Computing and Informatics, Vol. 37, 2018, 311–326, doi: 10.4149/cai_2018_2_311

LOAD BALANCING SCHEDULING ALGORITHM FOR CONCURRENT WORKFLOW

Sunita Singhal

Manipal University Jaipur, School of Computing & Information Technology
Jaipur, Rajasthan, 303007, India
e-mail: sunita.singhal@jaipur.manipal.edu

Jemin Patel

Birla Institute of Technology & Science, Pilani
&
Department of Computer Science & Information Systems
Pilani Campus, Pilani, Rajasthan, 333031, India

Abstract. Concurrent workflow scheduling algorithm works in three phases, namely rank computation, tasks selection, and resource selection. In this paper, we introduce a new ranking algorithm that computes the rank of a task, based on its successor rank and its predecessors average communication time, instead of its successors rank. The advantage of this ranking algorithm is that two dependent tasks are assigned to the same machine and as a result the scheduled length is reduced. The task selection phase selects a ready task from each workflow and creates a task pool. The resource selection phase initially assigns tasks using min-min heuristic, after initial assignment, tasks are moved from the highly loaded machines to the lightly loaded machines. Our resource selection algorithm increases the load balance among the resources due to tasks assignment heuristic and reassignment of tasks from the highly loaded machines. The simulation results show that our proposed scheduling algorithm performs better over existing approaches in terms of load balance, makespan and turnaround time.

Keywords: Rank computation, multiple workflow, scheduling algorithm, task allocation, concurrent workflow

1 INTRODUCTION

Heterogeneous computing systems consist of a heterogeneous collection of machines, connected over a high-speed network, communication protocols and programming environments which provide a variety of architectural capabilities that have a diverse execution requirements. Cluster computing, grid computing, and cloud computing [1] are examples of heterogeneous computing systems. One of the key challenges of heterogeneous computing systems is the scheduling problem. In heterogeneous computing system a user submits an application to the system, an application profiling tool [2] is used to extract the properties of an application. The application properties include a number of jobs forming the workflow, the size of each job in terms of the number of instructions and the number of bytes required to be exchanged between the two jobs in case of parent and child relationship between jobs, relationship (parent and child) among the tasks in a workflow. Analogical Benchmarking [3, 4] provides a measure of how well a resource can perform a given type of application. On the basis of available information and quality of service requirement, the scheduler schedules tasks to the available machines in the system.

A workflow scheduling problem is an NP-complete problem and it is solved using static scheduling algorithms, dynamic scheduling algorithms, single workflow and multiple workflow scheduling algorithms. Static scheduling algorithms [5] assume that all the information about tasks, and resources are known and remain constant. Resources performance are varying in dynamic scheduling algorithm [14]. Single workflow scheduling algorithms [13] schedule a single workflow at a time while multiple workflow scheduling algorithms [15] schedule more than one workflow at a time.

The multiple workflow scheduling algorithm is also known as concurrent workflow scheduling algorithm. The concurrent workflow scheduling algorithm works in three phases known as rank computation, task selection and resource selection. We have worked on rank computation and resource selection phases. Our rank computation algorithm computes the rank of a task, based on its successor rank and its predecessors average communication cost. Our resource selection algorithm assigns tasks to resources using Min-min [5] heuristic. After that, tasks are reassigned from the highly loaded machines to the lightly loaded machines.

The rest of the paper is organized as follows. Section 2 presents the related work. Section 3 presents the workflow model. Section 4 describes our proposed algorithm named Load balancing scheduling algorithm for concurrent workflow. The results are presented and discussed in Section 5. Finally, in Section 6 we conclude the work and suggest future research directions.

2 RELATED WORK

A multiple workflow scheduling algorithm is categorized as offline and online scheduling algorithms [7]. The offline workflow scheduling algorithms assume that all the

workflows are available before the scheduling starts. The online workflow scheduling algorithms assume that the workflow arrival time is not known in advance.

The two most popular ranking algorithms are Heterogeneous Earliest Finish Time (HEFT) [6] and Critical Path on a Processor (CPOP) [6] ranking algorithms. The HEFT ranking algorithm computes the rank of each task, on the basis of average communication and computation cost between the current task and its successor. The CPOP ranking algorithm uses the summation of the downward and upward rank of a task. CPOP ranking algorithm does not perform better than HEFT [6] ranking algorithm. The HEFT ranking algorithm does not consider communication time of its immediate predecessor. As a result, two dependent tasks might be assigned to different resources and increase the scheduled length.

Zhao and Sakellariou [7] presented offline multiple workflow scheduling algorithm for composition and fairness of workflow. Their composition scheduling algorithm composes many workflows into a single workflow. The tasks are selected from each workflow in a round robin fashion and assigned to the resource that completes earliest. Their fairness scheduling algorithm computes the task's rank based on the slowness of the workflow. Our work differs from their approaches. We worked on concurrent workflow where the arrival time of workflow was not known.

