GSJ: Volume 8, Issue 11, November 2020, Online: ISSN 2320-9186
www.globalscientificjournal.com

A Framework for Peer to Peer General Purpose GPU Distributed Computing

L.W. Amarasinghe[1*], I.G.A.S. Wijerathna[2]

Department of Statistics and Computer Science, Faculty of Science, University of Peradeniya

Department of Software Engineering, Sri Lanka Institute of Information Technology (SLIIT) Malabe, Sri Lanka.

*lakshithaw@sci.pdn.ac.lk

Abstract. In this present world, we can see a rapid revolutionary development in the field of distributed computing. As the complexity of the problems that the computers can also solve gradually increases. Hence, Advancements of technologies and more computational power is essential to complete more complex algorithms. Therefore, such kinds of algorithms need large network servers. But we can't anticipate such powerful resources are available to everyone. If there is a possibility to share its resources with another computer, it makes a great revolution. This research work proposed a peer-to-peer network that allows sharing idle processing power among other peers who are connected to the existing network. Hence, the shared processing power can be used to execute a complex problem by distributing it. The intention of this process to speed up the ordinary problem-solving procedure by parallel algorithms and parallelized data. In such scenario, proper scheduling and data splitting mechanism can improve the performance, cost of computation, and increase the reliability of the system. The application is based on a peer-to-peer network since there is a high probability of occurring faults. To reduce these errors, the system is implemented with fault tolerance mechanisms.

Keywords: GPGPU computing, Distributed computing parallel algorithms, peer-to-peer network, Scheduling, Fault tolerance, Machine learning, CUDA, OpenCL

1 Introduction

With the introduction of personal computers (in 1975's) to users, the processing power becoming cheaper as the semiconductor technology improved consequently with the time. According to Moore's law, that the number of transistors in integrated circuits approximately doubles every two years [1]. As the result of that most of the general-purpose computers are more capable than most advanced computers in decades ago.

With the development of internet technologies such as speed and bandwidth capabilities and usage of advanced computer hardware equipment by users in the modern world, there's an emerging growth towards finding new ways that allow sharing resources within a network of interconnected computers. That will in return cut down the cost of computing. Distributed computing is an environment in which a group of independent and geographically dispersed computer systems takes part to solve a complex problem, each by solving a part of a problem 1and then combining the result from all computers [2]. Public-resource computing (also known as "Global Computing" or "Volunteer Computing") uses these resources to do scientific supercomputing. This paradigm enables previously infeasible research. It also encourages public awareness of current scientific research [3]. General purpose computers are equipped with high-performance CPUs and GPUs. Most of the time they are underutilized, and that leads to resource and power wastage. If there is a possibility of harnessing the idle processing power, that can be used to solve complex problems such as scientific problems and other general problems.

2 Related Work

There were many general public based projects initiated to make it an efficient use of computer resources. BitTorrent is one project which benefitted by this paradigm that allows file sharing within peers. BOINC is the most popular project that takes the advantage of this technology and harnesses the idle power of CPU and GPU to processing intensive problem-solving. Graphical Processing Unit is a processor dedicated to the computer for Graphical and video related processing activities, but it also can be used for general purpose computing. However, most of the public resource computing projects which share the GPU is still not available for the general public. A working decentralized system of GPU process sharing mechanism is yet to be found. It is the purpose of this research to design and implement such a system. The planned procedure is to design the network layer separately at first, build the execution layer on top of that and create a user agent software. Ideally, the outcome would be a decentralized (peer-to-peer) distributed GPU and CPU computing platform that's globally scalable and available to the general public as a way to access high-performance computing.

BOINC (Berkeley Open Infrastructure for Network Computing) is a platform for public-resource distributed computing [3]. A BOINC project corresponds to an organization or research group that does public-resource computing. It is identified by a single master URL, which is the home page of its website and also serves as a directory of scheduling servers. Participants register with projects. The server complex of a BOINC project is centered around a relational database that stores descriptions of applications, platforms, versions, work units, results, accounts, teams, and so on. Server functions are performed by a set of web services and daemon processes. Scheduling servers handle RPCs from clients; it issues work and handles reports of completed results. Data servers handle file uploads using a certificate-based mechanism to ensure that only legitimate files, with prescribed size limits, can be uploaded. File downloads are handled by plain HTTP.

distributed.net was the Internet's first general-purpose distributed computing project: [4] that is attempting to solve large-scale problems using idle CPU or GPU time. distributed.net is working on RC5-72 (breaking RC5 with a 72-bit key), and OGR-28 (searching for the optimal 28-mark Golomb ruler). "dnetc" is the file name of the software application which users run to participate in any active distributed.net project. It is a command line program with an interface to configure it, available for a wide variety of platforms. distributed.net refers to the software application simply as the "client." In recent years distributed.net has been started to support on GPU Processing. RC5-72 Designed to run on NVidia-CUDA enabled hardware and ATI Stream-enabled hardware. There is Open CL based client has been implemented for distributed.net that is supported by all the leading GPU manufacturers (AMD, NVidia, and Intel) [5].

