Int. j. inf. tecnol. (January 2023) 15(1):239–247
https://doi.org/10.1007/s41870-022-01090-w

ORIGINAL RESEARCH

A new approach for global task scheduling in volunteer computing systems

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

Received: 7 May 2022 / Accepted: 22 August 2022 / Published online: 3 September 2022

Abstract Volunteer computing networks are made up of a large number of computing devices owned by volunteers who want to donate their computing resources to help with large-scale scientific projects. The performance of these devices varies due to the available computing power, churn effect, and device heterogeneity, making equal workload distribution a significant challenge for the central server. To ensure that each device receives work that is proportional to its computing capability, we propose a new global scheduling algorithm based on the observed performance data provided by all connected computing devices in a peer-to-peer volunteer network. The experimental simulation results show that the main task is distributed in such a way that all devices exert the same amount of effort in terms of power consumption and execution time, yielding a very low values of the relative standard deviation for each of these two metrics.

Keywords Volunteer computing · Distribution · Scheduling · Peer-to-peer

1 Introduction

There is a growing interest in developing alternative low-cost and low-energy computing environments. We can develop such environments by connecting a large number of computing devices in a network and then distributing tasks among them by using any parallel programming algorithm such as message passing interface (MPI) or parallel virtual machine (PVM). Distributed Systems are a great example of these networks where all of the connected devices must be trusted, reliable, and available all the time of execution, making such computing-dedicated networks difficult to establish, especially outside academic-oriented institutional laboratories.

Volunteer computing (VC), on the other hand, is a form of distributed systems made up of thousands of heterogeneous connected devices, but instead of exploiting computing resources in an intrusive manner like most distributed systems, VC aims to use only the underutilized computing power of connected devices offered by geographically distributed volunteers [5, 24].

Task scheduling is regarded as the backbone of every volunteer system, as it determines the sequence in which tasks are executed and the computing resources that will be used to complete each task.The most important consideration when developing any task scheduling algorithm is how to reduce makespan and waiting time [27].

In general, there are two types of scheduling: global scheduling, which determines the processor that should be assigned a predefined-size task, and local scheduling, which is performed locally and determines the order in which tasks are executed on the same processor.

Because volunteer systems rely solely on voluntary participation, each device has unpredictable computing power, making it hard to estimate the size of the task that should be allocated to each device in the network. As a result, there will be an unbalanced distribution of power consumption and execution time. In this paper, we propose a new global task scheduling approach that estimates the appropriate amount of work that any computing device in a VC network

✉ Ehab Saleh
ehabosaleh@gmail.com

Chandrasekar Shastry
bs.chandrasekar@jainuniversity.ac.in

[1] Department of Computer Science and Engineering, Jain University, Bengaluru, India

can handle at any given time. This approach is expected to work properly in peer-to-peer networks (P2P), where numerous peers can act as servers and perform the global scheduling for their sub-peers rather than performing the scheduling on a single main server.

We have coined the term *Accumulated Computing Power* to describe how the available computing power of the a super-peer gradually increases as more sub-peers join. This theoretical concept allows each super-peer's scheduler to estimate the size of work that can be handled in each sub-peer in such a way that all sub-peers exhibit the a level of effort that is commensurate with its computing capability.

We used the simulation tool SimGrid [11] to analyse and compare the findings in this paper.

The rest of this paper is structured as follows: Sect. 2 includes some related works on global scheduling in existed volunteer systems, Sect. 3 explains the proposed approach in details, Secti. 4 demonstrates the network configuration, Sect. 5 discusses the experimental results, Sect. 6 concludes the paper, and Sect. 7 discusses the scope of future research.

2 Related work

Despite the fact that there are different architectures of VC frameworks, their scheduling systems have the same goal of eventually completing all tasks by avoiding busy computing resources and assigning tasks to the most available ones.

Task scheduling policies in VC systems can be categorized into two calsses according to Durrani and Shamsi [19, 22]: naive and knowledge-based.