Various online workflow scheduling algorithms are designed by researchers and are known as the Rank Hybrid [9] scheduling algorithm, the Online Workflow Management (OWM) [15] scheduling algorithm and the Fair Share [8] scheduling algorithm.

The Rank hybrid scheduling algorithm first computes the rank of each task using HEFT ranking algorithm. After ranking, all the ready tasks are selected from the workflow and assigned to the machines in increasing order of rank. Therefore, it gives priority to the smaller workflow. While the OWM scheduling algorithm selects a single task from each workflow, instead of selecting all the ready tasks from each workflow, the OWM scheduling algorithm assigns tasks to the machine that takes minimum time instead of the machine that completes first. It means if the minimum time machine is not free, it keeps the tasks in a waiting queue. Therefore, the lowest priority task may be executed first. The Fair share scheduling algorithm also selects a single task from each workflow, after that the rank of each task is again computed based on the percentage of the remaining task of a workflow. The highest rank task is assigned first. Thus, it does not differentiate between longer and smaller workflow.

All these scheduling algorithms compute the rank using HEFT ranking algorithm and assign tasks to the machine using Minimum Completion Time (MCT) [5] scheduling algorithm, while our ranking algorithm considers its successor rank and its predecessors average communication time. After ranking the tasks of a workflow, a tasks pool is created, then these tasks are initially assigned using Min-min heuristic. After initial assignment of tasks, they are reassigned from the highly loaded machines to the lightly loaded machines. Our reassignment algorithm is also different from the Balance Minimum Completion Time (BMCT) [12] resource selection algorithm. The BMCT resource selection algorithm moves tasks from the highest completion time machine to other machines in the system. Thus, it may transfer

tasks to the machine that is not lightly loaded. As a result, the load may not be balanced.

3 WORKFLOW MODEL

A typical scientific application is represented using Directed Acyclic Graph (DAG) to model an application. A sample DAG of 10 tasks is shown in Figure 1 a). A workflow is represented by G = (v, e, p, w), where v and e are the set of tasks and directed edges, respectively. A node in the task graph represents a task that runs non-preemptively on any cluster. Each edge is denoted by e_{ij} corresponding to the data communication between t_i and t_j, where t_i is called the immediate parent task of t_j. A child task cannot be started until all of its parent tasks are completed. A task which does not have a parent task is called an entry task t_{entry}. A task that does not have a child task is called an exit task t_{exit}. w is a $v \times p$ computation matrix, where v is the number of tasks and p is the number of processors in the system. w_{ij} is the estimated execution time of task t_i on processor p_j. t_{pred} and t_{succ} are the sets of immediate predecessors and successors of task t_i, respectively. The average computation time of task t_i is calculated as follows:

$$w_i = \frac{\sum_{j \in P} w_{ij}}{p}. \tag{1}$$

Here w_i is the average computation time of t_i, p is the number of processor on which task can be executed, and P is the set of processors on which task can be executed. w_{ij} is the expected execution time of t_i on processor p_j. The average communication cost between edge i and edge j is defined as follows:

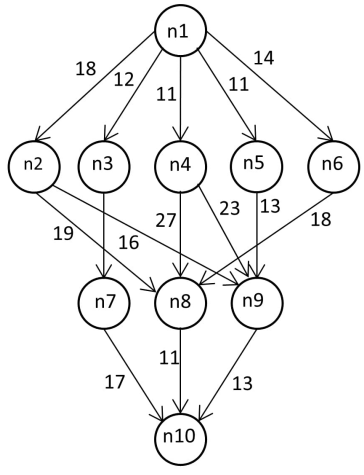
$$c_{ij} = \frac{L + data_{ij}}{B}. \tag{2}$$

Here c_{ij} is the average communication cost between task i and j, L is the average communication start-up cost of all processors and B is the average bandwidth of all links connecting to P. $data_{ij}$ is the amount of data that task t_i transfers to task t_j, when tasks are assigned to different processors.

The resources are heterogeneous computers, connected to each other.

4 LOAD BALANCING SCHEDULING ALGORITHM FOR CONCURRENT WORKFLOW

This algorithm works in three phases as shown in Figure 2. The first phase computes the rank of the tasks. The second phase selects a single ready task from each workflow in order to provide fairness among available workflow. The last phase first assigns the tasks using Min-min heuristic then computes the average load of the system. Based on the average load of the system, tasks are transferred from the



(a) DAG

Processor / Task	P1	P2	P3
n1	14	16	9
n2	13	19	18
n3	11	13	19
n4	13	8	17
n5	12	13	10
n6	13	16	9
n7	7	15	11
n8	5	11	14
n9	18	12	20
n10	21	7	16

(b) Computation Matrix

Figure 1. DAG and computation matrix

highly loaded machines to the lightly loaded machines in order to reduce the load imbalance.

