Periodica Polytechnica Electrical Engineering and Computer Science, 67(3), pp. 355–367, 2023

Using Heuristic Search Techniques to Reduce Task Migrations in Peer-to-Peer Volunteer Computing Networks

Ehab Saleh[1]*, Chandrasekar Shastry[1]

[1] Department of Computer Science and Engineering, Jain (Deemed-To-Be University), Outer Ring Rd, 560018 Bengaluru, Karnataka, India
* Corresponding author, e-mail: ehab.saleh@jainuniversity.ac.in

Received: 20 September 2022, Accepted: 23 March 2023, Published online: 16 May 2023

Abstract
Massive computations in today's computer applications necessitate the use of high-performance computing environments. Unfortunately, high costs and power management must be addressed while operating these environments. Volunteer computing (VC) enables the creation of a global network of computing devices capable of accumulating their computing power to outperform any supercomputer. VC refers to the use of underutilized computing resources donated by thousands of volunteers who want to actively participate in solving common research problems. However, VC systems experience unexpected and sudden loss of connections between volunteers' computing resources and the main server. In this case, the server must redistribute the work to new devices as they become available. This process is known as task migration, and it is already used in various volunteer frameworks to address the unavailability of computing resources. However, there is a tendency to limit the number of migrations since they are considered a technically complex and time-consuming process. In this paper, we employ heuristic search algorithms to reduce task migrations caused by loss of connections in Peer-to-Peer volunteer networks by locating an alternate network path to send output files to the server when the direct link is no longer available. The simulation results demonstrate that using a heuristic search algorithm eliminates all task migrations caused by loss of connections, resulting in less total execution time and power consumption.

Keywords
volunteer computing, Peer-to-Peer, task migration, Best First Search

1 Introduction

Distributed systems are made up of a massive number of heterogeneous computing devices that communicate via different networking channels and collaborate to solve a common task that requires a lot of computing power and a large memory capacity that a single device cannot provide. Since all of these devices' computing resources are dedicated solely to running their share of the main task, they must be reliable and available at all times during execution. As a result, developing such a distributed system can be challenging and necessitates careful power management.

To address these issues, developers have been interested in using alternative green computing resources to perform large tasks by utilising unused computing power donated by volunteers. This concept of voluntary participation is called volunteer computing (VC).

VC first appeared in 1996 in the project Great Internet Mersenne Prime Search [1], which is still attracting volunteers to help find new large Mersenne numbers.

The project "distributed.net" [2] is also a distributed network project that was found in 1997 to employ unused CPU or GPU resources to perform cryptanalysis tasks related to breaking RC5 cryptographic algorithm [3] and OGR-28 [4]. These two applications are still fully operational and attract volunteers from all over the world.

Several distributed platforms based on the concept of voluntary participation were developed to provide the community with low-cost, negligible-power consumption, and high-performance computing environments, such as Paraweb [5], Manta [6], Superweb [7], Bayanihan [8], Javelin [9], Javelin 2.0 [10], Popcorn [11] and Charlotte [12]. All of these platforms are Java-based volunteer frameworks developed to provide an alternative distributed computing environment for various parallel Java-based research projects. They rely on the concept of using virtual machines in the volunteers' operating systems. Unfortunately, many of these platforms are no longer in use for a variety of reasons, including the fact that some of them require users to keep their web browsers open all the time to execute the main task, while others do not support all computing architectures (heterogeneity), and the broker (middleware) does not support security or authentication mechanisms.

In 2002, the framework Berkeley Open Infrastructure for Network Computing (BOINC) was introduced to solve large scientific problems by employing the simple concept of VC.

BOINC projects have approximately 700,000 active devices. The total number of processing units is approximately 4 million CPUs and 560000 GPUs, with a total throughput of roughly 93 PetaFLOPS [13].

Other projects exist and use the BOINC framework, such as the World Community Grid [14] from IBM (that was running the project OpenPandemics – COVID-19 to eradicate the COVID-19 virus.), Rosetta@Home [15] and some others [13].

The simple concept of VC encourages thousands of volunteers from all over the world to join ongoing volunteer projects. As a result, the question of how long their devices can perform the given work arises, and it's likely that certain devices will be turned off for extended periods of time or that some of the network's links will be off for whatever reason, even after performing the allotted work. In this case, the server must abandon those devices and find new available ones in order to complete the share of the main task that has not been completed in a technically complex and time-consuming process known as "task migration".

Because the loss of connection between volunteers' devices and their server will prompt this server to initiate task migration, we propose in this study employing search algorithms to find an alternative route (path) for sending output files to the server.

To refer to this process in this paper, we coin the term "result migration", which can be achieved in a structured Peer-to-Peer (P2P) volunteer network where each node (peer) can act as both a server and a client at the same time, while also utilizing lookup tables such as Distributed Hash Table (DHT), which is distributed evenly among all nodes and stores information about all or part of the connected nodes (IP address, hash value of the work, the available computing power, etc.).

The rest of this paper is structured as follows: Section 2 includes some related works on reducing migrations in distributed systems as well as searching techniques in P2P systems, Section 3 includes an elaboration of the methodology used in this paper, Section 4 demonstrates in detail how we design our network, Section 5 discusses the experimental results, Section 6 concludes the paper, and Section 7 discusses the scope of future research.

2 Related work

Task migration is the process of moving a task from one computing item to another one. This concept is applied not only in distributed computing systems, but it can also be used as part of the local scheduler job in a multi-core processor where the task can be moved from one core to another. However, Section 2 only addresses the concept of task migration in distributed systems in terms of migration algorithms and techniques used for reducing the number of migrations. In addition, we review previous research on the use of searching algorithms in P2P networks to justify our objectives.

The concept of migrating tasks has been developed in a range of domains, including load balancing, with the goal of increasing resource utilization and reducing response time. Suen and Wong proposed in [16] a communication protocol and a fully distributed algorithm for task migration to reduce both the average response time and the communication overhead. The approach is based on finite projective planes, which Maekawa [17] used to reduce the communication overhead for a distributed mutual exclusion algorithm. Task migration will be performed by the proposed algorithm whenever it is possible, that is, whenever there is at least one lightly loaded processor and at least one heavily loaded processor.

Also, in Ma and Wang [18] suggested using a Java-based light-weight task migration to accelerate computation. The middleware used supports an asynchronous migration technique that allows migrations to occur virtually anywhere in the task.

Zhu and Socko [19] carried out several task migration experiments in order to investigate the impact of using process migration to improve system performance. All experiments were conducted on the distributed system Amoeba [20]. Because the cost of the migrations is compensated by the gain from using idle cycles, the results showed a slight improvement in system performance.

Sharma and Nitin [21] used in both scheduling algorithms, RM and EDF, as well as task migration, to improve total task execution. Initially, a global scheduler is used, which maintains a global task queue and independently distributes tasks to different processors. Then, the RM scheduling algorithm, which assigns priorities based on task time, is used to schedule global task queues. The results show that this combination helped the task meet its deadline.

Although the concept of task migration has already been used in various volunteer frameworks to address computing resource availability, there is a trend to limit the number of migrations or optimize the migration in order to save time and energy. In which, task migration entails several steps, including looking for available peers in the network and rescheduling tasks to be executed on these peers. As a result, the total execution time of the main task will increase in proportion to the number of peers involved in the migration process. This will have an impact on energy consumption because involving more peers in computing will increase the total amount of energy consumed during the execution of the main task.

