combined [7]. Thus, distributed computing networks harness the power of existing computing resources and grant access to significant computing power while averting the costs of a supercomputer.

Both supercomputers and distributed computing networks offer substantial computing power to researchers in various fields. However, for many organizations and researchers, the costs associated with building and maintaining or renting a supercomputer make distributed computing networks a more attractive option. An important goal of this work is to explore the benefits of distributed computing in two different computer clusters. First, an HTCondor pool consisting of sixteen Raspberry Pi single-board computers and one laptop is created. Distributed computing networks are well suited to problems that can be divided into a collection of smaller, independent computations. However, not all problems can be separated in such a fashion. Therefore, the scalability of distributed computing systems in problems like matrix computation is explored in the Raspberry Pi clusters.

The Raspberry Pi is a low-cost single board computer with convenient sensors and controller interfaces, as well as connectors to support various peripherals and high network connectivity. Despite its small physical size, the Raspberry Pi's computational performance, especially of the newer models equipped with more powerful ARM processors, can reach the gigaFLOPS level. The Raspberry Pi has been a popular device to support edge computing in internet-of-things (IoT). The computing power can be further boosted by creating clusters with multiple Raspberry Pi systems. In recent years, the CPU and energy consumption of various Raspberry Pi clusters was evaluated using the HPL benchmark suite [8], a portable version of the LINPACK linear algebra benchmark for distributed-memory computers. Example Raspberry Pi clusters include a cluster with 25 Raspberry Pi Model 2B boards in [9] and clusters with 16 Raspberry Pi Model 3B boards in [10]. In addition, a cluster with 20 Raspberry Pi Model B+ boards was built with the evaluation focused on cryptography libraries [11].

In our case, we use a matrix multiplication program to evaluate the calculating performance of the Raspberry Pi cluster. Matrix multiplication is widely used in various computing problems

such as linear systems solutions, coordinate transformation, and network theory. It is also one of the most fundamental and computing intensive operations in machine learning. Evaluating matrix multiplication can shed light on the capability and scalability of edge devices in artificial intelligence applications such as federated learning in IoT networks [12].

The second distributed computing network is set up with Windows computers in university campus labs, where each computer is configured to only execute HTCondor jobs after sitting idle for a set time. The power of distributed computing networks in computationally intensive problems is explored in both networks. The same matrix multiplication test was done on the desktop computer clusters.

In addition to evaluating the computing power of the distributed computing networks, another goal of this project is to configure distributed computer clusters to contribute to scientific computing both locally and around the world. For this purpose, a guideline is created, allowing campus users to securely access to the HTCondor central manager, submit computing jobs, and retrieve results once done by the HTCondor pool. Moreover, the HTCondor pool is configured alongside the BOINC installation on both computer clusters, allowing them to contribute to community research projects such as the IBM World Community Grid [13], when the computers would otherwise sit idle.

The remainder of this paper is structured as follows. Section 2 provides an overview of HTCondor and BOINC, the middleware used to set up the distributed computing clusters. Section 3 explains the steps of setting up and configuring Raspberry Pi single-board computers as well as campus lab computers as distributed computing networks. Section 4 evaluates the computing performance of the two distributed computing clusters through a matrix multiplication program. Finally, Section 5 summarizes our effort and discusses future work.

2. DISTRIBUTED COMPUTING MIDDLEWARE

To form independent computers together as a distributed network, middleware is needed to provide services to enable the various components of a distributed system to communicate and manage data, so that the computing power of the connected machines can be effectively harnessed to create a high-throughput computing environment. It is software that lies in between the operating system and applications, providing programming language and platform independence, as well as code reuse. This section introduces two types of middleware, HTCondor and BOINC, that are used to set up the distributed computing networks in this project.

2.1. HTCondor

HTCondor is developed by a team at the University of Wisconsin–Madison and is freely available for use [14]. Users and administrators may tailor the program's behaviour to unique systems, networks, and environments with great flexibility.

HTCondor is a workload management system that provides scheduling policy and priority, as well as resource monitoring and management [15]. The major processes of an HTCondor system are depicted in Figure 1. The user submits jobs to an agent. This agent keeps the jobs in persistent storage while looking for resources willing to run them. Agents and resources advertise themselves to a matchmaker, which then introduces potential compatible agents and resources. Once introduced, an agent will contact a resource and verify if the match is still valid. To execute a job, both the agent and resource must start a new process. At the agent, a shadow provides all of

the necessary details to execute a job. At the resource, a sandbox creates a safe environment to execute the job.

Figure 2 shows the state machine for the HTCondor program. The following description gives a broad overview of HTCondor's behaviour [15].

