WorkStealing Algorithm for Load Balancing in Grid Computing

Hadeer S.Hossam[1], Hala Abdel-Galil[2], Mohamed Belal[3]
Computer Science Department, Faculty of Computers and Artificial Intelligence, Helwan University, Egypt

***Abstract*—Grid computing is a computer network in which many resources and services are shared for performing a specific task. The term grid appeared in the mid-1990s and due to the computational capabilities, efficiency and scalability provided by the shared resources, it is used nowadays in many areas, including business, e-libraries, e-learning, military applications, medicine, physics, and genetics. In this paper, we propose WorkStealing-Grid Cost Dependency Matrix (WS-GCDM) which schedule DAG tasks according to their data transfer cost, dependency between tasks and load of the available resources. WS-GCDM algorithm is an enhanced version from GCDM algorithm. WS-GCDM algorithm balances load between all the available resources in grid system unlike GCDM which uses specific number of resources regardless how many resources are available. WS-GCDM introduces better makespan than GCDM algorithm and enhances system performance from 13% up to 17% when we experiment algorithms using DAG with dependent tasks.**

Keywords*—Grid computing; static scheduling; dynamic scheduling; load balancing; directed acyclic graph (DAG)

I. Introduction

Importance of grid computing comes from the need to access resources which are geographically distributed and cannot be moved or duplicated to the same location. Grid computing offer approaches to overcome these obstacles. By using a grid, distributed resources can be treated as if they are into single place [1,2]. Assigning tasks to processors /machines is an important issue as it improves the performance of the whole job so that our concern will be on scheduling resources in grid computing. Resources can be computers, storage space, instruments, software applications, and data, are all connected through the Internet and a middleware layer that provides basic services for security, monitoring, resource management, and so forth as shown in Fig. 1. This work concerned with the processing time efficiency. Resources available on grid are shared under policies that specify who is permitted to access resources, what are the resources that will be available for everyone, and under what conditions they will use these resources [3].

Nature of grid computing resources is dynamic; therefore achieving high performance is a challenge as new resource can be submitted to the grid or withdraw from the grid. There are number of factors, which can affect the grid application performance; load balancing is one of the most critical features of Grid infrastructure.

Scheduling tasks in grid computing is critical as it influences the execution of the whole application. The problem of mapping tasks in grid computing is to find proper assignment of tasks to the available processors in order to optimize system utilization and load balancing and to minimize execution time [4].

This paper is organized as follows. In Section 2, related works for the independent and dependent task scheduling algorithms are discussed. In Section 3, we clarified our problem statement. And, we describe our proposed algorithm and how the WS-GCDM balance task scheduling between the available resources. And, we discuss the experimental results under varying gridlets and number of resources between the GCDM and WS-GCDM algorithms. In Section 4, we provide our final conclusion and the detailed algorithm.

II. Related Work

Over the past few years, a lot of grid computing algorithms have been introduced. Such algorithms focused on arranging and allocating tasks to resources in a way that minimizes the execution time in order to enhance performance and data transfer cost between these resources. This section surveys previous work in scheduling tasks in grid computing.

After user submit an application, the scheduler divides this application into tasks. These tasks may be dependent on each other and need to be scheduled based on the precedence between tasks, or sometimes each task is stand-alone and can be scheduled without affecting other tasks. This categorized scheduling in grid computing according to task dependency.

As an example of independent task scheduling algorithms: Opportunistic Load Balancing (OLB) algorithm [5, 6], Minimum Execution Time (MET) algorithm [7], Minimum Completion Time (MCT) algorithm [8], Min-min algorithm [9], Max-min algorithm [10, 11, 12], Suffrage algorithm [13, 14].

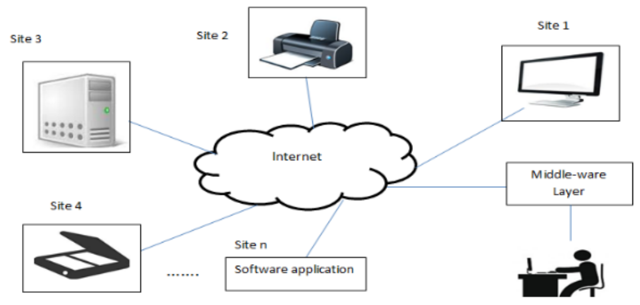

Fig. 1. Framework of Grid Computing.

