International Journal of Research in Engineering and Science (IJRES)
ISSN (Online): 2320-9364, ISSN (Print): 2320-9356
www.ijres.org Volume 6 Issue 4 Ver. II ǁ 2018 ǁ PP. 49-60

A Dynamic Scheduling Optimization Model (DSOM)

Joshua Agola[1], George Raburu[2]
[1] *Department of Computer science and Software Engineering*
Jaramogi Oginga Odinga University of Science and Technology
P.O. Box 210 Bondo 40601 Kenya
[2] *Department of Computer science and Software Engineering*
Jaramogi Oginga Odinga University of Science and Technology
P.O. Box 210 Bondo 40601 Kenya
Corresponding Author: Joshua Agola

Abstract: *Scheduling in computing is the rearrangement of machine instructions to obtain the optimal level services subject to various constraints. The scheduling activity is controlled by a scheduler (part of the Operating Systems that initiates and stops the computer programs). This study proposes a dynamic scheduling optimization model (DSOM) that handles unexpected machine breakages and balances the workload amongst the available machines at optimal time and makespan. The model mimics the multiprocessor scheduling theory that is formulated as a blocking parallel-machine job shop scheduling (BPMJSS) problem. In the developed model, any unexpected machine failure during load balancing is detectable and available machines are considered for rescheduling.*

Keywords - *Dynamic, Static, Load balancing, Shifting bottleneck algorithm, shortest processing time algorithm.*

Date of Submission: 02-05-2018 Date of acceptance: 17-05-2018

I INTRODUCTION

Scheduling has been a subject of a significant amount of literature in the operation research field since the early 1950s. The main objective of scheduling is an efficient allocation of shared resources over time to competing activities. Emphasis has been on investigating machine scheduling problems where jobs represent activities and machines represent resources. The problem is not only NP-hard, but also has a well-earned reputation of being one of the most computationally difficult combinatorial optimization problems considered to date. This intractability is one of the reasons why the problem has been so widely studied. Many researchers have developed optimization models to improve the efficiency (Masoud, M.,Kozan, E.,& Kent, G., 2010).However, there is still considerable work that can be done in this field to develop new techniques to achieve further efficiency improvements and get optimal solutions for the dynamic scheduling problem. The study reviewed recent work in dynamic scheduling and presents a new model designed to produce efficient schedules by solving the problems of unexpected machine break down and load balancing during scheduling. The new model includes the load balancing to optimize the performance of the system.

II RELATED WORK

2.1 Dynamic scheduling technique

Ouelhadj & Petrovic (2009) did a survey research on dynamic scheduling in manufacturing systems. They found that a vast majority of the literature dealing with production scheduling has primarily been focused on finding optimal or near-optimal predictive (static) schedules for simple scheduling models with respect to various criteria assuming that all problem characteristics are known. Such predictive schedules are often produced in advance in order to direct production operations and to support other planning activities. Unfortunately, most manufacturing systems operate in dynamic environments subject to various real-time events, which may render the predictive optimal schedule neither feasible nor optimal. Therefore, dynamic scheduling is of great importance for the successful implementation of real-world scheduling systems.

Apurva, Ketan, & Dipti (2010) developed and simulated Dynamic scheduling for real-time distributed systems using ant colony optimization and found that the proposed algorithm is equally efficient during under-loaded conditions.

In parallel computation, the scheduling and mapping tasks is considered the most critical problem which needs High Performance Computing (HPC) to solve it by breaking the problem into subtasks and working on those subtasks at the same time (Abdelkader & Omara, 2011). The application sub tasks are assigned to underline machines and ordered for execution according to its proceeding to grantee efficient use of

available resources such as minimize execution time and satisfy load balance between processors of the underline machine.

The task scheduling can be either static or dynamic. In the static scheduling algorithms, the decision is made prior the execution time when the resources requirement estimated, while the dynamic scheduling algorithms allocate/reallocate resources at run time (Alam &Varshney, 2016; Singh, Alam, & Sharma, 2015).

2.2 Load balancing

Load balancing (Gopinath &Vasudevan, 2015) is a method that distributes the workload among diverse nodes in the given environment such that it ensures no node in the system is over loaded or sits idle for any instant of time. An efficient load balancing algorithm will make sure that every node in the system does more or less same volume of work. The responsibility of load balancing algorithm is that to map the jobs which are set forth to the cloud domain to the unoccupied resources so that the overall available response time is improved as well as it providing efficient resource utilization. The main focus of load balancing in the cloud domain is in allocating the load dynamically among the nodes in order to satisfy the user requirements and to provide maximum resource utilization by assorting the overall available load to distinct nodes.

An appropriate or an ideal load balancing algorithm help in making use of the available resources most favorably, thereby ensuring no node is over loaded or under loaded. Load balancing enables scalability, avoids bottlenecks and also reduces time taken to give the response. Many load balancing algorithm (Moharana, Ramesh &Powar, 2013) have been designed in order to schedule the load among various machines. But so far there is no such ideal load balancing algorithm that has been developed which will allocate the load evenly across the system. It has been proved that allocating the tasks evenly across the system is considered to be an NP complete problem (Fernández-Baca, 1989).

2.3 Shifting bottleneck and Shortest processing time algorithms

(Adams, J., Balas, E., &Zawack, D. (1988) found in his project study that Cutting Planes are fast in optimizing mixed integer programming problems, but unreliable. While Branch and Bound algorithms are reliable but time consuming in optimization. The disadvantage of meta-heuristic techniques (Masoud, 2011) is that there is no guarantee that the best solution found will be the optimal solution. Dynamic programming is often nontrivial to write code that evaluates the sub-problems in the most efficient order. (Masoud, 2011) is his study proposed the use of shifting bottleneck and Shortest processing time to reduce the optimization time.

III THE MODEL

The model is formulated using the Job shop scheduling approach as shown in Figure 3.1. Job shop scheduling is implemented as an integration of mixed integer programming, constraint programming and dynamic disjunctive graph. Constraint programming is implemented using A*search while shortest processing time, shifting bottleneck and load balancing are applied on dynamic disjunctive graph to optimize the mixed integer programming. Flexibility of disjunctive graph is achieved through load balancing which will be checking the availability status of the resources during scheduling. In the event of resource unavailability, alternative resource (node) in the disjunctive graph will be chosen.