1. Naive task scheduling policies: The scheduler does not consider the history of workers when assigning new jobs. The following policies are the most used in this scheduler:
   - First Come First Served (FCFS): In this scheduling strategy, the scheduler assigns a new job to the worker who requests it, regardless of whether or not that worker is available [13, 15, 28].
   - Locality Scheduling (LS): Tasks are assigned in a preferential manner under this scheduling policy, with only workers who have the necessary task data being selected [3, 14].
   - Random Assignment (RA): The tasks are assigned completely randomly, making the performance of the computing system fluctuates from time to time [4, 18].
2. Knowledge-based task distribution policies: The scheduler in this model of distribution considers the history of workers when assigning new jobs. Here are some examples of task distribution policies based on knowledge:
   - World Community Grid's Policy: This scheduling policy is based on the average amount of time the workers take to complete their assigned task before the deadline (Turnaround time) [14, 26].
   - Work Fetch Policy: Each worker requests a number of tasks to be queued locally for execution, keeping each worker as busy as possible until the next connection with the server [6, 18].
   - Threshold-Based Policies: For scheduling purposes, there are two thresholds to be considered: one is to determine availability, while the other is to determine reliability [13].
   - Buffer None Policy: In the buffer-none policy, each worker receives one task from the server at a time, and when the task is completed, the worker requests another task from the server [26].
   - Buffer *N* Days Policy: Under this scheduling approach, the worker is assigned various tasks to be stored in a buffer and performed one by one in a specific order, and when they are nearly completed, the worker requests the server for more set of tasks. It is called buffer *N* because the worker stores *N* number days of work [26].
   - Weighted Round-Robin: Implementing round-robin scheduling approach among projects on various servers and allocating weights based on resource share [18].
   - Priority Round-Robin: Workers with similar computing resources are clustered together to ensure the accuracy of findings and, as a result, to shorten overall task's execution time. A priority queue is used to sort all workers according to their computing time, and workers with the highest priority are chosen first to complete unfinished tasks [21].

Several real-world VC systems have employed some of the aforementioned scheduling policies for task distribution:

Some java-based volunteer frameworks such as Bayanihan [25] and Charlotte [8] use *Eager Scheduling* [12]. It is based on buffer-none policy where packets of the task to be executed are stored in a pool on the server, and whenever there is a client available, it receives new packets when requested. In this case, the fastest clients get more work because they always finish earlier.

Networks of Workstations (NOW) [10] and ATLAS [7] use *Work Stealing* [9] as a global scheduling approach that is based on the idea of acquiring jobs from other nodes when they are busy. It employs a scheduling strategy similar to the buffer *n* days, in which each node has its own work queue where task items are placed, but when a sub-node runs out

of task items, it hunts for other uncompleted task items in other work queues and steals them.

Neary and Cappello proposed in [20] a combination of eager scheduling and work stealing in one scheduler to improve load balance and achieve fault tolerance and scalability.

XtremWeb [16] is a global computing framework that was designed to bring two new aspects to the VC systems: high performance that is based on efficient scheduling, and multi-applications that allows institutions to build their own computing systems. The scheduler is divided into two parts: the dispatcher, which selects a collection of tasks and delivers them to the scheduler; and the scheduler, which is in charge of assigning tasks to workers as they request them by implementing buffer-none policy.

GridMP [1] is also a commercial distributed computing platform that uses buffer-none policy. Many projects, primarily linked to cancer research and analyzing human protein folding, have been launched using the GridMP framework.

Most of the volunteer projects that are currently in operation are based on an open source framework called Berkeley Open Infrastructure for Network Computing (BOINC) [5, 24].

On the server side of BOINC, job scheduling is based on job runtime estimation because it is used to calculate the number of jobs required to fulfill the client's request and to estimate the turnaround time to ensure that the job deadline is not exceeded. This number of jobs that the client can request is determined not only by the estimated runtime but also by the platform that the client uses, as well as the hardware and software description.

Table 1 compares all of the VC frameworks previously mentioned in terms of scheduling policy and network architecture.

In comparison to our proposed scheduling approach, our golabal scheduler seeks to balance power consumption and execution time among all nodes rather than having the fastest node perform the left task's packets. We want the scheduler to distribute packets evenly over all network nodes, and if any packets remain to be done, the super-node will wait until all of its sub-nodes have completed their execution before redistributing these packets to them. To do so, all sub-nodes must provide their super-node with all of the essential performance metrics, which allow the scheduler to determine how much work each sub-node can perform.

Furthermore, all of the aforementioned schedulers and policies have only been implemented in client-server volunteer networks where only one server operates the scheduler, and because no previous study has addressed task scheduling in P2P volunteer networks, this paper will be the first in this field.

Table 1 Volunteer frameworks and task scheduling policies

Volunteer framework	Schedulingpolicy	Network structure
Bayanihan	Buffer none	Client–server
Charlotte	Buffer none	Client–server
NOW	Buffer n days	Client–server
ATLAS	Buffer n days	Client–server
BOINC	Buffer none/buffer n days	Client–server
GridMP	Buffer none	Client–server
Xtremweb	Buffer none	Client–server

3 Proposed method

As previously stated, the network is P2P, meaning that any peer can operate as a client and server at the same time, and it uses shared tables to store addresses and performance data of all nodes.

In the practical implementation, we need two shared tables: *Network Tree* and *Computing Tree*, where stored data is expressed as a collection of two tuples of the form $<$ *Key* : *Value-1, Value-2, Value-3,...*$>$.

