Pyp2pcluster: A cluster discovery tool

Tracey, R., Akinsolu, M. O., Elisseev, V., and Shoaib, S.

pyp2pcluster: A cluster discovery tool

1st Robert Tracey
IBM Research Europe
Wrexham Glyndŵr University
Warrington, U.K.
robert.tracey@ibm.com

2nd Mobayode O. Akinsolu
Wrexham Glyndŵr University
Wrexham, U.K.
mobayode.akinsolu@glyndwr.ac.uk

3rd Vadim Elisseev
IBM Research Europe
Wrexham Glyndŵr University
Warrington, U.K.
vadim.v.elisseev@ibm.com

4th Sultan Shoaib
Wrexham Glyndŵr University
Wrexham, U.K.
sultan.shoaib@glyndwr.ac.uk

***Abstract*—It is becoming increasingly common for laboratories and universities to share computing resources. Also as cloud usage and applications continue to expand, a hybrid cloud working model is fast becoming a common standard practice. In line with these present-day trends, we present in this paper an open-source Python library that provides information on high performance computing (HPC) clusters and systems that are available to a user via a peer to peer (P2P) infrastructure. These metrics include the size of system and availability of nodes, along with the speed of connection between clusters. We will present the benefits of using a P2P model compared to traditional client server models and look at the ease in which this can be implemented. We will also look at the benefits and uses of gathering this data in one location in order to assist with the managing of complex workloads in heterogeneous environments.**

***Index Terms*—python, P2P, cluster, HPC**

I. Introduction

Building a high-performance computer (HPC) and the required infrastructure to support it is a costly endeavour. For example Fugaku supercomputer which topped the Top 500 list in June 2020[1], cost roughly 1.2 billion dollars to build and consumes 29,899 kW of electric power[2]. This is an extreme case as it is one of largest in the world but still requires a very large investment. On a smaller scale JADE 2 a recently built supercomputer in the UK for ground breaking work in areas such as energy storage and improved drug design, cost roughly £5 million to build[3]. Even for a much smaller system this is a very large investment and requires backing from multiple institutions to fund and then run. That is why more and more groups are being created that share HPC resources such as EuroHPC[4] and Supercomputing Wales[5]. Also there is a greater focus on cloud environments and configuring them to be cloud HPC environments [6], these obviously have the advantage of being scale-able in order to save costs. In a paper by Gupta and Milojicic[7] the use of cloud as a HPC resource was looked at and the costs involved and an evaluation was made. They found that for a lot of applications the cloud is a suitable alternative and can be lower cost, but in some cases a physical HPC will still perform better.

Another consideration when choosing clusters and nodes can be the network connection between them. Scheduling jobs among multiple clusters can be done at the via a master scheduler that is aware of multiple clusters [8]. This can be done for many reasons, including access to more resources, free spaces on clusters and in order to take advantage of specific hardware for different parts of the job. When this is done, the network connections between the clusters become a very important factor. How this is measured is often disputed, however metrics such as latency, link speed and the number of hops are quite popular for carrying out this evaluation. In one example latency was used to look at the connection speed between nodes within countries and between countries to identify slow and fast connections [9]. This data was able to be used to help with network design and towards advancement in internet technologies.

For an end user this creates an enviable problem where they have too many resources at their disposal and are unsure where there job should be run. Choosing the correct cluster and resources is very important as if a cluster is chosen with too few available nodes or nodes that are not powerful enough then a job can be significantly delayed or even fail. This can also go the other way if a too powerful cluster is used, then resources are wasted and queue times are lengthened for other users [10]. This is often referred to as heterogeneous environments, running complex workloads in these environments is enhanced if you have available to hand information such as how many nodes a cluster has and the resources available to it.

For the project *Dynamic routing of data based on resource availability and benefit* that is currently being worked on we required an easy to use python library that would allow us to get data from the various nodes and clusters and utilize it for other algorithms and code. In this other project we need to gather data concerning the size and specification from many clusters and individual nodes and the speed of the networks between them. We will then be applying fuzzy logic to the data received from them in order to classify them by quality. This data will then be fed into further investigations looking at how jobs can be routed and scheduled to run on the best available cluster using swarm intelligence. All of this data will be collected and processed automatically so that it is regularly updated. In order for this to work we had very specific needs in a Python library which needed to be met these were:

- Easily adaptable for the needs of our project
- Be able to provide information from different types of clusters
- Be able to work with single nodes
- To consume as few resources as possible

- To be able to obtain network speed information between nodes.
- To be able to provide detailed information on a cluster and resources

The outcome of this is pyp2pcluster library. We identified that this library has other uses as well outside our project which could be beneficial to many users. Examples of this are to keep track of the health of a cluster for system administrators, or to monitor network connections and quickly spot outages that may occur. As mentioned it is also useful for users with access to multiple clusters and the advantage of being able to see the resources on them can be great, along with the fact that it is scriptable. This makes it very versatile for a variety of needs and users.