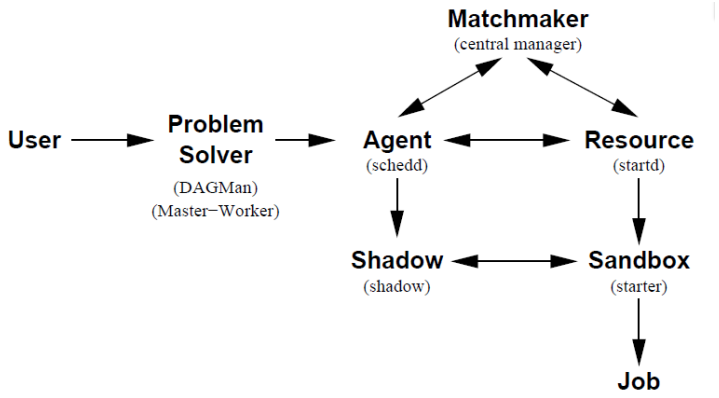


Figure 1. HTCondor kernel [6]

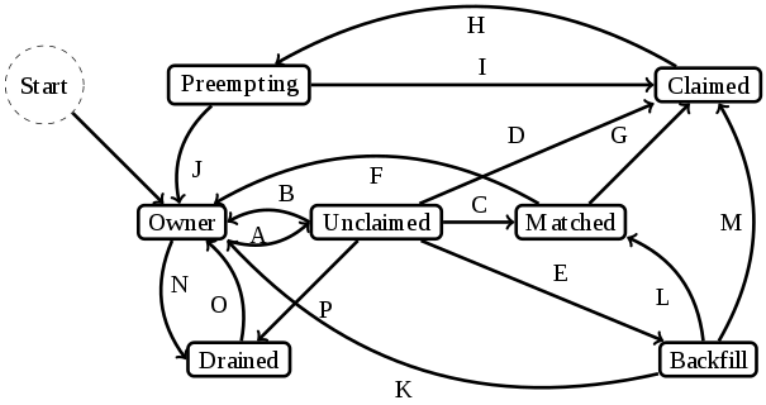


Figure 2. HTCondor program state machine [15]

When HTCondor starts, it immediately enters the *Owner* state. In this state, a user is controlling the computer, or, more generally, the machine is unavailable to run any HTCondor jobs. There are only two transitions out of the *Owner* state: HTCondor can initiate draining (N) to transition to the *Drained* state, or the START expression evaluates to TRUE (A) allowing a transition to the *Unclaimed* state.

One feature of HTCondor includes partitioning system resources into slots. A job may then be written to request resource slots from a machine such that only a fraction of the system's resources is occupied. However, draining, the process of pooling resource slots, is required if a job requires multiple resource slots. This is the purpose of the *Drained* state. Once HTCondor has finished pooling system resources, it transitions to the *Owner* state (O).

On the other hand, if the START expression evaluates to TRUE (A), that is, the machine is available to run HTCondor jobs, HTCondor enters the *Unclaimed* state where it sits idle. There are five transitions from the *Unclaimed* state: 1) the user can control the computer causing the START expression to evaluate to FALSE (B), 2) HTCondor can initiate draining (P), 3)

HTCondor can match an available job with the system's resources entering the *Matched* state (C), 4) HTCondor can match an available job with the system's resources entering the *Claimed* state (D), and 5) the START_BACKFILL expression evaluates to TRUE (E).

While in the *Matched* state, HTCondor begins the process for officially claiming a job and assigning it the machine's resources. There are only two transitions from the *Matched* state. HTCondor can complete the claiming protocol (G), or the START expression evaluates to FALSE (F) indicating that a user is now controlling the computer. In rare cases, HTCondor may immediately enter the *Claimed* state, bypassing the *Matched* state when certain background processes (condor_startd and condor_schedd) do not receive the match notification in the correct order.

In the *Claimed* state, HTCondor executes the job assigned by the central manager server. The only transition from the *Claimed* state is to *Preempting* state (H), typically meaning either that the job has finished or that a user is controlling the computer.

In the *Preempting* state, HTCondor will decide how to assign the system's resources after executing a job and exiting the *Claimed* state. If a job with a higher priority is available, HTCondor will reassign the system's resources and reenter the *Claimed* state (I). If a user is controlling the computer again, HTCondor will halt execution of all jobs and enter the *Owner* state (J).

Finally, in the *Backfill* state, HTCondor will utilize otherwise idle system resources to execute some pre-configured tasks. There are three transitions out of the *Backfill* state. HTCondor can match an available job with the system's resources and enter the *Matched* state (L) or *Claimed* state (M), or it will enter the *Owner* state after the START expression evaluates to FALSE (K) indicating that a user is now controlling the computer.