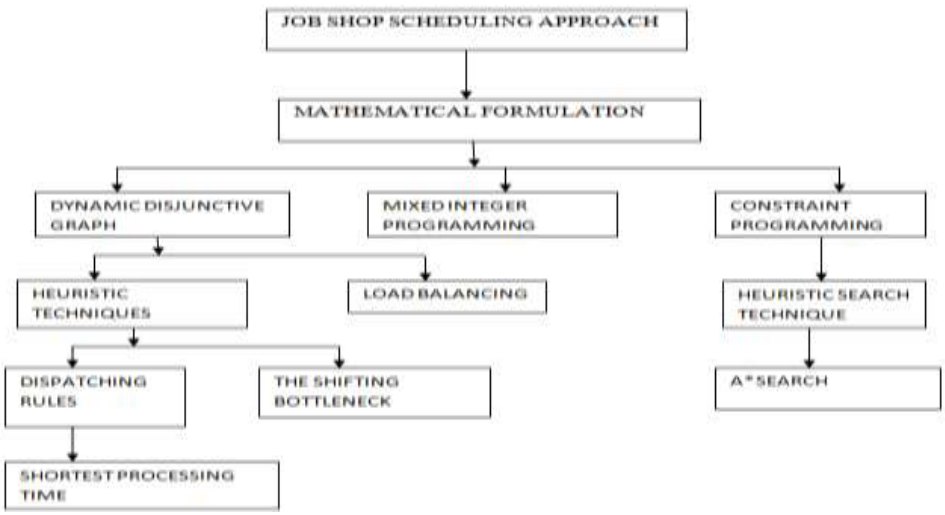


Figure 3.1: Job shop scheduling with dynamic disjunctive graph solution approach for optimizing scheduling problems

The process of scheduling jobs to machines in a shop floor is a complex issue. Some of the machine schedules would be conflicting in a time space while some machines would be overloaded. The conflict and workload of every machine available in a shop floor need proper scheduling to achieve the optimization of the objective functions (minimization of the makespan and the total waiting time). This can either be achieved by Static scheduling model in Figure 3.2 or Dynamic scheduling model in Figure 3.5

3.1Scheduling jobs to machines using Static scheduling model (without load balancing algorithm)

Static scheduling model solves the problem of conflicting scheduling by putting jobs (patients) on the waiting queue as shown in Figure 3.1. There is possibility of some machines having longer waiting queue (workload) than other machines.

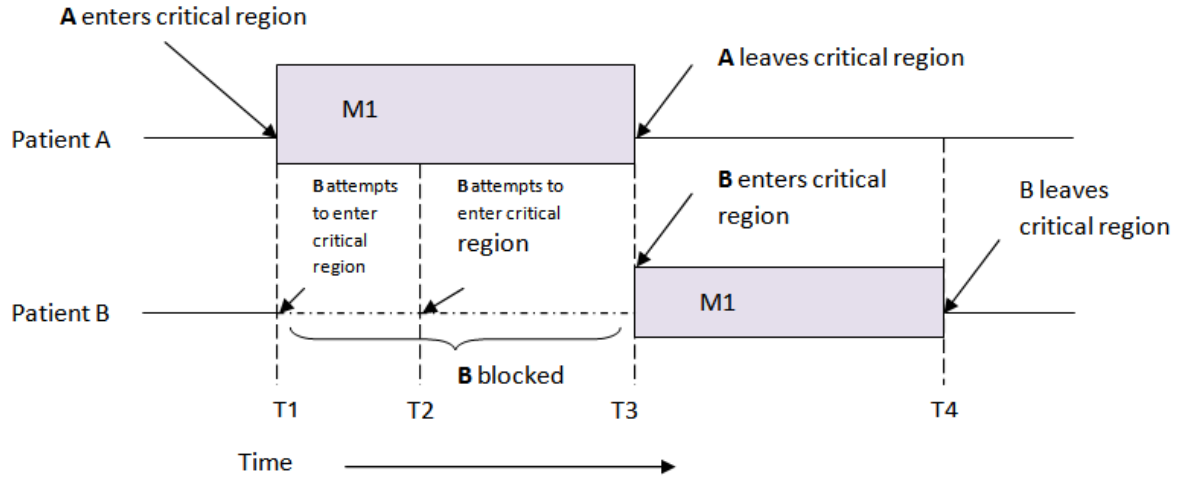


Figure 3.1: A case of static disjunctive graph that generates longer waiting queue in the environment of many patients

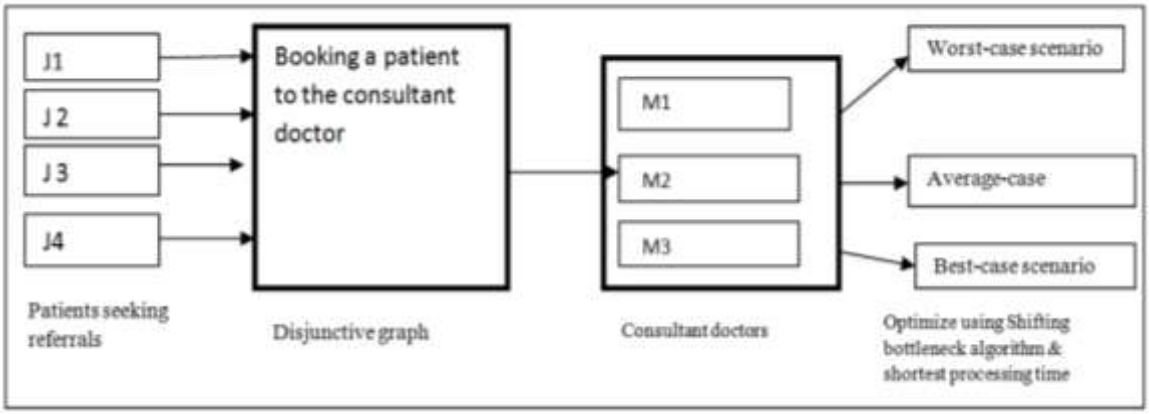


Figure 3.2: Static scheduling model

Solution to scheduling problem when applying the static scheduling model will give the possible outcome as in Figure 3.3. The scheduling process might end up with any of the queue size (workload) as indicated in the model. The outcome could either be in the: Worst-case analysis, or Average-case analysis, or Best-case analysis.