4.1 Rank Computation Phase

Our rank computation algorithm is called Modified Heterogeneous Earliest Finish Time (MHEFT) ranking algorithm. The MHEFT ranking algorithm computes the rank from downwards. t_{exit} task rank is defined as follows:

$$\text{rank}(t_{exit}) = w_{exit} + \max_{t_k \in t_{pred}\{t_{exit}\}} c(exit, k). \tag{3}$$

Here t_{pred} is the set of predecessors of t_{exit}, $c(exit, k)$ is the average communication cost between task t_{exit} and t_k, and w_{exit} is the average computation cost of task t_{exit}. The rank of a task t_i is recursively defined and computed as follows:

$$\text{rank}(t_i) = w_i + \max_{t_j \in t_{succ}\{t_i\}} (\, c(i,j) + \text{rank}(t_j)) + \max_{t_k \in t_{pred}\{t_i\}} c(i,k). \tag{4}$$

Here t_{succ} is the set of immediate successors of t_i, w_i is the average computation cost of task t_i. $c(i,j)$ is the average communication cost between task t_i and t_j, t_j is the successor of t_i, and $c(i,k)$ is the average communication cost between task t_i and t_k. t_{pred} is the set of predecessors of t_i.

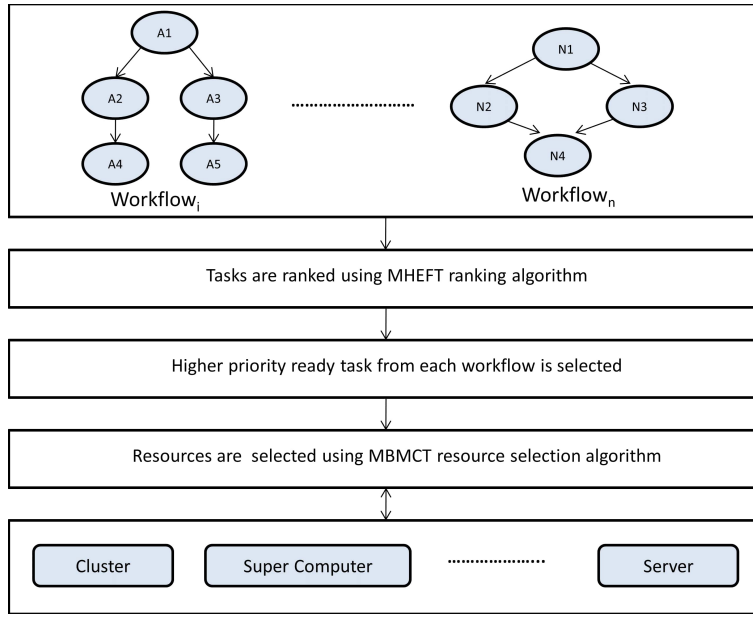


Figure 2. Pictorial diagram of load balancing concurrent workflow scheduling algorithm

4.1.1 Illustrative Example

Consider a sample DAG of 10 tasks and the computation matrix are given in Figures 1 a) and 1 b), respectively. An identical matrix is used by Topcuoglu et al. [6] to show the working of HEFT ranking algorithm. It has 10 tasks and 3 processors. Computed rank and Gantt chart of HEFT, CPOP and MHEFT ranking algorithms are shown in Table 1 and Figure 3, respectively.

Task	HEFT	CPOP	MHEFT
n1	108	108	108
n2	77	108	95
n3	79	105	91
n4	80	104	91
n5	69	93	80
n6	63	90	77
n7	42	105	65
n8	35	102	62
n9	44	108	67
n10	14	108	31