In this paper we present pyp2pcluster, this is a python library that runs as a Peer-to-Peer (P2P) client on the gateway nodes for all clusters that a user has access to. It allows speedy reports of cluster status and available hardware and also a way of testing network speeds between clusters. The remainder of our paper is organized as follows: the differences between peer-to peer networks and client server networks are discussed in Section 2, job management in HPC is examined in Section 3, existing libraries available for Python are reviewed in Section 4, the library, its output when used and the features it has and how they can be used are presented in Sections 5 and 6, and the concluding remarks alongside the planned future direction for our work are detailed in Section 7.

II. Peer-to-Peer Systems

Peer-to-Peer (P2P) networks are based on a distributed environment and structure rather than a single node. These types of systems were very popular in the late 1990s and early 2000s as a way of sharing files anonymously with services such as Kazaa, Napster, Limewire and also BitTorrent. P2P networks are also a way of sharing computing resources in a distributed manner as demonstrated by networks such as SETI@Home [11].

P2P networks fall in to two main categories: structured networks and unstructured networks. In a structured network, the nodes form a specific topology rather than just being randomly connected and they use a distributed hash table (DHT). This allows for specific nodes to be searched for within the network. This type of network works best with a more static design, as each node has to keep track of neighbouring nodes in order to keep the DHT updated correctly[12]. In an unstructured network, nodes can leave or join at will, as no central directory is kept. If a search for a specific file is made then it has to go through the whole network. This routine can cause issues in larger networks as they can easily become flooded by searches[13]. However, this type of network is better at handling large amounts of nodes leaving and joining at any time which makes it very flexible.

A. *P2P vs Client Server*

There are many advantages of a P2P-based system in comparison to traditional client-server-based communication. One of the main advantages, in the context of our work, is that each node can request and send data as needed. Unlike in a client-server method where the clients request data from a central location (i.e., the server) and the server can only respond as requested and cannot receive data. Using a distributed P2P network also decentralizes the infrastructure that helps with costs, existing nodes not dedicated nodes can be used, and also increases resilience at no extra cost. Also if one node goes down the others can still handle requests and pass data to each other. Finally one of the other key features of P2P over client-server is that in a client-server network as soon as a connection is made, a chunk of bandwidth is reserved regardless of whether it will be fully utilized or even if needed at all until the connection is terminated. On the other hand, in a P2P network, the connection only uses as much bandwidth as required when required, which keeps the network running smoother, especially with multiple connected clients.

An example of the differences in layout between client server and P2P networks can be seen in Fig. 1. Fig. 1 highlights one of the major benefits of a P2P cluster over a client-server network - it is not reliant on one central server and all nodes can send and receive data.

III. HPC Job Management

Job management and scheduling software is a vital part of working with a HPC cluster. The job scheduling software has the important job of keeping track of what resources are available across the cluster, status of the various nodes and what hardware they have available. Also, it keeps track of what jobs are currently running, which are waiting and for which users and if that user has the ability to use this cluster. By using this software, a user is able to submit jobs to nodes via queues, find information on available nodes and on running job status. Some common examples of job schedulers are IBM SpectrumLSF[14], Slurm[15] and OpenPBS[16]. In Fig. 2, it can be seen how a user interacts solely with the gateway node. This in turn uses job scheduling software to place the users job on the next available node or nodes.

There are many types of algorithms used for job scheduling on HPC systems [17]; the simplest being first-come first-serve (FCFS). In FCFS, the jobs are queued up as they arrive regardless of size or priority. It is simple and works. but it is not very efficient and can lead to small jobs getting stuck behind larger jobs, even if there is space for them to run. Many other algorithms exist to help with the scheduling of HPC systems that can manage jobs in a more efficient way. A regular challenge for HPC administrators is to fill clusters in the most efficient way possible by finding the perfect balance between priority and job size. This is especially the case as increasingly an important metric when monitoring a cluster is its utilization and the patterns that form from users jobs [18]. This information is vital for decision making about resources and efficient allocation of users jobs. This information is also used by decision makers such as managers and laboratory leaders about future purchases in regards to clusters and the resources that will be required for future work. The study of