Prediction techniques based on Machine Learning algorithms were proposed to reduce the migrations processes by predicting the availability of hosts based on their trace records over a specific time period.

The effectiveness of availability prediction has been studied in the context of P2P volunteer computing systems. Ramachandran et al. [22, 23] claimed that the effective prediction of the most available volunteers reduced the number of migrations during job execution. In their paper, they described a P2P desktop grid framework that uses resource availability prediction based on group availability data. Improving system functionality was by submitting jobs to machines that are more likely to be available at a given time.

The use of prediction models in data migration was discussed by Bhagwan et al. [24] and Knežević et al. [25]. They employed prediction and simple replica mechanisms to ensure data availability in P2P distributed systems and therefore reduce data migration.

Prediction techniques to reduce migrations have been proposed to predict the availability in other structures of volunteer networks. McGough et al. [26] proposed the use of machine learning algorithms to predict idle time in the HTC server-client volunteer computing network. Because the prediction will only target those volunteers who are less likely to abandon their volunteer work, the total amount of energy wasted was reduced by reducing the number of task migrations.

Mengistu et al. [27] proposed using machine learning techniques in volunteer cloud systems (VcaaS). Their study's experimental results show that a prediction-based fault tolerance approach is appropriate for these types of systems.

Iglesias et al. [28] proposed a prediction method to estimate a long-term availability of a group of computing resources in server-client volunteer networks. According to the authors, the prediction method handled the frequent disconnections between hosts by selecting only those hosts that are available at any given time, avoiding the need to migrate unfinished work or even data.

Some works used prediction methods to address host availability in volunteer distributed systems, but they did not explicitly mention the beneficial effects on task or data migration [29–33].

P2P systems uses search techniques to find sources that contain the desired data. Whatever this data may represent, there may be multiple locations of it. Some P2P systems are only interested in finding one copy of the data, whereas other systems consider all possible locations of all copies to determine the best and optimal solution. In other words, determining the best route to the location of the desired data.

Searching techniques used in P2P systems vary depending on the network topology, which divides these systems into two categories: structured and unstructured [34].

In unstructured P2P systems, most of the searching techniques are flooding-based algorithms where no peer knows the location of the desired data. Gnutella [35] is a P2P protocol that uses flooding to find a source in a network. It employs Breadth-First Search, in which the query peer sends a query request to all of its neighbors. If the destination peer is discovered, it will respond to this request with a positive replay; otherwise, all of these neighbor peers will repeat the same query request to all of their neighbor peers except the original query peer until the destination peer is discovered.

These flooding techniques can generate a large number of requests and messages, which increases network traffic.

Yang and Molina [36] proposed *alternative deepening* as a combination of artificial intelligence and searching in P2P networks. A maximum depth limit D is set, and the query is terminated when the query result (the destination peer) is satisfied or the maximum depth limit D is reached (the destination peer is not found).

Kalogeraki et al. [37] proposed two searching approaches in P2P network's. The first approach is a modified version of Breadth-First Search that is designed to reduce the number of messages required to search the network. The second approach, on the other hand, used an intelligent search mechanism that learns from the P2P network's past behavior to improve the scalability of the searching.

Lv et al. [38] ran several scenarios over Gnutella's original flooding algorithm prior to finally proposing *k-walker random walk*, a new approach of searching in unstructured P2P networks in which, instead of flooding the same request to all neighbors, only *k random* neighbors are chosen and thus only *k* copies of the requests are generated at each peer until the destination peer is found. Also, Jawhar and Wu [39] proposes another version of the *k-walker* algorithm in which Time-To-Live (TTL) is used to ensure that searching using *k-walker* stops when TTL expires. However, in the next round of the *k-walker* algorithm, other values will be assigned to both *k* and TTL.

Other approaches have been proposed in [40] and [41] which can be classified as Breadth-First Search-based and all aim to address flooding techniques in unstructured P2P systems.

In terms of structured P2P systems, the network architecture is precisely defined, and the relationship between each peer and data location is also represented in different data structures such as DHT. However, the final structure of these systems determines the searching technique used. For instance, in Chord network [42] the data structure is ring. It uses a *finger table* that contains addresses of different set of peers such as half of the peers away from it, one fourth of the peers away of it, until its immediate successor peer. When searching is done, the query is forwarded from the query peer to all successors in *finger table*, which then sends the same query to their peers using the addresses in *finger table* until the destination peer is found.

Bin et al. [43] proposed a new enhanced version of Chord protocols, which uses tables of neighbors' neighbors links in a dynamic P2P system with frequent peer arrivals and departures. It estimates a shorter path length from the source peer to the destination than the pure Chord protocol.

Pastry [44] used tree data structure for its DHTs. Similarly to the Chord, each Pastry's peer stores the addressing information of the closest geographically dispersed peers in the same space. However, instead of using *Chord*'s *finger tables*, it uses *routing table*, a *leaf set* and a *neighbourhood set*.

In comparison to our work, we aim to apply the concept of searching in the context of minimizing task migrations caused by loss of connection in P2P volunteer computing networks. To accomplish this, we combine heuristic search techniques and lookup tables, like DHTs, into a single algorithm to find an alternate path from the source peer to the destination peer. We afterwards use this algorithm to enable peers to send result packets to their super-peers once direct links are no longer available.

We use two different lookup tables; the first table stores the addresses information for all peers in our network and is used to aid routing, while the second contains performance metrics for all peers as well as addresses information. In the context of this paper, the first table is referred to as *Network Tree*, while the second table is referred to as *Computing Tree.*

We evaluate our approach using two metrics: total execution time and power consumption.

3 Methodology

To ensure that our approach is properly implemented, the following tasks must be performed in the order listed below.

3.1 Performance query

The first step in our approach is to investigate each network peer's performance and store the most important performance metrics in *Computing Tree*. The procedure that initiates this querying request is performed at each peer that functions as a server (super-peer), and peer is added or removed from *Computing Tree* based on a predefined threshold. For example, if the CPU availability is the chosen performance metric, peers are added to the *Computing Tree* if the provided availability ratio is greater than 50%; otherwise, peers are not considered available for computing at the time of querying.

It is worth noting that this query procedure is launched whenever *Computing Tree* needs to be updated. This can happen only when the super-peer has work that needs to be distributed, and if a sub- peer does not respond to the querying requests for whatever reason, it will not be considered for running the current task; However, the address details of this peer will still be preserved in *Network Tree* for future query requests.

3.2 Global scheduling

The second step in our method is to distribute the main task to all connected sub-peers. To accomplish this, each peer acting as a server is given a global scheduler whose goal is to calculate the workload portion of the main task to be distributed to each network sub-peer in such a way that all devices exert the same amount of effort in terms of power consumption and execution time. This global scheduler is described in [45]. It is based on calculating the *accumulated computing power* available at each peer that can serve as a server. This global scheduler is briefly explained below (Eq. (1)).

To estimate how many FLOPS each peer's processor *i* can perform per second, we use Eq. (1):

$$\mathrm{FLOPS}_i^{Local} = \mathrm{FLOPS}_i^{Peak} \times a_i \times c_i, \tag{1}$$

where a is the processor availability given as a float number between 0 and 1, FLOPS_i^{Peak} is the peak processor speed per core, and c is the number of cores.

