International Journal of Research in Computer Science
eISSN 2249-8265 Volume 2 Issue 3 (2012) pp. 41-49

www.ijorcs.org

MAX MIN FAIR SCHEDULING ALGORITHM USING IN GRID SCHEDULING WITH LOAD BALANCING

R.Gogulan[1], A.Kavitha[2], U.Karthick Kumar[3]

[1]*Phd Research scholar, School of Computer Sciences, Bharath University, Selaiyur, Chennai*
Email: r.gogul@lycos.com
[2]*Research scholar, School of Computer Sciences, Bharath University, Selaiyur, Chennai, Tamil Nadu*
Email: gogulan_kavitha@yahoo.com
[3]*Assistant Professor, Department of MCA & Software Systems, VLB Janakiammal College of Arts and Science Coimbatore, Tamil Nadu*
Email: u.karthickkumar@gmail.com

Abstract: *This paper shows the importance of fair scheduling in grid environment such that all the tasks get equal amount of time for their execution such that it will not lead to starvation. The load balancing of the available resources in the computational grid is another important factor. This paper considers uniform load to be given to the resources. In order to achieve this, load balancing is applied after scheduling the jobs. It also considers the Execution Cost and Bandwidth Cost for the algorithms used here because in a grid environment, the resources are geographically distributed. The implementation of this approach the proposed algorithm reaches optimal solution and minimizes the make span as well as the execution cost and bandwidth cost.*

Keywords: *Grid Scheduling, QOS, Load balancing, Fair scheduling, Execution Cost, Communication Cost.*

I. INTRODUCTION

Grid computing has been increasingly considered as a promising next-generation computing platform that supports wide area parallel and distributed computing since its advent in the mid-1990s [1]. It couples a wide variety of geographically distributed computational resources such as PCs, workstations, and clusters, storage systems, data sources, databases, computational kernels, and special purpose scientific instruments and presents them as a unified integrated resource [2]. The complete grid definition built using all main characteristics and uses may be considered important for several reasons [6]. Grids address issues such as security, uniform access, dynamic discovery, dynamic aggregation, and quality of services [7].

In computational grids, heterogeneous resources with different systems in different places are dynamically available and distributed geographically. The user's resource requirements in the grids vary depending on their goals, time constraints, priorities and budgets. Allocating their tasks to the appropriate resources in the grids so that performance requirements are satisfied and costs are subject to an extraordinarily complicated problem. Allocating the resources to the proper users so that utilization of resources and the profits generated are maximized is also an extremely complex problem. From a computational perspective, it is impractical to build a centralized resource allocation mechanism in such a large scale distributed environment.

In a Grid Scheduler, the mapping of Grid resources and an independent job in optimized manner is so hard. So the combination of uninformed search and informed search provide the good optimal solution for mapping a resources and jobs, to provide minimal turnaround time with minimal cost and minimize the average waiting time of the jobs in the queue. A heuristic algorithm is an algorithm that ignores whether the solution to the problem can be proven to be correct, but which usually produces a good solution.

Heuristics are typically used when there is no way to find an optimal solution, or when it is desirable to give up finding the optimal solution for an improvement in run time. A grid scheduler, often called resource broker, acts as an interface between the user and distributed resources. It hides the complexities of the computational grid from the user. The scheduler does not have full control over the grid and it cannot assume that it has a global view of the grid

Similarly, for resource suppliers, it is hard to evaluate the profit of putting resource into a grid without such a measurement. For both users and suppliers, joining a grid will incur more security and maintenance cost than having only their own computational resources to execute their own tasks. The remaining section of this paper is organized as follows. Section 2 explains the related work. Section 3 Notation and problem formulation is explain, Section 4 explain Existing Method and in section 5 detail the Proposed Method and section 6 describes comparison of tables and charts and in section 7 present the conclusion and future work.

II. RELATED WORK

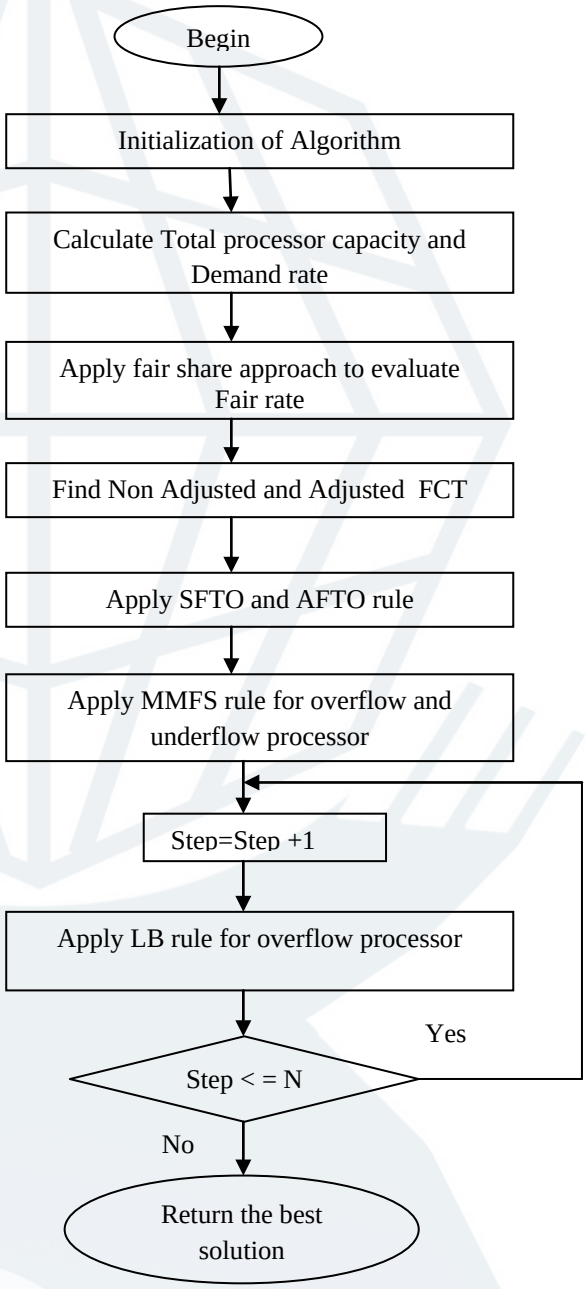
Fair Share scheduling [4] is compared with Simple Fair Task Order Scheduling, Adjusted Fair Task Order Scheduling and Max-Min Fair Share Scheduling algorithm are developed and tested with existing scheduling algorithms. K. Somasundaram, S. Radhakrishnan compares Swift Scheduler with First Come First Serve ,Shortest Job First and with Simple Fair Task Order based on processing time analysis, cost analysis and resource utilization[5]. Thamarai Selvi describes the advantages of standard algorithms such as shortest processing time, longest processing time, and earliest deadline first.