In our work, HTCondor is set up and configured on both the computer that serves as the central manager and the clients (i.e., Raspberry Pis and lab computers) that serve as computing resources. Users can submit computing jobs to the central manager, and if a client is not under control by any user for a set time, they are available to run HTCondor jobs.

2.2. BOINC

BOINC is an open-source middleware system for volunteer computing by a team at University of California, Berkeley [4]. Figure 3 shows the components of the BOINC system.

Volunteers install the BOINC client application on a computing device, choose to support one or more projects listed on the BOINC website [16], then attach the BOINC client to projects through the account managers. Each project has a server that distributes jobs to volunteer devices. Once attached to a project, the BOINC client periodically issues remote procedure calls (RPCs) to the project's server to report completed jobs and retrieve new job assignments. The client then downloads application and input files, execute jobs, and uploads output files. The BOINC client runs jobs at the lowest process priority and limits the total memory footprint, with the purpose of computing invisibly to the volunteer.

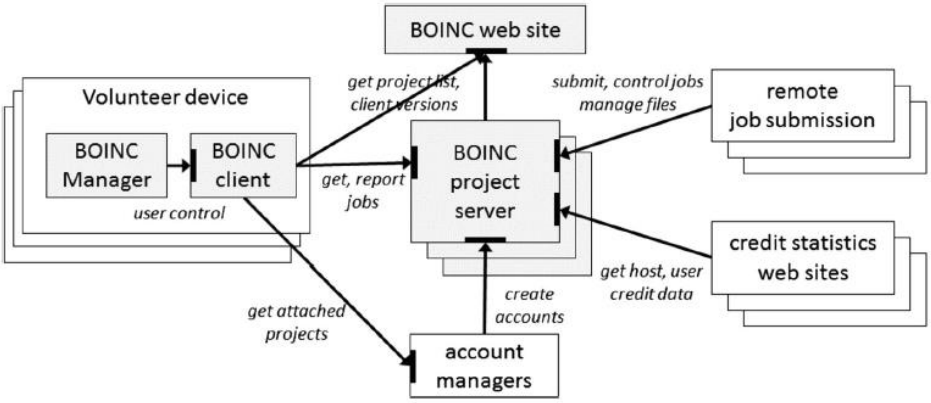


Figure 3. The components and RPC interfaces of the BOINC system [4]

The BOINC client software consists of three programs that communicate via RPCs over a TCP connection: 1) The core client manages job execution and file transfers. The BOINC installer initiates the client to run when the computer starts up. It exports a set of RPCs to the GUI and the screensaver. 2) A GUI (the BOINC Manager) assists volunteers in controlling and monitoring computation of individual jobs. 3) An optional screensaver shows application graphics for running jobs when the computer is idle.

In our work, the BOINC client software is installed on the Raspberry Pis and lab computers, making them as volunteering computing devices. BOINC client is set up as a pre-configured *Backfill* task to contribute to the IBM World Community Grid project. Therefore, if the computers are not under control by a user for a set time, and there is no current HTCondor jobs to run, they will contribute a promised portion of computing resources to the World Community Grid project.

3. COMPUTER CLUSTER CONFIGURATION

In this section, the procedures of configuring the middleware on the Raspberry Pi single-board computers as well as the campus lab computers to form distributed computing networks are explained.

3.1. Configuring the Raspberry Pi Cluster

3.1.1. HTCondor Installation and Verification

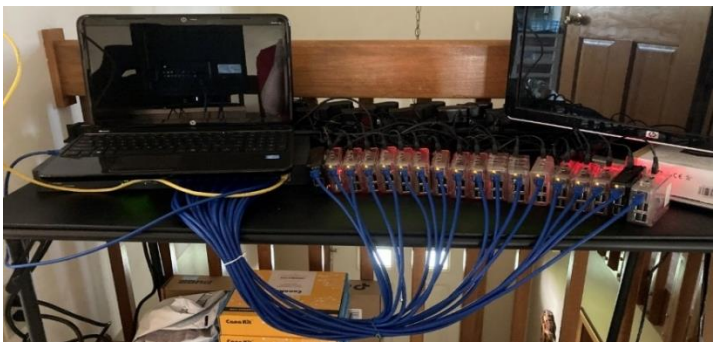


Figure 4. Raspberry Pi HTCondor pool

Our first distributed computing cluster consists of sixteen Raspberry Pi single board computers and an HP Pavilion laptop, interconnected with a TP-Link Gigabit Ethernet switch. This cluster is depicted in Figure 4. Among the single board computers, there are eleven Raspberry Pi 2B, four Raspberry Pi 3B, and one Raspberry Pi 3B+ systems, each with a quad-core ARM processor and Ubuntu Server 20.04 LTS flashed to the SD card. The laptop is installed with Ubuntu Desktop 20.04.