Table 1. Rank of scheduling algorithms

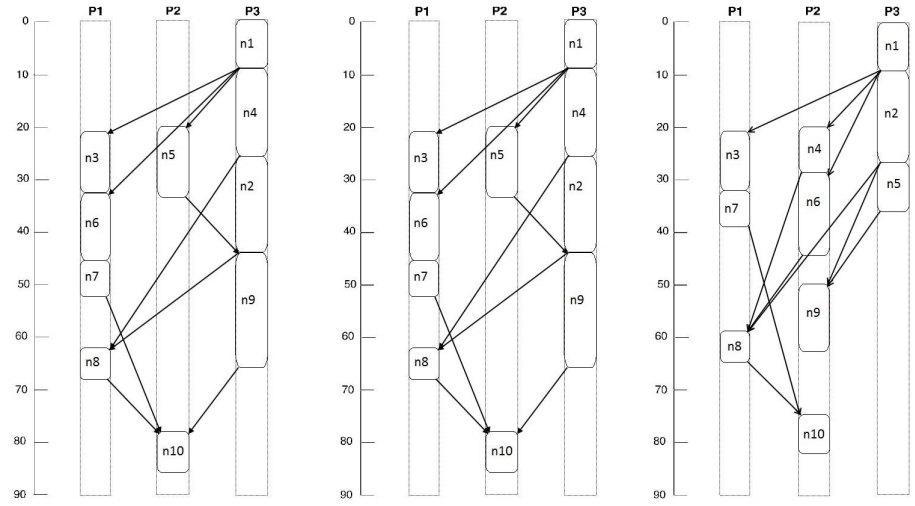


Figure 3. Gantt chart of ranking. a) HEFT ranking algorithm (schedule length = 86) b) CPOP ranking algorithm (schedule length = 86) c) MHEFT ranking algorithm (schedule length = 82).

The scheduled length of MHEFT ranking algorithm is 82 in this example, while CPOP and HEFT ranking algorithms scheduled length is 86 in both cases. The HEFT ranking algorithm computes the rank as follows:

$$\text{rank}(t_i) = w_i + \max_{t_j \in t_{succ}\{t_i\}} (c(i,j) + \text{rank}(t_j)). \tag{5}$$

The Critical Path on a Processor (CPOP) ranking Algorithm computes the rank as follows:

$$\text{rank}(t_i) = \max_{t_j \in t_{pred}\{t_i\}} (w_j + \ c(i,j) + \text{rank}(t_j)). \tag{6}$$

4.2 Task Selection Phase

This phase selects the highest priority ready task from each workflow and a task pool is created.

4.3 Resource Selection Phase

Our resource selection phase algorithm is called the Modified Balance Minimum Completion Time (MBMCT) resource selection algorithm. The MBMCT resource selection algorithm pseudo code is shown in Algorithm 1.

The MBMCT resource selection algorithm first finds the initial assignment of tasks using Min-min heuristic. Min-min heuristic first assigns the task that has

the minimum expected execution time then updates the remaining task's expected execution time, this step is repeated till all the tasks are assigned. After completion of the initial assignment, the average makespan is computed (line 5). Based on the average makespan, the machines are categorized into lightly loaded, moderately loaded and highly loaded machines (line 6–7). The moderately loaded machines load is close to ($\pm\delta$) average load of the system. The lightly loaded machines load is less than moderately loaded machines. The highly loaded machines load is greater than the average load of the system. The reassignment procedure is called till makespan is changed (line 8–12).

The reassignment procedure finds a task that can be transferred from the highly loaded machines to the lightly loaded machines. As a result, the load imbalance is decreased among machines. The pseudo code of the reassignment procedure is shown in Algorithm 2.

The reassignment procedure finds a task which can be reassigned from M_i machine (a machine that belongs to the highly loaded machines) to the lightly loaded machine. Then, it computes the completion time of M_i machine and lightly loaded machine (line 5–6). If completion time of M_i machine and lightly loaded machine is less than the makespan, the makespan is updated (line 7–9). It also keeps information about the machine and the task, where and which task will be moved (line 10–11). At the end, it returns the updated makespan (line 17).

Algorithm 1 Pseudo code of MBMCT resource selection algorithm

```
1: Select ready task from each workflow;
2: Assign tasks to machine using Min-min;
3: repeat
4:    makeSpan ← computeMakespan();
5:    avrgMakeSpan ← computeAverage();
6:    machLow ← findLowMachine(avrgMakeSpan, δ);
7:    machHigh ← findHighMachine(avrgMakeSpan, δ);
8:    newMakeSpan ← reAssignment(machLow, machHigh, makeSpan);
9:    if newMakeSpan < makeSpan then
10:      moveTask();
11:   end if
12: until newMakeSpan < makeSpan;
```

The time complexity of MBMCT resource selection algorithm is $O(k * m * n^2)$. Where k is the number of iterations when the makespan value change, m is the number of machines, and n is the number of tasks.

5 SIMULATION AND RESULTS

In this section, we discuss how to generate workflow, what are the performance parameters, and compare our results with existing algorithms. A Java based simulator is designed.

Algorithm 2 Pseudo code of reassignment procedure

```
for all M_i ∈ machHigh do
  for all M_j ∈ machLow do
    for T_k ∈ M_i do
      comTime'' ← computeFinishTime(M_j);
      comTime'' ← comTime''+ expected execution time of T_k on M_j;
      comTime' ← makeSpan− expected execution time of T_k on M_i;
      if comTime'' < makeSpan then
        if comTime' < makeSpan then
          makeSpan ← comTime'';
          task ← T_k;
          machine ← M_j;
        end if
      end if
    end for
  end for
end for
return makeSpan
```

5.1 Workflow Generation

Random workflows are generated based on a given approach [6]. The parameters to generate a workflow are the number of nodes, the average computation cost, the computation to communication ratio, and the arrival rate is shown in Table 2. The average computation cost denotes an average number of instructions required to execute a task, the computation and communication ratio denotes the average data transfer required between two tasks, the heterogeneity factor denotes the heterogeneity in processor speed, and the shape decides the out degree of nodes in a workflow. The arrival rate is simulated as the corresponding percentage of tasks completed from the recently arrived workflow. For example, 10 % of the inter-arrival time means workflow X will be inserted after completion of 10 % tasks of workflow Y.