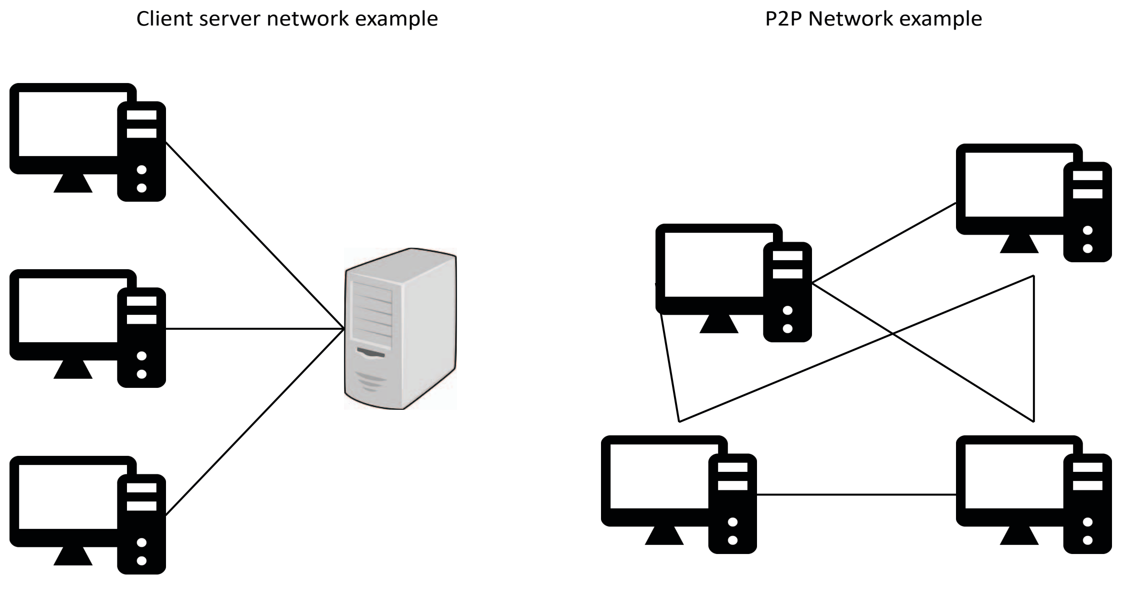

Fig. 1. Client server vs P2P networks

utilization has also led to research into how job scheduling and utilization can affect energy efficiency[19].

As well as knowing what management system a user has to interact with, the next challenge is ensuring that the cluster required resources available. These can include the type and number of central processing units (CPUs) available, available memory per node, the number of nodes, and the number of accelerators that are available. Accelerators are being used increasingly in HPC environments to enable specialist code to run faster and more efficiently than on a CPU alone. These include Field Programmable Gate Arrays (FPGA) [20], Graphics Processing Unit (GPU) [21] and three-dimensional (3-D) Memory [22].

As more systems at multiple locations are made available to users and they move to more heterogeneous environments. In order to support this job managers that support multiple clusters have started to be developed. Examples of these are Globus[23] and Flux[24]. These tools are able to see multiple clusters and schedule jobs on them from one central location.

IV. Existing Libraries

Many Python P2P libraries currently exist with features that could be used for the project this tool was built for. In order for these Python libraries to be suitable for us to work with, they need to be usable with the latest versions of Python, be in active development, have good documentation and be flexible so that they can be adapted to the challenge we have. We will look at some of the libraries we found in greater detail below. A brief list of them and their current status of development can be found in table I.

Library	Current status
pyp2p	Not active
pydevp2p	Not active
dispersy	Not active
trinity	Not active
py-libp2p	Active
Zyre	Active
libp2p	Active

TABLE I
Python P2P libraries

A. Non active libraries

Firstly. there is a *pyp2p* library[25], which is a simple working library that has abilities to bypass network address translation's (NAT) to make linking P2P networks easier. It has some documentation and some good example codes that allows potential users to demo on the same node if only a single node is being used for tests and experimentation. It also supports broadcast communication and direct communication which is useful if there is a need to switch the P2P network from structured to unstructured. The major drawback of the *pyp2p* library[25] is that it has not been updated in a long time, in this case, the last commit was 2016 and it has only been tested on Python 2.6 and 3.3. This means that it could have issues with modern libraries that require newer versions of Python and lots of work could end up being spent debugging these issues rather than doing actual development work.

The next library is the *pydevp2p* library[26]. This is a low-level library that was built to help with Ethereum mining, but could easily in theory, be adapted for other purposes. It