Dependent task scheduling algorithms use the Directed Acyclic Graph (DAG) to represent dependency between tasks. Algorithms schedule dependent tasks such as Fully Decentralized P2P Grid scheduling (FDPGS) algorithm proposed by Piyush Chauhan and Nitin [15]. Sorted nodes in leveled DAG division (SNLDD) algorithm proposed in [16]. Grid costs and dependence matrix (GCDM) algorithm was proposed in [17]. CCF (Cluster ready Children First) algorithm proposed by Florin Pop and Valentin Cristea [18]. Communication inclusion generational scheduling (CIGS) algorithm proposed by Communication Inclusion Generational Scheduling (CIGS) algorithm [19], Dynamic Critical Path for Grid (DCP-G) and rescheduling DCP-G (Re-DCP-G) [20], and Grid Workflow Scheduling (GWS) algorithm [21].

This work is concerned with the task scheduling problem. We present application subtasks as a Directed Acyclic Graph (DAG) then these subtasks are mapped to the available processors. Moreover, we are trying to balance load across the available resources by using workstealing algorithm. WorkStealing algorithm (WS-GCDM) redistributes the initial scheduling work over idle processors, and as long as all processors have tasks to execute, no scheduling overhead occurs. This is the advantage of using receiver-initiated scheduling rather than sender-initiated scheduling as every resource maintains it as work queue and idle processors try to steel work from other resources.

III. Proposed Algorithm

Architecture of proposed WS-GCDM algorithm has two main components: Resource Discovery and Workflow Task Scheduling.

Resource Discovery component is responsible for discovering grid resources. It continuously collects information about which resources are available to the system. After finishing this task, it sends the collected information to the Grid Information Service (GIS) about the registered resources. The information includes number of processing elements (PEs), million instructions per seconds (MIPS), architecture and their operation systems.

Workflow Task scheduling component is responsible for scheduling workflow tasks. It receives Directed Acyclic Graph (DAG) from the user and based on the collected information about the registered resources from GIS it instantiates static scheduling phase. Ready tasks are then dispatched to the mapped resources based on their priority on the workflow. When resource becomes idle, it tries to steal tasks from the busy resources and the initial static scheduling change.

A. *Overview of WS-GCDM **A**lgorithm*

A WS-GCDM tries to minimize the execution time of the whole workflow application (makespan). The proposed WS-GCDM algorithm consists of three phases: Resource Discovery, Static Scheduling, and Rescheduling by WorkStealing phase.

Resource discovery phase is responsible for collecting information about the available resources. Then static scheduling phase tries to map DAG tasks to the appropriate resources based on the collected information stored in the GIS with the aim to reduce data cost transfer, which is the cost of transferring data from task to another task, and execution time of the whole application. The schedule is then submitted to Execution Manager (EM), which is responsible for getting ready tasks and executing these tasks to the scheduled resources. Final phase is the rescheduling by WorkStealing. This phase is responsible for monitoring the status of the available resources. When resource becomes idle it tries to find resource with tasks on its queue. Once it succeeded to finds this resource it tries to steal tasks from that resource which become a victim. We call idle resource which stole work from other resources stealer. Stealer resources try to steal tasks which can be finished as earlier as possible in order to search for other tasks to execute it.

B. *Static Task Scheduling*

Static task scheduling considers dependencies and data cost transfer between tasks in the workflow. Consider we have N tasks in the DAG workflow. Dependency (D) and data cost transfer (C) matrices (N*N) then created.

In this stage number of resources that will be used initially will be as the number input nodes in the DAG. Steps of static task scheduling are:

- Determine input nodes.
- Mapping input nodes.
- Mapping remaining nodes in the DAG.

1) Determine input nodes: In this step input nodes are determined considering D matrix. Input nodes are the nodes which did not depend on any other nodes. Also, sometimes, they are called entry nodes.

2) Mapping input nodes: After determining input nodes, number of resources that will be used initially is determined. In this step each input node is mapped to separate resource. If number of registered resources is less than number of input nodes, then we start mapping the outnumber tasks with the previously mapped tasks. Each processor now has its own set containing nodes that will be executed by this it.

3) Mapping remaining nodes in the DAG: In the step all the remaining tasks in the workflow DAG is mapped to the used resources in the previous step. DAG is divided into levels and nodes will be scheduled level by level in order to perceive priorities between nodes. Let us consider we have m levels in the DAG, this mean that node in level i (0<=i<m) have higher priority than nodes in level i+1. Considering C matrix and sets of each processor nodes in the next level can be determined, and then these nodes can be scheduled considering their parent nodes. There are two cases for each node while scheduling it:

- Node has only one parent. This node is not shared and scheduled to processor which has its parent on its set.
- Node has more than one parent. This node is shared between other parent nodes and will be scheduled according to the following rule:

a) If parent nodes are mapped to the same resource, then this node will be added to this processor set.

b) If parent nodes are mapped to different resources, then task will be added to the processor that are more transferring cost to its nodes.