Parameter Name	**Range**
Number of nodes	30 to 100
Average computation cost	10 to 200
Computation to communication ratio	0.1 to 10
Heterogeneity factor of resources	0.2 to 0.5
Shape	0.1 to 0.45
Arrival rate	10 to 50
Number of processors	3 to 30

Table 2. Workflow parameters

5.2 Performance Parameters

The performance parameters are Makespan, Schedule Length Ratio (SLR), Load balance, and Turnaround Time Ratio (TTR). Makespan is the difference between execution start time and execution finish time of a workflow. The turnaround time is the super-set of the makespan because it also includes the waiting time of workflow. TTR is the ratio of turnaround time to makespan. The SLR is a normalized makespan based on the critical path. SLR is defined as follows:

$$SLR = \frac{Makespan}{\sum_{t_i \in cp_{min}} \min_{p_j \in P}(w_i, j)}. \tag{7}$$

The denominator in SLR is the minimum computation cost of the critical path tasks (cp_{min}). There is no makespan less than the denominator of the SLR equation. Therefore, the algorithm with lowest SLR is the best algorithm. Average SLR values over several workflows are used in our experiments. Load balance of resource$_i$ is computed as follows:

$$LoadBal_i = \sum_{j \in K}(Makespan_j - ResUti_i). \tag{8}$$

Here K is the set of workflows. Makespan$_j$ is the makespan of workflow$_j$ and $ResUti_i$ is the sum of total time spent by a resource on execution of all assigned tasks of a workflow$_j$. Average load balance is defined as follows:

$$AvgLoadBal = \frac{\sum_{i \in P} LoadBal_i}{p}. \tag{9}$$

5.3 Simulation Results and Analysis

The value of δ is selected $\pm 5\,\%$ of makespan based on experiment. The results presented are averaged out over thirty simulations of each type of workflow. Figures 4 and 5 show the results of different numbers of resources for average makespan and average SLR, respectively. In each case, we have generated 100 workflows of 50 nodes. Workflows are a combination of different communication and computation ratio. At the x-axis, it has a number of resources. In Figure 4, it can be observed that MHEFT ranking algorithm average makespan is less than HEFT ranking algorithm. The difference of average makespan increases as the number of resources are increased. A similar trend is followed in average SLR, as shown in Figure 5. Even when the number of resources is less the MHEFT ranking algorithm's average SLR is better than the HEFT ranking algorithm.

Figures 6 and 7 present the results of different inter-arrival time for the number of iterations and average makespan, respectively. In Figure 7, it can be observed that MBMCT resource selection algorithm's average makespan is less than BMCT resource selection algorithm in each inter-arrival time. The number of iterations of

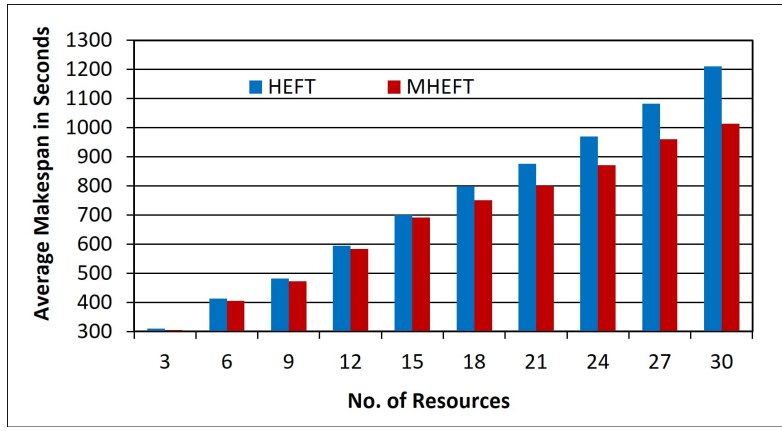


Figure 4. Results of different number of resources for average makespan

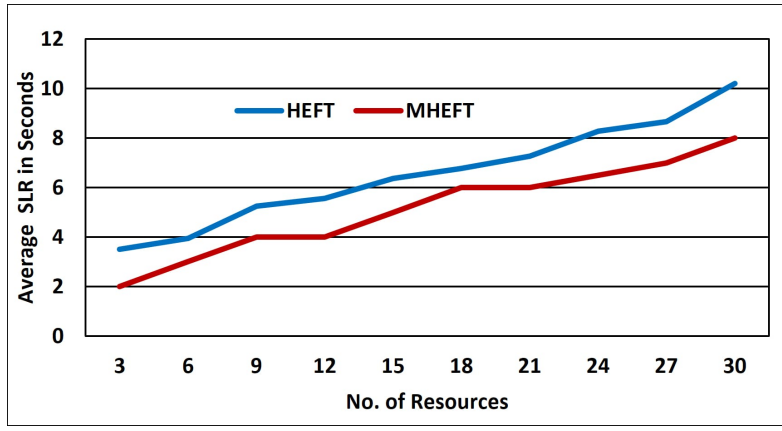


Figure 5. Results of different number of resources for average schedule length ratio