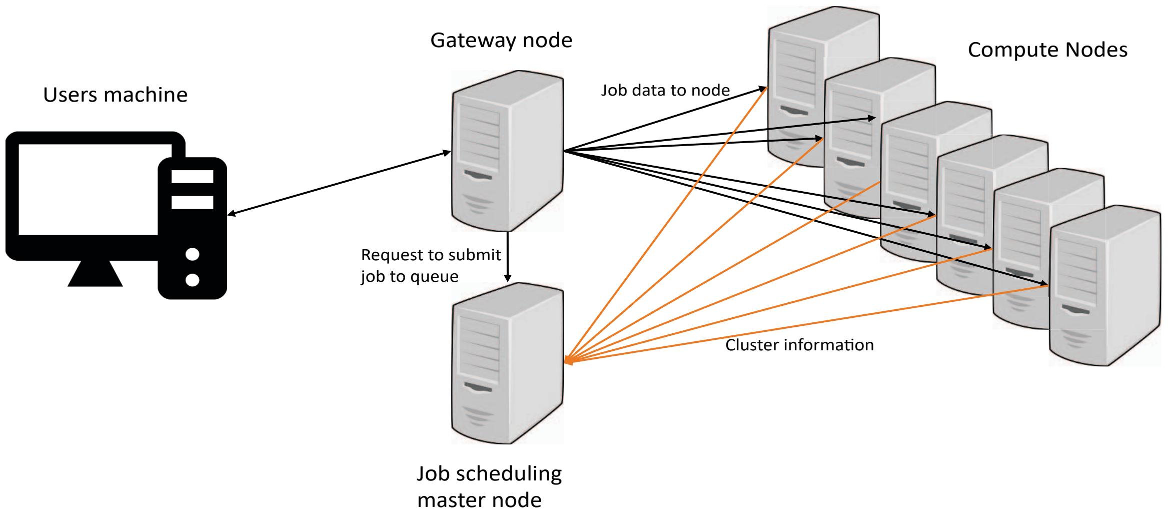

Fig. 2. Job management for HPC system

uses RLPx network layer, a specialist network layer built for Ethereum mining. The *pydevp2p* library[26] has many features such as built in encryption for both transportation of messages and handshakes between nodes which increases security and privacy across the network. Despite the aforementioned pros, the *pydevp2p* library[26] suffers from a few major issues. Similar to the pyp2p library[25], it has not been updated in a long time, in this case, since 2018. This suggests that it is not under active development. Secondly and most importantly, it has no documentation at all, indicating that potential users need to reach out to the developer directly or work through the library code to decipher how to use the library. As a result of these drawbacks, this library was not considered in our work which aims to the purposes of our project.

Another library examined is the *dispersy* library[27]. This library allows users to create a fully decentralized network that can scale up to hundreds of thousands of nodes. It also transverses easily over NAT's, making networking easier for users on organizational networks. For this library, each node runs all the algorithms needed, helping to keep it robust and an elliptic curve cryptography is used to keep communications secure. The $dispersy$ library[27] has been proven to work well over wireless fidelity (WiFi) and cellular networks, making it very practical for mobile devices and remote devices. However, unlike the previous two libraries, the development of the *dispersy* library[27] has ceased. As a matter of fact, on GitHub, there is a notice that no further development will happen to the library [27]. Hence, this library is not suitable for the aim of the work carried out in this paper.

The *trinity* library[28] was also reviewed. Like the $pydevp2p$ library[26], this library was also written to support Ethereum mining. However, in this case, it was more application-specific, so much so that it likely can not be used for anything else. The *trinity* library[28] has a lot of documentation, enabling potential users to be able to make use of it proficiently, relatively quickly. The major drawback of this library is that it is no longer under active development. Therefore, it is impractical to use it for our work.

All the libraries discussed above had various issues that made them unsuitable for use in our work. A common drawback to all the libraries is the lack of update and/or development[25, 26, 27, 28]. How often a library is updated is a critical factor when choosing what to use it, as this can impact the number of bugs that needs to be fixed before the library can be used. Aside from bugs, compatibility with other libraries and the current versions of Python, and the level of help and information available from other users and developers are also directly related to the updates and recurrent development of the library. In the next section, some other libraries that are in active development still are reviewed.

B. *Active libraries*

The *py-libp2p* library [29] is a library that is under very active development. It has many features, such as support for multiple transport protocols which makes it very flexible. It also has a good set of documentation that details how the library can be used and its many features accessed. The main cons of this library are its complexity, considering the intended use in our work, and too many features and settings that avert rapid development. The *py-libp2p* library [29] is also geared towards Ethereum mining who funds it, suggesting that it is leans more towards mining, rather than traditional P2P networks.

Another library under active development is the *Zyre* library[30] that is written in C. Despite this, this library can still be considered as it has bindings for many other languages, so that these other languages can use its interfaces and this includes Python. The *Zyre* library[30] has extensive documentation and many features such as groups for peers, ease of joining and leaving the network for nodes, and it can work on multiple operating systems. However, it was designed and built to run on local networks, rather than over the internet. This would make it useful for a single site with multiple nodes, but for our work, this is not suitable.

Finally we will look at libp2p[31] this is a very interesting and different library for a number of reasons. Firstly it is modular so it allows you to bring in only the parts you need and extend it as needed. It can use multiple transport types and supports encryption in communication between nodes. The main issue and blocker for the use in our work is that that although it has been released for a few languages the Python implementation is still being worked on and has therefore not been released yet. This obviously stops it from being useful for our project at this time.

From the review of the current libraries, it can be seen that none of them meet the required criteria for our work, as mentioned earlier in Section I. As a result, we have written and developed a P2P library in this paper to meet the said requirements. This allows for the flexibility of having a library that can be coded and documented for the specific versions of Python used in our work.

V. PYP2PCLUSTER

The *pyp2pcluster* library has been designed to be very simple to use and to be as flexible as possible. The library has been tested running on Python versions 3.6, 3.7 and 3.8, and it requires only one extra library to be installed, i.e., the *tcp_latency* library. Because the type of available resources and clouds to be used by a user changes very often, the library is designed as an unstructured P2P network, so no DHT exists for all nodes. In Fig. 3, the flow of data to the node is illustrated, a selection of choices are available to the user (more details on these choices are provided later on), and then the required data is returned to the master node.