Pal Nilsson and Michal Pioro have discussed Max Min Fair Allocation for routing problem in a communication Network [8]. Hans Jorgen Bang, Torbjorn Ekman and David Gesbert has proposed proportional fair scheduling which addresses the problem of multiuser diversity scheduling together with channel prediction[9]. Daphne Lopez, S. V. Kasmir raja has described and compared Fair Scheduling algorithm with First Come First Serve and Round Robin schemes [10]. Load Balancing is one of the big issues in Grid Computing [11], [12]. B. Yagoubi, described a framework consisting of distributed dynamic load balancing algorithm in perspective to minimize the average response time of applications submitted to Grid computing.

Grosu and Chronopoulos [13], Penmatsa and Chronopoulos [14] considered static load balancing in a system with servers and computers where servers balance load among all computers in a round robin fashion. Qin Zheng, Chen-Khong Tham, Bharadwaj Veradale to address the problem of determining which group an arriving job should be allocated to and how its load can be distributed among computers in the group to optimize the performance and also proposed algorithms which guarantee finding a load distribution over computers in a group that leads to the minimum response time or computational cost [12].

III.NOTATION AND PROBLEM FORMULATION

- Initialization of Algorithm:Number of task and number of resource are initialized at the beginning of the algorithm.
- Calculate the total processor capacity and demand rateis calculated from workload by difference between deadline and grid access delay.
- Evaluate fair rate: From the max min fair share approach calculate fair rate depend on the number of processor and processor capacity.
- Non Adjusted and Adjusted FCT: By using fair rate adjusted and non-adjusted fair completion time is calculated as per SFTO and AFTO.
- SFTO and AFTO rule: Non adjusted fair completion time is used in SFTO to order the processor in increasing order and adjusted fair completion time is used in AFTO to order the processor in increasing order.
- MMFS rule: MMFS is applied here to compensate the overflow and underflow processor.
- LB rule: After MMFS rule LB rule is applied only for overflow processor to reduce the overall completion time of the processor.



Let N be the number of tasks that have to be scheduled and workload w_i of task T_i, i=1, 2… N is the duration of the task when executed on a processor of unit computation capacity. Let M be the number of processors and that the computation capacity of processor j is equal to c_j units of capacity. The total computation capacity C of the Grid is defined [4] as

$$C = \sum_{j=1}^{M} c_j \quad (1)$$

Let d_{ij} be the communication delay between user i and processor j. More precisely, d_{ij} is the time that elapses between the times a decision is made by the resource manager to assign task T_i to processor j and the arrival of all files necessary to run task T_i to processor j.

Each task T_i is characterized by a deadline D_i that defines the time by which it is desirable for the task to complete execution. Let γ_j be the estimated completion time of the tasks that are already running on or already scheduled on processor j. γ_j is equal to zero when no task has been allocated to processor j at the time a task assignment is about to be made; otherwise, γ_j corresponds to the remaining time until the completion of the tasks that are already allocated to processor j. We define the earliest starting time of task T_i on processor j[4] as

$$\delta_{ij}=max\{d_{ij},\gamma_j\} \quad (2)$$

δ_{ij} is the earliest time at which it is feasible for task T_i to start execution on processor j. We define the average of the earliest starting times of task T_i over all the M available processors[4] as

$$\delta_i = \frac{\sum_{j=1}^{M} \delta_{ij} c_j}{\sum_{j=1}^{M} c_j} \quad (3)$$

where δ_i as the grid access delay for task T_i. In the fair scheduling algorithm, the demanded computation rate X_i of a task T_i will play an important role and is defined [4] as

$$X_i = \frac{w_i}{D_i - \delta_i} \quad (4)$$

Here, X_i can be viewed as the computation capacity that the Grid should allocate to task T_i for it to finish just before its requested deadline D_i if the allocated computation capacity could be accessed at the mean access delay δ_i.

VI. EXISTING METHOD

The scheduling algorithms do not adequately address congestion, and they do not take fairness considerations into account. For example, the ECT rule, tasks that have long execution time have a higher probability of missing their deadline even if they have a late deadline. Also, with the EDF rule, a task with a late deadline is given low priority until its deadline approaches, giving no incentive to the users.

To overcome these difficulties, in this section provide an alternative approach, where the tasks requesting service are queued for scheduling according to their fair completion times. The fair completion time of a task is found by first estimating its fair task rates using a max-min fair sharing algorithm.

A. Estimation of the Task Fair Rates

Max-Min Fair Sharing scheme, small demanded computation rates X_i get all the computation power they require, whereas larger rates share leftovers. Max-Min Fair Sharing algorithm is described as follows.