CrowdCL, an open-source, web-based, cross-platform, volunteer computing framework [13]. CrowdCL allows developers to write high-performance web applications that can be executed as background processes within a web page. This avoids the need for active installation on user's computers and allows developers to create and distribute cross-platform volunteer computing applications that take advantage of more and more ubiquitous hardware accelerators.

3 Methodology

The main intention of the system is to build a non-centralized peer-to-peer network that can be used for sharing of computer resources. In the primary skeleton, the system will be implemented in a distributed computing aspect for effective use of computational power and scheduling workloads on nodes that are connected to the network. To implement these features itself, it reveals with some technical aspects and technologies such are,

A. Network

1) The system is to handle the functionalities of the network without a centralized server thus all the maintenance of a proper communication mechanism is to achieve in a peer to peer manner. The system is capable of supporting few nodes connected into a vast area of nodes connected spanning over a large area.

2) Peer Management

When a peer wants to join to the network, it should follow certain procedures to have a successful connection with the network. To initiate the connection, the new peer needs to know the address of at least one of the nodes which are already in the network. After that, the peer that needs to connect to the network has to send a join request to the peer which is already in the network. While connecting a Kademlia Id will be generated automatically by considering Node Id, UDP Port, and IP Address. If the peers that are already in the network's k-bucket have space to the joining peer, it will be added to the list. Else it has to refresh the K-bucket to free up space in the k-bucket. If that fails another peer in the list is fetched randomly for the connecting process. To refresh a K-bucket, a peer will send a message requesting a response from the peers in the list. In case of a peer failed to respond that peer will be eliminated from the k-bucket.

Depending on the size of the network number of connection that a single node can maintain can be managed manually. After when a new peer joined the existing network, it needs to identify other peers in the network to do this the new peer asks for the addresses of other nodes from the connected peer. On receiving the list, the new node tries to establish the connection with the other nodes.
To actively maintain the network with within a specific time, limit a hello message is distributed by each peer to its neighboring peer to notify the active state of the peer. If a node unable to send a hello message within the time frame that node will be removed from the routing table.
The rating of each peer connected is also advertised in the network in the DHT. So when distributing a workload, a node can look at its DHT and find the peer which it wants to hand over its workload to process.
For connection termination, it can occur in 2 different ways, either a node can exit from the network selecting shutdown which will also inform other peers that this node is going to leave the network then after its address would be removed from other routing tables. Moreover, also a node could have left the network without following the exit procedure and to leave the network itself. In such situations, the node which is left would be eliminated from the routing tables of the other nodes after a certain specific time frame.
The system also uses application-specific protocols to communicate with other peers, to ask for tasks, respond the status of the task, and send and receive back the workload the system uses specific messages with separate CODE attribute to identify each of the messages. Using of such kind of protocols makes the system efficient and cut down the bandwidth usage of the network.

B. Workload Allocator

The user able to upload a source code and related data to the application. Workload allocator module used to parallelize the data to process by other peers. Large workloads will be split into independent manageable parts. Even though the Data will be split automatically, the system unable to capture any dependencies between split data. If there are any dependencies available, the user should define rules to split the data to avoid errors. The user can define rules and parameters in XML format. Defined rules should be uploaded to the system. Split workloads will be managed by the scheduler.