Each node or cluster runs a script in the background which calls the library, and this does all of the sending and receiving of data. In the example shown in Listing 1, we have the code for a standard node. This code has three simple parts: the first couple of lines import the *pyp2pcluster* library and also the *sys* library to support command line arguments. Then variables for port and also the system ID are taken from the command line arguments when the script is executed. The *sys* is commonly the host name, but it can be anything in order to identify this cluster or node. This name is currently not used in the code but has been added in case DHT features ever needed to be introduced. The library is used to create a object that will loop and wait for connections to use the library. Once this is running, requests can be sent to this node and if needed, the node running this loop will request from other nodes as well.

```
from pyp2pcluster import pyp2pconn as pyp2p
import sys

port = int(sys.argv[2])
sysid = sys.argv[1]

loop = pyp2p(port,sysid)
loop.mainloop()
```

Listing 1. Example of creating a node using pyp2pcluster

A. Connecting to the nodes

Connecting to the nodes is a simple process due to the fact that each P2P node is just listening for text input. This is then used to process specific commands and actions. By default, if a message is sent that does not match an action that the node is aware of, then it echoes it back to the sender. This feature is useful for debugging and basic connection checking - a very basic form of this can be demonstrated using the *nc* command. The *nc* command demonstrates that a server is able to reply. It can be viewed as the simplest way to connect to a node and example can be seen in Listing 2.

```
echo "test_message" | nc 192.168.0.103 44444
```

Listing 2. Example of using nc command to connect to cluster

Listing 2 can be expanded further for use in Python which provides more opportunities for larger and more complex scripts to be written and developed. In Listing 3, a function that sends data, receives the basic reply and then stores it as a variable is being executed. As can be seen in Listing 3,the pickle library is used to decode any messages sent or received, as they are received in binary format.

```
import socket
import pickle

def testmessage(host, port):
    message = ["tester"]
    with socket.socket(socket.AF_INET, \
     socket.SOCK_STREAM) as s:
        s.connect((host, port))
        messageb = pickle.dumps(message)
        s.sendall(messageb)
        data = s.recv(2024)
        outy = pickle.loads(data)
    return outy

output = testmessage("192.168.0.103", 44444)
```

Listing 3. Example of creating a node using pyp2pcluster

Now that a basic connection has been made and tested as revealed in Listing 3, it is important to look at the specific messages that will run actual commands on the P2P nodes. This is discussed in Section V-B.

B. Cluster information

The cluster information call is done via a message that is made up of three parts and sent to the P2P node. The first part is the command *getcluster* - this tells the node that it is a

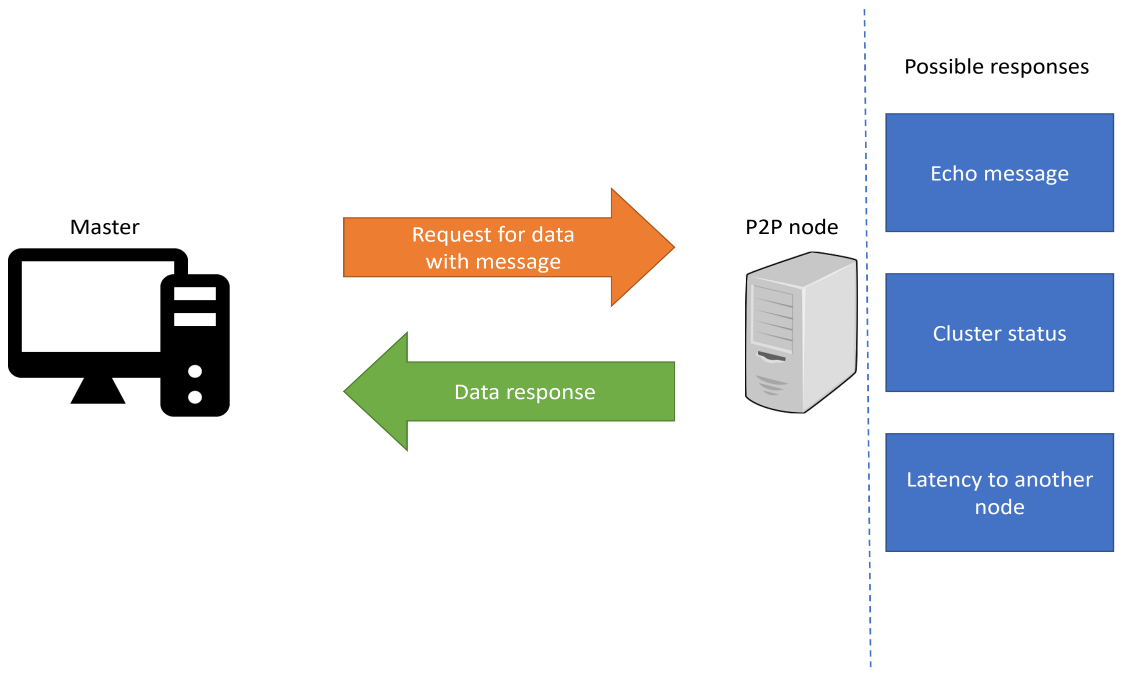

Fig. 3. communication to API

cluster information call, the second part is the type of cluster - this currently accepts three arguments: *lsf*, *slurm* and *single* (*lsf* and *slurm* correspond to the type of clusters that can be talked to if they are present on the node, and the library uses a combination of commands for each cluster type to generate an output, and *single* is for getting data from single nodes that can be used but have no job scheduling software installed).