MBMCT resource selection algorithm are slightly greater than the BMCT resource selection algorithm because MBMCT resource selection algorithm moves a task to the lightly loaded machine, while BMCT resource selection algorithm may transfer a task to the moderately loaded machine.

In Figures 8, 9 and 10, LB* scheduling algorithm is the combination of MHEFT ranking algorithm and MBMCT resource selection algorithm. LB scheduling algorithm is the combination of HEFT ranking algorithm and MBMCT resource selection algorithm. To assess the proposed algorithms, we have generated 100 workflows of 70 nodes, that is the combination of different average computation cost and communication ratio. The average computation and communication ratio is randomly chosen from a particular set. In all the performance parameters, LB scheduling al-

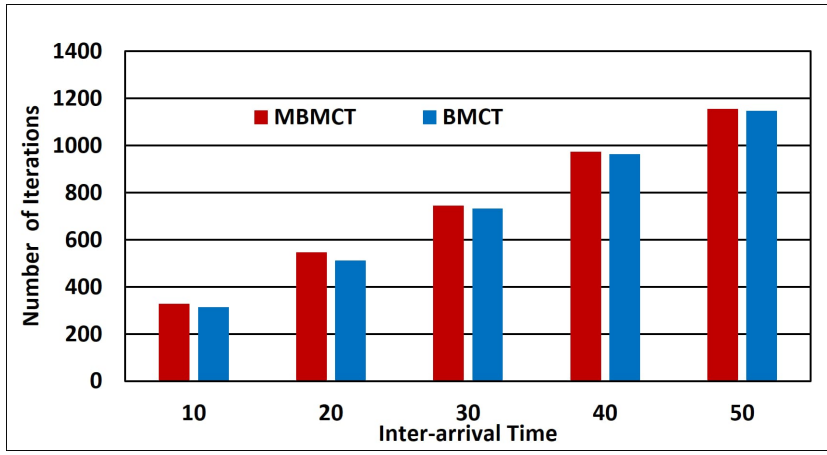


Figure 6. Results of different inter-arrival time for number of iterations

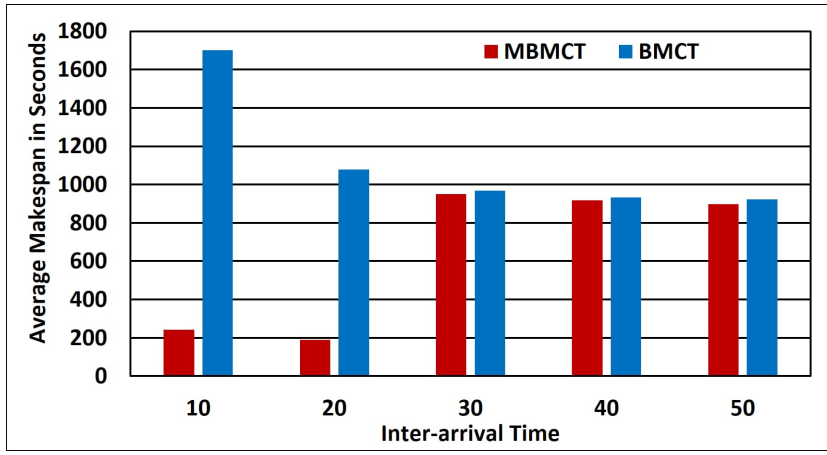


Figure 7. Results of different inter-arrival time for average makespan

gorithm performs better or equally to existing algorithms. It performs better when the inter-arrival time is shorter because in shorter inter-arrival time more number of tasks are scheduled thus, more move is possible. LB* scheduling algorithm performs better than LB scheduling algorithm due to better ranking of tasks in a workflow.

Figure 8 displays the results of different inter-arrival time for average makespan. The LB* scheduling algorithm average makespan is 10 % of the Fair share scheduling algorithm in case of 10 and 20 inter-arrival time. The overall average makespan of LB* scheduling algorithm is 15 % better than the LB scheduling algorithm.

