New Backfilling algorithm for multiprocessor scheduling with mathematics algo.

Ajay Jain* Dharmendra Saxena **

Deepshikha Group Of Colleges, Jaipur

In this study, we propose an efficient algorithm for the multiprocessor job scheduling problem. From a given list of jobs, Jobs are queued according to the decreasing order of their durations. Depending upon the job duration, jobs are divided into multiple threads for processing. Multi-thread jobs are processed based on the concept of 'gang scheduling'. To minimize the idle time of the processors, backfilling approach is incorporated into the algorithm.

Keyword: multiprocessor, Greedy, Gang scheduling, Backfilling

Introduction

The size of real-life scheduling problems is too large to be solved by single processor computer systems. Therefore, multiprocessor scheduling has become necessary to solve these problems. Multiprocessor is the coordinated processing of program by more than one processor. These scheduling problems are known to be NP-hard even in the very restricted situations (Ullman, 1975; Hou, 1994). Approximation algorithm has been developed in response to the challenge of solving these problems to optimality. However, in order to be useful in practical situations, these solutions have to be supported by theoretical analysis showing how good the solutions are.

In this study, we propose an algorithm based on the concept of gang scheduling to solve a multiprocessor scheduling problem. Gang scheduling is a scheduling approach in which multiple threads of a given job are scheduled simultaneously on different processors. Start time and finishing time of each thread it is unavoidable that certain processors may remain idle due to this requirement. Our proposed algorithm will try to reduce these idle times by bringing some jobs to the queue to fill the gaps occurring in idle processors. This approach is known as backfilling (Feitelson et al. 2004).

Gang scheduling and backfilling technique have been used by several researchers in solving multiprocessors scheduling problems. Zhang et al. (2000) have analyzed the behavior of well-known queuing policies such as first–come-first-Served (FCFS), when backfilling is used.

.Ward et al. (2002) have discussed the difference between two basic approaches of backfilling: conservative backfilling and aggressive backfilling, Conservative backfilling allows a lower priority job to run if it will not delay the highest priority job. Siyambalapitiya and Sandirigama (2011) have proposed an improvement to FCFS approach using gang scheduling.

In this paper, we propose an improved algorithm for multiprocessor scheduling which incorporates the idea of backfilling into gang scheduling. It is a further improvement to decreasing-ascend algorithm proposed earlier (Siyambalapitiya and Sandirigama, 2010). The idea is to reduce the idle times of the processors by bringing to give an idea of the quality of solution obtained by comparing it with a lower bound for an optimal solution.

The remainder of this paper is organized as follows: in section 2, we explain the methodology of developing the proposed algorithm. The proposed decreasing backfilling algorithm is presented in section 3. Results and discussion are presented in section 4 and the conclusion is given in section 5.

2. Methodology

We assume that at any given time, we have a set of m jobs to be processed by n identical processors and m > n. It is assumed that the actual or expected processing time for each job is known in advance. First, we arrange the list of jobs in the descending order of job completion times. It is assumed that when the processing time of the jobs exceeds a certain value, the jobs is splits into two or more threads. As a result, at any given time, there could be a mixture of single treads threads jobs and multi-threads jobs to be processed. As we assume that the jobs are processed according to gang scheduling approach, each of the threads in a particular job has to be processed simultaneously using identical processors.

When we arrange the jobs in the descending order of job completion times, the jobs with the higher number of threads will be placed at the front of the queue. It is assumed that the processors are numbered in a certain order and each job requires a number of processors equal to the number of threads in that job. Then we start assigning jobs to processors according to this number to this numbering system. We also keep an array contain the earliest possible start time for the next job in each processor. Once a job is completed, we update this array of earliest possible start times.

After each assignment, we check whether the number of available processors is sufficient to schedule the next job in the queue. If the number of idle processors is less than the number of threads in the next job to be processed, we look for a job later in the queue which could be accommodated and bring this job to the front. Then this job is given priority and it is scheduled as the next job. This procedure is known as brought to the front of the queue by delaying a

higher priority job requiring more resources which are not currently available. The idea of backfilling is to reduce the idle time of the processors thereby trying to reduce the overall processing time of the jobs waiting in the queue.

Once jobs are assigned to all the processors, we arrange the possible start time for the next job for each processor in the ascending order. The next job is assigned to the processor with the earliest possible start time. This process is continued until all the jobs are processed. Then the total job completion time is the latest completion time of a job in the queue.