4) Rescheduling by work stealing phase: Workflow scheduling should be balanced to the available resources. GCDM fail to balance load across the resources, as this algorithm did not consider all the available resources and use only specified number of resources which is equal to the input nodes. Therefore, we can find resources idle while other resources may be overloaded. This may influence the execution of the whole application makespan; which is the time when all jobs are completed.

WS-GCDM balance load across the available resources by using WorkStealing algorithm. Tasks are submitted to the mapped resources initially by using sender-initiated policy, which means scheduler is responsible for sending tasks to the resources. After this stage each resource will have its own queue containing ready tasks that will be executed by this resource. When execution of tasks begins on the mapped resources WorkStealing algorithm will be triggered. Idle resources, which are called stealer nodes, will try to find resources with tasks that can be stolen; these resources are called victim nodes. After succeeding to find this resource stealer node will steal Least Waiting Tasks (LWT) from victim and add this task to its own queue. LWT are tasks that have minimum waiting counter which represent number of tasks needed to be finished to start executing this task. We choose the LWT in order to make resource finish this task earlier than possible and try to steal another task from other victim nodes. Steal nodes become receiver initiated and by this utilization of idle resources are enhanced as it helps to make these resources as busy as possible.

5) WS-GCDM pseudo code: As in this heading, they should be Times New Roman 11-point boldface, initially capitalized, flush left.

This section describes the pseudo code of the WS-GCDM algorithm for scheduling workflow tasks which is shown in Algorithm 1.

Algorithm 1. Work Stealing Grid Cost Dependancy Matrix (WS-GCDM) algorithm

```
1.Input: Directed acyclic graph.
2.Construct dependency matrix
3.Construct data transmission cost matrix
4. Loop from i=0 to number of gridlets in dependency matrix
     If gridlet does not depend on any other node
        Add this gridlet to inputGridlets.
5. Number of resources= the number of gridlets in inputGridlets
6.Loop from i=0 to number of inputGridlets
   Schedule each gridlet to processor.
7.While there are gridlets in DAG not scheduled
     1.If gridlet is not shared between resources then:
        i. Assign task to the processor that the tasks dependent on before it in set.
     2.Else if gridlet is shared between resources then:
          If parent nodes are mapped to the same resource
          then
             Add task to the parents processor set.
          If parent nodes are mapped to different resources
          then
             Add task to the processor that are more t
                ransferring cost to its nodes.
8.Construct gridletResourceCharacteristics for each
  gridlet which contains
  Direct Parent list
  Waiting counter
  Direct children list
9.Construct gridletProcessor list for each processor.
10.Begin
    I. Loop from i=0 to number of gridlets on every
       gridletProcessor list
              If gridlet waiting counter=0
               Submit gridlet to its scheduled
        resource
               Add this gridlet to runningGridlets
   II. Loop from i=0 to runningGridlets
        Loop from i=0 to number of gridletProcessor
        While gridletProcessor.size==0
        Create random number from 0 to

number of processors

            Loop from i=0 to
            gridletProcessor[randomNumber].size()

        Get gridlet with minimum waiting counter

  III. Add this gridlet to gridletProcessor list of this
       resource
   IV. Get direct children of those runningGridlets
    V. Add direct children to waitingGridlets
   VI. Loop from gridlet=0 to number of
       waitingGridlets
       Send message to waitingGridlets to decrement
       their waiting counter.
       Get direct children of those waitingGridlets
       Add direct children to waitingGridlets
       Go to step vi
  VII. Go to step i
```

Initially the first phase of statically scheduling tasks to available resources.

Steps 1-3 take DAG and construct its data and cost matrix. Then Steps 4-6 determines input or entry nodes and assigns each input node to the specified number of resources. Step 7 schedules the remaining tasks in DAG to processors according to the previously described algorithm in section Static task scheduling. Step 8 adds parameters to gridlet beside its characteristics. Direct Parent list contains Parents which are directly connected to this gridlet. Waiting counter contains number of tasks needed to be executed to start executing this gridlet and is calculated by incrementing waiting counter of its direct parents waiting by one. Direct children list contains children of this gridlet which are directly connected with it. Step 9 construct waiting queue for each processor which contains tasks that will be executed by this processor.