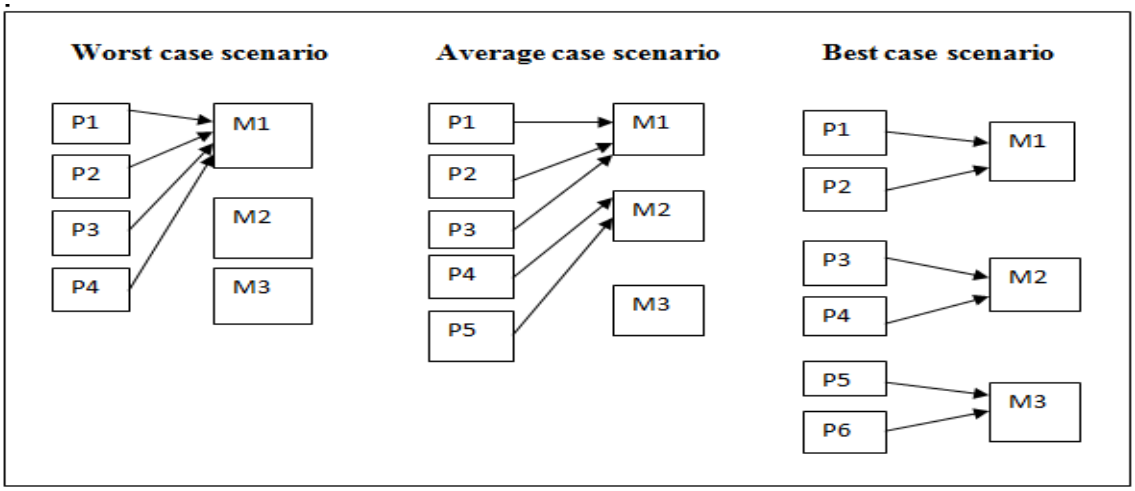


Figure 3.3: Possible outcome of static scheduling model

3.2 Scheduling jobs to machines using the proposed Dynamic scheduling model

In Figure 3.4: Jobs (Patients) A and B are both directed to the same machine (consultant doctor) at the same time. Using dynamic disjunctive graph, job (patient) B would be blocked. But instead of taking it to blocked queue to wait for machine (consultant doctor) (m1) to be free, the next machine (consultant doctor) with lesser workload out of (m2, m3 ….M.) would be selected using load balancing algorithm and allocation would be based on the most appropriate machine (consultant doctor) with comparatively small queue (workload) that would optimize the objective functions.

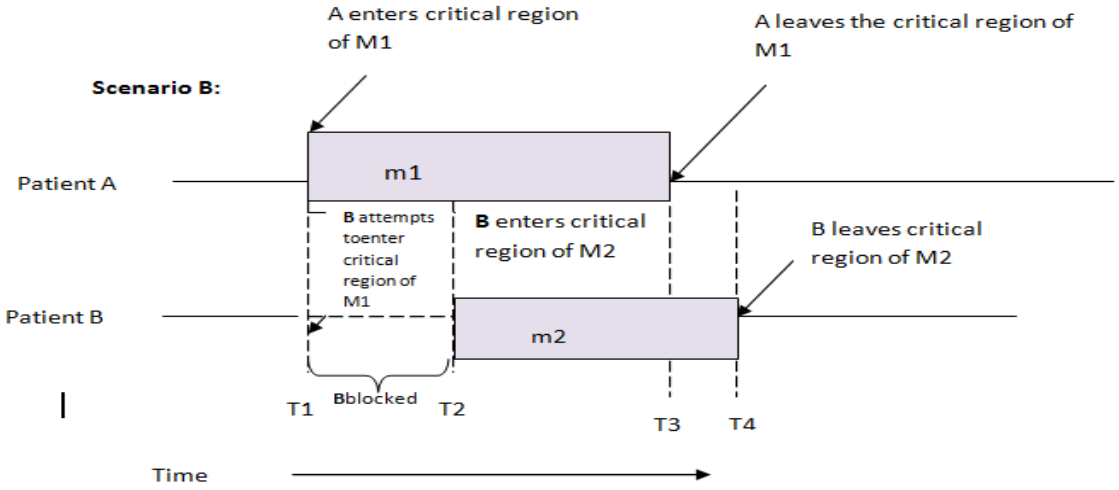


Figure 3.4: A case of dynamic disjunctive graph that assigns the jobs to machines based on the load balancing

3.2.1 Load balancing

The study was aimed at the application of load balancing in developing a dynamic scheduling optimization model. One of the important mechanisms for utilizing and sharing the CPUs optimally is the policy of balancing the load amongst the processors. This type of load balancing can be achieved by transferring some of the tasks from a heavily loaded processor to a lightly loaded processor. Load balancing algorithms may either be static or dynamic, depending upon the rules they follow. Dynamic algorithms on the other hand, distribute tasks using the current state information of the system. The load balancing improves the performance of the system by using the processing power of the entire system more efficiently (Barmon, C., Faruqui, M.N., & Battacharjee, G.P., 1991).

In the study a dynamic load balancing algorithm is developed which improves the performance of the whole scheduling process. The algorithm takes into account the need for transfer of the task as well as the state of the processor to which the transfer is made. The decision of transferring a task depends on the estimate of the total number of tasks waiting for execution. The algorithm requires only the knowledge of the current state of

the waiting queue size of every processor in the neighborhood to distribute the load among processors which is achieved by the following algorithm:

```
Enter number of patients to be scheduled, say int newnumPatient

Enter the number of Consultant doctors available in the hospital, say int newnumDoc

                    if((newnumPatient%newnumDoc)==0)

                    {int  m = (newnumPatient/newnumDoc);}

                    else

                    {int  m=(newnumPatient/newnumDoc) + 1;}

   for(int counter=1; counter<=newnumPatient;counter++) {
    int patient = randomPatient.nextInt(newnumPatient);
    //the random generated customers were then added
    generatedPatients.add(patient);
}
    //used the sublist method once i got the size of the array and split it into m parts
for (int start =0; start < generatedPatients.size();  start += m) {
    int end = Math.min(start + m, generatedPatients.size());
    List<Integer> sublist = generatedPatients.subList(start,  end);
    System.out.println(" Doctor's waiting queue" + " " + sublist);
```

3.2.2 Application of load balancing algorithm to scheduling problem examples:

(i) Scheduling Six (6) patients to Three (3) consultant doctors (Best-case scenario)

```
How many Patients to be booked?:
6
Number of available Consultant Doctors to serve the patients?:
3
 Doctor's waiting queue1:  [1, 3]
 Doctor's waiting queue2:  [4, 4]
 Doctor's waiting queue3:  [2, 0]
```

(ii) Scheduling Four (4) patients to Three (3) consultant doctors (Worst-case scenario)

```
How many Patients to be booked?:
4
Number of available Consultant Doctors to serve the patients?:
3
Doctor's waiting queue [1, 1]
Doctor's waiting queue [2, 2]
```

(iii) Scheduling Five (5) patients to Three (3) consultant doctors (Average-case scenario)

```
how many Patients to be booked?:
5
Number of available Consultant Doctors to serve the patients?:
3
 Doctor's waiting queue [4, 2]
 Doctor's waiting queue [1, 2]
 Doctor's waiting queue [1]
```

The outcome of load balancing algorithm in dynamic scheduling model is shown in Figure 3.5.