To determine the quality of solution obtained, we make use of a lower bound proposed by Siyambalapitiya and Sandirigama (2010 a). Let m be the number of jobs and n be the number of Processors. Let t_i be the estimated processing time for the i^{th} job and

T be the total processing For all jobs. Then T= $\sum_{i=1}^{m}$ ti. If L is the lower bound, then

L=T/n. let C be the total time elapsed for the best feasible solution for the given problem, then

P=$\left(\frac{c-L}{L}\right)100$

Where p is the lower bound. Then, we can say that the optimal solution for the given problem should lie within p% of the lower bound.

3. **Proposed Algorithm (Decreasing Backfill-Gang Algorithm)**

We state the proposed algorithm known as decreasing backfill-gang algorithm as follows:

Step 1: From a queue of jobs by arranging the list of jobs in the descending order of job durations.

Step 2: Define the maximum allowed length of a thread. Divide the jobs into threads on this basis. Compute the duration of threads for each job.

Step 3: Start assigning jobs to processors according to the position of the job in the queue. If the number of currently free processors is insufficient to schedule the next job in the queue, move to the back of the queue and look for a job that requires exactly the number of free processors. When such a job is found, stop moving and bring this job to the front of the queue and schedule it. Other jobs in the queue will be shifted to accommodate this job. If a job which fits exactly is not available, select a suitable job that comes closest to this requirement.

Step 4: For each processor, compute the earliest possible start time for the next job. Arrange start times for each processor in the ascending order. Assign jobs for processors in the ascending order of earliest start times.

Step 5: Continue assigning jobs to processors until all the jobs are scheduled. The job with the latest completion time gives the total processing time for all the jobs.

4. Results and Discussion

Several test problems have been solved by the proposed decreasing backfill-gang algorithm. These results were compares with the FCFS-gang algorithm proposed earlier (Siyambalapitiya and Sandirigama, 2011). These results are shown in table 1.

Table 1: Comparison of Performance of Different Algorithm

Problem number	Jobs	Processors	Threads length	Lower bound	FCFS-gang	Decreasing Backfill-gang algorithm
1	12	4	10	25	29.0	26.5
2	17	4	10	46	53.5	47.5
3	20	4	10	64	79.7	67.17
4	22	4	10	76	83.9	76.25
5	32	4	12	125	146.4	127.42
6	25	6	15	111	139.8	115.33
7	31	6	10	73	81.8	76.5
8	40	6	25	387	483.4	389.75
9	50	6	25	397	477.0	406.83
10	65	6	30	623	757.7	636.00
11	65	6	30	720	817.1	729.00
12	40	8	20	247	288.9	251.00
13	50	8	15	155	177.6	161.42
14	60	8	10	148	182.3	149.33
15	65	8	15	164	181.0	166.5
16	45	10	15	146	173.8	151.3
17	75	10	10	261	314.4	265.32
18	95	10	30	847	979.0	860.87
19	60	12	15	98.4	112.5	102.00
20	90	12	15	210	242.6	212.13

The percentage gap between the lower bound and the solution obtained from decreasing backfill-gang algorithm is given in table 2.