The demanded computation rates X_i, i =1, 2. . . N of the tasks are sorted in ascending order, say, $X_1 < X_2 < \_ \_ \_ < X_N$. Initially, we assign capacity C/N to the task T_1 with the smallest demand X_1, where C is the total grid computation capacity. If the fair share C/N is more than the demanded rate X_1 of task T_1, the unused excess capacity of $C/N - X_1$ is again equally shared to the remaining tasks N-1 so that each of them gets an additional capacity $(C / N + (C / N – X_1)) / (N – 1)$.

This may be larger than task T_2 needs, in which case, the excess capacity is again equally shared among the remaining N-2 tasks, and this process continues until there is no computation capacity left to distribute or until all tasks have been assigned capacity equal to their demanded computation rates. When the process terminates, each task has been assigned no more capacity than it needs, and, if its demand was not satisfied, no less capacity than any other task with a greater demand has been assigned. We denote by $r_i(n)$ the non adjusted fair computation rate of the task T_i at the nth iteration of the algorithm. Then, $r_i(n)$ is given[4] by

$$r_i(n) = \begin{cases} X_i & if\ X_i < \sum_{k=0}^{n} O(k) \\ \sum_{k=0}^{n} O(k) & if\ X_i \geq \sum_{k=0}^{n} O(k) \end{cases} ;\ n \geq 0, \quad (5)$$

Where

$$O(k) = \frac{C - \sum_{i=1}^{N} r_i(n-1)}{Card\{N(n)\}},\ n \geq 1 \quad (6)$$

With

$$O(0) = C/N. \quad (7)$$

Where, N (n) is the set of tasks whose assigned fair rates are smaller than their demanded computation rates at the beginning of the nth iteration, that is,

$$N(n) = \{T_i: X_i > r_i(n-1)\}\ and\ N(0) = N, \quad (8)$$

Whereas, the function card (.) returns the cardinality of a set. The process is terminated at the first iteration no at which either O (n_0) = 0 or the number card {N (n_0)} =0. The former case indicates congestion, whereas the latter indicates that the total grid computation capacity can satisfy all the demanded task rates [4], that is,

$$\sum_{i=1}^{N} X_i < C \qquad (9)$$

The non adjusted fair computation rate r_i of task T_i is obtained at the end of the process as

$$r_i = r_i(n_0). \qquad (10)$$

B. Fair Task Queue Order Estimation

A scheduling algorithm has two important things. First, it has to choose the order in which the tasks are considered for assignment to a processor (the queue ordering problem). Second, for the task that is located each time at the front of the queue, the scheduler has to decide the processor on which the task is assigned (the processor assignment problem). To solve the queue ordering problem in fair scheduling, SFTO and AFTO are discussed.

C. Simple Fair Task Order

In SFTO, the tasks are ordered in the queue in increasing order of their non adjusted fair completion time's t_i. The non adjusted fair completion time t_i of task T_i is defined[4] as

$$t_i = \delta_i + w_i / r_i \qquad (11)$$

where t_i can be thought of as the time at which the task would be completed if it could obtain constant computation rate equal to its fair computation rate r_i, starting at time δ_i .

D.Adjusted Fair Task Order

In the AFTO scheme, the tasks are ordered in the queue in increasing order of their adjusted fair completion times t_i^a. The AFTO scheme results in schedules that are fairer than those produced by the SFTO rule; it is, more difficult to implement and more computationally demanding than the SFTO scheme, since the adjusted fair completion times t_i^a are more difficult to obtain than the non adjusted fair completion times t_i.

i.Adjusted Fair Completion Times Estimation:

To compute the adjusted fair completion times t_i^a, the fair rate of the active tasks at each time instant must be estimated. This can be done in two ways. In the first approach, each time an unused processor capacity is available; it is equally divided among all active tasks. In the second approach, the rates of all active tasks are recalculated using the max-min fair sharing algorithm, based on their respective demanded rates.

The estimated fair rate of each task is a function of time, denoted by $r_i(t)$. Here, introduce a variable called the round number, which defines the number of rounds of service that have been completed at a given time. A non-integer round number represents a partial round of service. The round number depends on the number and the rates of the active tasks at a given time. In particular, the round number increases with a rate equal to the sum of the rates of all active tasks, equal to $1 / \sum_i r_i(t)$. Thus, the rate with which the round number increases changes and has to be recalculated each time a new arrival or task completion takes place. Based on the round number, we define the finish number $F_i(t)$ of task T_i at time t as in [4]

$$F_i(t) = R(\tau) + w_i / r_i(t). \qquad (12)$$

Where τ is the last time a change in the number of active tasks occurred, and $R(\tau)$ is the round number at time τ. $F_i(t)$ is recalculated each time new arrivals or task completions take place. Note that $F_i(t)$ is not the time that task T_i will complete its execution. It is only a service tag that we will use to determine the order in which the tasks are assigned to processors.

The adjusted fair completion times t_i^a can be computed as the time at which the round number reaches the estimated finish number of the respective task. Thus, in [4]

$$t_i^a : R(t_i^a) = F_i(t_i^a) \qquad (13)$$

Where, the task adjusted fair completion times determine the order in which the tasks are considered for assignment to processors in the AFTO scheme: The task with the earliest adjusted fair completion time is assigned first, followed by the second earliest, and so on.