C. Scheduler

The main advantage of proposed system is that the overloaded peer can share its workload with the other free peers. In order to share the workloads with the other peers, it should be happening perfectly otherwise the computation time will be increased and lead to wrong results. To reduce this failure, the algorithm should be work like an external scheduler. Scheduling means order of jobs which satisfy the metrics like users' satisfaction and completion time and so on [6]. Better scheduling algorithm can improve the performance, cost of computation, load balancing and increase the reliability of the system. The algorithm should consider the factors like CPU, GPU computation power, idle time of the peers and the free peers in the non-centralized network, and also the algorithm should be able to determine how many nodes to deliver the workload and select the best nodes to deliver the workload depending on load and capacity of the peers. Once the workload has been allocated to the peers they execute and send the results to the scheduler. When all the results have been received by the scheduler the algorithm should be able to aggregate the results to get the final outcome. Since scheduling is a NP hard problem, finding an exact solution for larger problems is not possible. Instead an approximate solution can be achieved. Many heuristic scheduling algorithms have been proposed that provide approximate solutions to problems. [7, 8]. There are many heuristic methods for static and dynamic scheduling among peers. But many of the algorithms are not considers about the current load on the resource. The main objective of heuristic algorithms is to minimize makespan which is the maximum completion time spent for execution of a batch of tasks [9].
The existing peer-to-peer framework is using a Hybrid scheduling algorithm. As the name implies the hybrid algorithm is considering two major impacts of the scheduling heuristics. Such are,

1. Minimum completion time to complete the task
2. Current load on the resource

The architecture of the scheduler is given in figure 1. The users submit the tasks to the scheduler and the assigns the submitted tasks to the available resources based on the load of the resource and the consideration of time taken to complete the task.

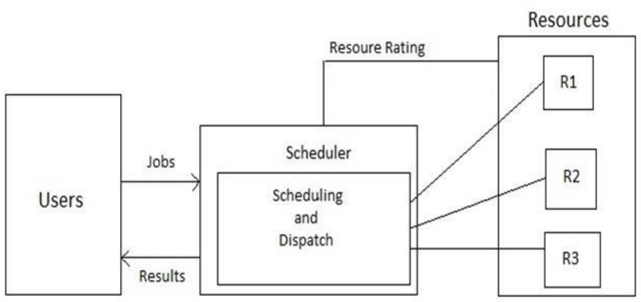

Fig. 1. Scheduler architecture

The scheduler gets the resource rating and efficiency to calculate the estimated execution time.

$$efficiency = \frac{how\ long\ take\ to\ complete\ task\ with\ 1\ core}{how\ long\ take\ to\ complete\ with\ all\ cores}$$

The scheduler gets the resource rating and efficiency to calculate the estimated execution time.

$$efficiency = \frac{how\ long\ take\ to\ complete\ task\ with\ 1\ core}{how\ long\ take\ to\ complete\ with\ all\ cores}$$

Rating	With 1 core	With all cores	Efficiency
2.4	636.4	323.4	1.97

Table 1. Machine rating vs Efficiency

Where, Estimating the time,

$$estimated\ time = \frac{1/task\ efficiency}{machine\ rating}$$

Every time when the task is received the arrival time of the tasks will be captured in an array known as **arrivals.** Machine availability time, the time at which machine finishes all previously assigned tasks **Mat.** The completion time of the task will be calculated as,

$$Completion\ Time\ = arrival\ time + estimated\ \ calulation\ time$$

The main intension of the proposed algorithm is to minimize the makespan as well as to balance the load on the resources. In this work, the tasks are mutually independent and the estimated execution time and arrival time of the tasks are randomly generated for the simulation process.

Number of machines (m)	16
Number of Tasks (n)	512
Meta Set Size (S)	29
Machine available time (mat[])	new int[m]

Table 2. Network Resource Characteristics and Scheduling parameters

D. Workload Executer

After delivering a workload to a peer node, execution is handled by the workload executer module. The workload consists of the data and the instructions for the execution (source code). Source code can be a GPU kernel or any other Java code block. The kernel should be compiled before execution. Compilation of the code is done by OpenCL [12] and CUDA [11] compilers. This allow to users able to run throughput intensive code even without a high-end GPU. After completing the processing, results will be sent back to the originator.

E. Fault Tolerance

To provide a reliable and robust environment, it is important that means of resilience come with the system. The major concern in a peer-to-peer computation time-sharing system is the reliability, as a single node failure fails all running applications on the node [10]. In a volunteer computing platform, there can be misbehaving nodes that may lead to an inaccurate result. The validity of the results is of utmost importance and cannot be ignored. All the workloads submitted to the system should be executed efficiently. Since the nodes in the network are free to connect and disconnect from the network anytime, a node can disappear from the network even without completing allocated workload execution. To maintain reliability of the system there should be Result Validation and Work Resilience mechanism.

1) Work Resilience

During the scheduling process, workload allocator node submits the source code and portion of data to the worker nodes those are willing to accept tasks. After submitting the task to be executed by a peer, the originator node waits for the results. However, the worker node may have disconnected from the network the reason can be network failure, system crash or user leaving the network. There should be a mechanism to keep track of the allocated workload, and in case of failure, the workload should be reallocated and execute again.