Figure 9 demonstrates the results of different inter-arrival time for average load balance. LB* scheduling algorithm average load balance is 47 %, 37 %, 41 %, and

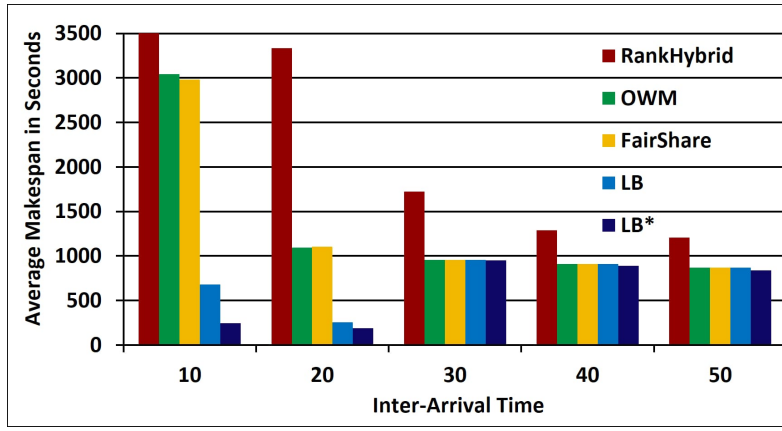


Figure 8. Results of different inter-arrival time for average makespan

26 % less than the LB, the Fair share, the OWM and the Rank hybrid scheduling algorithm, respectively.

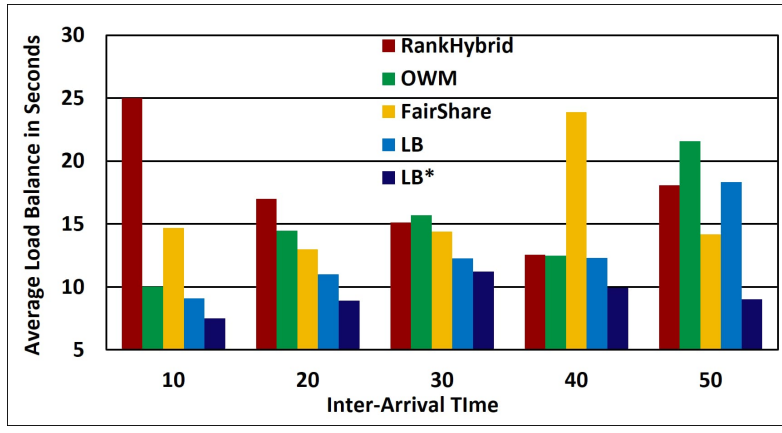


Figure 9. Results of different inter-arrival time for average load balance

Figure 10 shows the results of different inter-arrival time for average TTR. The average TTR of LB* scheduling algorithm is 8 %, 50 %, 50 %, and 73 % less than the LB, the Fair share, the OWM and the Rank hybrid scheduling algorithm respectively.

6 CONCLUSION

We have proposed the load balancing scheduling algorithm for the concurrent workflow that is the combination of modified heterogeneous earliest finish time ranking

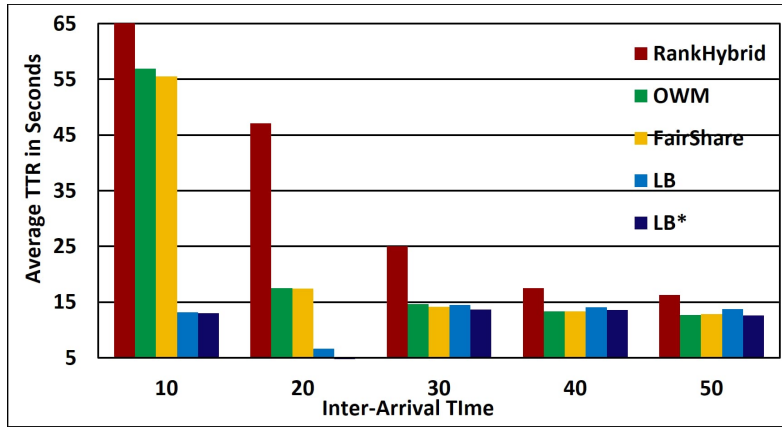


Figure 10. Results of different inter-arrival time for average turnaround time ratio

algorithm and modified balance minimum completion time resource selection algorithm. The modified heterogeneous earliest finish time ranking algorithm considers a communication time of parent task that plays a significant role when tasks are communication intensive in a workflow. Similarly, the modified balance minimum completion time resource selection algorithm transfers the tasks between the highly loaded machines to the lightly loaded machines instead of transfer tasks to all the machines in the system. As a result, faster load balance is achieved in a shorter time. The load balancing scheduling algorithm performs better than other existing scheduling algorithms in a shorter inter-arrival time. In future, we intend to test our algorithm on the real large-scale workflow and heterogeneous environment.

REFERENCES

[1] Foster, I.—Kesselman, C.: The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann Publishers, USA, 1999.