The accumulated available FLOPS of a given super-peer i with n sub-peers can be calculated as follows in Eq. (2):

$$\mathrm{FLOPS}_i^{Total} = \mathrm{FLOPS}_i^{Local} + \sum \mathrm{FLOPS}_k^{Local}. \quad (2)$$

Now, for a super-peer I with a task t that has an estimated number of FLOPS *estFlopCount* we can calculate the workload FLOPS that can be allocated to each peer (super-peer and sub-peers) by using Eq. (3):

$$\mathrm{FLOPS}^{Load'} = estFlopCount_{i,t} \,/\, \mathrm{FLOPS}_i^{Total}. \quad (3)$$

Equation (3) is only applied if all peers have one core per processor with availability 100%, but since each peer i provides different performance depending on the availability a and the number of cores c, we will use Eq. (4) to calculate the workload FLOPS that can be allocated to a peer i:

$$\mathrm{FLOPS}_i^{Load} = \mathrm{FLOPS}^{Load'} \times \mathrm{FLOPS}_i^{Peak} \times a_i \times c_i. \quad (4)$$

Equation (4) can be easily written in the following form (Eq. (5)):

$$\mathrm{FLOPS}_i^{Load} = \mathrm{FLOPS}^{Load'} \times \mathrm{FLOPS}_i^{Local}. \quad (5)$$

3.3 Task migration

Following the distribution of the appropriate workload to all available network devices, the super-peer will now expect results from these devices and will enter a loop while waiting for all result packets to be provided. The result packet includes not only the result (or a portion of it), but also some important details, such as the completion status, which indicates whether or not the workload was executed completely. If a portion of the workload is still incomplete, it will be delivered to the server in the same packet. When all of the sub-peers have finished execution, the server will start a *Performance Query* to update *Computing Tree* before initiating the task migration procedure.

The pseudo-code that follows describes in detail all of the aforementioned tasks, as well as all of the global variables that must be defined in order for our approach to be implemented (see in Algorithm 1).

Algorithm 1 Task migration algorithm

Require: Network with only one super-node(N), Task(est_flop_count), Threshold(threshold)
Ensure: Workload per node

```
1:  loop main_loop
2:      isExecuted ← false
3:      unexecutedWork ← empty list
4:      availableNodes ← empty list
5:      completionStatus ← empty list
6:      workload ← empty list
7:      PerformanceEnquiring(N,threshold)
8:      GlobalScheduling(est_flop_count,availableNodes)
9:      for node n in availableNodes do
10:         if isExecuted is false then
11:             availableNodes.remove(n)
12:             unexecutedWork.add(workn)
13:         end if
14:     end for
15:     if unexecutedWork.size() > 0 then
16:         est_flop_count ← unexecutedWork.FLOPs()
17:         Repeat main_loop
18:     else
19:         Exit main_loop
20:     end if
21: end loop
22: procedure PerformanceEnquiring(N,threshold)
23:     for node n in N do
24:         if n.computingPower > threshold then
25:             availableNodes.add(n)
26:         else
27:             avaliableNodes.remove(n)
28:         end if
29:     end for
30: end procedure
31: procedure GlobalScheduling(est_flop_count,availableNodes)
32:     for node i in availableNodes do
33:         FLOPS$_i$_Local ← FLOPS$_i$_Peak × a$_i$ × c$_i$
34:         if node i is super-node then
35:             FLOPS_Total ← FLOPS$_i$_Local
36:         end if
37:     end for
38:     for node j is sub-node to node i do
39:         FLOPS_Total ← FLOPS_Total + FLOPS$_j$_Local
40:     end for
'41:    FLOPS_Load ← est_flop_count/FLOPS_Total
42:     for node i in N do
'43:        workload.add(FLOPS_Load × FLOPS$_i$_Local)
44:     end for
45: end procedure
```

3.4 Heuristic search

When considering AI for search problems, there are a few common terms to be aware of:

- Agent: it is the entity that perceives and reacts to the environment around it.
- State: it describes the configuration of the agent in his environment.
- Initial state: it is the state in which the agent begins.
- Action: it is a choice that we can make in any state.
- Transition model: it is the description of what state we get after applying an action in the current state.

- State space: it is a space of all the states that we can get from applying any actions in the initial state.
- Goal test: the way we make a comparison in the current state to figure out if we reach the goal state.
- Path cost: it is the numerical cost that is associated with a specific path.
- Optimal path: it is the solution that has the lowest cost from the initial state to the goal state.

However, when we adapt these common terms to our problem, where we use a P2P network of connected computer devices, we get the following:

- Agent: the network sub-peer (worker) transmitting result files to the super-peer (server).
- State: in our case, it represents the addresses and performance information in DHT tables.
- Initial state: it is the address from which the result file must be transmitted.
- Action: transferring the output files to the next connected peer in the network.
- Transition model: A protocol that makes two peers in the network connect and exchange data. Our transition model mimics the functionality of the TCP protocol.
- State Space: it represents the addresses of all the direct connected sub-peers in the network.
- Goal test: the method by which we compare addresses to determine if the next address is the address of the target super-peer.
- Path cost: in our case, we want to use the bandwidth given in MBps as a weight assigned to each link.
- Optimal path: over all possible paths from the sub-peer to the super-peer, it is the network path with the maximum bandwidth.

Generally, we can divide AI search algorithms into two categories based on the parameters we pass to them:

1. Uninformed search: algorithms in this category attempt to identify a solution without any domain-specific knowledge or other information about the state. Some of the most commonly used uninformed search algorithms are: Breadth First Search [46] and Depth First Search [47].
2. Informed search/heuristic search: Algorithms in this category are mostly employed to locate the optimal path according to additional provided parameters, such as the state space, costs, and other parameters that aid the algorithm in finding the proper solution from the initial state to the goal state. The most commonly used algorithms that employ this heuristic concept are [48]: Best First Search (Greedy Search) and A* Search.

We will use the heuristic algorithm Best First Search in this paper since it benefits from uniformed search in both Breadth First Search and Depth First Search algorithms and exceeds A* algorithm in speed while doing the search.

Greedy Best First Search algorithm's pseudo-code is as follows in Algorithm 2.

The proposed algorithm is entirely based on determining all possible paths between the sender and the recipient. However, among all possible paths, Best First Search algorithm selects only the optimal path based on the weighted criteria.

In simulation, we adapted BFS to choose the optimal path with the maximum bandwidth over all other discovered paths. However, in practice, this cannot be guaranteed because peers use different communication channels with different bandwidths and sharing policies. Also, it is hard to estimate how the bandwidth slows down as the number of users on the network increases. However, we guarantee that the search query will use the only path whose bandwidth is not slowing down as much as the other paths.

Algorithm 2 Best First Search algorithm

```
Require: Network(G), Start Node(s), Goal Node(g)
Ensure: Path with maximum bandwidth
1:  openList ← s
2:  closedList ← empty list
3:  path ← empty list
4:  while openList is not empty do
5:      b ← best node from openList
6:      openList.remove(b)
7:      closedList.add(b)
8:      if b is g then
9:          path.add(b)
10:         return path
11:     end if
12:     N ← neighbors(b)
13:     for n in N do
14:         if n is n neither closedList nor openList then
15:             openList.add(n)
16:         else if n is in openList then
17:             if path with current parent ← path with old parent then
18:                 Replace parents of n
19:             end if
20:         else if n is not in closedList then
21:             openList.add(n)
22:         end if
23:     end for
24: end while
25: return path
```

4 Network configuration

We chose a structured P2P network of virtual 1500 connected peers, with each peer representing a volunteer device in the real world.

Each peer can act as both a server and a client at the same time and there are common tasks that can be performed, such as responding to any query request, and performing the allotted work. However, some tasks, such as distributing the allotted work, can only be performed in one mode of the peer, which is only implemented in peers that act solely as servers.