E. Max-Min Fair Scheduling

In MMFS ,the task are non preemptable, the sum of the rates of the tasks assigned for execution to a processor may be smaller than the processor capacity, and some processors may not be fully utilized. A processor with unused capacity will be called an underflow processor. In an optimal solution, tasks assigned to underflow processors have schedulable rates that are equal to their respective fair rates, $r_i^s = r_i$. The overflow O_j of processor j is defined [4]as

$$O_j = max\{0, \sum r_i - c_j\} \qquad (14)$$

$i \in P_j$

And the underflow U_k of processor k as

$$U_k = max \{ 0, \sum_{i \in P_k} r_i - c_k \} \quad (15)$$

Processors for which $O_j > 0$ will be referred to as overflow processors, whereas underflow processors are those for which $U_k < 0$. In an optimal solution, we have

$$\sum_{i \in P_j} r_i^s = c_j \text{ for all } j \text{ for which } O_j > 0 \quad (16)$$

i. Processor Assignment

This algorithm combines processors of capacity overflow with processors of capacity underflow to obtain a better exploitation of the overall processor capacity. More specifically, given an assignment of tasks to processors, we consider the rearrangement where a task of rate r_l assigned to an overflow processor is substituted for a task of rate r_m assigned to an underflow processor. After the task rearrangement, the overflow (underflow) capacity of the processors is updated as[4] follows:

$$\left.\begin{aligned} R_j &= O_j - € \\ R_k &= U_k - € \end{aligned}\right\} \quad (17)$$

Where

$$€ = r_m - r_l. \quad (18)$$

To expresses the task rate difference between the two selected tasks, where R_j and R_k are the updated processor residuals. If $R_j > 0$, processor j remains at the overflow state after the task rearrangement, whereas if $R_j < 0$, processor j turns to the underflow state. A reduction is accomplished only if the task rate difference satisfies the following equation in [4]:

$$€ : O_j^1 + O_k^1 \quad (19)$$

Where $O_j^1 = \max(0, R_j)$ and $O_k^1 = \max(0, R_k)$.This satisfies the processor requirements.

V.PROPOSED METHOD

A. Load Balancing

The existing method is good for fair completion time but the load is not balanced. That is sometimes processor task allocation is excessive than the other, it may take more time to complete the whole job. For this difficulty, here we propose a new algorithm called Load Balance Algorithm to give uniform load to the resources. The overflow O_n and O_p of processors n and p is defined a

$$O_n = max \{0, \sum_{i \in P_n} r_i - c_n\} \quad (20)$$

$$O_p = max \{0, \sum_{i \in P_p} r_i - c_p\} \quad (21)$$

Where $O_n > 0$and $O_p > 0$ referred as n and p are overflow processors. If $O_n > O_p$ then n processor take more time to complete the job. If $O_p > O_n$ then p processor take more time to complete the job. This will cause more time to complete the full job. To recover these problems Load Balance Algorithm proceeds to rearrange the fair rates of the caused processor, so it will reduce overall completion time. The proposed algorithm combines maximum of two processors of capacity of first overflow with processors of capacity of second overflow to obtain a better exploitation of the overall processor capacity. More specifically, given an assignment of tasks to processors, we consider the rearrangement if $O_n > O_p$ then a task of rate r_x assigned to an overflow processor O_n is substituted for a task of rate r_y assigned to an overflow processor O_p. After the task rearrangement, the overflow capacity of the processors is updated as follows:

$$\left.\begin{aligned} R_n &= O_n - € \\ R_p &= O_p - € \end{aligned}\right\} \quad (22)$$

Where $€ = r_x - r_y$. It expresses the task rate difference between the two selected tasks, where R_n and R_p are the updated processor residuals. If $R_n > R_p$, processor n remains at the more completion time. So it will continue from step (1) up to R_n is more or less equal to R_p.

B. Execution Cost

Here, we also implement Execution Cost for all Algorithm used in this thesis. The Execution Cost is defined by $C_{exe}(P_j)$ that is execution cost of j^{th} processor.

$$C_{exe}(P_j) = P(t_i^a)_j * cost_j \quad (23)$$

Where $P(t_i^a)$ is fair completion time of processor j.

C. Communication Cost

Here, we also implements the Communication Cost is defined as

$$C_b(P_j) = C_{exe}(P_j) + F(P_j) \quad (24)$$

Where $C_{exe}(P_j)$ is execution cost of processor j and $F(P_j)$ is Fitness of processor j.

VI.RESULTS

This paper proposes Load Balancing in MMFS to obtain better load balancing. Here, cost rate range from 5 – 10 units is randomly chosen and assigned according to speed of the processor. Speed of the processor ranges from 0 – 1MIPS are randomly assigned to M processor. The proposed method is compared with existing one with different number of processors and tasks. Here number of processor taken are 8, 16, 32 and 64 matrixes with number of task as 256, 512, 1024, and 2048MI. Below table shows the

comparison results of load balance in MMFS with existing algorithm such as EDF, SFTO, AFTO and MMFS for 8, 16, 32, and 64 processors. The proposed work is approximately gives 45% - 25% less than EDF and 7% - 5% less than SFTO and AFTO and 5% - 2% less than MMFS for makespan. Also, MMFS + LB approximately show 30% - 25% less than EDF and 7% - 6% less than SFTO and AFTO 2% - 1% less than MMFS for Execution cost and Bandwidth cost. The result shows better performance for Higher Matrix also. The following are the comparison result of existing and proposed method.