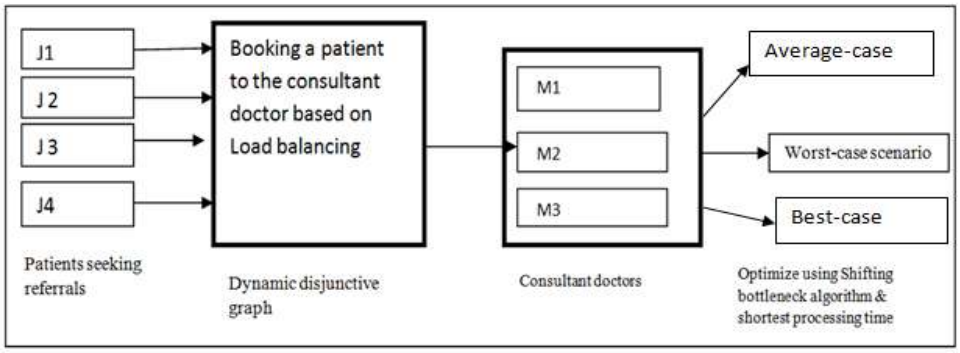


Figure 3.5: Dynamic scheduling model

Solution to scheduling problem when applying the dynamic scheduling model in Figure 3.5 will give the outcome as in Figure 3.6

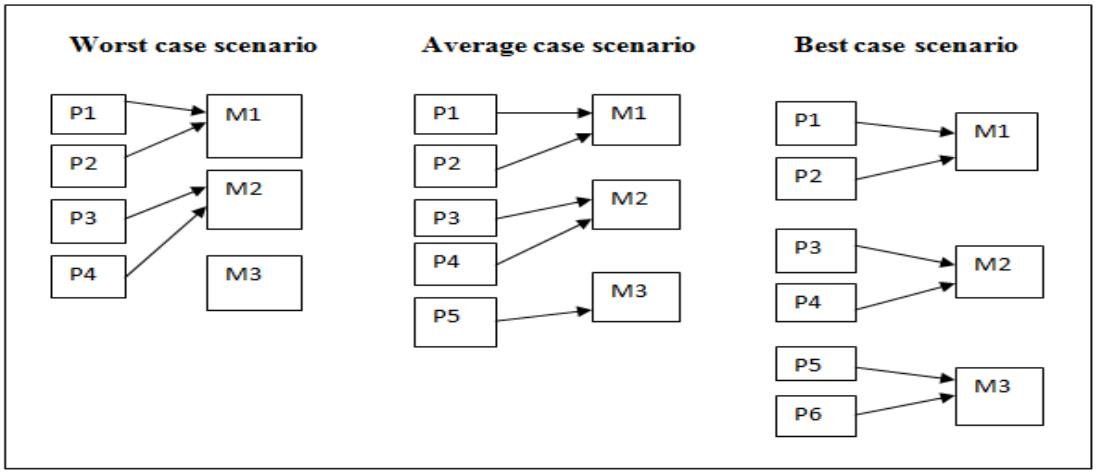


Figure 3.6: Possible outcome of dynamic scheduling model

3.2.3. Simulating Static scheduling model based on developed Java random numbers generator and LEKIN software (which has got the Shifting bottleneck heuristic and Shortest processing time heuristic) to optimize the objective functions

The static scheduling model in Figure 3.7assigns jobs to machines without load balancing

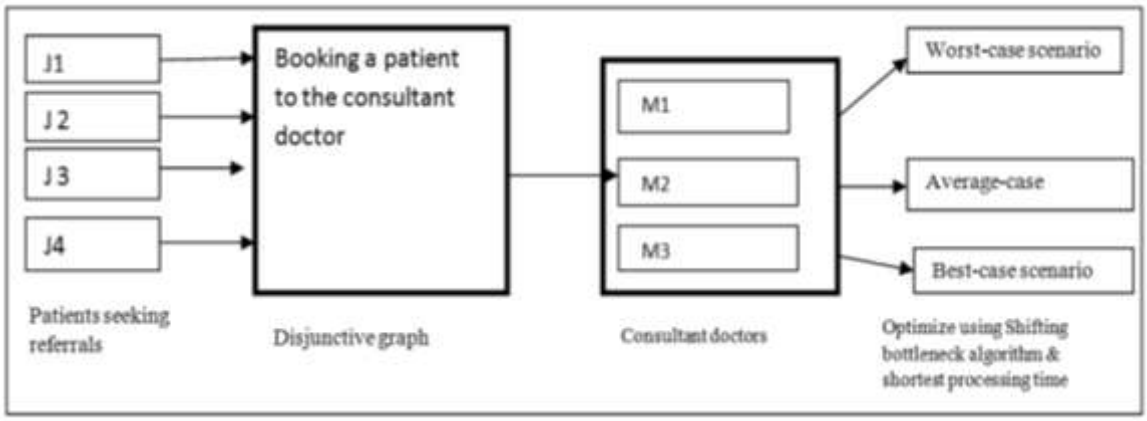


Figure 3.7: Static scheduling model assigns jobs to machines without considering machine's workload

Solution to scheduling problem when applying the static scheduling model will give the possible outcome as in Figure 3.8. The outcome could either be: Worst-case analysis, or Average-case analysis, or Best-case analysis.