Instead of randomly flooding messages to all peers, following a structured overlay will reduce the effort that each peer expends in routing to find other peers or resources in the same network. This can be accomplished by storing peer-related information in lookup and hash tables that are accessible to all network peers.

Distribute Hash Table (DHT) is a large example of a lookup table used in structured P2P networks that provides lookup service to all network participants. Data is stored in the <*Key:Value*> format, where "*Key*" represents a peer's actual identifier and the associated "*Value*" can be any type of information, including keys of other peers in the network.

To update DHT contents, two main operations or functions must be performed: *PUT* (*Key, Value*) is used to add a new peer to the table, and *GET* (*Key*) is used to return the actual content of the value field that is associated with Key value.

In our simulation, we employ two lookup tables: *Network Tree* and *Computing Tree*. They both enable the usage of the same operations as real DHTs.

Fig. 1 depicts the virtual network overlay that will be used in the simulation stage. The network shape follows a structured topology that is constructed as a result of the real-world epidemic spread of VC's middle-ware.

To evaluate and compare results, we use the simulation framework *SimGrid* [49], which allows us to write code in C++ or Python that reflects the actual behavior of the proposed distributed system or protocols.

The structure of our network is specified in an external xml file that describes the technical specifications of each peer and the type of connection that exists between them. The number of cores, power consumption, speed of all cores running concurrently (FLOPS).

Furthermore, links are distinguished by their bandwidth, latency, and sharing policy. All links' bandwidth was set to 100 MBps, and the sharing policy was set to full-duplex, which means that any two connected peers can transmit data in both directions at the same time using the TCP protocol.

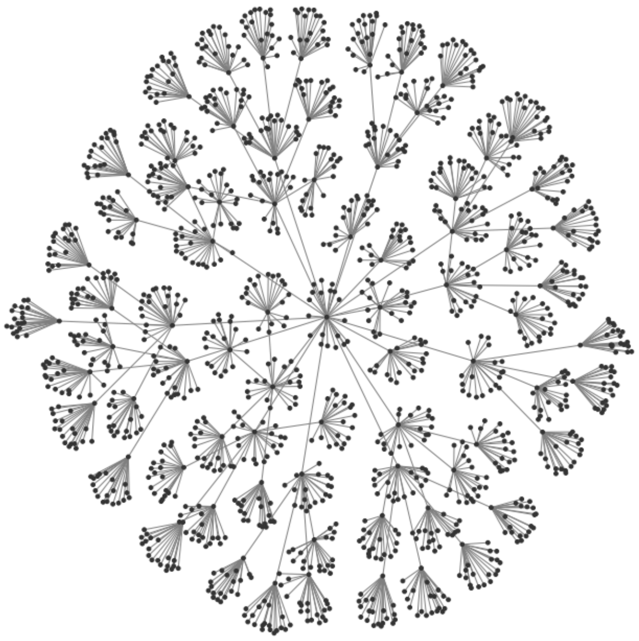

Fig. 1 The network overlay diagram used in the simulation stage

Regarding CPU's availability, each peer is given an availability file describing the CPU's available percentage taken every 5 minutes over a period of one month.

To provide the availability files of all the connected peers in the network, we have used the dataset GWA-T-13 Materna [50], which contains performance metrics described as trace files of over 1500 VMs of the distributed *Materna Data Centers*, Dortmund, Germany.

Fig. 2 shows the available CPU power expressed in percent throughout a 24-hour period on November 5, 2015. We can notice that the CPU was idle for the majority of the day, with the exception of the time between 10:00:00 and 14:00:00, when the observed availability shows an unstable availability rate, causing it to fall, reaching its lowest value at 11:10:00.

In our experiments, to avoid using the host's CPU in an intrusive manner and in order to limit CPU heat, we have set a fixed threshold (50%) that the CPU must cease performing the allotted work if its available percentage falls below this threshold.

Each peer performs differently as they are different in both availability and the number of cores per processor. This was employed to ensure the heterogeneity of our network.

In SimGrid, we use the plugin *plugin_host_energy* [51] to track each peer's power consumption behavior. It estimates the amount of power consumed by each peer in the network by adding the static and dynamic parts of the consumed power. The plugin uses Eq. (6) to calculate the total power consumption for a given machine *i* that has the frequency *f*, the workload *w*, and the usage percentage *u*:

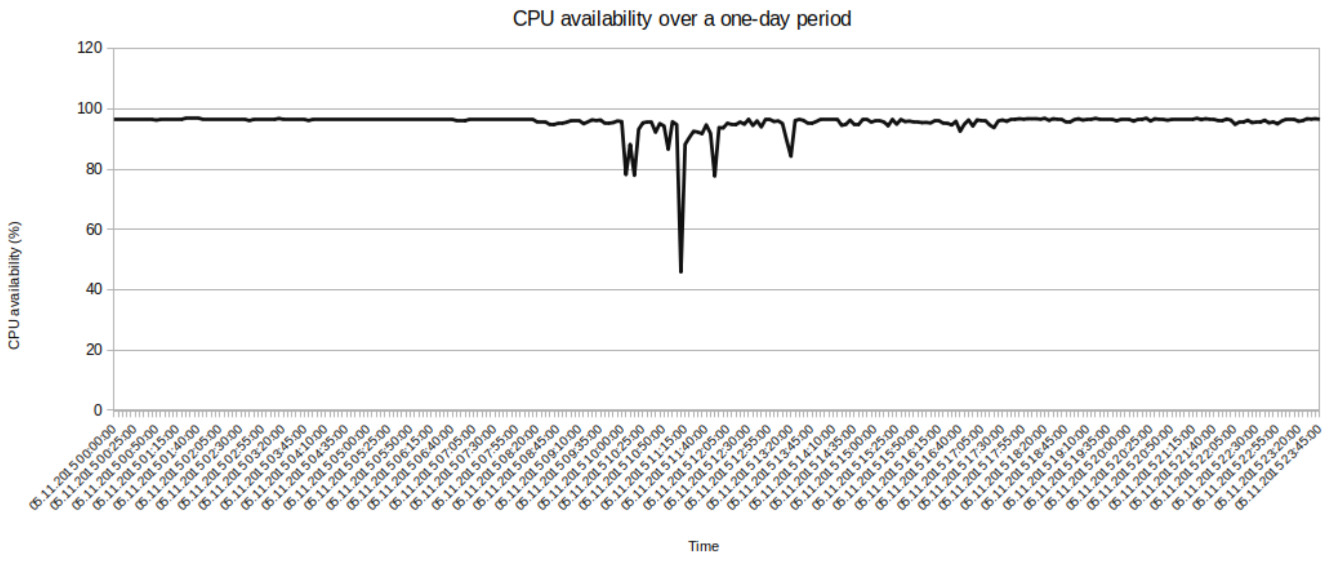

Fig. 2 A one day sample of CPU availability

$$P_{i,f,w} = P_{i,f}^{static} + P_{i,f,w}^{dynamic} \times u. \quad (6)$$

The static part represents power usage when the peer is idle, turned off, or turned on, and the dynamic part represents power consumption when the CPU is working. Whereas the static part of the total consumed energy is simple to compute, the dynamic part is linearly proportional to the CPU load [52].