Step 10 send gridlets with waiting counter =0 which means this gridlet become ready to be executed to running queue. After sending task to the running queue of the mapped processor, message from this task need to be sent to each child in the direct children list. This message is to inform the child that parent who sends message start executing and to be ready for executing. This means that this child needs to decrease its waiting counter by one. Also, this child needs to send message to each child in its direct children list to decrease its waiting counter and continue sending message from child to child until we reach the exit node. While there are tasks in the running queue idle resources search for victim nodes and try to steal work from it.

6) Implementation: In our work, we use GridSim toolkit simulator. The GridSim toolkit provides modeling and simulation of entities in distributed computing systems, users, resources, and resource brokers (schedulers) for design and evaluation of scheduling algorithms. It was originally conceived by Buyya [22].

There are five different entities used in the system:

- User.
- Gridlets.
- Resources.
- Grid Information Service (GIS).
- Grid Broker.

The makespan, resource busy time percentage, and system improvement are the used measure to compare the performance of GCDM and WS-GCDM algorithms.

Makespan is the total time elapsed between the start time of executing the first task in the DAG workflow to the completion time of last task.

Resource busy time percentage which is the percentage of time the resource is busy in executing tasks in its ready queue.

System improvement rate that specifies the performance improvement rate of WS-GCDM algorithm with respect to GCDM algorithm can be measured as the difference between GCDM makespan and WS-GCDM makespan over the makespan of WS-GCDM algorithm.

Table I shows the characteristics of all resources that will be used in our simulation environment. We have used a subset of resources of the World-Wide Grid (WWG) testbed, as used in [23].

TABLE I. THE WWG TESTBED RESOURCES

Resource ID	Resource Name	MIPS Rating	No of PEs	Operating System Architecture
5	Resource_0	515	4	OSF1 Compaq AlphaServer
9	Resource_1	377	4	Solaris Sun Ultra
13	Resource_2	377	4	Solaris Sun Ultra
17	Resource_3	377	2	Solaris Sun Ultra
21	Resource_4	380	2	Linux Intel Pentium/VC820
25	Resource_5	410	6	IRIX SGI Origin 3200
29	Resource_6	410	16	IRIX SGI Origin 3200
33	Resource_7	410	16	IRIX SGI Origin 3200
37	Resource_8	380	2	Linux Intel Pentium/VC820
41	Resource_9	410	4	IRIX SGI Origin 3200

The following subsections show six different experiments for evaluating the performance of WS-GCDM algorithm. DAG in each experiment is generated randomly. The gridlets in the generated DAG varying in their length and input file size. In addition, the first five experiments are done using only the first five resources from Table I and different number of gridlet (twenty gridlets, forty gridlets, sixty gridlets, eighty gridlets and one hundred gridlets).

Experiment six shows the performance of GCDM and WS-GCDM algorithms under different number of resources.

IV. EXPERIMENTS

A. Experiment One

We implement GCDM and WS-GCDM algorithms on random DAG with 20 tasks and two input tasks.

Fig. 2 and Fig. 3 indicate the busy time percentage of the available five resources for GCDM and WS-GCDM algorithms, respectively.

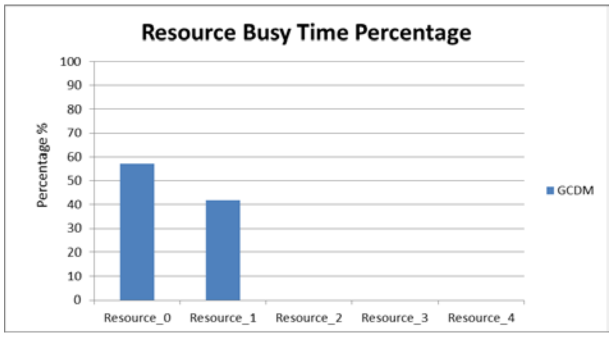

Fig. 2. Resource Busy Time Percentage for GCDM Algorithm on Random DAG with 20 Tasks.

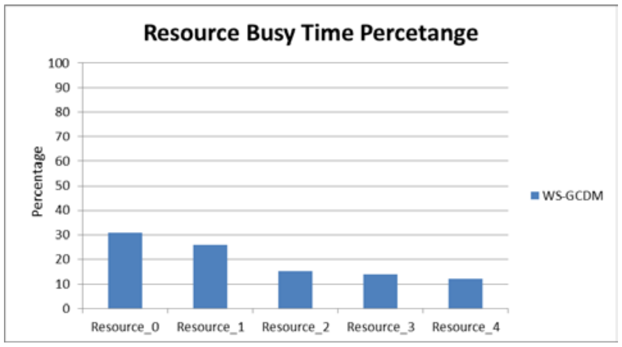

Fig. 3. Resource Busy Time Percentage for WS-GCDM Algorithm on Random DAG with 20 Tasks.