Table: 1 Performance comparison of proposed MMFS + LB with existing algorithm for EDF, SFTO, AFTO + MMFS for 8 processors

Scheduling Algorithm	Resource Matrix	Makespan	Execution Cost	Communication Cost
EDF	256 x 8	917.82	5506.91	6424.73
SFTO		447.74	4477.44	4925.19
AFTO		444.39	4468.54	4912.94
MMFS		439.61	4446.77	4886.39
MMFS + LB		418.13	4181.27	4599.4
EDF	512 x 8	1121.32	7849.21	8970.53
SFTO		1022.36	5111.8	6134.16
AFTO		1010.09	5050.45	6060.53
MMFS		858.54	4292.71	5151.26
MMFS + LB		836.72	4183.58	5020.3
EDF	1024 x 8	1825.33	10951.97	12777.3
SFTO		1651.45	13211.63	14863.08
AFTO		1686.17	11803.21	13489.38
MMFS		1643.32	13180.96	14824.28
MMFS + LB		1599.82	12798.55	14398.36
EDF	2048 x 8	3596.42	25174.94	28771.36
SFTO		3280.39	26243.11	29523.5
AFTO		3247.63	25981.06	29228.69
MMFS		3137.59	25100.75	28238.34
MMFS + LB		3095.82	24766.55	27862.37

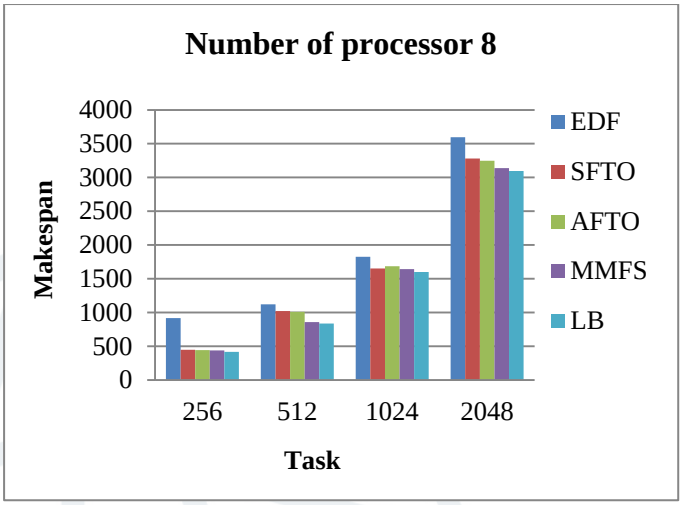


Fig 1: Performance comparison of proposed MMFS + LB with existing algorithm for EDF, SFTO, AFTO + MMFS for Makespan

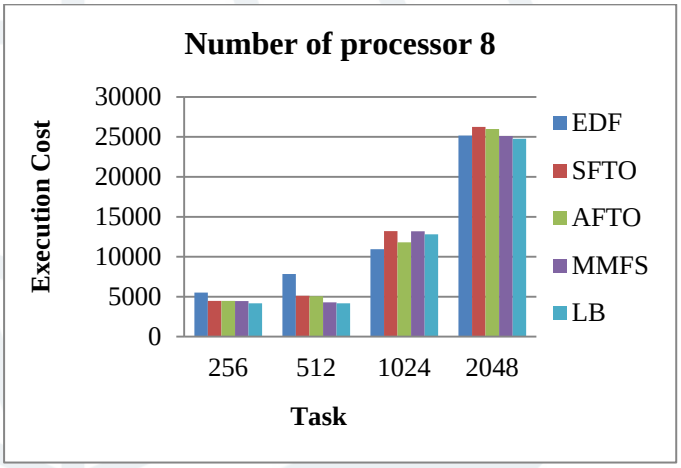


Fig 2: Performance comparison of proposed MMFS + LB with existing algorithm for EDF, SFTO, AFTO + MMFS for Execution Cost

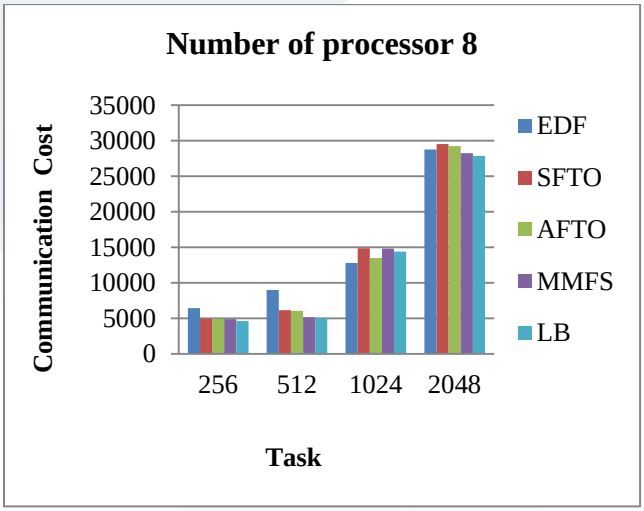


Fig 3: Performance comparison of proposed MMFS + LB with existing algorithm for EDF, SFTO, AFTO + MMFS for Bandwidth Cost

Table: 2 Performance comparison of proposed MMFS + LB with existing algorithm for EDF, SFTO, AFTO + MMFS for 16 processors

Scheduling Algorithm	Resource Matrix	Makespan	Execution Cost	Communication Cost
EDF	256 x 16	1466.72	7332.11	8798.53
SFTO		304	1520	1824
AFTO		300.65	1511.1	1811.75
MMFS		295.87	1489.33	1785.2
MMFS + LB		209	1045	1254
EDF	512 x 16	1366.48	13664.81	15031.29
SFTO		553.89	5538.91	6092.8
AFTO		555.37	5553.75	6109.12
MMFS		545.76	5508.24	6054
MMFS + LB		483.57	4835.69	5319.26
EDF	1024 x 16	1540.27	9241.6	10781.86
SFTO		1309.94	6549.72	7859.66
AFTO		1296.35	6481.77	7778.13
MMFS		1301.81	6519.05	7820.86
MMFS + LB		1231.43	6157.14	7388.57
EDF	2048 x 16	3352.67	23468.72	26821.39
SFTO		2742.53	24682.76	27425.29
AFTO		2761.98	27619.81	30381.79
MMFS		2734.4	24652.09	27386.49
MMFS + LB		2641.04	23769.35	26410.39