Table 1 groups the peers based on the number of cores they have. The energy consumption model for each peer is also given to each group. Each model has four parameters, which are as follows:

- Idle: wattage when the peer is up but without performing anything.
- Epsilon: wattage when all cores are not performing the allotted work but they are not in an idle state.
- Allcores: wattage when all cores of the peer are performing the allotted work.
- Off: wattage when the peer is turned off.

5 Simulation results

To conduct the experiments we considered two scenarios: in the first scenario, we make the network execute a given task with size of one ExaFLOPS (10^{18} FLOPS) without taking into account the loss of connectivity between peers, forcing the server to migrate the whole work to other available peers. Whereas in the second scenario, we make the network execute the same size of task but after enabling result migration at each peer, allowing it to find an alternative route if the connection to the super-peer is lost.

In both scenarios, each peer's CPU can be in one of the following types of execution:

- The CPU executes all of the allotted work until it is finished.
- The CPU suspends execution when the monitored usage percentage exceeds a predefined threshold.
- The CPU does not engage in executing the whole task when the monitored usage exceeds a predefined threshold.

The work can be considered partially or totally unfinished according to the following cases:

- When a sub-peer suspends the execution of its work, it provides the super-peer with the portion of the work (in FLOPS) that was not completed, as well as the output files of the other portion that was successfully completed just before the suspension.
- When a super-peer does not get any output files from a sup-peer, the overall work is considered unfinished. The super-peer deems the sub-peer to be no longer available in the network in this case.

When the super-peer detects that all or part of the allotted work has yet to be executed, it initiates the task migration process. However, if all peers have result migration

Table 1 Specification of peers based on the number of cores

		Power Consumption Parameters (Watt)			
No. cores/peer	No. peers	Idle	Allcores	Epsilon	Off
One-core CPU	202	100	140	120	10
Two-core CPU	953	100	160	120	10
Four-core CPU	242	100	200	120	10
Six-core CPU	39	100	240	120	10
Eight-core CPU	64	100	280	120	10

enabled, the super-peer may only receive output files if the sub-peers discover a new route to it, eliminating the requirement for task migration.

Charts in Fig. 3 and Fig. 4 show the execution time of each of the 1500 peers in both scenarios after the whole task is successfully executed.

Fig. 3 shows that the execution time of nearly 100 sub-peers is missing, which occurred because the connection with the super-peer of these sub-peers was lost, causing the super-peer to consider these sub-peers as no longer available and that their work needed to be redistributed again. In fact, the super-peer opted to initiate task-migration and redistribute the entire incomplete work to the sub-peers between 480 and 580 because they became available earlier, causing their execution time to be significantly longer than the rest of the sub-peers.

On the other hand, in Fig. 4 we can notice that loss of connection was recovered in each sub-peer, and thus, result migration was initiated only when it was required, and there was an available path can be used to provide the output files to the super-peers, reducing the need for task migration and thus reducing the execution time of each peer.

In proportion to the execution time, Fig. 5 and Fig. 6 show the consumed energy in both scenarios for each of the participating peers after the whole task is successfully executed.

Fig. 5 shows that when result migration is off, the consumed power in these sub-peers that lose connection with their super-peer is considered wasted power since the super-peer has to implement task migration and redistribute the unfinished work again to other available sub-peers.

The consumed power when result migration is used to dispatch output files to the super-peers is depicted in Fig. 6. When compared to Fig. 5, we can conclude that as the number of migrations decreases, so does the amount of power consumed by each peer in the network.

Table 2 shows the total execution time and overall estimated power consumption when the two scenarios were conducted. As a result of enabling result migration process