[2] Hoschek, W.—Jaen-Martinez, J.—Samar, A.—Stockinger, H.—Stockinger, K.: Data Management in an International Data Grid Project. Proceedings of the First International Workshop on Grid Computing (GRID 2000), Bangalore, India, 2000. Lecture Notes in Computer Science, Vol. 1971, 2000, pp. 77–90, doi: 10.1007/3-540-44444-0_8.

[3] Wolski, R.—Spring, N. T.—Hayes, J.: Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. Future Generation Computer Systems, Vol. 15, 1999, No. 5-6, pp. 757–768, doi: 10.1016/S0167-739X(99)00025-4.

[4] Gong, L. G.—Sun, X. H.—Watson, E. F.: Performance Modeling and Prediction of Nondedicated Network Computing. IEEE Transactions on Computers, Vol. 51, 2002, No. 9, pp. 1041–1055.

[5] Braun, T. D.—Siegel, H. J.—Maciejewski, A. A.—Hong, Y.: Static Resource Allocation for Heterogeneous Computing Environments with Tasks Having Dependencies, Priorities, Deadlines, and Multiple Versions. Journal of Parallel and Distributed Computing, Vol. 68, 2008, No. 11, pp. 1504–1516, doi: 10.1016/j.jpdc.2008.06.006.

[6] Topcuoglu, H.—Hariri, S.—Wu, M.-Y.: Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. IEEE Transactions on Parallel and Distributed Systems, Vol. 13, 2002, No. 3, pp. 260–274, doi: 10.1109/71.993206.

[7] Zhao, Y.—Wilde, M.—Foster, I.—Voeckler, J.—Dobson, J.—Gilbert, E.—Jordan, T.—Quigg, E.: Virtual Data Grid Middleware Services for Data-Intensive Science. Concurrency and Computation: Practice and Experience, Vol. 18, 2006, No. 6, pp. 595–608.

[8] Arabnejad, H.—Barbosa, J.: Fairness Resource Sharing for Dynamic Workflow Scheduling on Heterogeneous Systems. Proceedings of the 10th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA), 2012, pp. 633–639, doi: 10.1109/ISPA.2012.94.

[9] Yu, Z.—Shi, W.: A Planner-Guided Scheduling Strategy for Multiple Workflow Applications. Proceedings of the IEEE International Conference on Parallel Processing-Workshops (ICPP-W '08), 2008, pp. 1–8.

[10] Zhao, H.—Sakellariou, R.: Scheduling Multiple DAGs onto Heterogeneous Systems. Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS), Rhodes Island, Greece, 2006, pp. 1–14, doi: 10.1109/IPDPS.2006.1639387.

[11] Bansal, S.—Hota, C.: Efficient Refinery Scheduling Heuristic in Heterogeneous Computing Systems. Journal of Advances in Information Technology, Vol. 2, 2011, No. 3, pp. 159–164, doi: 10.4304/jait.2.3.159-164.

[12] Sakellariou, R.—Zhao, H.: A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems. Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS), Santa Fe, New Mexico, USA, 2004, pp. 1–13, doi: 10.1109/IPDPS.2004.1303065.

[13] Yu, J.—Buyya, R.—Ramamohanarao, K.: Workflow Scheduling Algorithms for Grid Computing. In: Xhafa, F., Abraham, A. (Eds.): Metaheuristics for Scheduling in Distributed Computing Environments. Springer, Studies in Computational Intelligence, Vol. 146, 2008, pp. 173–214.

[14] Alam, T.—Raza, Z.: A Dynamic Load Balancing Strategy with Adaptive Thresholds DLBAT for Parallel Computing System. International Journal of Distributed Systems and Technologies, Vol. 5, 2014, No. 1, pp. 54–69.

[15] Hsu, C.-C.—Huang, K.-C.—Wang, F.-J.: Online Scheduling of Workflow Applications in Grid Environments. Future Generation Computer Systems, Vol. 27, 2011, No. 6, pp. 860–870.

Sunita Singhal has been working as Associate Professor, School of Computing and Information Technology of Manipal University Jaipur, India since 2016. Before that she worked at the Birla Institute of Technology and Science (BITS), Pilani Campus, Pilani, India from 2005. She earned her Ph.D. degree from the BITS Pilani in computer science engineering. Her current research focuses on cloud computing, high performance computing, internet of things and distributed computing. She has published several papers in international conferences and journals over the past few years. She acted as a member of the organizing committee and program committee of many international conferences and workshops.

Jemin Patel earned his M.Eng. degree in software systems from the Birla Institute of Technology and Science (BITS), Pilani Campus, Pilani, India. He received his B.Tech. degree in computer engineering from the Dharmsinh Desai University, Nadiad, Gujarat, India in 2013. He is currently doing his internship at Amazon. His research interests include distributed computing and cloud computing.