Makespan of GCDM=10563.763 sec, while makespan of WS-GCDM=9128.416 sec.

From the makespan results we can conclude that WS-GCDM improve performance rate with nearly 15%.

B. *Experiment Two*

We experiment GCDM and WS-GCDM algorithms on random DAG with 40 tasks and three input tasks.

Fig. 4 and Fig. 5 indicate the busy time percentage of the available five resources for GCDM and WS-GCDM algorithms, respectively.

Makespan of GCDM= 46630.65 sec, while makespan of WS-GCDM= 39797.880 sec.

From the makespan results we can conclude that WS-GCDM improve performance rate with nearly 17%.

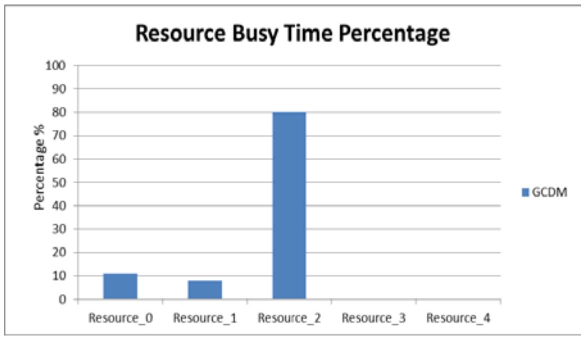

Fig. 4. Resource Busy Time Percentage for GCDM Algorithm on Random DAG with 40 Tasks.

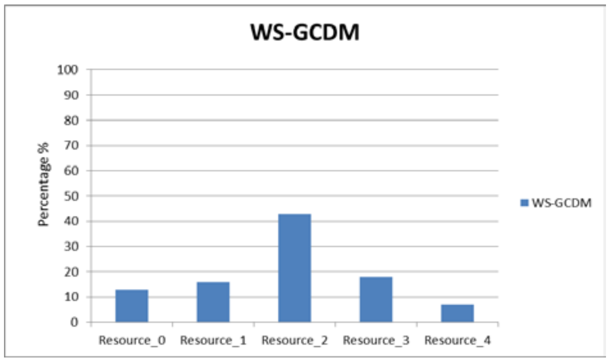

Fig. 5. Resource Busy Time Percentage for WS-GCDM Algorithm on Random DAG with 40 Tasks.

C. *Experiment Three*

We experiment GCDM and WS-GCDM algorithms on random DAG with 60 tasks and three input tasks.

Fig. 6 and Fig. 7 indicate the busy time percentage of the available five resources for GCDM and WS-GCDM algorithms, respectively.

Makespan of GCDM= 64844.101 sec, while makespan of WS-GCDM= 57213.422 sec.

From the makespan results we can conclude that WS-GCDM improve performance rate with nearly 13%.

D. *Experiment Four*

We experiment GCDM and WS-GCDM algorithms on random DAG with 80 tasks and four input tasks.

Fig. 8 and Fig. 9 indicate the busy time percentage of the available five resources for GCDM and WS-GCDM algorithms, respectively.

Makespan of GCDM= 75219.907 sec, while makespan of WS-GCDM= 65978.606 sec.

From the makespan results we can conclude that WS-GCDM improve performance rate with nearly 14%.

E. *Experiment Five*

We experiment GCDM and WS-GCDM algorithms on random DAG with 100 tasks and two input tasks.

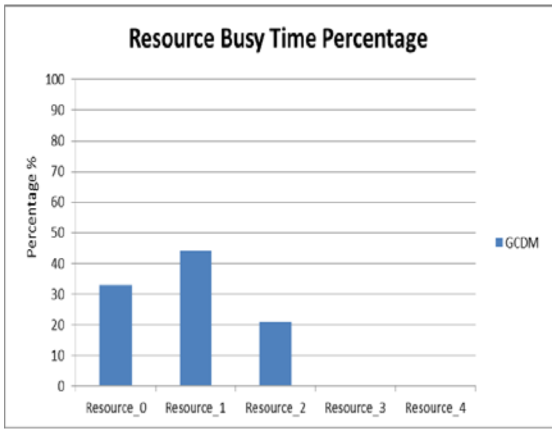

Fig. 6. Resource Busy Time Percentage for GCDM Algorithm on Random DAG with 60 Tasks.