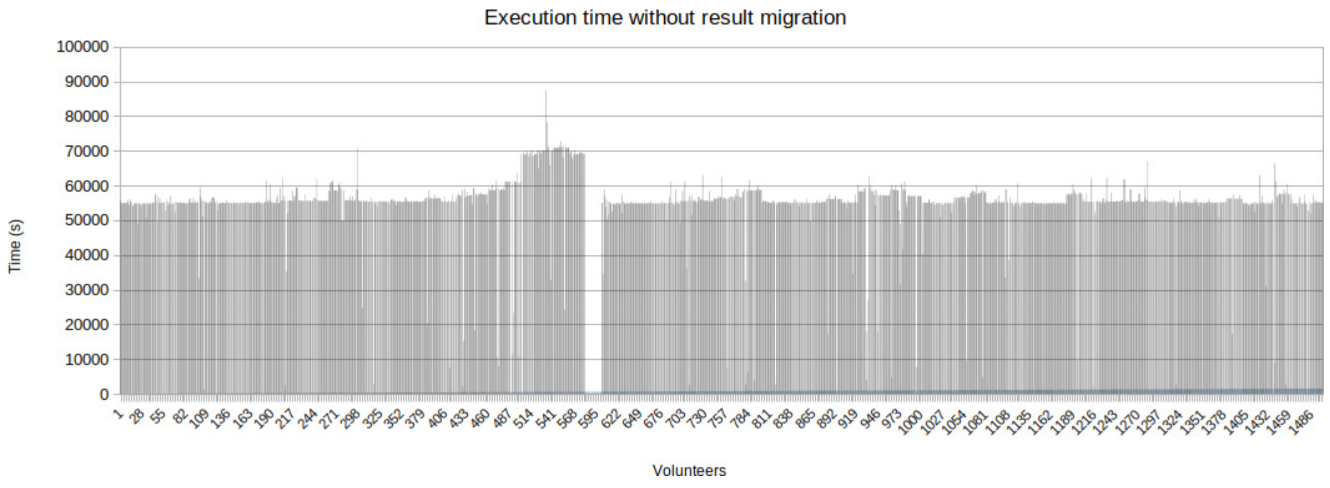

Fig. 3 Execution time for each peer without enabling result migration

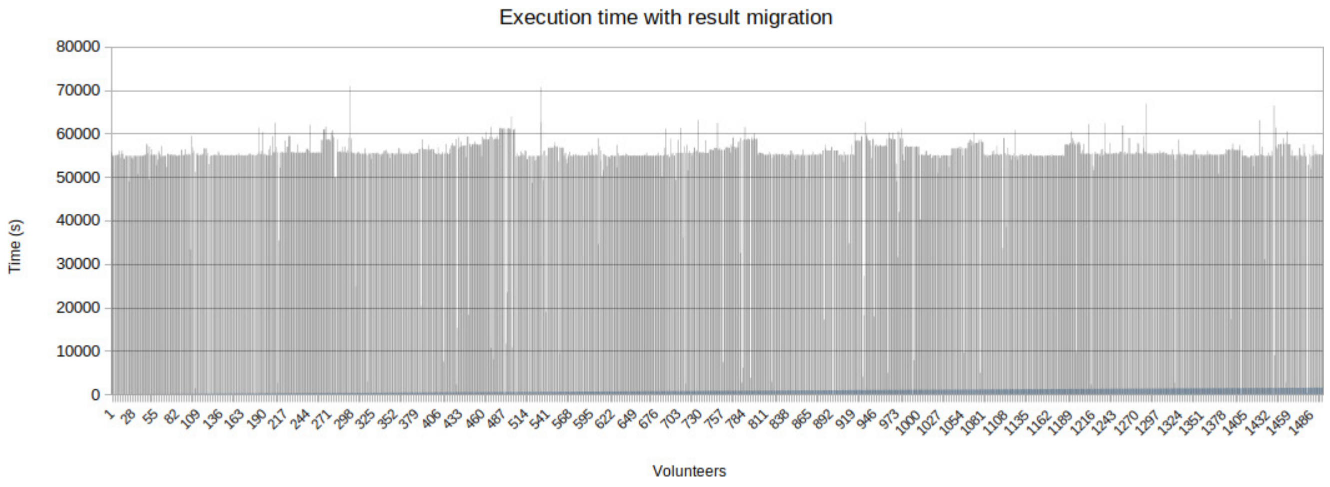

Fig. 4 Execution time for each peer with enabling result migration

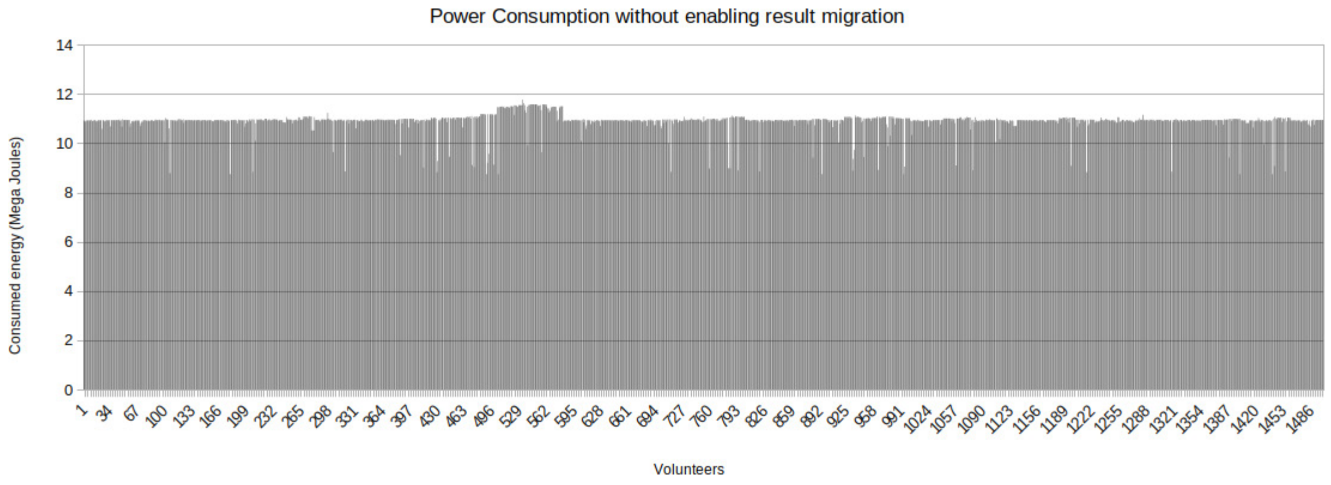

Fig. 5 Power consumption for each peer without enabling result migration

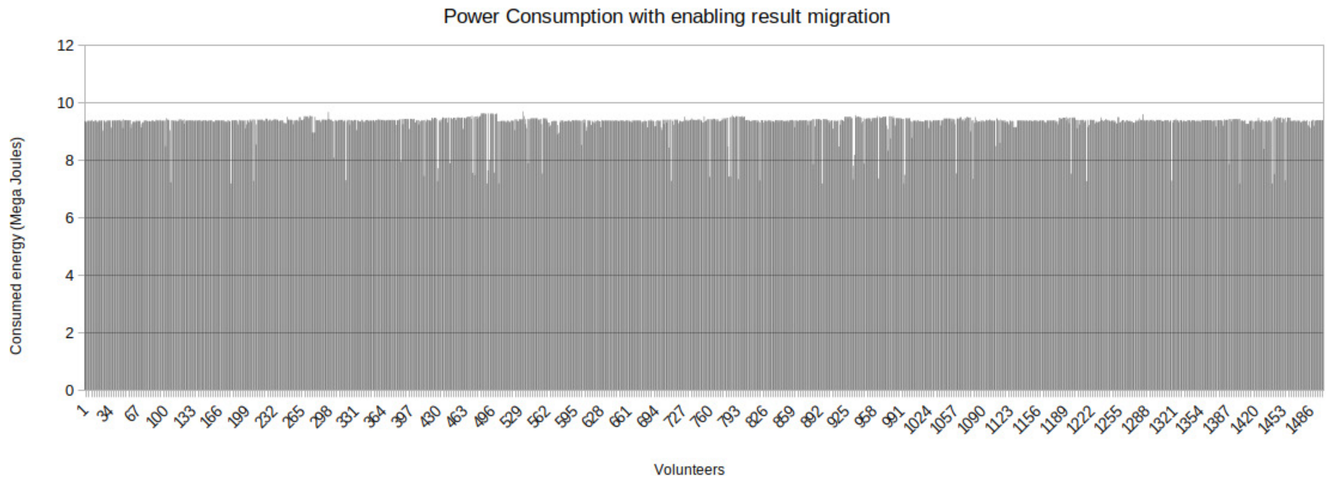

Fig. 6 Power consumption for each peer with enabling result migration

Table 2 The total execution time and estimated power consumption in both scenarios

	Total execution time (seconds)	Total power consumption (giga Joules)
Without result migration	87457.5	16.36
With result migration	71796.2	13.97

at each peer in the second scenario, 100% of task migrations, which the super-peers perform due to losing connections were eliminated, resulting in an 18% reduction in total execution time and a 14.6% reduction in total power consumption, taking into consideration that the number of task migrations initiated as a result of losing connections with the super-peers represents just 5% of the total migrations that may occur.

In other words, in the first scenario, we can consider that the same task was executed twice, once by peers that have lost their links and are unable to provide results to their super-peer, and once by peers that are located in the same sub-network and execute the same workload that is considered unfinished. However, in the second scenario, if the sub-peers are able to send results packets to the super-peer despite losing connections, the task will not be duplicated because there is another discovered path to send results.

6 Conclusion

In this paper, we coined the term "result migration" to describe the implementation the concept of searching in the context of minimizing task migrations caused by loss of connection links in P2P volunteer computing networks. We combined heuristic search techniques and lookup tables into a single algorithm to enable peers to send result packets to their super-peers once direct links are no longer available. The experimental results showed that enabling this strategy at each peer eliminated all of task migrations that the super-peers had to perform as a result of loss of connections, ensuring that task will not be executed more than once, resulting in fewer task migrations and a reduction in total execution time and power consumption.

7 Future scope

The goal of this research is to propose a new approach for reducing task migrations caused by loss of connections in P2P VC systems. However, the suggested method can be extended to provide more efficient performance, and our approach can always be improved into a more sophisticated version.

As a result of the current approach, the following ideas for the feature have been proposed:

- Best-First-Search is initiated only at each sub-peer when it has results to send to its super-peer, but the direct link is turned off. However, losing of connections can occur at any time during task execution, such as before or after the super-peer sends a querying request or before sending the corresponding sub-job. In these cases, peers who do not respond will be considered no longer available. In such cases, we believe that deploying the heuristic search to be launched whenever a connection is lost can also help improve both total execution time and power consumption. However, without a reduction in the number of task migrations.
- If some peers are unavailable for an extended period of time, the super-peer will not consider them for any computation duties; however, the addresses of these peers will be saved to inquire about their availability when a new task is to be executed. In reality, this long-term unavailability reflects the fact that these peers abandoned the project and are no longer interested in computing. To save space in lookup tables, we can suggest removing these peers' addresses after a certain period of time.