Figure 1: Flow chart for Decreasing backfill-Gang Algorithm

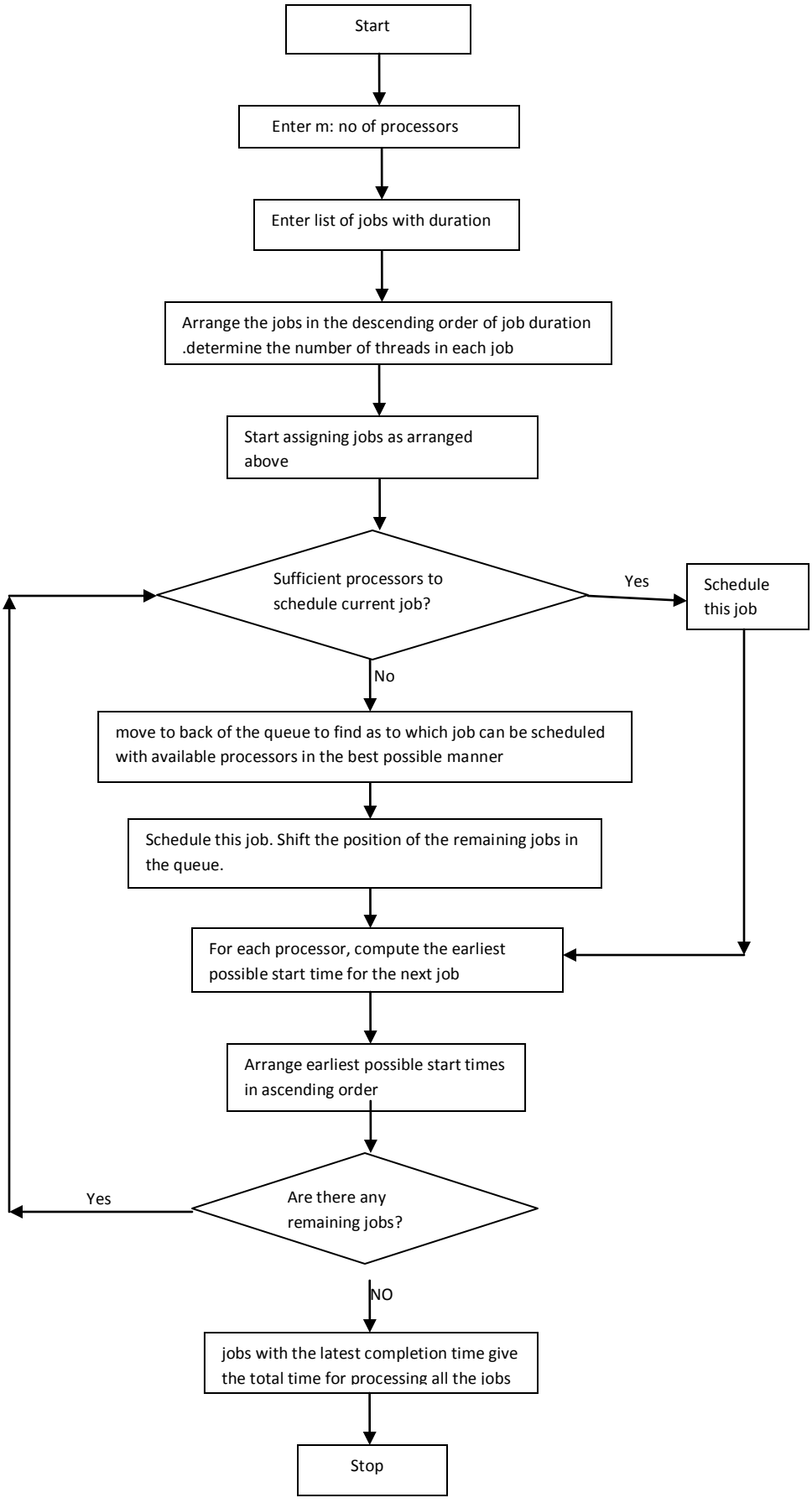

Table 2: percentage Gap between the lower bound and the decreasing backfill-Gang Algorithm

Problem	1	2	3	4	5	6	7	8	9	10
% Gap	6.00	3.26	4.95	0.33	1.94	3.90	4.79	0.71	2.48	2.09
Problem	11	12	13	14	15	16	17	18	19	20
% Gap	1.23	1.62	4.14	0.92	1.52	3.63	1.66	1.64	3.66	1.01

According to these results, we can observe that the solution to all the test problems lie within 6% from the relevant lower bound. In fact, 80% of the test problems have a percentage gap of less than 4% from the lower bound. This shows that the optimal solutions should lie within this range.

Conclusion

The above result show that the decreasing backfill-gang algorithm gives superior solution compared to FCFS-gang algorithm. It is observed that the percentage gap is extremely good, showing that the solutions obtained are extremely close to the optimal solutions. Therefore, It can be concluded that the incorporation of backfilling into gang scheduling approach allows us to obtain a promising solution method to solve multiprocessor scheduling problems.

Reference

1. Feitelson D G al. (2004), “parallel job scheduling –A status Report”, job scheduling Strategies for parallel Processing, Lecture Notes in computer Science, Vol. 3834, pp. 1-16
2. Hou E S H (1994), “A Genetic Algorithm for multi-processor Scheduling”, IEEE Transactions on parallel and Distributed system, Vol. 5, No. 2, pp. 113-120.
3. Siyambalapitiya R and Sandirigama M (2010), “A New Greedy Algorithm for multiprocessor Scheduling”, Journal on Software Engineering, Vol. 1, No. 3, pp. 7-12.
4. Siyambalapitiya R and Sandirigama M (2011), “Improvements to First-Come-First-served Multi-processor scheduling with Gang scheduling”, Journal on Software Engineering, Vol. 5, No. 3, pp. 1-7.
5. Ullaman J D (1975), “NP-Complete Scheduling Problems”, Journal of Computer and system Science, Vol. 10, No.3, pp. 384-393.
6. Ward W A et al. (2002), “Scheduling jobs on parallel Systems Using a relaxed Backfill Strategy”, Job Scheduling Strategies for parallel Processing, Lecture Notes in Computer Science, Vol. 2537, pp. 88-102.
7. Zhang Y et al. (2000),”Improving parallel Job Scheduling by Combining Gang Scheduling and Backfilling Techniques”, 14th International Parallel and Distributed Processing Symposium, Mexico, pp. 133-142.