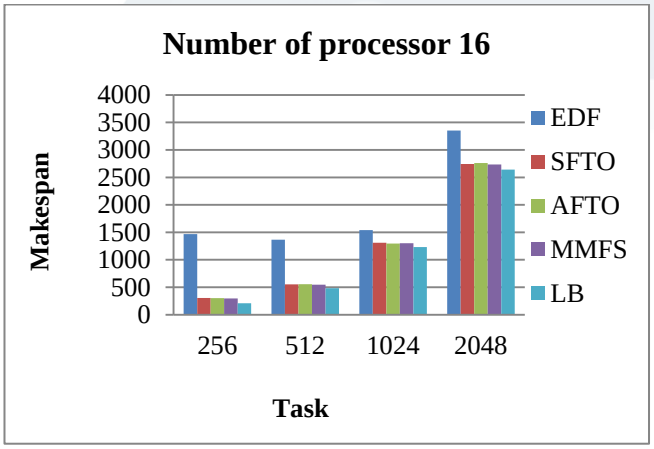


Fig 4: Performance comparison of proposed MMFS + LB with existing algorithm for EDF, SFTO, AFTO + MMFS for makespan

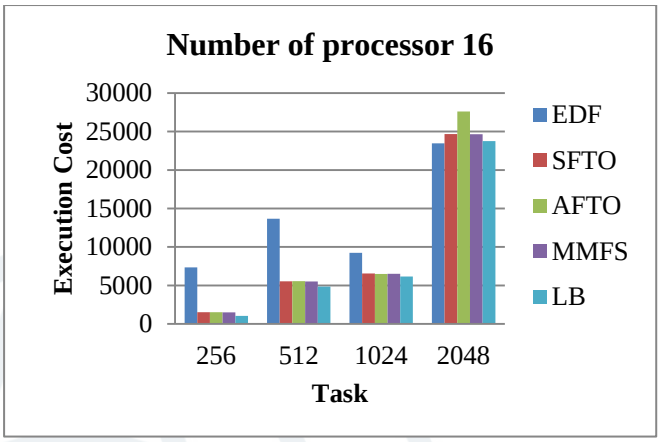


Fig 5: Performance comparison of proposed MMFS + LB with existing algorithm for EDF, SFTO, AFTO + MMFS for Execution Cost

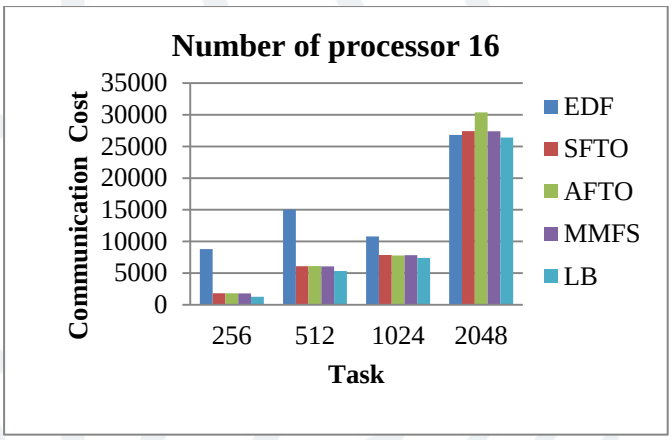


Fig 6: Performance comparison of proposed MMFS + LB with existing algorithm for EDF, SFTO, AFTO + MMFS for Communication Cost

Table 3: Performance comparison of proposed MMFS + LB with existing algorithm for EDF, SFTO, AFTO + MMFS for 32 processors

Scheduling Algorithm	Resource Matrix	Makespan	Execution Cost	Communication Cost
EDF	256 x 32	206.05	1648.40	1854.45
SFTO		183.43	917.15	1100.58
AFTO		180.08	908.25	1088.33
MMFS		175.30	886.48	1061.78
MMFS + LB		114.64	573.22	687.86
EDF	512 x 32	744.80	5958.36	6703.16
SFTO		580.60	5225.43	5806.04
AFTO		577.25	5216.53	5793.79
MMFS		574.27	3445.64	4019.91
MMFS + LB		464.54	2787.23	3251.77
EDF	1024 x 32	966.47	7731.78	8698.26
SFTO		912.96	4564.78	5477.73
AFTO		937.07	7512.53	8451.59
MMFS		904.83	4534.11	5438.93
MMFS + LB		863.54	4317.72	5181.26
EDF	2048 x 32	2675.05	18725.38	21400.44
SFTO		2427.97	21851.76	24279.74
AFTO		2375.23	21377.09	23752.32
MMFS		2370.11	21330.99	23701.10
MMFS + LB		2262	20359.85	22622.06

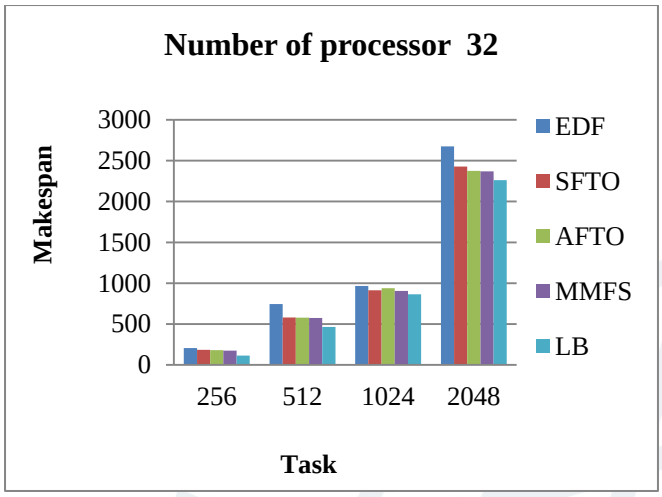


Fig 7: Performance comparison of proposed MMFS + LB with existing algorithm for EDF, SFTO, AFTO + MMFS for makespan