HTCondor is installed on both the laptop, configured as the central manager, and the Raspberry Pis, as the clients. A central manager has the ability to submit jobs to the pool, allow jobs to be executed using its resources, and add security configurations (e.g., password) to the pool. To run any application using this distributed computing network, a user must have access to the laptop. Each Raspberry Pi is configured as an HTCondor execute node, allowing jobs to be performed using its resources. After configuring the Raspberry Pis and the laptop, the status of the pool was queried using the condor_status command.

To test the HTCondor pool, a simple test program called sleep.sh was written and submitted. This program made an executing machine sleep for six seconds and print its hostname. By checking the hostname that was returned, the machine executing the job could be identified, and the pool's functionality could be verified. To execute this program on the pool, the submit file in Figure 5 was written.

```
# sleep.sub -- simple sleep job

executable              = sleep.sh
log                     = sleep.log
output                  = outfile-$(Process).txt
error                   = errors-$(Process).txt
requirements            = (TARGET.Arch == "armv7l")
should_transfer_files   = Yes
queue 100
```

Figure 5. Test program submission file

According to the submission file, the sleep.sh program was run 100 times. Each time sleep.sh ran, stdout was redirected to an output file. Similarly, stderr was redirected to the error file. The program would only run on computers with an “armv7l” architecture, in our case the Raspberry Pis. Additionally, any necessary files would be transferred to the executing machine. The job was then submitted to the pool for execution. The status of the pool during the job was monitored using condor_status and condor_q commands. The output of running these commands is shown in Figure 6. The output files were verified to contain the hostname of every Raspberry Pi in the HTCondor pool. Note that the status of the processor cores of the laptop and only one Raspberry Pi is included; the rest are omitted due to space.

```
$ condor_submit sleep.sub
Submitting
job(s).....................................................................................
..................
100 job(s) submitted to cluster 19.

$ condor_status
Name                  OpSys      Arch   State     Activity LoadAv Mem   ActvtyTime

slot1@htcondorcm.Home LINUX      X86_64 Unclaimed Idle      0.020  956  0+22:44:43
slot2@htcondorcm.Home LINUX      X86_64 Unclaimed Idle      0.000  956  0+22:45:08
slot3@htcondorcm.Home LINUX      X86_64 Unclaimed Idle      0.000  956  0+22:45:08
slot4@htcondorcm.Home LINUX      X86_64 Unclaimed Idle      0.000  956  0+22:45:08
slot1@pfwpi2.Home     LINUX      armv7l Claimed   Busy      0.000  213  0+00:00:00
slot2@pfwpi2.Home     LINUX      armv7l Claimed   Busy      0.000  213  0+00:00:03
```

```
slot3@pfwpi2.Home      LINUX      armv7l Claimed   Busy      0.000  213  0+00:00:02
slot4@pfwpi2.Home      LINUX      armv7l Claimed   Busy      0.000  213  0+00:00:02
...               ...             ...             ...             ...           ...

                     Total Owner Claimed Unclaimed Matched Preempting Backfill  Drain

        X86_64/LINUX     4     0       0         4       0          0        0       0
        armv7l/LINUX    64     0      64         0       0          0        0       0

               Total    68     0      64         4       0          0        0       0
$ condor_q
-- Schedd: htcondorcm.Home : <192.168.0.98:9618?... @ 10/04/21 23:12:31
OWNER    BATCH_NAME       SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
htcondor CMD: sleep.sh  10/4  23:11     59     41       _     100 19.0-99

41 jobs; 0 completed, 0 removed, 0 idle, 41 running, 0 held, 0 suspended
```

Figure 6. Pool configuration verification and testing result on the Raspberry Pi cluster

In addition to verifying and testing the HTCondor pool, the pool's performance was also quantified. Using the condor_status command, the MIPS (millions of instructions per second) rating for each CPU core was calculated. In total, the HTCondor pool is capable of 232200 MIPS. For reference, this score is approximately double that of an AMD 5950x (101163 MIPS), one of the most powerful consumer processors available today [17].

