A Critic on Real-Time Scheduling Strategies

Manupriya Hasija, TIT&S, Bhiwani, Haryana, India
Akhil Kaushik TIT&S, Bhiwani, Haryana, India
Parveen Kumar BITS, Pilani, Rajasthan

Abstract— **The present scenario of the computing era is dominated by the real-time systems. The real-time systems are gaining popularity in the field of job scheduling as both uni-processor and multi-processor architectures to satisfy the requirements of time-constrained applications like flight controls, avionics, multimedia data streaming, etc. The application areas are also growing by huge amounts due to increasing human dependency on machines. Hence to balance between ever-increasing needs and exponentially-rising overheads; the necessity of new approaches, which can ensure the best results not only limited to reliable logical results but also acceptable temporal results too, is expected to increase. This paper reviews various existing scheduling algorithms suggesting further dissimilar techniques and endeavors to compare and suggest which approach best suits which scenarios based on their primary pluses.**

Keywords- ***Scheduling algorithms, FCFS, SJF, EDF, G-EDF, Backfilling, Conservative Backfilling, EASY, Best-Gap*****.**

Introduction

Today, real-time embedded systems find applications in many diverse areas, including automotive electronics, avionics, telecommunications, space systems, medical imaging, and consumer electronics. Real-time systems are driven by a profit motive and they are in huge demand due to rapid technological developments in mostly applications all around the world. A real-time system as defined as an information processing system which has to respond to externally generated input stimuli within a finite amount of time with the maximum accuracy. The correctness depends not only on the logical result but also on temporal accuracy to the same extent; the failure to respond in time is as bad as the wrong response [1]. For example in avionics, flight control software must execute within a fixed time interval in order to accurately control the aircraft. In automotive electronics there are tight time constraints on engine management and transmission control systems that derive from the mechanical systems that they control.

Thus for the sake of best results, the point under consideration especially for avoiding deadline misses is efficient scheduling. Real-time scheduling has its origins in early 1960s. Scheduling is of huge importance because it largely influences the resource utilization and overall performance of the system

Scheduling algorithms can be classified as follows.

a. *Preemptive*: A scheduling algorithm is *preemptive* if the release of a new job of a higher priority task can preempt the job of a currently running lower priority task. For example, RM and EDF.
b. *Non-preemptive*: In *non-preemptive* scheme, a currently executing task always completes its execution before another active task starts execution.
c. *Cooperative*: Tasks may only be preempted at defined scheduling points within their execution. Effectively, execution of a task consists of a series of non-preemptable sections.

A preemptive scheduling algorithm can succeed in meeting deadlines where a non-preemptive scheduling algorithm fails but a non-preemptive scheduling algorithm has naturally the advantage of no run-time overhead caused by preemptions.

The focus of this paper will only be on real- time scheduling algorithms and their merits and demerits. The proposed work here reviews few algorithms and uncovers further scopes of work if present.

I. EDF (EARLIEST DEADLINE FIRST)

The EDF scheduling algorithm is a priority- based algorithm in which the job with the earlier deadline is assigned the higher priority, and a higher priority request always preempts the lower priority one. This scheduling algorithm is an example of priority- based algorithms with *dynamic priority* assignment in the sense that the priority of the request is not fixed at the start instead is assigned at the execution time and vary with the arrival of new jobs and completion of the jobs already in queue. EDF is also referred to as *Deadline monotonic* scheduling algorithms sometimes. For a set of preemptive tasks, EDF will surely find a schedule if it exists [2]. But EDF for non-preemptive tasks is not yet studied widely. However, EDF algorithm for non- preemptive periodic and aperiodic tasks have produced nearly optimal results when the system is lightly loaded, but for overloaded systems EDF shows poor performance.

Later in the decade, many researchers showed interest and their efforts lead to the development of various other scheduling algorithms either or entirely new concepts of the extensions or modifications of EDF. A few of such algorithms are reviewed in this work.

II. G-EDF (GROUP-EDF):

G-EDF is a variation of EDF that was proposed by Wenming Li [3]. G-EDF groups together the tasks with similar deadlines (i.e. deadlines that are very close to each other) and the SJF (Shortest Job First) algorithm is used to schedule the task within a group. G-EDF showed a good improvement in the success ratio (number of tasks that

have been successfully scheduled to meet their deadlines) especially in overloaded conditions. It gives good results for lightly loaded conditions also and has the computational complexity approximately same as that for EDF.

III. FCFS (FIRST COME FIRST SERVE)

FCFS [4] also referred as FIFO (First In First Out) algorithm comes under the category of Queuing algorithm and is the most basic algorithm under this category. In FIFO queuing, all tasks are treated equally and are placed in the queue depending upon the submission (arrival) time (order in which the tasks arrived). FCFS is very easy to implement and places extremely low computational load on the system as compared to other scheduling algorithms. Similar to EDF, FCFS shows impressive performances as long as the Queue is smaller in size, but with increase in size of queue, the results deteriorate quickly. Hence FCFS is not an attractive alternative on its own, but if combined with other schemes, it can give an unconventional scheduling algorithm with fabulous results.