In Network Tree, we store the IP address of a super-node as a key, and in the values field we store the IP addresses of all connected sub-nodes. Computing Tree, on the other hand, has a structure identical to Network Tree, except that we store the computing capability of connected sub-nodes in the values field.

Any node can access these two tables at any time, and any updates to the configuration, such as changes in available computing power or nodes joining and leaving the network, will be reflected in these two tables instantly.

Rather than considering each node in terms of the computing power it provides [floating point operations per second (FLOPS)], we define *Accumulated Computing Power* as the summation of the super-node's computing power with all of the connected sub-nodes' computing power.

It is accumulated because the overall computing power may be updated gradually as sub-nodes join or leave the network, or the available computing power of any connected node may change over time. This concept is solely used to make it easier for the scheduler in each super-node to properly distribute the task among all of the sub-nodes based on the accumulated computing power available.

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

$$FLOPS_i^{Local} = FLOPS_i^{Peak} \times a_i \times c_i. \quad (1)$$

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

The accumulated available FLOPS of a given super-node i with n sub-nodes can be calculated as follows :

$$FLOPS_i^{Total} = FLOPS_i^{Local} + \sum_{j=1}^{n} FLOPS_j^{Local} \quad (2)$$

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

$$FLOPS^{Load'} = est\_flop\_count_{i,t}/FLOPS_i^{Total} \quad (3)$$

Equation (3) is only applied if all nodes have one core per processor with availability 100%, but since each node 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 node i :

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

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

$$FLOPS_i^{Load} = FLOPS^{Load'} \times FLOPS_i^{Local} \quad (5)$$

In real-world applications, the programmer must estimate the actual *est_flop_count* of his algorithm, which necessitates the consideration of a number of additional factors that are outside the scope of this paper.

The proposed algorithm's pseudocode is as follows:

Algorithm 1 Global Task Scheduling Algorithm

Require: Network with only one super-node(N), estimated task size (est_flop_count), processor's peak theoretical floating-point($FLOPS^{Peak}$), number of cores per processor (c), cpu availability (a).
Ensure: Workload per node

```
1: Workload ← empty list
2: for node i in N do
3:     FLOPS_i^Local ⇐ FLOPS_i^Peak × a_i × c_i
4:     if node i is supernode then
5:         FLOPS^Total ⇐ FLOPS_i^Local
6:     end if
7: end for
8: for node j is subnode to node i do
9:     FLOPS^Total ⇐ FLOPS^Total + FLOPS_j^Local
10: end for
11: FLOPS^Load' ⇐ est_flop_count/FLOPS^Total
12: for node i in N do
13:     Workload.add(FLOPS^Load' × FLOPS_i^Local)
14: end for
15: return Workload
```

There are three main loops: the first loop starts at *line 2* and computes $FLOPS^{Local}$ for each node in the network N, the second loop starts at *line 8* and computes $FLOPS^{Total}$ for the super-node, and the third loop starts at *line 12* and computes the workload that should be assigned to each node in the network (including the super-node).

The pseudocode only applies if we have a single super-node and its associated sub-nodes. However, in a P2P network with multiple super-nodes that could be sub-nodes to other super-nodes, we apply the algorithm in each super-node separately beginning first with low level super-nodes.

4 Network configuration

To conduct the simulation, we constructed a P2P computing network consists of 1500 virtual computing nodes, with each node representing a volunteer's device in the real world. We described the specifications of each node and the type of connection that exists between them in an external xml file.

We used the dataset GWA-T-13 Materna [2] to get the availability files for all of the network's connected nodes. It contains performance data described as trace files of over 1500 VMs of the distributed Materna Data Centers in Dortmund-Germany.

Figure 1 shows the available computing power percentage of three randomly chosen processors from the dataset. We can see that each processor has an unique availability over time, which will have an impact on the available $FLOPS_i^{Local}$ calculated from Eq. (1).

We have also set a threshold (50%) which the CPU pause executing the allocated job if its available percentage falls below this threshold to avoid using the host's CPU in an intrusive manner and to limit CPU heat.

As we can see in Table 2, all nodes are divided into groups based on their number of cores. Each group has its power consumption model that has four parameters as follows:

1. **Idle**: Wattage when the node is up, but without performing anything.
2. **Epsilon**: Wattage when all cores are not performing the allotted work, but they are not in idle state.
3. **Allcores**: Wattage when all cores of the node are performing the allotted work.
4. **Off**: Wattage when the node is turned off.

Any node's power consumption in the network can be represented as a sum of two parts: the static part describes power usage while the node is turned on or off, and the dynamic part describes power consumption while the CPU is operating. The static part is easy to observe, but the dynamic part is proportional to the CPU load [23].