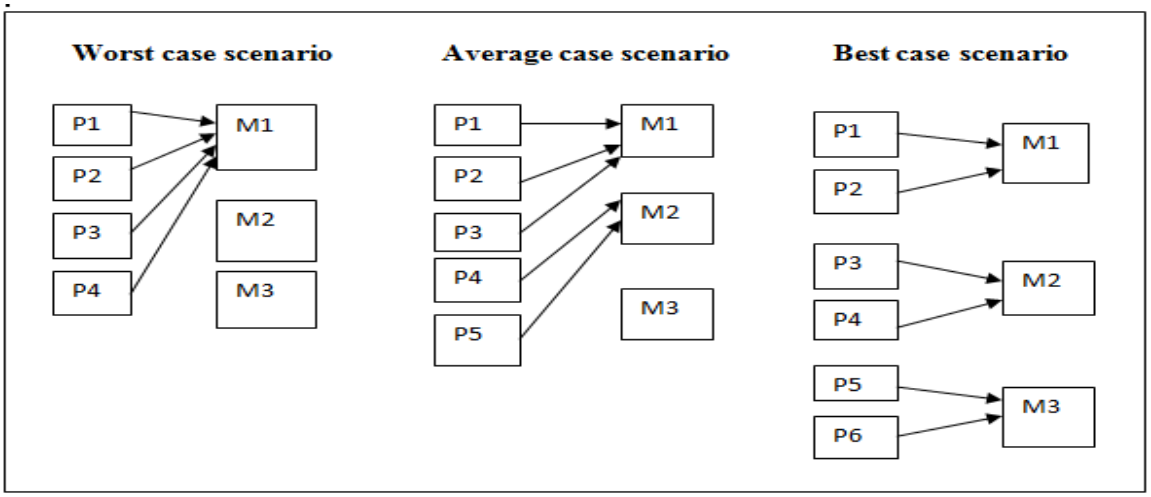


Figure 3.8: Possible outcome of static scheduling model

Worst case scenario analysis

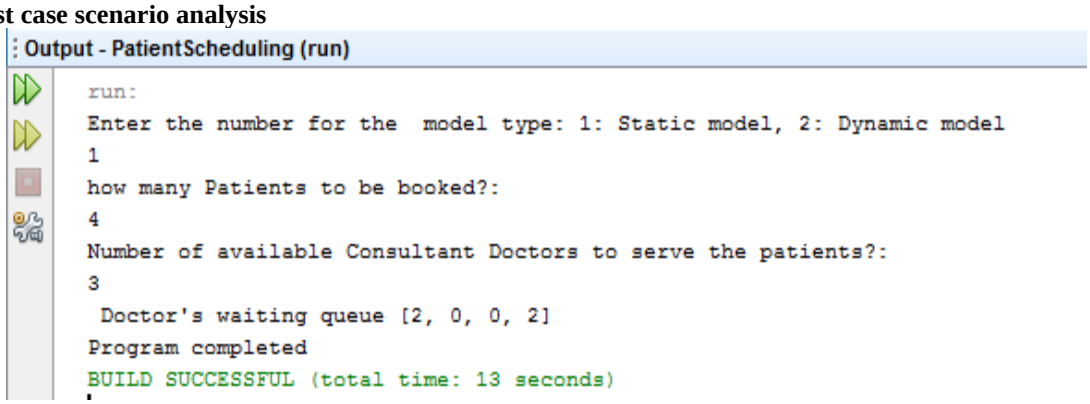
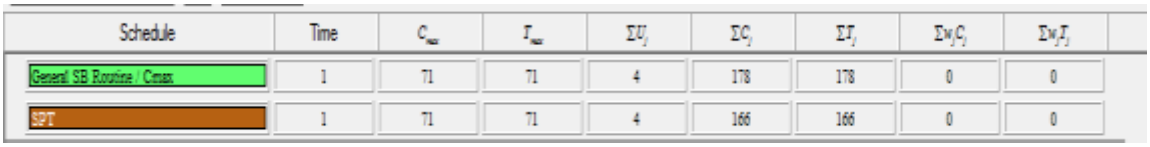


Figure 3.8ai: Randomly generated test data for the waiting queue size in the worst case scenario



Schedule	Time	C_{max}	T_{max}	ΣU_j	ΣC_j	ΣT_j	$\Sigma w_j C_j$	$\Sigma w_j T_j$
General SB Routine / Cmax	1	71	71	4	178	178	0	0
SPT	1	71	71	4	166	166	0	0

Figure 3.8aii: Simulation optimization results of the makespan (Cmax) and the total waiting time ($\sum T_i$) when shifting bottleneck algorithm (SB) and Shortest processing time algorithm (SPT) are used.

Average case scenario analysis

In the figure 3.8bi, there are only two (2) queues for two machines (M1 andM2). Only machineM1 and M2 are having jobs while M3 is idle. M1 is having three (3) jobs while M2 is having two (2) jobs.

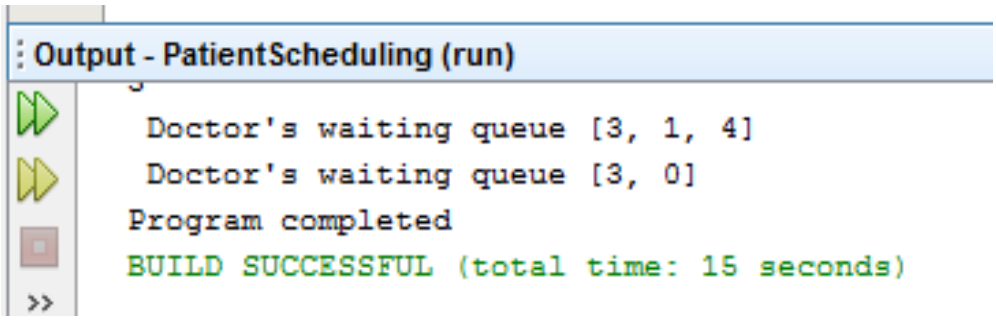


Figure 3.8bi: Randomly generated test data for the waiting queue size in the average case scenario

Schedule	Time	C_{max}	T_{max}	ΣU_j	ΣC_j	ΣT_j	$\Sigma w_j C_j$	$\Sigma w_j T_j$
General SB Routine / Cmax	1	50	50	5	192	192	0	0
SPT	1	71	71	5	196	196	0	0

Figure 3.8bii: Simulation optimization results of the makespan (Cmax) and the total waiting time ($\sum T_i$) when shifting bottleneck algorithm (SB) and Shortest processing time algorithm (SPT) are used

Best-case scenario analysis

In the figure 3.8ci, there are three (3) queues for three machines (M1, M2 and M3) with evenly distribution of jobs. M1, M2 and M3 are having two (2) jobs each.

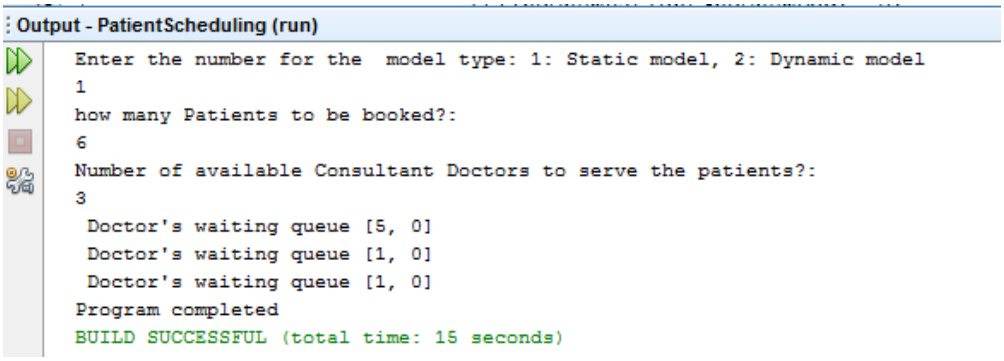


Figure 3.8ci: Randomly generated test data for the waiting queue size in the average case scenario