3.1.2. BOINC Installation and Verification

In addition to installing HTCondor, the Raspberry Pis were set up with BOINC running simultaneously as a pre-configured *Backfill* task to contribute to the IBM World Community Grid project. After installing the BOINC package, several configuration files in the /etc/boinc-client were edited. First, the remote hosts file (remote_hosts.cfg) was edited to allow a separate computer (i.e., the laptop) with a specified IP address to control the BOINC client remotely. After saving the remote hosts file, the core client configuration file (cc_config.xml) was edited to enable remote connections. Then, the password file (gui_rpc_auth.cfg) was edited with a password to authenticate remote BOINC connections. Finally, BOINCTasks was installed on the laptop specified in the remote hosts file to view the BOINC client installed on each Raspberry Pi. Figure 7 depicts all the Raspberry Pis (circled in red) are listed in the BOINCTasks program.

Figure 7. BOINCTasks computer view

The Raspberry Pis were also visible from the World Community Grid website [13] depicted in Figure 8 providing additional verification that they were contributing to research projects. The device names are circled in red – please note that not all devices are included due to space. Additionally, through the World Community Grid website, the behaviour of the devices can be

altered by editing device profiles, and new devices can be added by copying the account key and project URL.

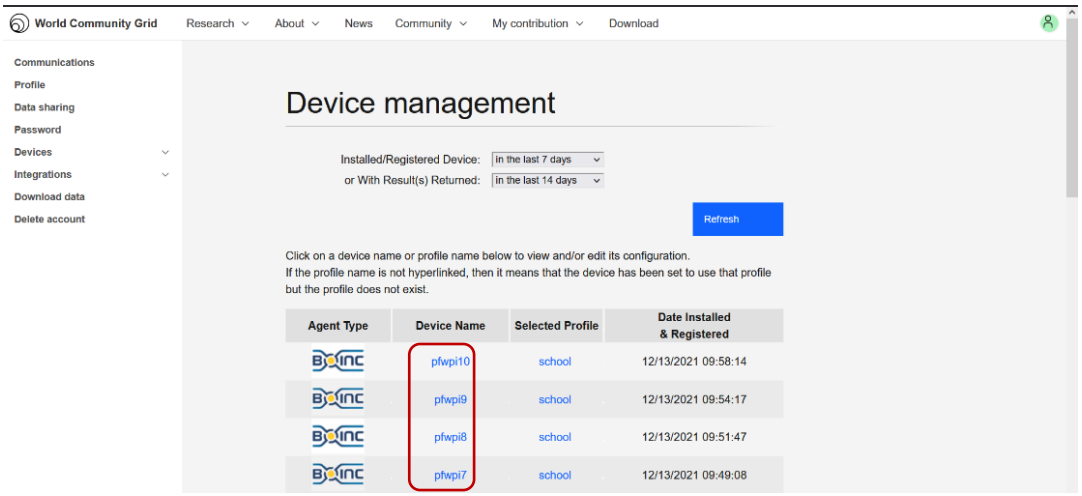


Figure 8. World Community Grid device management page

3.2. Configuring Campus Lab Computers

After successful installation of HTCondor and BOINC on the Raspberry Pi cluster, the next step is to configure the lab computers on campus as a distributed computing cluster. We obtained the permission to install HTCondor and BOINC on the Windows computers in the computer labs limited to one campus building. In these labs, there are 252 desktop computers with 1,140 physical CPU cores, contributing to a total computing capability of 19.872 teraFLOPS. This aggregated computing power is about 0.937% performance of the 500th supercomputer on the current Top500 most powerful supercomputer systems list [18]. In addition, a campus cloud server was reserved as the HTCondor central manager server.

Unlike the Linux version of the software, the Windows version of HTCondor comes bundled with an installer. This installer automatically configures the computer using a configuration file. However, some manual configuration (the same process used for the Raspberry Pi cluster) is still required to specify the security settings. After configuring HTCondor, the installation was verified by entering the `condor_status` command in the command line.

Additionally, a simple test program called sleep.bat was written and submitted to the HTCondor central manager. Similar to the sleep program in Section 3.1.1, this program made the executing machine sleep for six seconds and print its hostname. While the batch of jobs was running, the status of each CPU core on lab computers were monitored by entering the `condor_status` command again, with similar running result as in Figure 6. By observing each CPU core status, we verified that the software was installed and configured properly.

In addition to exploring the computing power of the distributed computing networks, another goal of this project is to configure distributed computer clusters to contribute to scientific computing both locally and around the world. For this purpose, a guideline is created to allow campus users to securely access to the HTCondor manager, submit computing jobs there, and retrieve results once the jobs are completed by the HTCondor pool. This provides a convenient way for campus users to utilize lab computers as powerful computing resources.