The third part or option of the message used for the cluster information call sets how the data will be returned, and the options are *table* and *summary*. *table* gives a dictionary output of all the nodes and lists the characteristics for each one individually. This can then be used for scripts or converted to a dataframe, if working with a library like *pandas*. *summary* returns a summary with three values: the average number of CPUs over the whole cluster, the average memory across the cluster and the total number of nodes in the cluster. This was added for ease in scripting. Also one of the main purposes of this library was to support the implementation of fuzzy logic that fits appropriately to it. In Listing 4, it can be seen that a request is being made to an *lsf* cluster, for the data to returned in a table format.

```
import socket
import pickle

def clustertest(host, port, cluster,mode):
    message = ["getcluster",cluster,mode]
    with socket.socket(socket.AF_INET, \
     socket.SOCK_STREAM) as s:
        s.connect((host, port))
        messageb = pickle.dumps(message)
        s.sendall(messageb)
        data = s.recv(1024)
        outy = pickle.loads(data)
    return outy

clustertest("192.168.0.125", 44444,"lsf","table")
```

Listing 4. Example of requsting cluster information

C. Latency between clusters

The latency call is designed to test the latency between clusters. As a result, it does not test for latency between the nodes on the same cluster. It is designed in such a way that a P2P node or cluster can reach out to another, and carry out a latency test between them. This has several applications, but most importantly, it allows the development of a map of how connectivity looks between clusters. This map can prove useful for splitting jobs between clusters or if resources for a job such as a database are to be hosted elsewhere, then an idea of the quality of network between the clusters can be easily observed or visualized. Such an observation or visualization can help with the decision of which clusters to use or where resources should be hosted.

In Listing 5, an example of how to request the latency is shown. The message sent to the P2P node is made up of two parts: the command *nodetest* which tells the node that checks the latency on another node, and the secondary node to be targeted from the primary (this can be sent as a hostname or IP address; a hostname will only work if it can be resolved by the primary node either through lookup or DNS). The result is returned to the user as a float. If latency needs to be checked from the user's machine to any node, then this can easily be done without the library by just using the *tcp_latency* library to talk to the node directly.

```
import socket
import pickle

def nodetest(host, port, target):
    message = ["nodetest",target]
    with socket.socket(socket.AF_INET,\
     socket.SOCK_STREAM) as s:
        s.connect((host, port))
        messageb = pickle.dumps(message)
        s.sendall(messageb)
        data = s.recv(1024)
        outy = pickle.loads(data)
    return outy

nodetest("192.168.0.125", 44444,"192.168.0.111")
```

Listing 5. Example of requsting latency between two nodes

VI. Output

In this section, the output and results when the library is running on a collection of nodes are presented and discussed. In Table II, the full cluster output for a system using the table option can be seen. Each host is listed with CPU cores, amount of memory and how many GPUs it has (if it has any). In Listing 6, an example of the summary of *lsf* is shown, and the items are listed in the order of average number of CPU cores, average amount of memory and the total number of nodes. Notice that numbers of GPUs are not available in this view.

Hostname	CPU's	Memory	GPU's
LoginNode	128	64.0G	0
Host1	32	1.1T	4
Host2	32	1.1T	4
Host3	32	1.1T	4
Host4	32	1.1T	4
Host5	20	512.0G	4
Host6	19	512.0G	4
Host7	20	512.0G	4
Host8	20	512.0G	4
Host9	20	1.0T	0
Host10	20	1.0T	0
Host11	20	1.0T	0
Host12	20	1.0T	0
Host13	20	1.0T	0
Host14	20	1.0T	0
Host15	20	1.0T	0
Host16	20	1.0T	0
Host17	20	1.0T	0
Host18	20	1.0T	0
Host19	20	1.0T	0
Host20	20	1.0T	0

TABLE II
Output of cluster information command on LSF system

```
{1: 27.381, 2: 913173.7142857143, 3: 21}
```

Listing 6. Summary output of LSF cluster

Hostname	CPU's	Memory(MB)	GPU's
Node1	1	196	0
Node2	1	199	0
Node3	1	199	0
Node4	1	200	0

TABLE III
Output of cluster information command on Slurm system

```
{1: 1.0, 2: 198.5, 3: 4}
```

Listing 7. Summary output of Slurm cluster

In Table III, the full output from a *slurm* cluster is shown using the *table* option. It is noticeable that the output is similar to *aid* in reading and scripting. The only difference is that memory sizes are in *int* format and represent megabyte (MB), rather than human readable format like the *LSF* output. The summary seen in Listing 7 again follows the same pattern as the *LSF* summary in order to make scripting as simple as possible.