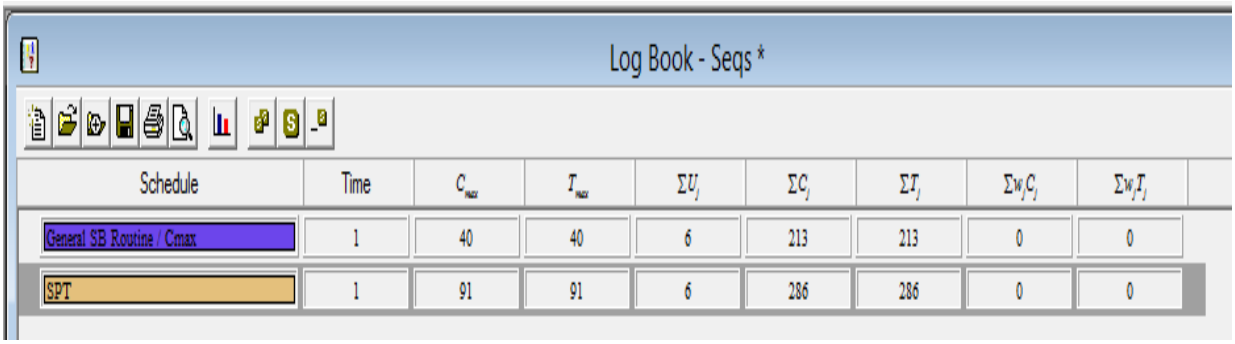


Figure 3.8cii:Simulation optimization results of the makespan (Cmax) and the total waiting time ($\sum T_i$) when shifting bottleneck algorithm (SB) and Shortest processing time algorithm (SPT) are used

3.2.4 Simulating Dynamic scheduling model based on developed Java random numbers generator and LEKIN software (which has got the Shifting bottleneck heuristic and Shortest processing time heuristic) to optimize the objective functions

The dynamic scheduling model in Figure 3.9assigns jobs to machines based on the load balancing.

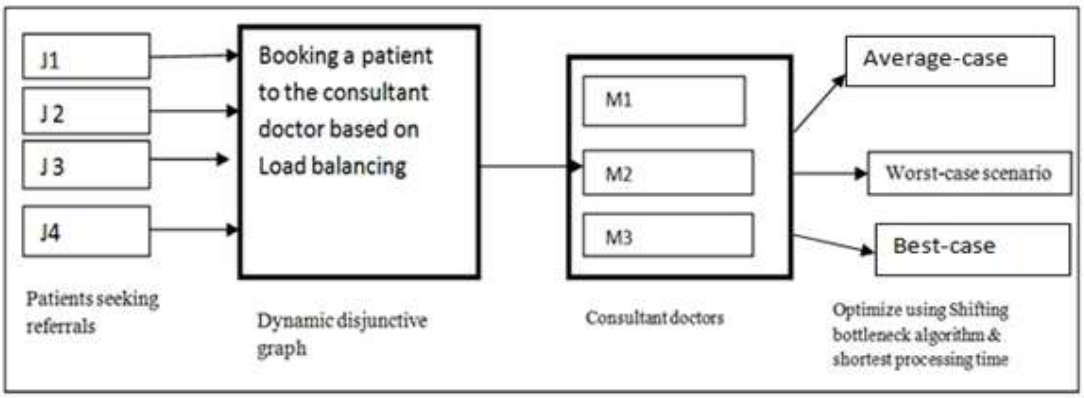


Figure 3.9: Dynamic scheduling model with load balancing

Solution of scheduling when applying the dynamic scheduling model will give the outcome as in Figure 3.10.

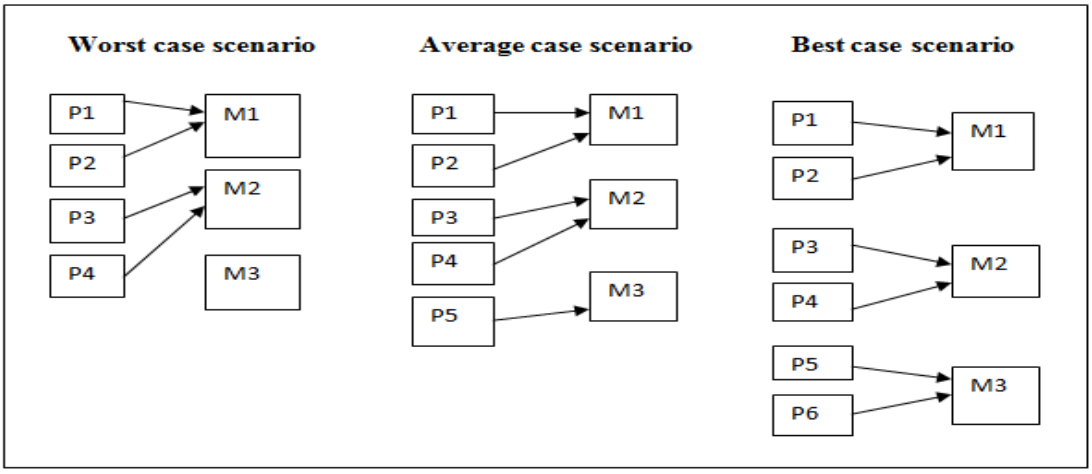


Figure 3.10: The possible outcome of dynamic scheduling model

In the worst-case scenario in the example, there are two queues for machine 1 and machine 2 having two jobs each. Machine 3 is idle.

Worst case scenario analysis

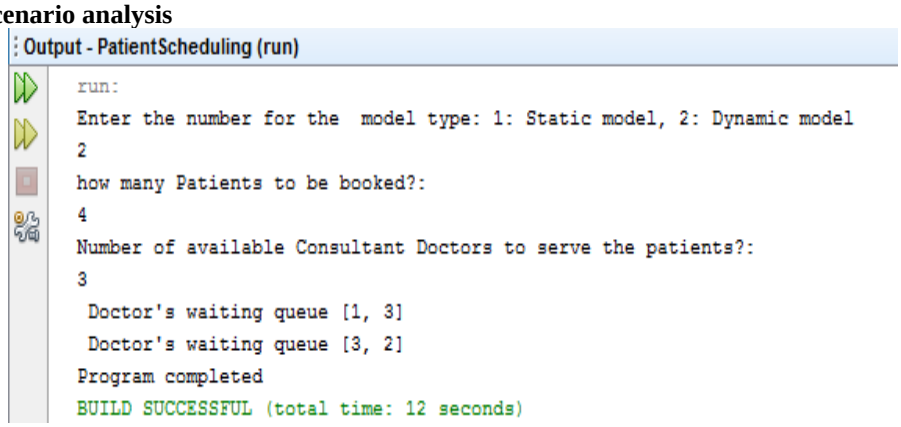


Figure 3.10ai: Randomly generated test data for the waiting queue size in the worst case scenario.