Besides HTCondor, BOINC was also installed on the lab computers. Like HTCondor, the Windows version of BOINC is accompanied by an installer which automatically configures the behaviour of the BOINC client. By checking the World Community Grid device management page, similar as in Figure 8, the BOINC installation could be verified.

After successful installation and configuration of HTCondor and BOINC, each lab computer connected to the distributed computing network is configured to always be in one of the following three states with decreasing priority: *owner* - under control by a legitimate user; *busy* - working on a submitted job to HTCondor pool; or *backfill* - contributing to World Community Grid.

Currently the office of information technology services on campus is in charge of managing and the distributed computing network with the lab computers. Security audits will be done periodically to make sure no intruders would access the computers and cause harm to the campus network. Additionally, information regarding the distributed computing network will be spread to faculty to raise awareness about the computing resource available for their research.

4. COMPUTING PERFORMANCE EVALUATION

To evaluate the computing power and the scalability of the two constructed distributed computing clusters, a simple matrix multiplication program was written and run using the HTCondor pool.This program computes the product of two square matrices of the same size. The program required three arguments to specify the dimensions of the matrix, number of CPU cores involved in calculation, and the portion of the product matrix to calculate. For example, if arguments 64, 1 and 0 were provided, a single CPU core would calculate the product of two 64x64 matrices. If arguments 128, 64, and 0 were provided, the program would split the multiplication of two 128x128 matrices into 64 parts and compute the first portion of the matrix product. Tests were conducted using 1, 2, 4, 8, 16, 32, and 64 cores to multiply two 1024x1024, 2048x2048, or 4096x4096 matrices on the two networks: Raspberry Pi cluster and the lab computer cluster. The time required for each test was determined by examining the timestamps of the first and last log file entries. For each combination of matrix size and number of cores, three trials were conducted on the Raspberry Pi cluster, whereas ten trials were conducted on the lab computer cluster. The tests were run when the clients are not under control of any users. The average execution time for each test was calculated, with results listed in Table 1.

Table 1. Matrix multiplication execution times on both distributed computing networks.

Matrix Size	**Cores**	**Average Time (MM:SS)**	
		Raspberry Pi Cluster	**Lab Computer Cluster**
4096x4096	64	01:02	00:17
	32	01:42	00:19
	16	02:41	00:40
	8	04:26	00:53
	4	08:17	01:24
	2	15:28	03:09
	1	27:47	05:31
2048x2048	64	00:25	00:07
	32	00:19	00:05
	16	00:20	00:08
	8	00:36	00:13
	4	00:59	00:15
	2	01:53	00:42
	1	03:35	00:59

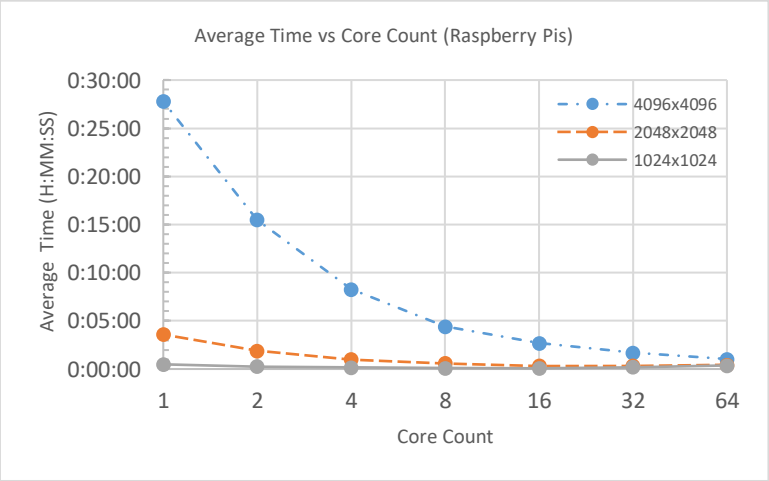
1024x1024	64	00:22	00:04
	32	00:12	00:03
	16	00:04	00:07
	8	00:06	00:06
	4	00:09	00:07
	2	00:16	00:08
	1	00:29	00:12

In general, as the number of cores increases, the execution time decreases. However, as the matrix size decreases and the number of cores increase, this relationship is no longer true. This observation is best visualized in Figure 9.