In Listing 8, the output from a single node can be seen. For single nodes, there is no *table* option because none is really needed, so only the *summary* option is available. *summary* command returns two numbers: the number of CPU cores on the machine and the amount of memory in kilo bits (kb).

```
['4', '2027368']
```

Listing 8. Summary output of single node cluster

In table IV we have populated a table with the latency between the master (user's machine) and other nodes and then the latency they have between each other as well by using the P2P library. All of this data was gathered using either a direct command from the master node or by requesting via the P2P library. One thing to note is that the node *dtfi4* is not running any P2P software at present but the other nodes are still able to do latency tests to it. This is useful for testing speeds to nodes you may not fully control so are therefore unable to run the P2P software or to test to nodes you maybe thinking of renting/purchasing.

Source	Destination	Latency(ms)
Master	flux	2.800941
Master	nova	3.694057
Master	sandbox	1.065969
flux	nova	2.099037
flux	dtfi4	11.276484
nova	dtfi4	11.115551

TABLE IV
Latency between nodes in P2P network

In Fig. 4, the data from Table IV is applied to our network diagram of nodes. This helps to visualize the received data and gives an indication of the relative speed of connection between nodes based on the latency.

VII. Conclusions and future work

In this paper, we have developed and demonstrated a new Python library called *pyp2pcluster* and shown its features and capabilities. We have looked at the ways of connecting nodes and the data the libarary is able to collect. So far, we have mainly focused on getting this library running and sharing a basic set of data that is required for the project *Dynamic routing of data based on resource availability and benefit* that we are working on, which as mention previously will be to apply fuzzy logic to this data in order to ascertain the quality of a cluster. This means that as shown it currently only collects a basic subset of data about the size of the nodes and their

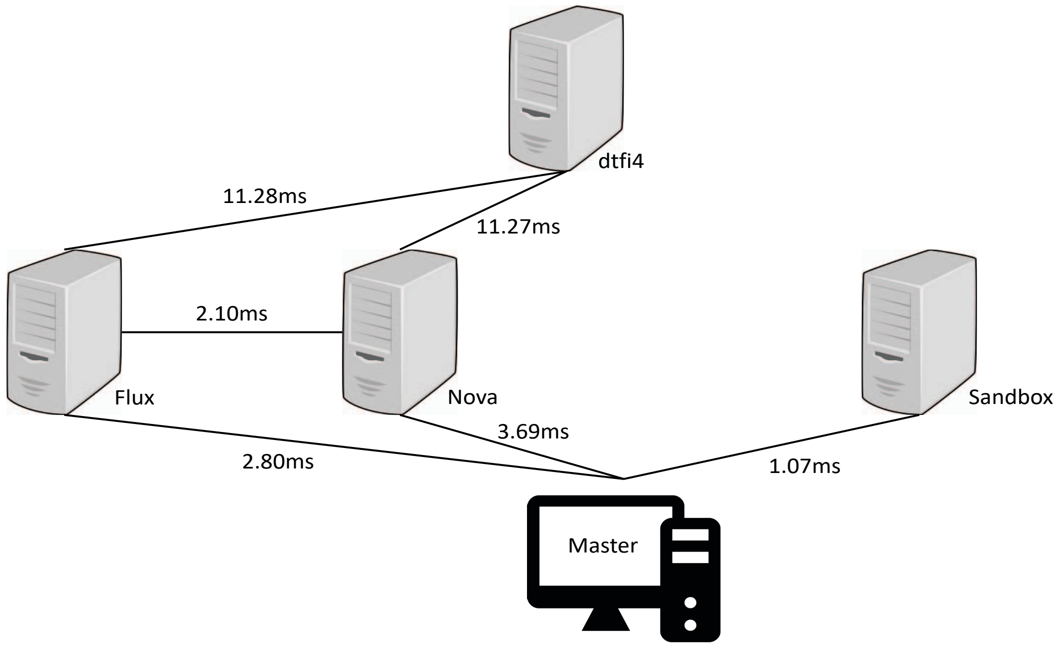

Fig. 4. Latency of nodes

specification but can do this in an automatic continuous way so the data can be regularly updated. As demonstrated the library has other applications outside the scope of the main project and would work well in assisting the arranging and scheduling of jobs in a multi cluster or meta-scheduling environment. This assistance as well as finding the best cluster for performance can also help with the energy efficient scheduling. In the future, we plan to expand the scope of this library so that more information can be shared between the nodes such as queue and job status which are easily obtainable from queue management software as the length of queue will certainly affect which cluster is chosen. We plan to also expand the tool to communicate with cluster management software and in the case of single nodes with the OS in order to get software information as this is also an important factor. We also plan to release a beta version of this library for others to use, with plans to keep active development.

References

[1] TOP500. Top500 June 2020. https://www.top500.org/lists/top500/2020/06/.

[2] BIANCA. Fugaku is now the world's fastest supercomputer but is set to lose the crown soon. http://tinyurl.com/kwazph23.