To keep track of the workload, originator node uses Task Status Messages. It checks the status of the worker node periodically. Every worker replies the status of the execution. In case of no reply from the worker node, originator node waits for specific time to results to arrive, and if there is no reply from worker node in given period, the worker will be removed from originator's routing table, and workload will be assigned for another peer.

Fetching for a new node and send the workload to that peer is a time consuming process and it reduces the efficiency of the system. As a solution for this, a primary node and backup node will be used. Primary nodes assigned such that every primary node has a backup node. Most reliable nodes in the network will be selected as primary nodes. Those nodes are selected based on Machine Learning algorithm. Originator node sends the workload to the primary peer and the backup node. However, workload execution only starts at the primary peer. If a primary peer vanishes from the network and it will be detected by the originator, and it will simply send start execution message to the backup node. Hens the additional workload transfer time is saved.

2) Peer Redundancy

In some situations, there can be a fault in the originator node, and it will disappear from the network before receiving the results. In such scenarios, there should be a recovery mechanism for results generated by primary peers.

When a new peer is joining to the existing network, it chooses some other node as a redundant peer. Originator node always synchronizes its state with redundant peer. In case of the originator is vanish from the network, results will be forward to the redundant peer. That peer will store the results. When originator peer is available at the network, the state can be recovered by the redundant peer. Hens the results will be delivered to the originator.

3) Fault Prevention

Tolerance for faults of the network is based on the reliability factor of the network nodes. If nodes are highly reliable, there is less chance to occur a fault in the system. Nodes that are selected as primary nodes of the network should be highly reliable; so that decrease occurrence of a faulty situation. There should be a proper mechanism for selection of primary nodes in the peer-to-peer network. It has been identified the reliability of the node is based on these factors,

- Device Rating
- Total bytes received
- Total bytes sent
- Total workloads received
- Total workloads completed
- Last seen time of the node

Device rating factor is used to measure the processing capability of a node in the network. A Device is a unit of a computer that capable of processing data. There can be more than one devices exist in a given node, e.g., CPU, Internal GPU, Discrete GPU. However, only one device can be used as the worker because processing in multiple devices can lead to performance degradation, synchronizing problems between processes and also programs will be overly complex and hard to implement. So most desirable device should be used as the worker. According to the device rating formula, devices with the highest score perform better. Device rating is calculated and stored in the node. The rating can be accessed through the routing table of peers.

$$Device\ Rating = \frac{(Maximum\ Core\ Count * \ Maximum\ Core\ Speed * \ Global\ Memory)}{3}$$

Other factors, **Total workloads received**, **Total workloads completed**, **Total bytes sent**, **Total bytes received**, **Last seen time** of the node calculated and stored in the node. Those statistics will be consumed in Naïve Bayes, based Machine Learning algorithm and reliability of the node is predicted as a probability. Highest reliable nodes are appointed as primary nodes, and it leads to achieving fault prevention of the network.

4 Results and Discussion

For the testing purposes, different types of simulations were introduced. Different simulations are undertaken to test the internal modules of the overall system. Tested modules compared to existing similar applications for efficiency and improvements identified and implemented.

A. Scheduling Testing

The simulation results of the proposed algorithm are compared with some of the existing scheduling algorithms described here. Existing some of the scheduling algorithms are MET (Minimum Execution Time), MCT (Minimum Completion Time), Min-Min algorithm, Min-Mean Algorithm, Max-Min Algorithm, Min-Var Algorithm, and Sufferage Algorithm. The proposed algorithm differs from these existing algorithms in such a way that these algorithms do not deal with load balancing, but the proposed algorithm takes load balancing as an important factor along with those parameters considered in the existing algorithms. Average Makespan for 5000 simulations for the same set of tasks in the same network. Makespan comparison graphs are given below in Figure 2 and 3.