There are two (2) queues for machines (M1 and M2) while machine (consulting doctor) M3 is idle.

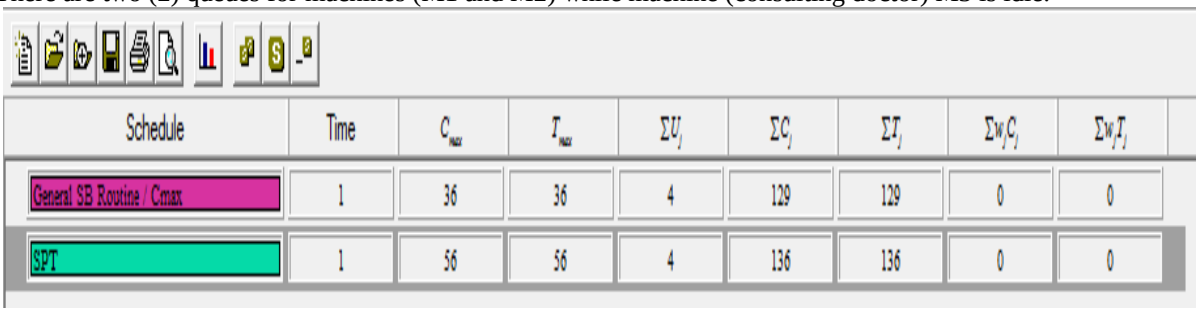


Figure 3.10aii: Simulation optimization results of the makespan (Cmax) and the total waiting time ($\sum T_i$) when shifting bottleneck algorithm (SB) and Shortest processing time algorithm (SPT) are used SB gives far much better results than SPT in the worst case scenario for the makespan and the total waiting time

Average case scenario analysis

In the figure 3.10ai,there are three (3) queues for three machines(M1,M2 and M3) with M1 and M2 having two (2) jobs each while M3 is having one(1) job only.

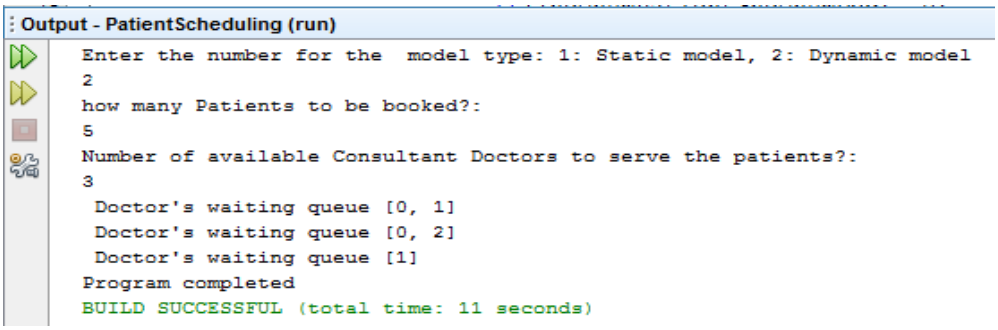


Figure 3.10bi: Randomly generated test data for the waiting queue size in the average case scenario

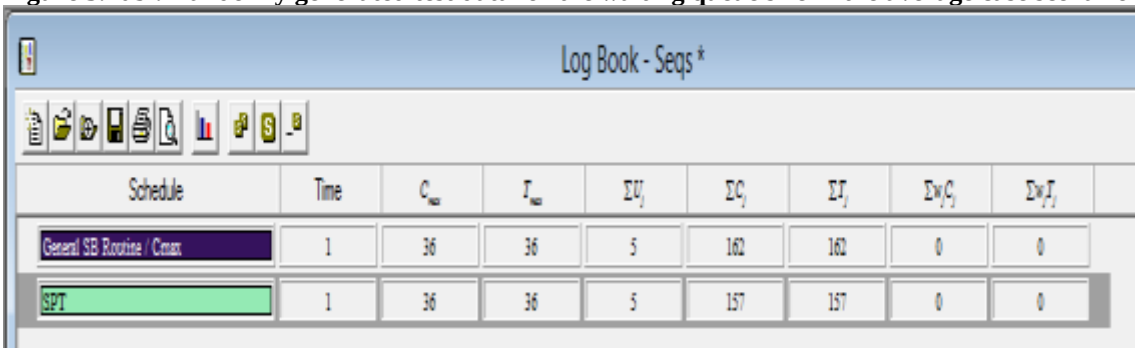


Figure 3.10bii: Simulation optimization results of the makespan (Cmax) and the total waiting time ($\sum T_i$) when shifting bottleneck algorithm (SB) and Shortest processing time algorithm (SPT) are used

The figure 3.10bii compares the makespan (Cmax) of the model when simulated using the Shifting bottleneck heuristic algorithm (SB) and Shortest processing time heuristic algorithm (SPT). The makespan for the two algorithms are the same at 36 units of time while their total waiting times are different by 5 units of time. The total waiting time for shifting bottleneck algorithm is 162 units of time while for Shortest processing time algorithm is 157 units of time

Best-case scenario analysis

In the figure 3.10ci, there are three (3) queues for three machines (M1, M2 and M3) with evenly distribution of jobs. M1, M2 and M3 are having two (2) jobs each.

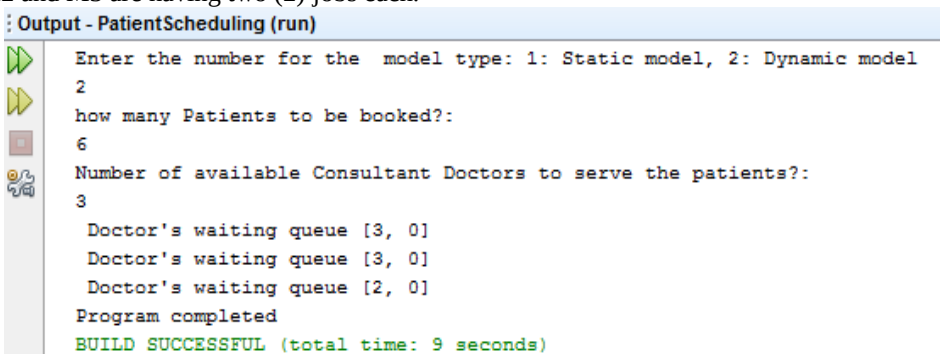


Figure 3.10ci: Randomly generated test data for the waiting queue size in the best case scenario