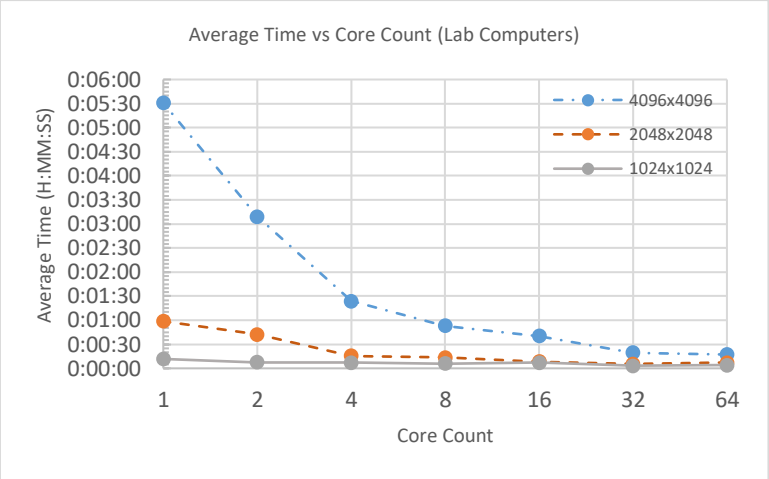
For 1024x1024 and 2048x2048 matrices, the optimum number of cores was 16 or 32 in the two distributed computing networks, not the largest number of cores (64). This counterintuitive observation is most plausibly explained by HTCondor's overhead. For HTCondor to complete a job, it must match the job with a client, send the program to execute, wait for the client to execute the program, and receive the computation results. The total time to complete a single job is expressed using (1).

$$T_{total} = T_{match} + T_{send} + T_{exe} + T_{receive} = T_{overhead} + T_{exe} \quad (1)$$

Where T_{total} is the total job completion time, T_{match} is the time required to match a job with a client, T_{send} is the time required to send the client the required resources, T_{exe} is the time required by the client to complete the job, and $T_{receive}$ is the time required to send the results back to the central manager server. $T_{overhead} = T_{match} + T_{send} + T_{receive}$, referring to the total overhead time involved but not actually in executing the job itself.



(a)



(b)

Figure 9. Task completion time vs number of cores in (a) the Raspberry Pi cluster and (b) the lab computer cluster

Assuming in the same distributed computing network, $T_{overhead}$ stay approximately constant for different jobs, the time benefits of a distributed computing network may be predicted based on the relationship between $T_{overhead}$ and T_{exe}. For example, to execute the same job, assuming highly parallelizable, if the total number of computing cores in a distributed computing network is doubled, the job execute time T_{exe} will be halved. In the limit, if T_{exe} is significantly longer than $T_{overhead}$, the total job completion time approaches ½ of before, which is the ideal result of doubling the number of cores. On the other hand, if $T_{overhead}$ is comparable to or longer thanT_{exe}, the work will not benefit much from additional cores and further division into smaller jobs. Additionally, if T_{exe} is too small, meaning the job is simple and can be done extremely fast, the execute nodes could return results before the central manager has completed the match process for the remaining jobs. In this scenario, the total completion time for a batch of jobs could increase. Consequently, the work will only benefit from additional cores and further division into smaller jobs if $T_{overhead}$ is significantly shorter than T_{exe}.

5. CONCLUSIONS AND FUTURE WORK

In conclusion, an HTCondor pool was successfully created using a collection of sixteen Raspberry Pi single-board computers and one laptop. The pool's functionality was verified using a simple test job. Computing intensive programs such as matrix multiplication can be efficiently split into smaller jobs to be distributed to the Raspberry Pis and executed independently. The computing power of the Raspberry Pi cluster can be greatly boosted by forming a distributed computing network, making it suitable to carry out compute-intensive applications such as federated learning in power-constrained IoT devices.

A similar HTCondor pool was constructed to form a second distributed computing system composed of Windows computers in campus labs. As a result, university faculty and students have access to a significant amount of computing power by submitting computing jobs to the HTCondor central manager server. In addition to installing and configuring HTCondor, BOINC was also installed on the Raspberry Pi cluster and lab computer cluster as a *Backfill* task to contribute to scientific computing projects such as the IBM World Community Grid. In this configuration, researchers partnered with scientific computing projects associated with BOINC

will have more resources available from the community to help with high-demand computing needed for research such as climate, cancer, and vaccine study.

Additionally, the scalability of distributed computing networks in matrix multiplication was investigated on the Raspberry Pi cluster and lab computer cluster. In general, as the number of cores increases, the execution time decreases. However, as the matrix size decreases and the number of cores increase, this relationship is no longer valid. This counter-intuitive observation is most plausibly explained by HTCondor's overhead. Additionally, the relationship between HTCondor's overhead (primarily the time required to match jobs with resources) and the execution time of a job may predict the benefit of further parallelization. Regardless, the power of distributed computing systems is apparent: many small computers can create a body of computing power on par with most powerful consumer hardware available on the market.