References

[1] Mersenne Research, Inc. "Welcome to GIMPS, the Great Internet Mersenne Prime Search", [online] Available at: https://www.mersenne.org/ [Accessed: 19 July 2022]

[2] distributed.net "The Organization", [online] Available at: https://www.distributed.net/Main_Page [Accessed: 19 July 2022]

[3] distributed.net "Project RC5", [online] Available at: https://www.distributed.net/RC5 [Accessed: 19 July 2022]

[4] distributed.net "Aggregate Statistics", [online] Available at: https://stats.distributed.net/projects.php?project_id=28 [Accessed: 19 July 2022]

[5] Brecht, T., Sandhu, H., Shan, M., Talbot, J. "Paraweb: Towards world-wide supercomputing", In: EW 7: Proceedings of the 7th Workshop on ACM SIGOPS European Workshop: Systems Support for Worldwide Applications, Connemara, Ireland, 1996, pp. 181–188. ISBN 978-1-4503-7339-5 https://doi.org/10.1145/504450.504484

[6] van Nieuwpoort, R., Maassen, J., Bal, H. E., Kielmann, T., Veldema, R. "Wire-area parallel computing in Java", In: JAVA '99: Proceedings of the ACM 1999 Conference on Java Grande, San Francisco, CA, USA, 1999, pp. 8–14. ISBN 978-1-58113-161-1 https://doi.org/10.1145/304065.304087

[7] Alexandrov, A. D., Ibel, M., Schauser, K. E., Scheiman, C. J. "Superweb: research issues in Java-based global computing", Concurrency: Practice and Experience, 9(6), pp. 535–553, 1997. https://doi.org/10.1002/(SICI)1096-9128(199706)9:6<535::AID-CPE307>3.0.CO;2-1

[8] Sarmenta, L. F. G., Hirano, S. "Bayanihan: building and studying web-based volunteer computing systems using Java", Future Generation Computer Systems, 15(5-6), pp. 675–686, 1999. https://doi.org/10.1016/S0167-739X(99)00018-7

[9] Christiansen, B. O., Cappello, P., Ionescu, M. F., Neary, M. O., Schauser, K. E., Wu, D. "Javelin: Internet-based parallel computing using Java", Concurrency: Practice and Experience, 9(11), pp. 1139–1160, 1997. https://doi.org/10.1002/(SICI)1096-9128(199711)9:11<1139::AID-CPE349>3.0.CO;2-K

[10] Neary, M. O., Phipps, A., Richman, S., Cappello, P. "Javelin 2.0: Java-based parallel computing on the internet", In: 6th International Euro-Par Conference, Munich, Germany, 2000, pp. 1231–1238. ISBN 978-3-540-67956-1 https://doi.org/10.1007/3-540-44520-X_174

[11] Nisan, N., London, S., Regev, O., Camiel, N. "Globally distributed computation over the internet - the POPCORN project", In: Proceedings. 18th International Conference on Distributed Computing Systems (Cat. No.98CB36183), Amsterdam, Netherlands, 1998, pp. 592-601. ISBN 0-8186-8292-2 https://doi.org/10.1109/ICDCS.1998.679836

[12] Baratloo, A., Karaul, M., Kedem, Z. M., Wijckoff, P. "Charlotte: Metacomputing on the web", Future Generation Computer Systems, 15(5-6), pp. 559–570, 1999. https://doi.org/10.1016/S0167-739X(99)00009-6

[13] Anderson, D. P. "BOINC: A platform for volunteer computing", Journal of Grid Computing, 18(1), pp. 99–122, 2020. https://doi.org/10.1007/s10723-019-09497-9

[14] World Community Grid "World community grid", [online] Available at: https://www.worldcommunitygrid.org/ [Accessed: 20 July 2022]

[15] Rosetta@home "You don't have to be a scientist to do science", [online] Available at: https://boinc.bakerlab.org/ [Accessed: 20 July 2022]

[16] Suen, T. T. Y., Wong, J. S. K. "Efficient task migration algorithm for distributed systems", IEEE Transactions on Parallel and Distributed Systems, 3(4), pp. 488–499, 1992. https://doi.org/10.1109/71.149966

[17] Maekawa, M. "A √N Algorithm for Mutual Exclusion in Decentralized Systems", ACM Transactions on Computer Systems, 3(2), pp. 145–159, 1985. https://doi.org/10.1145/214438.214445

[18] Ma, R. K. K., Wang, C.-L. "Lightweight Application-Level Task Migration for Mobile Cloud Computing", In: 2012 IEEE 26th International Conference on Advanced Information Networking and Applications, Fukuoka, Japan, 2012, pp. 550–557. ISBN 978-1-4673-0714-7
https://doi.org/10.1109/AINA.2012.124

[19] Zhu, W., Socko, P. "Migration impact on load balancing-an experience on Amoeba", In: Proceedings of 5th IEEE International Symposium on High Performance Distributed Computing, Syracuse, NY, USA, 1996, pp. 531–540. ISBN 0-8186-7582-9
https://doi.org/10.1109/HPDC.1996.546224

[20] Fireball Amoeba "Amoeba: A Distributed Operating System", [online] Available at: https://fsd-amoeba.sourceforge.net/amoeba.html/ [Accessed: 05 November 2022]

[21] Sharma, R., Nitin. "Task Migration with EDF-RM Scheduling Algorithms in Distributed System", In: 2012 International Conference on Advances in Computing and Communications, Cochin, India, 2012, pp. 182–185. ISBN 978-1-4673-1911-9
https://doi.org/10.1109/ICACC.2012.42

[22] Ramachandran, K., Lutfiyya, H., Perry, M. "Decentralized resource availability prediction for a desktop grid", In: 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, Melbourne, VIC, Australia, 2010, pp. 643–648. ISBN 978-1-4244-6987-1
https://doi.org/10.1109/CCGRID.2010.54

[23] Ramachandran, K., Lutfiyya, H., Perry, M. "Decentralized approach to resource availability prediction using group availability in a P2P desktop grid", Future Generation Computer Systems, 28(6), pp. 854–860, 2012.
https://doi.org/10.1016/j.future.2010.10.006

[24] Bhagwan, R., Tati, K., Cheng, Y.-C., Savage, S., Voelker, G. M. "Total recall: System support for automated availability management", In: NSDI '04: Proceedings of the First Symposium on Networked Systems Design and Implementation, San Francisco, CA, USA, 2004, pp. 337–350. ISBN 9781931971195 [online] Available at: https://www.usenix.org/legacy/events/nsdi04/tech/bhagwan.html [Accessed: 05 November 2022]

[25] Knežević, P., Wombacher, A., Risse, T. "DHT-Based Self-adapting Replication Protocol for Achieving High Data Availability", In: Second International Conference on Signal-Image Technology and Internet-Based Systems, Hammamet, Tunisia, 2009, pp. 201–210. ISBN 978-3-642-01349-2
https://doi.org/10.1007/978-3-642-01350-8_19

[26] McGough, A. S., Forshaw, M., Brennan, J., Al Moubayed, N., Bonner, S. "Using machine learning to reduce the energy wasted in volunteer computing environments", In: 2018 Ninth International Green and Sustainable Computing Conference (IGSC), Pittsburgh, PA, USA, 2018, pp. 1–8. ISBN 978-1-5386-7467-3
https://doi.org/10.1109/IGCC.2018.8752115

[27] Mengistu, T. M., Che, D., Alahmadi, A., Lu, S. "Semi-Markov process based reliability and availability prediction for volunteer cloud systems", In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 2018, pp. 359–366. ISBN 978-1-5386-7236-5
https://doi.org/10.1109/CLOUD.2018.00052