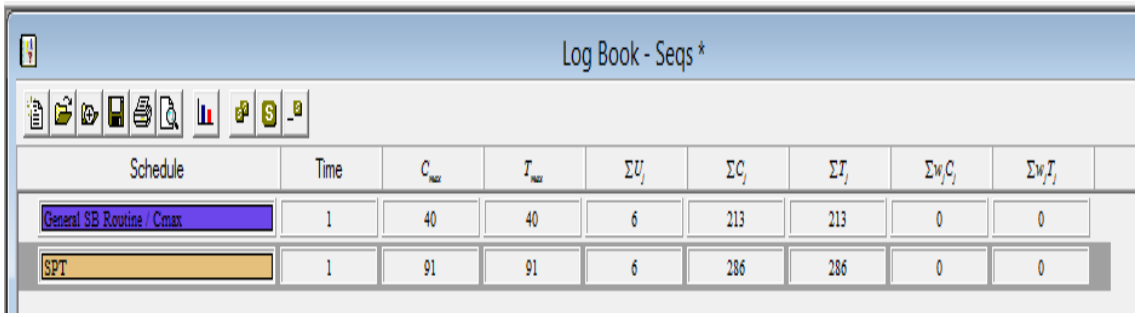


Figure 3.10cii: Simulation optimization results of the makespan (Cmax) and the total waiting time ($\sum T_i$) when shifting bottleneck algorithm (SB) and Shortest processing time algorithm (SPT) are used. SB gives far much better results than SPT in the best case scenario for the makespan and the total waiting time

IV OPTIMIZATION RESULTS IN UNITS TIME OF THE STATIC AND DYNAMIC SCHEDULING MODELS

CASE ANALYSIS SCENARIO	OPTIMIZATION ALGORITHM	STATIC MODEL		DYNAMIC MODEL	
Worst-case scenario	Shifting bottleneck	71	178	36	129
	Shortest processing Time	71	166	56	136
Average-case scenario	Shifting bottleneck	50	192	36	162
	Shortest processing Time	71	196	36	157
Best-case scenario	Shifting bottleneck	40	213	40	213
	Shortest processing Time	91	286	91	286

Table 4.1: Performance analysis of static scheduling model and dynamic scheduling model using Shifting bottleneck algorithm and Shortest processing Time algorithm

Optimization algorithm	**Objective Function**			
	Static scheduling Model		**Dynamic scheduling model**	
	Makespan	Total waiting time	Makespan	Total waiting time
Shifting bottleneck (**SB**)	53.7	194.3	37.3	168
Shortest Processing Time (**SPT**)	77.7	216	61	193

Table 4.2: Performance analysis of static scheduling model and dynamic scheduling model

V DISCUSSION

The two developed scheduling models were tested for their optimization performances using the algorithms that were recommended (Mamoud, 2012) to reduce the optimization time: the shifting bottleneck algorithm and the shortest processing time algorithm

Shifting bottleneck algorithm and Shortest processing Time (SPT) algorithm were proposed to optimize the objective functions within the reasonable time (Masoud, 2012). In this study, the investigation revealed that shifting bottleneck algorithm (SB) performs better than SPT on both static scheduling model and dynamic scheduling model. The performances of both the algorithms are enhanced when load balancing technique is introduced to produce dynamic model. The study also realized that the shifting bottleneck algorithm performs generally much better on dynamic scheduling model than the static scheduling model. The makespan and the total waiting time was reduced by 23.7 and 25 units of time respectively. That is an indication that shifting bottleneck performs better than SPT on the two models.

VI CONCLUSIONS

Dynamic scheduling model gives much better results with both shortest processing time (SPT) and shifting bottleneck algorithms than static scheduling model. The shifting bottleneck algorithm performs much better on both the static scheduling model and dynamic scheduling model as well. Its performance is far much better than SPT on dynamic scheduling model and able to cope with load balancing which take care of any machine break down and distribution of jobs to the machines during run time.

The shifting bottleneck algorithm and load balancing are recommended for the development of dynamic scheduling model to find the optimal solution within a reasonable time. Though the research study does not take care of job break down during run time. the integration of the shifting bottleneck and load balancing algorithms improves the quality solutions and decrease the CPU's time .

VII RECOMMENDATIONS

The future research work should consider developing dynamic scheduling model to cope up with job break down during dynamic scheduling.

REFERENCES

[1]. M. Masoud, E. Kozan, and G. Kent. Scheduling techniques to optimize sugarcane rail systems. *ASOR Bulletin*. 29, 2010, 25-34

[2]. D. Ouelhadj, and S. Petrovic, Survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling*, 12(4), 2009, 417-431.

[3]. S. Apurva, K. Ketan, and S. Dipti, Dynamic scheduling for real-time distributed systems using ant colony optimization. *International Journal of Intelligent Computing and Cybernetics*, 3(2), 2010, 279-292

[4]. D. M. Abdelkader and Omara. Dynamic task scheduling algorithm with load balancing for heterogeneous computing system. *Egyptian Informatics Journal*, 13, 2011, 135–145.

[5]. M. Alam, A.K Varshney. A new approach of dynamic load balancing scheduling algorithm for homogeneous multiprocessor system. *Int. J. Appl. Evol. Comput*. 8 (2), 2016, 61-75.

[6]. K. Singh, M. Alam and S.K Sharma. A survey of static scheduling algorithm for distributed computing system. *Int. J. Comput. Appl*. 129 (2), 2015, 25-30.

[7]. G. P. P. Gopinath and S.K. Vasudevan. An in-depth analysis and study of Load balancing techniques in the cloud computing environment. *Journal of Procedia Computer Science*, 50, 2015, 427 – 432.

[8]. S.S. Moharana,, R.D. Ramesh, and D. Powar. Analysis of load balancers in cloud computing. *International Journal of Computer Science and Engineering (IJCSE)ISSN 2278-9960*, 2(2), 2013, 101-108.

[9]. D. Fernández-Baca. Allocating modules to processors in a distributed system. *Journal of IEEE Transactions on Software Engineering*, 15(11), 1989, 1427-1436.

[10]. J. Adams, E. Balas and D. Zawack. The Shifting Bottleneck Procedure for Job Shop Scheduling. *Management Science*. 34, 1988, 391-401.

[11]. C. Barmon, M.N. Faruqui, and G.P.Battacharjee. Dyanamic load balancing algorithm in distributed system. *Journal of microprocessing and microprogramming*, 29, 1991, 273-285

[12]. M. Masoud. A job shop scheduling approach for optimizing sugarcane rail operation. *Flexible services and manufacturing Journal*, 23(2), 2011, 181- 196.

Joshua Agola." A Dynamic Scheduling Optimization Model (DSOM)." International Journal of Research in Engineering and Science (IJRES), vol. 06, no. 04, 2018, pp. 49–60.