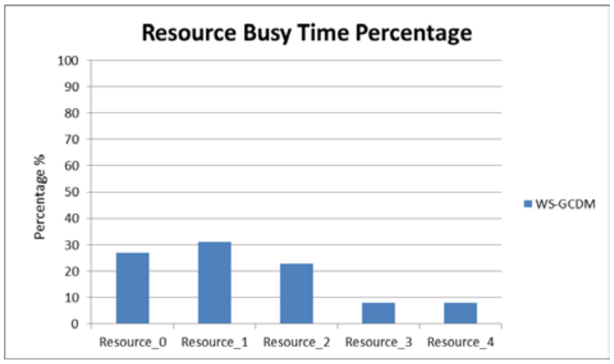

Fig. 7. Resource Busy Time Percentage for WS-GCDM Algorithm on Random DAG with 60 Tasks.

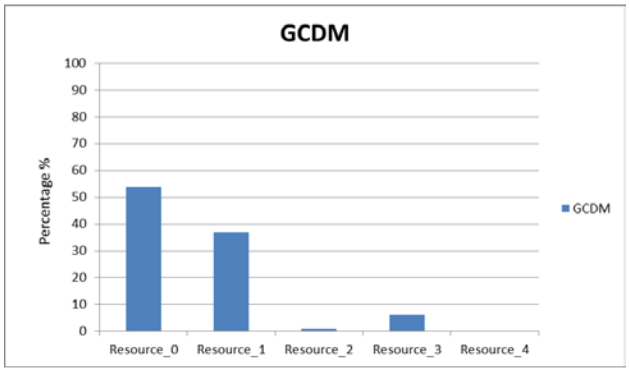

Fig. 8. Resource Busy Time Percentage for GCDM Algorithm on Random DAG with 80 Tasks.

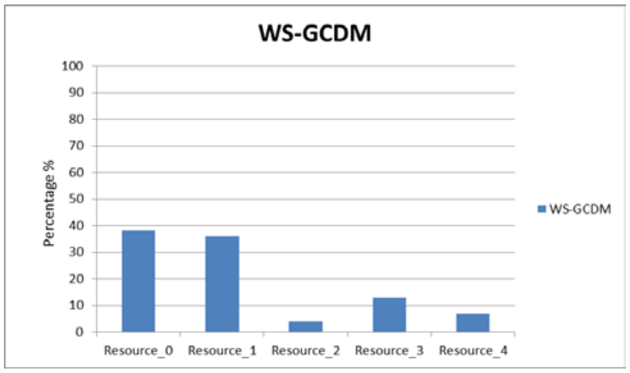

Fig. 9. Resource Busy Time Percentage for WS-GCDM Algorithm on Random DAG with 80 Tasks.

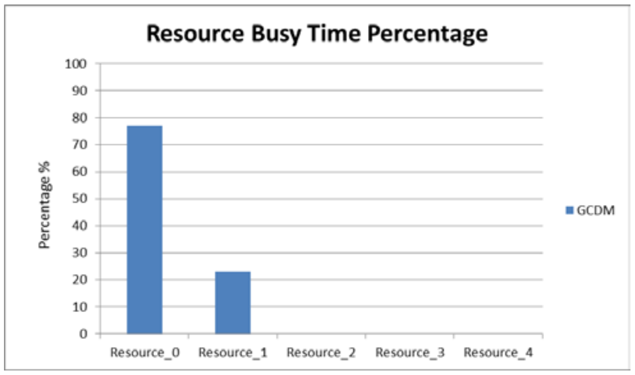

Fig. 10. Resource Busy Time Percentage for GCDM Algorithm on Random DAG with 100 Tasks.

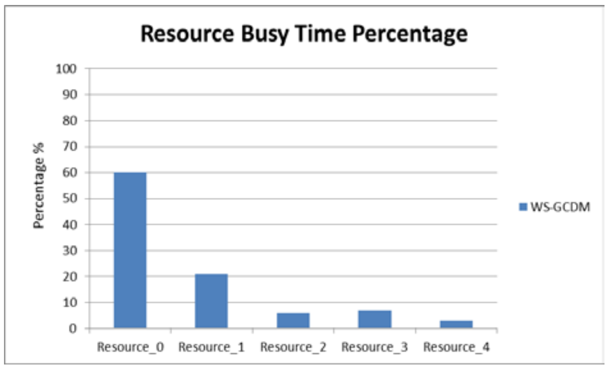

Fig. 11. Resource Busy Time Percentage for WS-GCDM Algorithm on Random DAG with 100 Tasks.

Fig. 10 and Fig. 11 indicate the busy time percentage of the available five resources for GCDM and WS-GCDM algorithms, respectively.