[28] Lázaro, D., Kondo, D., Marquès, J. M. "Long-term availability prediction for groups of volunteer resources", Journal of Parallel and Distributed Computing, 72(2), pp. 281–296, 2012.
https://doi.org/10.1016/j.jpdc.2011.10.007

[29] Ren, X., Lee, S., Eigenmann, R., Bagchi, S. "Prediction of resource availability in fine-grained cycle sharing systems empirical evaluation", Journal of Grid Computing, 5(2), pp. 173–195, 2007.
https://doi.org/10.1007/s10723-007-9077-5

[30] Yang, D., Cao, J., Yu, C., Xiao, J. "A multi-step-ahead CPU load prediction approach in distributed system", In: 2012 Second International Conference on Cloud and Green Computing, Xiangtan, China, 2012, pp. 206–213. ISBN 978-1-4673-3027-5
https://doi.org/10.1109/CGC.2012.32

[31] Lili, S., Shoubao, Y., Liangmin, G., Bin, W. "A Markov chain based resource prediction in computational grid", In: 2009 Fourth International Conference on Frontier of Computer Science and Technology, Shanghai, China, 2009, pp. 119–124. ISBN 978-1-4244-5466-2
https://doi.org/10.1109/FCST.2009.32

[32] Kondo, D., Andrzejak, A., Anderson, D. P. "On correlated availability in internet-distributed systems", In: 2008 9th IEEE/ACM International Conference on Grid Computing, Tsukuba, Japan, 2008, pp. 276–283. ISBN 978-1-4244-2578-5
https://doi.org/10.1109/GRID.2008.4662809

[33] Kianpisheh, S., Kargahi, M., Charkari, N. M. "Resource availability prediction in distributed systems: An approach for modeling non-stationary transition probabilities", IEEE Transactions on Parallel and Distribute Systems, 28(8), pp. 2357–2372, 2017.
https://doi.org/10.1109/TPDS.2017.2659746

[34] Lua, E. K, Crowcroft, J., Pias, M., Sharma, R., Lim, S. "A survey and comparison of peer-to-peer overlay network schemes", IEEE Communications Surveys & Tutorials, 7(2), pp. 72–93, 2005.
https://doi.org/10.1109/COMST.2005.1610546

[35] Ripeanu, M. "Peer-to-peer architecture case study: Gnutella network", In: Proceedings First International Conference on Peer-to-Peer Computing, Linköping, Sweden, 2001, pp. 99–100. ISBN 0-7695-1503-7
https://doi.org/10.1109/P2P.2001.990433

[36] Yang, B., Gracia-Molina, H. "Improving search in peer-to-peer networks", In: Proceedings 22nd International Conference on Distributed Computing Systems, Vienna, Austria, 2002, pp. 5–14. ISBN 0-7695-1585-1
https://doi.org/10.1109/ICDCS.2002.1022237

[37] Kalogeraki, V., Gunopulos, D., Zeinalipour-Yazti, D. "A local search mechanism for peer-to-peer networks", In: CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management, McLean, VI, USA, 2002, pp. 300–307. ISBN 978-1-58113-492-6
https://doi.org/10.1145/584792.584842

[38] Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S. "Search and replication in unstructured peer-to-peer networks", In: ICS '02: Proceedings of the 16th international conference on Supercomputing, New York, NY, USA, 2002, pp. 84–95. ISBN 978-1-58113-483-4
https://doi.org/10.1145/514191.514206

[39] Jawhar, I., Wu, J. "A Two-Level Random Walk Search Protocol for Peer-to-Peer Networks", In: Proceedings of the 8th World Multi-Conference on Systemics, Cybernetics and Informatics, Orlando, FL, USA, 2004, pp. 1–5. ISBN 9789806560130

[40] Rhea, S. C., Kubiatowicz, J. "Probabilistic location and routing", In: Proceedings. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies, New York, NY, USA, 2002, pp. 1248–1257. ISBN 0-7803-7476-2 https://doi.org/10.1109/INFCOM.2002.1019375

[41] Tsoumakos, D., Roussopoulos, N. "Adaptive probabilistic search for peer-to-peer networks", In: Proceedings Third International Conference on Peer-to-Peer Computing (P2P2003), Linköping, Sweden, 2003, pp. 102–109. ISBN 0-7695-2023-5 https://doi.org/10.1109/PTP.2003.1231509

[42] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., Balakrishnan, H. "Chord: a scalable peer-to-peer lookup protocol for Internet applications", IEEE/ACM Transactions on Networking, 11(1), pp. 17–32, 2003. https://doi.org//10.1109/TNET.2002.808407

[43] Bin, D., Furong, W., Ma, J., Jian, L. "Enhanced Chord-Based Routing Protocol Using Neighbors' Neighbors Links", In: 22nd International Conference on Advanced Information Networking and Applications - Workshops (aina workshops 2008), Gino-wan, Japan, 2008, pp. 463–466. ISBN 978-0-7695-3096-3 https://doi.org/10.1109/WAINA.2008.53

[44] Rowstron, A., Druschel, P. "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems", In: IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware 2001), Heidelberg, Germany, 2001, pp. 329–350. ISBN 978-3-540-42800-8 https://doi.org/10.1007/3-540-45518-3_18

[45] Saleh, E., Shastry, C. "A new approach for global task scheduling in volunteer computing systems", International Journal of Information Technology, 15(1), pp. 239–247, 2023. https://doi.org/10.1007/s41870-022-01090-w

[46] Bundy, A., Wallen, L. "Breadth-First Search", In: Bundy, A., Wallen, L. (eds.) Catalogue of Artificial Intelligence Tools, Springer, 1984, p. 13. ISBN 978-3-540-13938-6 https://doi.org/10.1007/978-3-642-96868-6_25

[47] Kozen, D. C. "Depth-First and Breadth-First Search", In: The Design and Analysis of Algorithms, Springer, 1992, pp. 19–24. ISBN 978-1-4612-8757-5 https://doi.org/10.1007/978-1-4612-4400-4_4

[48] Grosan, C., Abraham, A. "Informed (Heuristic) Search", In: Intelligent Systems: A Modern Approach, Springer, 2011, pp. 53–81. ISBN 978-3-642-21003-7 https://doi.org/10.1007/978-3-642-21004-4_3

[49] Casanova, H., Giersch, A., Legrand, A., Quinson, M., Suter, F. "Versatile, scalable, and accurate simulation of distributed applications and platforms", Journal of Parallel and Distributed Computing, 74(10), pp. 2899–2917, 2014. https://doi.org/10.1016/j.jpdc.2014.06.008

[50] TU Delft: The Grid Workloads Archive "Gwa-t-13 materna", [online] Available at: http://gwa.ewi.tudelft.nl/datasets/gwa-t-13-materna [Accessed: 01 March 2022]

[51] Heinrich, F. C., Cornebize, T., Degomme, A., Legrand, A., Carpen-Amarie, A., Hunold, S., Orgerie, A. C., Quinson, M. "Predicting the energy-consumption of MPI applications at scale using only a single node", In: 2017 IEEE International Conference on Cluster Computing (CLUSTER), Honolulu, HI, USA, 2017, pp. 92–102. ISBN 978-1-5386-2327-5 https://doi.org/10.1109/CLUSTER.2017.66

[52] Orgerie, A.-C., de Assuncao, M. D., Lefevre, L. "A survey on techniques for improving the energy efficiency of large-scale distributed systems", ACM Computing Surveys, 46(4), 47, 2014. https://doi.org/10.1145/2532637