- Average makespan for MET heuristic for 5000 simulations is = 56435712
- Average makespan for MCT heuristic for 5000 simulations is = 33682145
- Average makespan for MaxMin heuristic for 5000 simulations is = 56714123
- Average makespan for Sufferage heuristic for 5000 simulations is = 04215948
- Average makespan for MinMin heuristic for 5000 simulations is = 03946808
- Average makespan for MinVar heuristic for 5000 simulations is = 03970885
- **Average makespan for Hybrid heuristic for 5000 simulations are = 03932970**

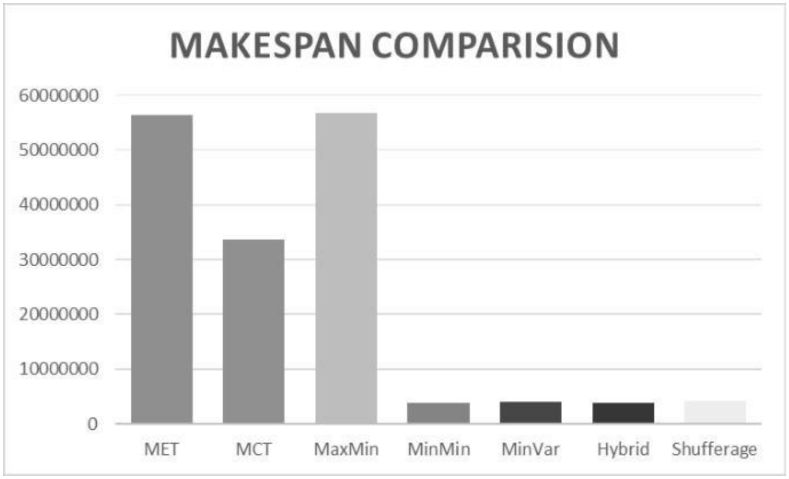

Fig. 2. Makespan comparison graph I

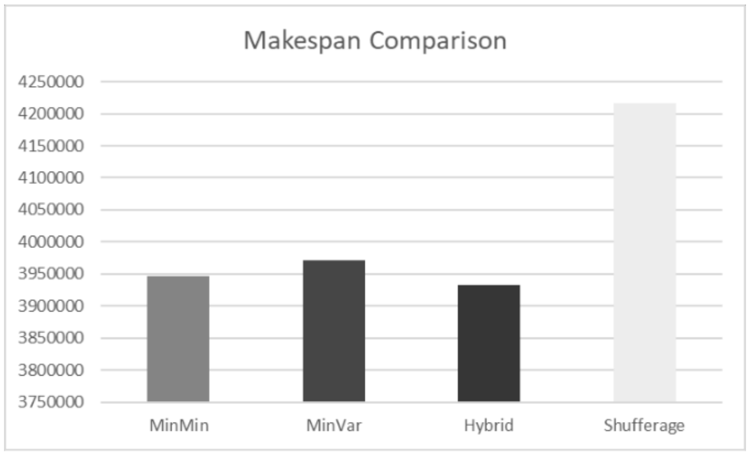

Fig. 3. Makespan comparison graph II

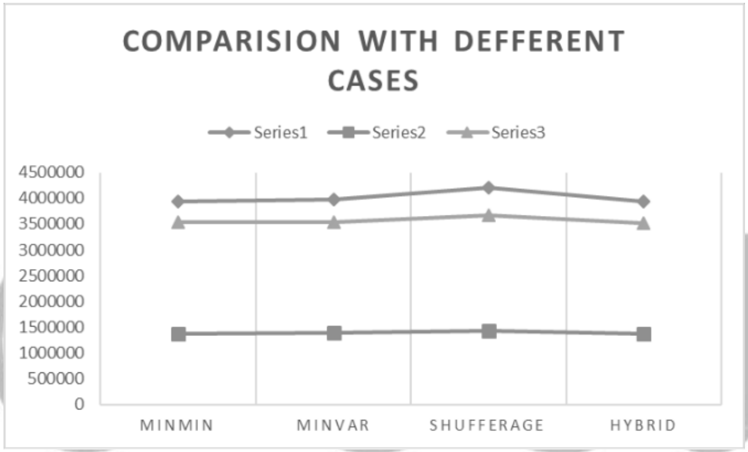

Fig. 4. Makespan comparison graph III for different scenarios

B. *Peer-to-Peer Network Testing*

For the network connectivity testing at first, the network is built with nine peers in the sense of virtual nodes and physical nodes. The originator node sends a text message to the connected peers in the network. Peers on the network can able receive the text message as well as the originator able to receive the acknowledge notification from all the peers. Secondly, this time the network is built with five physical nodes and the task was to receive a large sized file send by the originator. After sending the file all the peers able to receive the large sized file in short timing and sends back the notification to the originator successfully.