Makespan of GCDM= 87280.950 sec, while makespan of WS-GCDM= 76115.103 sec.

From the makespan results we can conclude that WS-GCDM improve performance rate with nearly 14%.

F. *Experiment Six*

In this experiment we compare load balance between resources and makespan of GCDM and WS-GCDM algorithms when number of resources varies. We make this evaluation using DAG with 100 tasks.

From Fig. 12, we can conclude that makespan of GCDM under different number of resources does not change a lot as it uses only limited number of resources considering number of input tasks in the DAG whereas makespan of WS-GCDM under different number of resources always are less than makespan of GCDM as all resources are busy and has gridlets to execute.

G. *Experiment Seven*

In this experiment we compare our proposed algorithm WS-GCDM with Adaptive Workflow Scheduling (AWS) algorithm [21]. We compare makespan of WS-GCDM and makespan of AWS under different number of tasks.

From Fig. 13, we can conclude that makespan of WS-GCDM is less than makespan of AWS.

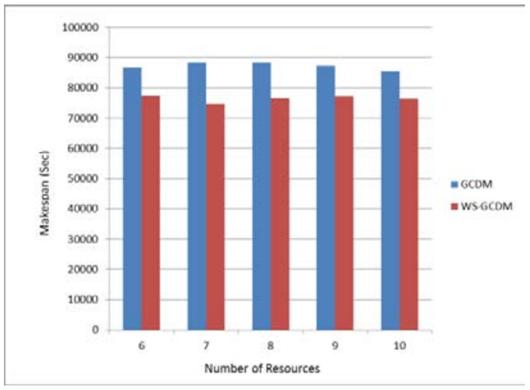

Fig. 12. Makespan of GCDM and WS-GCDM under different Number of Resources.

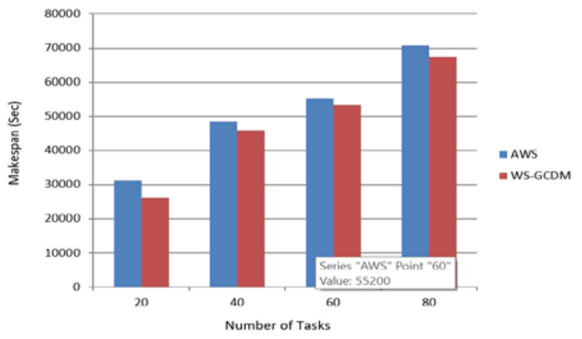

Fig. 13. Makespan of AWS and WS-GCDM under different Number of Resources.

V. Contributions

Load Balancing is one of most important features of Grid Middleware for efficient execution of intensive applications. The efficiency of load balancing of the algorithm decides the efficiency of grid middleware.

Our proposed algorithm WS-GCDM focus on balancing load among resources Work-Stealing Grid Cost Dependency Matrix (WS-GCDM) algorithm which is enhanced version from Grid Cost Dependency Matrix (GCDM) algorithm.

WS-GCDM and GCDM are implemented and compared using the same data. It is found that WS-GCDM improves performance of the GCDM and the makespan in case of using WS-GCDM is better than GCDM.

Also, we compared, this makespan of the proposed algorithm WS-GCDM with Adaptive Workflow Scheduling (AWS) algorithm and from the experiments results WS-GCDM introduces better load balancing and enhances the makespan than GCDM and AWS algorithms as all the available resources are utilized. It enhances system performance by nearly 15%, 17%, 13%, 14% and 14% when we experiment algorithms using DAG with 20,40,60,80 and 100 dependent tasks, respectively.

References

[1] Dr. K. S. Kanna, Dr. P. Devabalan, S.Hariharasitaraman and P. Deepa, Some Insights on Grid Computing-A Study Perspective, International Journal of Pure and Applied Mathematics Volume 118 No. 8 2018, 47-50.

[2] 7 things you should know about grid computing, Educause65 learning initiative, January 2006.

[3] Fangpeng Dong and Selim G. Akl, Scheduling Algorithms for Grid Computing: State of the Art and Open Problems, School of Computing, Queen’s University Kingston, Ontario January 2006.

[4] I.I. Kurochkin and E.A. Gerk, Modeling of task scheduling in desktop grid systems at the initial stage of development, Proceedings of the VIII International Conference "Distributed Computing and Grid-technologies in Science and Education, 2018.

[5] George Amalarethinam. D.I, Vaaheedha Kfatheen .S, “Max-min Average Algorithm for Scheduling Tasks in Grid Computing Systems”, (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 3 (2) , 2012.