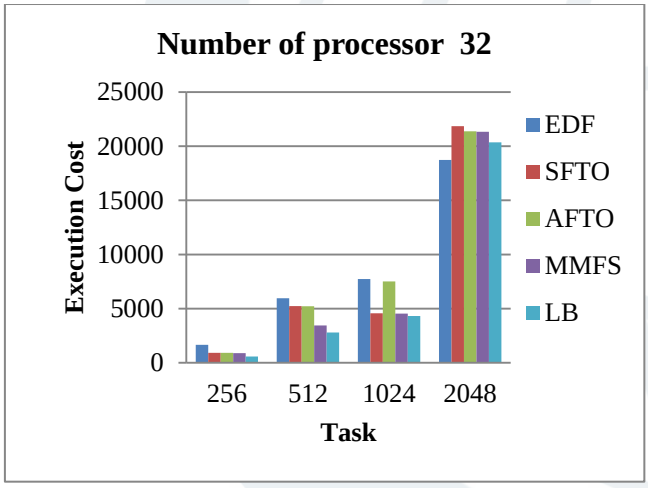


Fig 8: Performance comparison of proposed MMFS + LB with existing algorithm for EDF, SFTO, AFTO + MMFS for Execution Cost

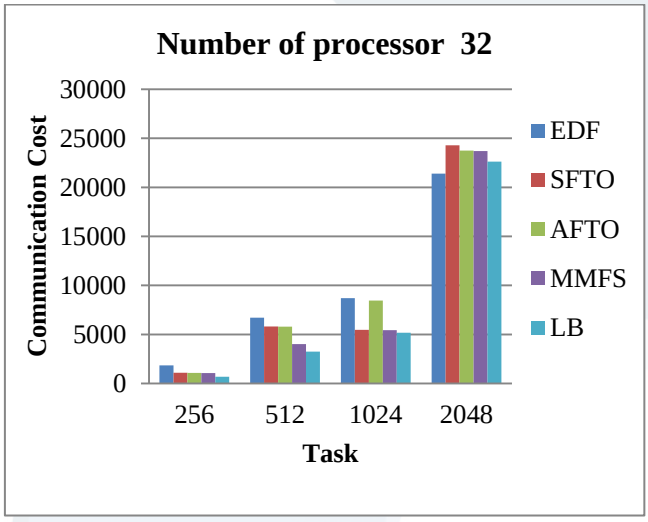


Fig 9: Performance comparison of proposed MMFS + LB with existing algorithm for EDF, SFTO, AFTO + MMFS for Communication Cost

TABLE 4: Performance comparison of proposed MMFS + LB with existing algorithm for EDF, SFTO, AFTO + MMFS for 64 processors

Scheduling Algorithm	Resource Matrix	Makespan	Execution Cost	Communication Cost
EDF	256 x 64	305.35	2748.13	3053.47
SFTO		281.93	1691.60	1973.53
AFTO		278.58	1682.70	1961.28
MMFS		273.80	1660.93	1934.73
MMFS + LB		211.45	1268.7	1480.15
EDF	512 x 64	966.67	5800.02	6766.69
SFTO		600	3000	3600
AFTO		596.65	2991.10	3587.75
MMFS		591.87	2969.33	3561.20
MMFS + LB		450	2700	3150
EDF	1024 x 64	968.49	5810.95	6779.44
SFTO		978.87	9788.75	10767.62
AFTO		975.52	9779.85	10755.37
MMFS		970.74	9758.08	10728.82
MMFS + LB		795.34	7953.36	8748.69
EDF	2048 x 64	2984.98	23879.85	26864.83
SFTO		2630.08	26300.85	28930.93
AFTO		2626.73	26291.95	28918.68
MMFS		2621.95	26270.18	28892.13
MMFS + LB		2330.86	23308.63	25639.50

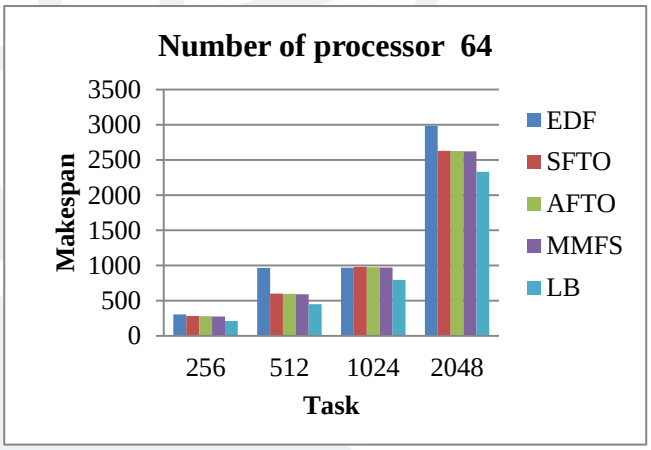


Fig 10: Performance comparison of proposed MMFS + LB with existing algorithm for EDF, SFTO, AFTO + MMFS for Makespan