SimGrid comes with a plug-in called *plugin_host_energy* [17] that calculates the amount of power consumed by each node in the network by adding the static and dynamic parts of the consumed power. The plugin uses Eq. (6) to calculate

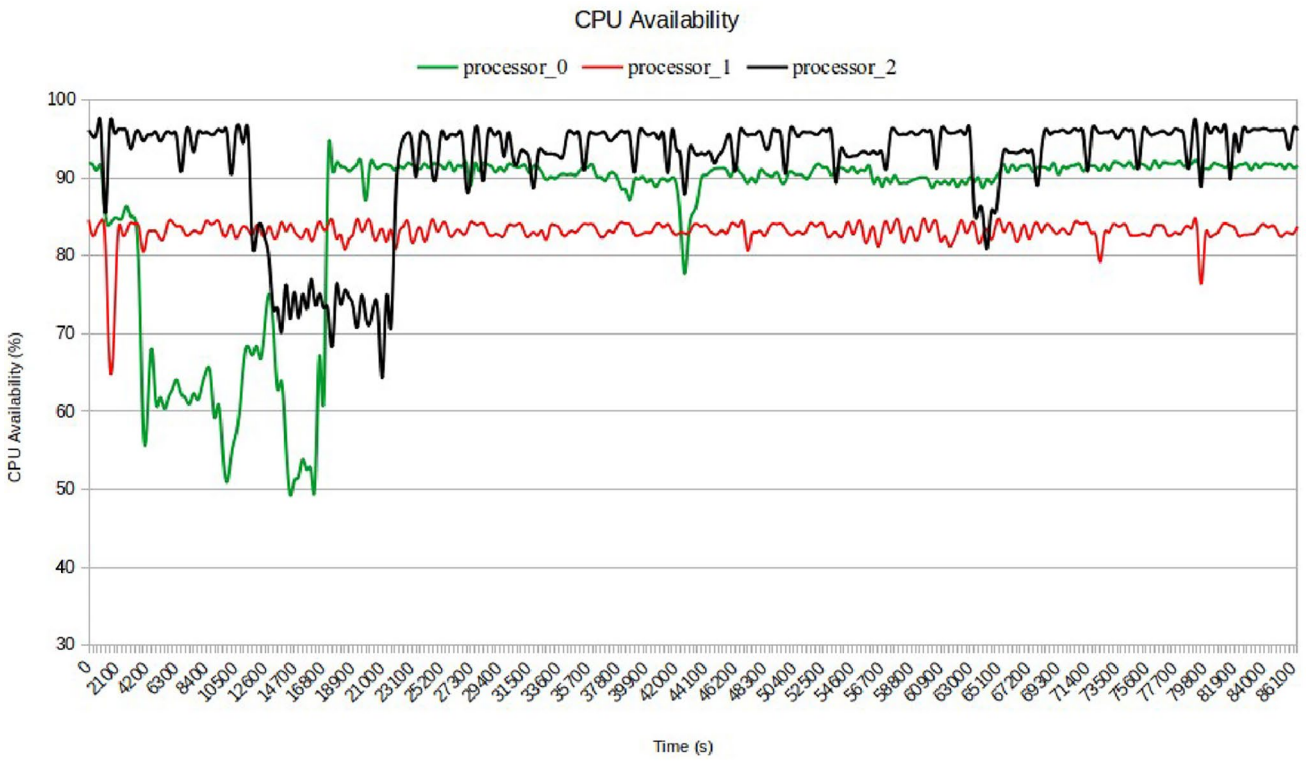

Fig. 1 A one-day sample of the availability of three random processors

the total power consumption for a given processor i that has the frequency f, the workload w and the usage percentage u.

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

5 Simulation results

To conduct experiments and evaluate our approach, we considered various experiments in which we distributed different sizes of tasks given in FLOPS, which are: one TeraFLOPS, ten TeraFLOPS, one hundred TeraFLOPS, one PetaFLOPS, ten PetaFLOPS, and one hundred PetaFLOPS. In each experiment, we measured the execution time and the power consumption of each node, as well as the relative standard deviation to determine how evenly the tasks were distributed.

Bars in Fig. 2 represent the execution time in each node when we use the proposed scheduling approach to distribute a 100 PetaFLOPS task to all of the participating nodes. We can see that the values are relatively close, with the exception of a few nodes where the processor suspended execution because the available computing power fell below the predefined threshold, causing other nodes in the same network zone to spend extra time completing the left job. On the other hand, the observed consuming energy values are depicted in Fig. 3. We can see that even though each node has its own energy specifications that vary depending on the workload and the number of working cores, the values are quite close to each other. We evaluate the relative standard deviation (RSD) for each experiment in our simulation, which measures how far the observed samples deviate from the mean value of a given population. We seek the smallest value of RSD in each experiment, and the higher the value we get, the further the samples are from the mean value.