C. *Fault Tolerance Testing*

The first simulation process, an active node which is already executing a given task was deliberately crashed. In such scenario, in a little while the workload on the backup node start execution. Another simulation process, multiple nodes which are executing task were deliberately crashed. All the tasks are executed by the backup nodes without any errors as well as the overall total processing time after crash was acceptable. Backup node execution occurs when a node which was currently executing task fails, the same work will be given to another node for execution after a specific timeout.

D. *Overall System Testing*

The overall system testing includes testing the functionality of all the internal modules of the system. The considered scenario was a 2D array multiplication of size of 33x13200 with five nodes in the network. The array was given in CSV file format to the workload allocator. Allocator lookups for the defined rules in the XML file and also the connected peers. The defined array was split into five independent parts and send to the scheduler. Scheduler wraps the array parts into task objects which were scheduled to the different peers. Peers who are scheduled with tasks are received the array parts, executed and sends the results back to the scheduler. It aggregates all the output parts, generated the actual output and send back to the originator. This overall process took about 10 seconds to complete. Though the process takes little time to complete if the number of peer increases it will take low time rather than resulted time.

5 Conclusion and Future Work

Distributed computing makes the existing systems very efficient and time-saving. It also reduces the cost of computing as it shares the available resources with the users. That makes use of limited resources in an optimum manner. Making the system in a peer to peer even enables a single user to connect to the system easily and harness the resources in the network. The implemented system is capable of supporting few number of peers in a small geographical area and expand it to the vast area consisting a huge number of peers. Therefore, the system is built in a manner which features robust, scalable and reliable resource sharing distributed system. Addressing fault tolerance even increases the robustness of the system. By using such systems cost for processing is making cheaper. As future works, the system can be developed in such a way that made more reliable and available. Memory buffers that provide fault tolerance can be increased.

Compliance with Ethical Standards

- Conflict of Interest: The authors declare that they have no conflict of interest.
- Ethical approval: This article does not contain any studies with human participants or animals performed by any of the authors.
- Informed consent: Informed consent was obtained from all individual participants included in the study.

References

1. G. E. Moore, "Lithography and the Future of Moore's Law", Optical/Laser Microlithography VIII: Proceedings of the SPIE, vol. 2440, pp. 2-17, 1995-Feb.-20.
2. Ajit Sul1, Vishal Razdan, Satyendra Tiwari, Rupali Pashte, "Survey On Distributed Computing Platform," in International Research Journal of Engineering and Technology, 2016.
3. D. P. Anderson. BOINC: a system for public-resource computing and storage. Fifth IEEE/ACM International Workshop on Grid Computing, 2004
4. distributed.net,'distributed.net', 1997. [Online]. Available: http://www.distributed.net/Main_Page. [Accessed: 14-MAR-2017].
5. Wikipedia, 'distributed.net', 2017. [Online]. Available: https://en.wikipedia.org/wiki/Distributed.net. [Accessed: 14-MAR-2017].
6. Heena Setia & Abhilasha Jain (2016) Literature Survey on Various Scheduling Approaches in Grid Computing Environment 1st IEEE International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES-2016)
7. N. Fujimoto and K. Hagihara, "A comparison among grid scheduling algorithms for independent coarse-grained tasks", Proceedings of the 2004, International Symposium on Applications and the Internet Workshops (SAINTW'04), vol. 26-30, (2004), pp. 674-680.
8. F. Xhafa and A. Abraham, "Computational models and heuristic methods for Grid scheduling problems", Future Gener, Comput. Syst., vol. 26, (2010), pp. 608-621.
9. L.Y. Tseng, Y. H. Chin and S. C. Wang, "The anatomy study of high performance task scheduling algorithm for Grid computing system", Journal of Computer Standards & Interfaces, DOI: 10.1016/j.csi.2008.09.017, vol. 31, no. 4, (2010), pp. 713-722.
10. T.Tangmankhong, P. Siripongwutikorn T. Achalakul, "Peer-to-Peer fault tolerene framework for a grid computing system," in International Joint Conference on Computer Science and Software Engineering (JCSSE), 2012.
11. Khronos, "OpenCL: The open standard for parallel programming of heterogeneous systems." [Online]. Available: http://www.khronos.org/opencl/ [Accessed: 15-JUL-2017]
12. Wikipedia, 'CUDA', 2017. [Online]. Available: https://en.wikipedia.org/wiki/CUDA. [Accessed: 15-JUL-2017].
13. T. MacWilliam and C. Cecka, "CrowdCL: Web-based volunteer computing with WebCL," 2013 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, 2013, pp. 1-6, doi: 10.1109/HPEC.2013.6670348.