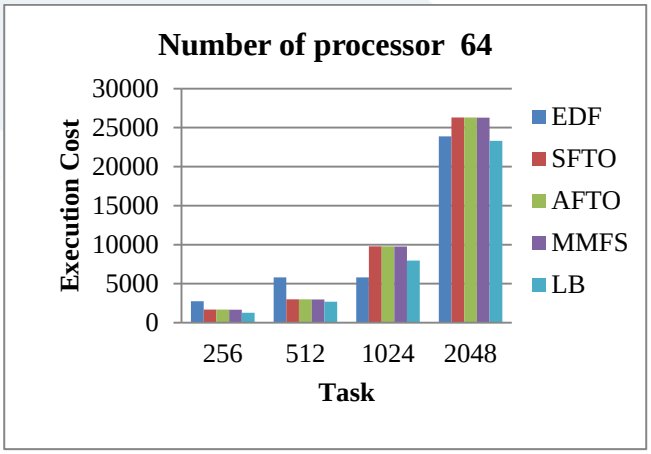


Fig 11: Performance comparison of proposed MMFS + LB with existing algorithm for EDF, SFTO, AFTO + MMFS for Execution Cost

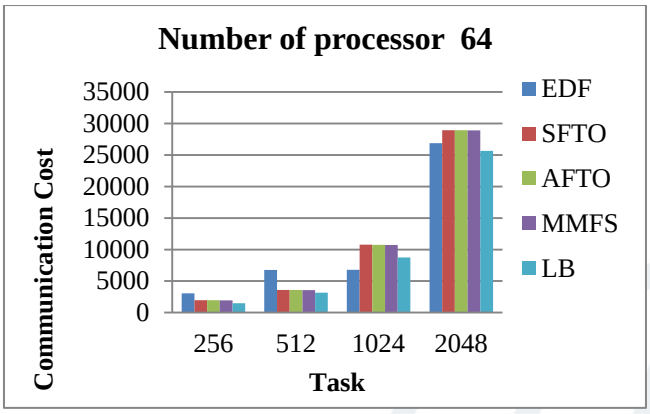


Fig 12: Performance comparison of proposed MMFS + LB with existing algorithm for EDF, SFTO, AFTO + MMFS for Communcation Cost

VII. CONCLUSION

In this paper, Load Balancing algorithm is compared with normal scheduling algorithm such as Earliest Deadline First, and Fair Scheduling algorithm such as SFTO, AFTO and MMFS. Our proposed algorithm shows better result for execution cost and bandwidth cost also. Result shows that load balancing with scheduling produces minimum makespan than others. Future work will focus on that how fair scheduling can be applied to optimization techniques, QoS Constrains such as reliability can be used as performance measure.

VIII. REFERRENCES

[1] Rajkumar Buyya, David Abramson, and Jonathan Giddy A Case for Economy Grid Architecture for Service Oriented Grid Computing

[2] Foster.I.,Kesselman.C(1999) The Grid: Blueprint for a New Computing Infrastructure. *Morgan Kaufmann* Publishers, *USA.*

[3] Wolski.R, Brevik.J, Plank.J, and Bryan.T, (2003) Grid Resource Allocation and Control Using Computational Economies, In Grid Computing: Making the Global Infrastructure a Reality. *Berman, F, Fox, G., and Hey, T. editors, Wiley and Sons,* pp. 747--772, 2003. doi:10.1002/0470867167

[4] Doulamis.N.D.Doulamis. A.D, Varvarigos. E.A. Varvarigou. T.A (2007) Fair Scheduling Algorithms in Grids *.IEEE Transactions on Parallel and Distributed Systems, Volume18, Issue 11Page(s):1630 – 1648.*

[5] K.Somasundaram, S.Radhakrishnan (2009) Task Resource Allocation in Grid using Swift Scheduler. *International Journal of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844 Vol. IV.*

[6] Miguel.L, Bote-Lorenzo, Yannis.A Dimitriadis, And Eduardo Gomez-Sanchez(2004) Grid Characteristics and Uses: a Grid Definition. *Springer-Verlag LNCS 2970, pp. 291-298.*

[7] Parvin Asadzadeh, Rajkumar Buyya1, Chun Ling Kei, Deepa Nayar, And Srikumar Venugopal Global Grids and Software Toolkits:A Study of Four Grid Middleware Technologies.

[8] Pal Nilsson1 and Michał Pi´Oro Unsplittable max-min demand allocation – a routing problem.

[9] Hans Jorgen Bang, Torbjorn Ekman And David Gesbert A Channel Predictive Proportional Fair Scheduling Algorithm.

[10] Daphne Lopez. S. V. Kasmir Raja (2009) A Dynamic Error Based Fair Scheduling Algorithm For A Computational Grid. *Journal Of Theoretical And* Applied *Information Technology JATIT.*

[11] Qin Zheng, Chen-Khong Tham, Bharadwaj Veeravalli (2008) Dynamic Load Balancing and Pricing in Grid Computing with Communication Delay.*Journal in Grid Computing*. doi:10.1007/s10723-007-9093-5

[12] Stefan Schamberger (2005) A Shape Optimizing Load Distribution Heuristic for Parallel Adaptive fem Computations.*Springer-Verlag Berlin Heidelberg* .

[13] Grosu, D, Chronopoulos. A.T (2005) Noncooperative load balancing in distributed systems.*Journal of Parallel Distrib. Comput.* ***65****(9), 1022–1034.* doi:10.1016/j.jpdc.2005.05.001

[14] Penmatsa, S., Chronopoulos, A.T (2005) Job allocation schemes in computational Grids based on cost optimization.*In: Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium,Denver*. doi:10.1109/IPDPS.2005.264

How to cite

R.Gogulan, A.Kavitha, U.Karthick Kumar, "Max Min Fair Scheduling Algorithm Using In Grid Scheduling With Load Balancing". *International Journal of Research in Computer Science*, 2 (3): pp. 41-49, May 2012. doi:10.7815/ijorcs.23.2012.028