[6] Naglaa M. Redaa, A. Tawfikb, Mohamed A. Marzokb, Soheir M. Khamis, “Sort-Mid tasks scheduling algorithm in grid computing1”, Journal of Advanced Research Volume 6, Issue 6, November 2015.

[7] P. K. Suri, Sunita Rani, “GRID Distance and Execution Time based Scheduling Algorithm”, International Journal of Computer Engineering and Technology (IJCET), ISSN 0976-6367(Print), ISSN 0976 - 6375(Online), Volume 5, Issue 7, July (2014).

[8] Adil Yousif, Sulaiman Mohd Nor, Abdul Hanan Abdualla, and Mohammed Bakri Bashir, “Job Scheduling Algorithms on Grid Computing: State-of- the Art”, International Journal of Grid Distribution Computing Vol. 8, No.6, (2015).

[9] T. Kokilavani, Dr. D.I. George Amalarethinam, “Load Balanced Min-Min Algorithm for Static Meta-Task Scheduling in Grid Computing”, International Journal of Computer Applications (0975 – 8887) Volume 20– No.2, April 2011.

[10] Navdeep Kaur, Khushdeep Kaur, “Improved Max-Min Scheduling Algorithm”, IOSR Journal of Computer Engineering (IOSR-JCE) e-ISSN: 2278-0661, p-ISSN: 2278-8727, Volume 17, Issue 3, Ver. 1 (May – Jun. 2015).

[11] Zhongping Zhang, Yupeng Feng, Shan Zhang and Ying Sun, “Heuristic Grid Resource Scheduling Algorithm based on Group of Task and Secondary Distribution”, International Journal of Security and Its Applications Vol.9, No.8 (2015).

[12] Salman Meraji and M. Reza Salehnamadi,” A Batch Mode Scheduling Algorithm for Grid Computing”, Journal of Basic and Applied Scientific Research, J. Basic. Appl. Sci. Res., 3(4)173-181, 2013.

[13] Naglaa M. Reda, “An Improved Sufferage Meta-Task Scheduling Algorithm in Grid Computing Systems International Journal of Advanced Research (2015), Volume 3, Issue 10, 123 -129.

[14] Nabeel Zanoon, Nashat Al Bdour and Evon Abu-Taieh, “Survey of Algorithm: Scheduling Systems and Distributed Resource Management in Grid”, International Journal of Computer Applications (0975 – 8887) Volume 98– No.1, July 2014.

[15] Piyush Chauhan and Nitin, “Decentralized Scheduling Algorithm for DAG Based Tasks on P2P Grid”, Hindawi Publishing Corporation Journal of Engineering Volume 2014, Article ID 202843,14 pages, January 2014.

[16] Nirmeen A. Bahnasawy, Magdy A. Koutb, Mervat Mosa and Fatma Omara, “A new algorithm for static task scheduling for heterogeneous distributed computing systems” African Journal of Mathematics and Computer Science Research Vol. 4(6), pp. 221- 234, June 2011.

[17] Amir M Bidgoli and Zahra Masoudi Nezad, “A new scheduling algorithm design for grid computing tasks”, 5th SASTech 2011, Khavaran Higher- education Institute, Mashhad, Iran.

[18] Florin Pop andValentin Cristea, “Intelligent Strategies for Dag Scheduling Optimization in GRID Environments”, 16th International Conference on Control Systems and Computer Science (CSCS16’07).

[19] Ian Foster, Carl Kesselman and Steven Tuecke, “The Anatomy of the Grid Enabling Scalable Virtual Organizations”, To appear: Intl J. Supercomputer Applications, 2001.

[20] Mustafizur Rahman , Rafiul Hassan , Rajiv Ranjan , and Rajkumar Buyya , “Adaptive workflow scheduling for dynamic grid and cloud computing environment”, Published online 4 March 2013 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3003.

[21] Ritu Garg and Awadhesh Kumar Singh, “Adaptive workflow scheduling in grid computing based on dynamic resource availability”, Engineering Science and Technology, an International Journal (2015).

[22] Jia, Rajkumar Buyaa, “Workflow Scheduling Algorithms for Grid Computing”, Technical Report, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, May 31, 2007.

[23] Mustafizur Rahman , Rafiul Hassan , Rajiv Ranjan , and Rajkumar Buyya , “Adaptive workflow scheduling for dynamic grid and cloud computing environment”, Published online 4 March 2013 in Wiley Online Library (wileyonlinelibrary.com). DOI:10.1002/cpe.3003.