For a population that has σ as standard deviation and μ as mean, RSD is calculated as follows.

$$CV = \frac{\sigma}{\mu} \tag{7}$$

For each of the experiment tested, the RSD percentage of the execution time is shown in Fig. 4. We can see that RSD values are negligible in all cases except for the experiments of distributing 10 PetaFLOPS and 100 PetaFLOPS among all nodes, where RSD values are also too small (2.3% and 5% respectively), illustrating that the execution times of all nodes are strongly clustered around the mean in all experiments.

Figure 5 shows RDS values of the power consumption for each experiment. Similarly, despite the relative increase in RSD values reaching approximately 1.2% when we

Table 2 Specification of nodes based on the number of cores

No.cores/node	No.nodes	Peak GFLOPS/core	Power consumption parameters (Watt)			
			Idle	Allcores	Epsilon	Off
One-core CPU	202	5	100	140	120	10
Two-core CPU	953	5	100	160	120	10
Four-core CPU	242	5	100	200	120	10
Six-core CPU	39	5	100	240	120	10
Eight-core CPU	64	5	100	280	120	10

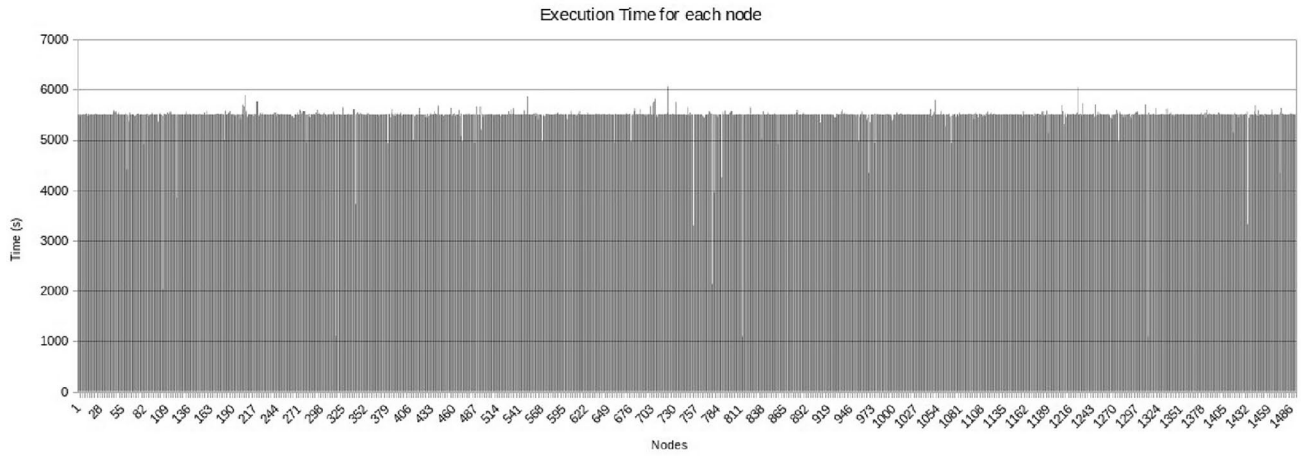

Fig. 2 The execution time when using the proposed scheduling approach

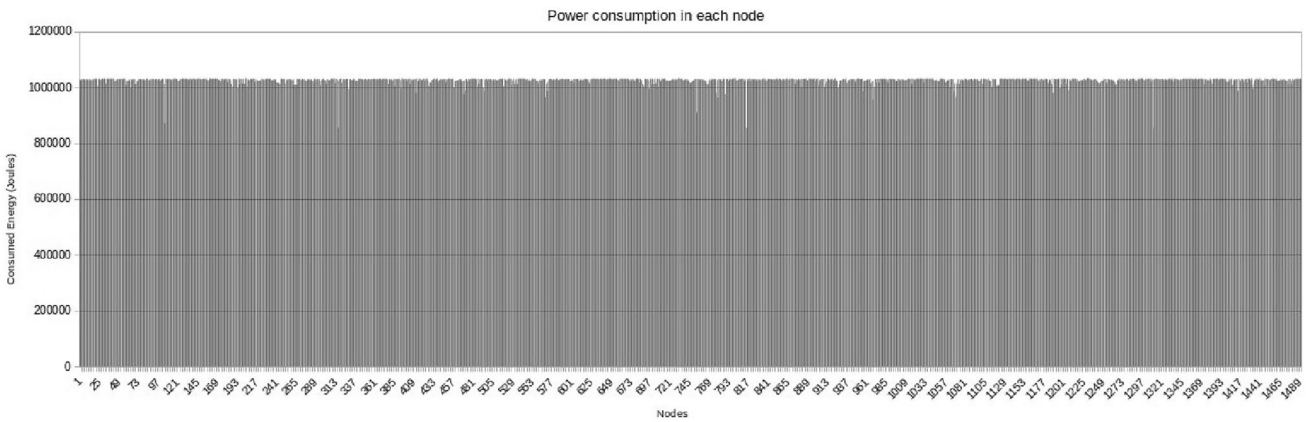

Fig. 3 The power consumption when using the proposed scheduling approach

distribute 100 PetaFLOPS, RSD values indicate low percentages, showing that nodes consume power equally.