Future directions of the work include evaluating different types of programs in distributed computing networks. Unlike matrix multiplication, not all compute-intensive programs can be parallelized easily. The power of distributed computing, especially of resource-constrained IoT devices like the Raspberry Pi should be evaluated for different types of computations (e.g., convolution and filtering) involved in edge computing. Furthermore, as security is always a concern for any distributed network, mechanisms such as encryption, authentication, and load balancing need to be carefully studied.

ACKNOWLEDGEMENTS

The authors would like to thank the Chapman Scholars Program, the Electrical and Computer Engineering Department, and the Office of Information Technology Services at Purdue University Fort Wayne for financial and technical support for this project.

REFERENCES

[1] S. S. Vazhkudai, *et al.* (2018) "The design, deployment, and evaluation of the CORAL pre-exascale systems." in *Proc. Supercomputing 2018 (SC18): 31th Int. Conf. on High Performance Computing, Networking, Storage and Analysis*, Dallas, TX.

[2] Summit - Oak Ridge leadership computing facility, [online]. Available: https://www.olcf.ornl.gov/summit/ (accessed May 2022).

[3] HPE GreenLake for high performance computing platform, [online]. Available: https://www.hpe.com/us/en/greenlake/hpc.html (accessed May 2022).

[4] D. P. Anderson, (2020) "BOINC: A platform for volunteer computing," *Journal of Grid Computing*, vol. 18, pp. 99-122, doi: 10.1007/s10723-019-09497-9.

[5] S. M. Larson, C. D. Snow, M. Shirts, and V. S. Pande, (2009) "Folding@Home and Genome@Home: Using distributed computing to tackle previously intractable problems in computtaional biology," *arXiv preprnt*, doi: 10.48550/arXiv.0901.0866.

[6] D. Thain, T. Tannenbaum, and M. Livny, (2005) "Distributed computing in practice: The Condor experience," *Concurrency and* Computation*: Practice and Experience*, vol. 17, iss. 2-4, pp. 323-356, doi: 10.1002/cpe.938.

[7] H. Mujtaba, "Folding@Home now at almost 2.5 Exaflops to fight COVID-19 – Faster than top 500 supercomputers in the world," [online]. Available: https://wccftech.com/folding-home-almost-2-5-exaflops-fight-covid-19-faster-than-top-500-world-supercomputers/ (accessed May 2022).

[8] A. Petitet, R. C. Whaley, J. Dongarra, A. Cleary, "HPL – A portable implementation of the high-performance linpack benchmark for distributed-memory computers," [online]. Available: https://www.netlib.org/benchmark/hpl/ (accessed May 2022)

[9] M. F. Cloutier, C. Paradis, and V. M. Weaver, (2016) "A Raspberry Pi cluster instrumented for fine-grained power measurement," *Electronics*, vol. 5, no. 4, 61, doi: 10.3390/electronics5040061

[10] P. J. Basford *et al.*, (2020) "Performance analysis of single board computer clusters," *Future Generation Computer Systems*, vol. 102, pp. 278–291, doi: 10.1016/j.future.2019.07.040

[11] D. Hawthorne, M. Kapralos, R. W. Blaine, and S. J. Matthews, (2020) "Evaluating cryptographic performance of Rapsberry Pi clusters," in *Proc. 2020 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1-9, doi: 10.1109/HPEC43674.2020.9286247.

[12] S. Savazzi, M. Nicoli and V. Rampa, (2020) "Federated Learning with cooperating devices: A consensus approach for massive IoT networks," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4641-4654, doi: 10.1109/JIOT.2020.2964162.

[13] World Community Grid, [online]. Available: https://www.worldcommunitygrid.org/ (accessed May 2022).

[14] HTCondor Overview, [online]. Available: https://htcondor.org/htcondor/overview/ (accessed May 2022).

[15] Center for High Throughput Computing, University of Wisconsin–Madison, "Policy Configuration for Execute Hosts and for Submit Hosts — HTCondor Manual 9.4.0 documentation," [online]. Available: https://htcondor.readthedocs.io/en/latest/admin-manual/policy-configuration.html (accessed December 2021).

[16] BOINC: Compute for Science, [online]. Available: https://boinc.berkeley.edu, (accessed May 2022).

[17] AMD Ryzen 9 5950X Benchmarks, [online]. Available: https://openbenchmarking.org/vs/Processor/AMD%20Ryzen%209%205950X%2016-Core (accessed December 2021).

[18] TOP500 List - November 2021, [online]. Available: https://www.top500.org/lists/top500/list/2021/11/ (accessed May 2022).