[3] HARTREE STFC. New milestone brings the uk's largest AI supercomputer one step closer. https://www.hartree.stfc.ac.uk/Pages/New-milestone-brings-the-UK's-largest-AI-supercomputer-one-step-closer.aspx.

[4] EUROHPC. EuroHPC. https://eurohpc-ju.europa.eu/index_en.

[5] SUPERCOMUTING WALES. Supercomuting wales. https://www.supercomputing.wales/.

[6] M. A. S. Netto, R. N. Calheiros, E. R. Rodrigues, R. L. F. Cunha, and R. Buyya, "HPC cloud for scientific and business applications: Taxonomy, vision, and research challenges," *ACM Comput. Surv.*, vol. 51, no. 1, jan 2018. [Online]. Available: https://doi.org/10.1145/3150224

[7] A. Gupta and D. Milojicic, "Evaluation of HPC applications on cloud," in *2011 Sixth Open Cirrus Summit*, 2011, pp. 22–26.

[8] W. M. Jones, W. B. Ligon, L. W. Pang, and D. Stanzione, "Characterization of bandwidth-aware meta-schedulers for co-allocating jobs across multiple clusters," *The Journal of Supercomputing*, vol. 34, no. 2, pp. 135–163, Nov 2005. [Online]. Available: https://doi.org/10.1007/s11227-005-2337-x

[9] A. Formoso and P. Casas, "Looking for network latency clusters in the LAC region," in *Proceedings of the 2016 Workshop on Fostering Latin-American Research in Data Communication Networks*, ser. LANCOMM '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 10–12. [Online]. Available: https://doi.org/10.1145/2940116.2940130

[10] R. Kuchumov and V. Korkhov, "Fair resource allocation for running HPC workloads simultaneously," in *Computational Science and Its Applications – ICCSA 2019*, S. Misra, O. Gervasi, B. Murgante, E. Stankova, V. Korkhov, C. Torre, A. M. A. Rocha, D. Taniar, B. O. Apduhan, and E. Tarantino, Eds. Cham: Springer

International Publishing, 2019, pp. 740–751.

[11] University of California. SETI@home. https://setiathome.berkeley.edu/.

[12] S. Sarmady, “A survey on Peer-to-Peer and DHT,” 2010. [Online]. Available: https://arxiv.org/abs/1006.4708

[13] H. Barjini, M. Othman, H. Ibrahim, and N. I. Udzir, “Shortcoming, problems and analytical comparison for flooding-based search techniques in unstructured P2P networks,” *Peer-to-Peer Networking and Applications*, vol. 5, no. 1, pp. 1–13, Mar 2012. [Online]. Available: https://doi.org/10.1007/s12083-011-0101-y

[14] IBM. IBM Spectrum LSF Suites. https://www.ibm.com/uk-en/products/hpc-workload-management.

[15] Schedmd. Slurm support and development. https://www.schedmd.com/.

[16] Altair Engineering, Inc. Openpbs. https://www.openpbs.org/.

[17] Y. Fan, “Job scheduling in high performance computing,” 2021. [Online]. Available: https://arxiv.org/abs/2109.09269

[18] N. Chan, “A resource utilization analytics platform using grafana and telegraf for the savio supercluster,” in *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning)*, ser. PEARC ’19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3332186.3333053

[19] “V. Elisseev et al., Energy Aware Scheduling Study on BlueWonder, E2SC@SC18.”

[20] T. Nguyen, C. MacLean, M. Siracusa, D. Doerfler, N. J. Wright, and S. Williams, “FPGA-based HPC accelerators: An evaluation on performance and energy efficiency,” *Concurrency and Computation: Practice and Experience*, vol. n/a, no. n/a, p. e6570. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.6570

[21] N. DeBardeleben, S. Blanchard, L. Monroe, P. Romero, D. Grunau, C. Idler, and C. Wright, “GPU behavior on a large HPC cluster,” in *Euro-Par 2013: Parallel Processing Workshops*, D. an Mey, M. Alexander, P. Bientinesi, M. Cannataro, C. Clauss, A. Costan, G. Kecskemeti, C. Morin, L. Ricci, J. Sahuquillo, M. Schulz, V. Scarano, S. L. Scott, and J. Weidendorfer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 680–689.

[22] M. Ujaldón, “HPC Accelerators with 3D memory,” in *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*, 2016, pp. 320–328.

[23] Globus. Globus. https://www.globus.org.

[24] Llnl. Flux: Building a framework for resource management. https://computing.llnl.gov/projects/flux-building-framework-resource-management.

[25] StorjOld. pyp2p-github. https://github.com/StorjOld/pyp2p.

[26] ethereum. pydevp2p-github. https://github.com/ethereum/pydevp2p.

[27] Tribler. dispersy-github. https://github.com/Tribler/dispersy.

[28] ethereum. trinity-github. https://github.com/ethereum/trinity.

[29] libp2p. py-libp2p-github. https://github.com/libp2p/py-libp2p.

[30] zeromq. zyre-github. https://github.com/zeromq/zyre.

[31] Protocol Labs. libp2p. https://libp2p.io/.