6 Conclusion

In this paper, we proposed a new approach for global scheduling in P2P volunteer networks. We introduced the term *Accumulated Computing Power* to describe how the current super-node's available computing power gradually increases as more sub-nodes join, allowing the global scheduler at each super-node to adapt the heterogenous architecture of its sub-nodes and estimate the proper size of the task that can be handled in each sub-node, while ensuring that all sub-nodes exhibit the same level of effort in terms of power consumption and execution time. According to the experimental simulation results, the main

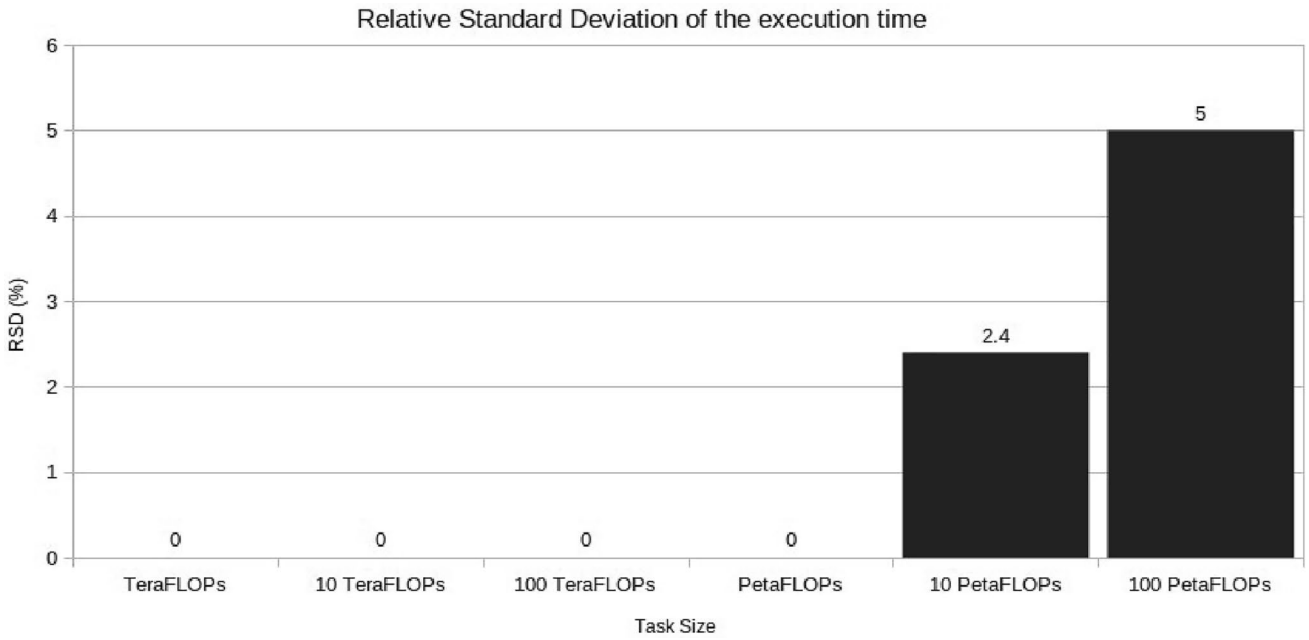

Fig. 4 RSD of the execution time for each experiment

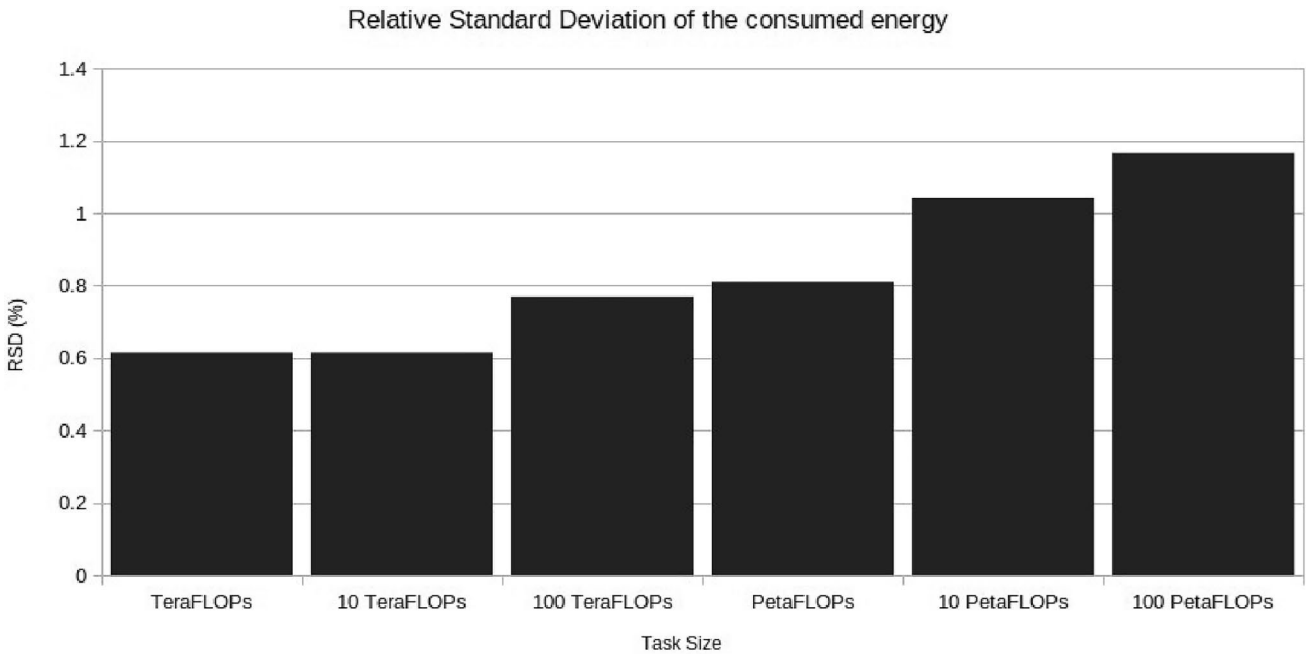

Fig. 5 RSD of the power consumption for each experiment

task was distributed in such a way that each node received a work that is proportional to its computing capability, resulting in a very low values of the relative standard deviation for both the execution time and the power consumption (Figs. 4 and 5).

7 Future scope

This study aims to develop a new global scheduling technique for P2P VC systems. Each node in the network receives amount of work proportional to its available computing power. However, the suggested method can be

extended to give more efficient performance and there is always some possibility of improving our approach to a more sophisticated version.

As a result of the present approach, some ideas for the feature have been introduced:

- Availability prediction: Every time the server initiates the scheduling procedure, it must first investigate the performance metrics of all its sub-nodes, which increases the overall task execution time. We believe that employing prediction algorithms to estimate the availability of these nodes ahead of time will save time that would otherwise be spent enquiring about performance metrics.

  Using prediction algorithms necessitates using the history of all computing devices to train the prediction model prior to initiate our approach.
- Computational Complexity: The entire proposed algorithm relies on the size of the task to be executed. SimGrid simulation tool allows us to pass the task size in number of floating operations. In the real implementation of our approach, it is a big challenge that necessitates calculating the computing complexity of the task before launching our scheduling algorithm.

References

1. Aggregating computational resources. https://stats.distributed.net/projects.php?project_id=28. Accessed 30 05 2022
2. Gwa-t-13 materna. http://gwa.ewi.tudelft.nl/datasets/gwa-t-13-materna. Accessed 07 2021
3. Anderson DP, Korpela E, Walton R (2005) High-performance task distribution for volunteer computing. In: First international conference on e-science and grid computing (e-Science'05). pp 8–203. https://doi.org/10.1109/E-SCIENCE.2005.51
4. Anderson DP (2011) Emulating volunteer computing scheduling policies. In: 2011 IEEE international symposium on parallel and distributed processing workshops and Phd forum. pp 1839–1846. https://doi.org/10.1109/IPDPS.2011.343
5. Anderson DP (2019) Boinc: a platform for volunteer computing. J Grid Comput 18:99–122
6. Anderson DP, McLeod J (2007) Local scheduling for volunteer computing. In: 2007 IEEE international parallel and distributed processing symposium. pp 1–8. https://doi.org/10.1109/IPDPS.2007.370667
7. Baldeschwieler JE, Blumofe RD, Brewer EA (1996) ATLAS: An infrastructure for global computing. In: Proceedings of the 7th workshop on ACM SIGOPS European workshop: systems support for worldwide applications, EW 7. Association for Computing Machinery, New York, pp 165–172. https://doi.org/10.1145/504450.504482
8. Baratloo A, Karaul M, Kedem ZM, Wijckoff P (1999) Charlotte: metacomputing on the web. Future Gener Comput Syst 15:559–570
9. Blumofe RD, Leiserson CE (1999) Scheduling multithreaded computations by work stealing. J ACM 46(5):720–748. https://doi.org/10.1145/324133.324234
10. Blumofe RD, Lisiecki PA (1997) Adaptive and reliable parallel computing on networks of workstations. In: Proceedings of the USENIX annual technical symposion. pp 133–147
11. Casanova H, Giersch A, Legrand A, Quinson M, Suter F (2014) Versatile, scalable, and accurate simulation of distributed applications and platforms. J Parallel Distrib Comput 74(10):2899–2917 http://hal.inria.fr/hal-01017319
12. Chen Hl, King CT (1996) Eager scheduling with lazy retry for dynamic task scheduling. https://doi.org/10.1007/BFb0024755
13. Estrada T, Flores DA, Taufer M, Teller PJ, Kerstens A, Anderson DP (2006) The effectiveness of threshold-based scheduling policies in boinc projects. In: 2006 second IEEE international conference on e-science and grid computing (e-Science'06). pp 88–88. https://doi.org/10.1109/E-SCIENCE.2006.261172
14. Estrada T, Fuentes O, Taufer M (2008) A distributed evolutionary method to design scheduling policies for volunteer computing. SIGMETRICS Perform Eval Rev 36(3):40–49. https://doi.org/10.1145/1481506.1481515
15. Estrada T, Taufer M, Anderson D (2009) Performance prediction and analysis of boinc projects: an empirical study with emboinc. J Grid Comput 7:537–554. https://doi.org/10.1007/s10723-009-9126-3
16. Fedak G, Germain C, Neri V, Cappello F (2001) Xtremweb: a generic global computing system. In: Proceedings first IEEE/ACM international symposium on cluster computing and the grid. pp 582–587. https://doi.org/10.1109/CCGRID.2001.923246
17. Heinrich FC, Cornebize T, Degomme A, Legrand A, Carpen-Amarie A, Hunold S, Orgerie A-C, Quinson M (2017) Predicting the energy-consumption of mpi applications at scale using only a single node. In: 2017 IEEE international conference on cluster computing (CLUSTER). pp 92–102. https://doi.org/10.1109/CLUSTER.2017.66
18. Kondo D, Anderson DP, Vii JM (2007) Performance evaluation of scheduling policies for volunteer computing. In: Third IEEE international conference on e-science and grid computing (e-Science 2007). pp 415–422. https://doi.org/10.1109/E-SCIENCE.2007.57
19. Kopal N (2018) Secure volunteer computing for distributed cryptanalysis. Ph.D. thesis. https://doi.org/10.19211/KUP9783737604277
20. Neary MO, Cappello P (2005) Advanced eager scheduling for java-based adaptive parallel computing: research articles. Concurr Comput Pract Exp 17(7–8):797–819
21. Ngo SH, Fukushi M, Jiang X, Horiguchi S (2008) Efficient scheduling schemes for sabotage-tolerance in volunteer computing systems. In: Proceedings of the 22nd international conference on advanced information networking and applications, AINA '08. IEEE Computer Society, USA, pp 652–658. https://doi.org/10.1109/AINA.2008.129
22. Nouman Durrani M, Shamsi JA (2014) Volunteer computing: requirements, challenges, and solutions. J Netw Comput Appl 39:369–380. https://doi.org/10.1016/j.jnca.2013.07.006
23. Orgerie AC, de Assunção MD, Lefèvre L (2013) A survey on techniques for improving the energy efficiency of large-scale distributed systems. ACM Comput Surv (CSUR) 46:1–31
24. Saleh E, Rajesh SL (2021) High-performance cryptanalysis: a comparative study of code-breaking techniques. In: Proceedings of the international conference on innovative computing and communication (ICICC). https://doi.org/10.2139/ssrn.3833299
25. Sarmenta LF, Hirano S (1999) Bayanihan: building and studying web-based volunteer computing systems using java. Future Gener Comput Syst 15(5):675–686. https://doi.org/10.1016/S0167-739X(99)00018-7
26. Toth D, Finkel D (2009) Improving the productivity of volunteer computing by using the most effective task retrieval policies. J Grid Comput 7:519–535. https://doi.org/10.1007/s10723-009-9133-4

27. Yadav S, Mohan R, Yadav P (2018) Fuzzy based task allocation technique in distributed computing system. Int J Inf Technol. https://doi.org/10.1007/s41870-018-0172-6
28. Zamudio R, Brooks CL, Armen R, Kerstens A, Teller PJ, Taufer M, Flores DA, Estrada TP (2007) Moving volunteer computing towards knowledge-constructed, dynamically-adaptive modeling and scheduling. In: 2007 IEEE international parallel and distributed processing symposium. IEEE Computer Society, Los Alamitos, pp 478. https://doi.org/10.1109/IPDPS.2007.370668