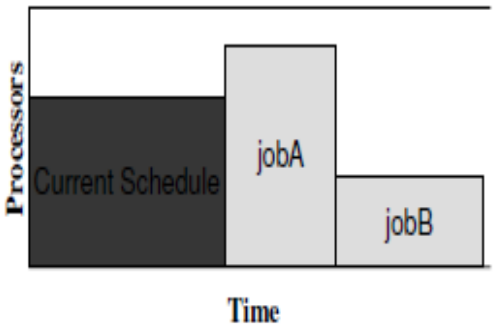

Fig.1. An example of a simple FCFS schedule

IV. SJF (SHORTEST JOB FIRST)

SJF is one of the most widely used scheduling algorithms in real-time systems to avoid deadline misses. The selection function for SJF [16] algorithm is the task with the shortest expected execution time. SJF is a non-preemptive scheduling priority- based algorithm where the task with shortest deadline is assigned the highest priority. Even though SJF is widely used, it is not preferred by real-time systems due to lack of preemption, which makes it unsuitable especially for time sharing environments. Similar to EDF and FCFS, SJF shows effective performances only in lightly loaded conditions, and the success rate and/or the schedulability shows a steep downfall when the number of tasks increases. Furthermore, one more drawback makes SJF unsuitable for real-time systems, *Starvation* (i.e. longer tasks may have to wait for quite long period for their turn).Hence, SJF focuses on queuing based on their period, irrespective of the priority or arrival time of the tasks.

V. BACKFILLING

Backfilling [5, 6] was originally introduced to extend the FCFS approach to improve the resource utilization and has been implemented in most production schedulers. Backfilling allows a task that is lower in the queue (priority based) to start before the task that is at the head of the queue. Backfilling identifies the "gaps" in the 2D chart and moves forward the jobs that have future "internal" reservations. By this, the CPU idle time can be reduced by many folds, if tasks that can fit into the gaps that exist in the queue. It also improves the average turnaround time.

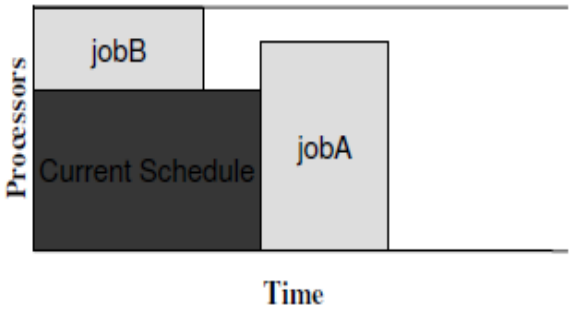

Fig.2. An example of backfill in a FCFS backfilling schedule

Figure 2 shows a similar situation to Figure 1, except job B is now allowed to start due to backfilling. A backfilling scheduler allows internal reservations for some of the tasks and these reservations as well as blocked off time for tasks, provide a schedule in which jobs can backfill. Backfilling allows a job to fit into the hole only till it doesn't violate any other reservations. Backfilling approach proved to be very efficient, it motivated the development of variation of backfilling like Conservative backfilling, aggressive backfilling etc.

VI. CONSERVATIVE BACKFILLING

Conservative backfilling [8] is generally used when predictability is required. It makes reservations for all queued tasks instead of doing it only for the first one. This version of backfilling focuses on eliminating the risk of starvation, thus proceeding subject to checking that it does not delay any previous job in the queue. The number of reservations that can be made can be set using some parameters managed by the system administrator in Maui scheduler [7]. This strategy improves the system utilization by allowing the jobs requiring a few available resources to overtake the jobs requiring more resources or the ones requiring them for long time.

VII. EASY (EXTENSIBLE ARGONNE SCHEDULING SYSTEM)

EASY is a popular variant of backfilling developed for the IBM SP2 supercomputer [9]. EASY implements an aggressive version of backfilling. In this approach, any job can be used to backfill provided it does not delay the first job in the queue. Since the queuing delay for the job

at the head of the queue depends only on jobs that are already running, and these jobs will eventually either terminate or be terminated when they exceed their estimated runtime, starvation is eliminated. But there arises an issue, that jobs other than first may be repeatedly delayed by newly arriving jobs that skip them in the queue, which reduces predictability. Mu'alem and Feitelson [10] compared Conservative backfilling with the EASY approach and for most cases EASY backfilling showed better performance than the conservative approach.

VIII. VARIANTS OF BACKFILLING:

a. *Dynamic backfilling:* The scheduler can overrule a previous reservation if introducing a slight delay will improve utilization considerably [11].

b. *Slack Based Backfilling:* In this variation of backfilling priorities are supported [12]. Slack is assigned to every waiting task according to the priorities and this slack determines how long the task will have to wait for being executed. Slack time of higher priority tasks is little.

c. *Selective Backfilling:* This approach is similar to Slack Based Backfilling, where slack is set to a value that limits the slowdown to a desired threshold, and this slowdown value is used to select the jobs for reservation. Usually, the jobs whose expected slowdown exceeds the threshold are provided the reservations [13].

d. *Multiple-Queue Backfilling:* Lawson and Smimi [14] proposed that each task is assigned to a queue according to its expected execution time and each queue is assigned to a disjoint partition of the parallel system on which only jobs from this queue can be executed.

IX. BEST GAP (BG)

BG [15] is the policy similar to conservative backfilling. The first suitable gap is searched for, at the arrival of each task. As, conservative backfilling chooses the first gap if more than one gaps are available on different clusters, BG chooses the best one. The best gap is decided based on the evaluation done. The newly arriving job is added into the existing schedule after finding the suitable cluster and proper time slot for the job. A cluster is considered suitable if it has enough CPUs to execute the task. If two or more suitable gaps are found, tie is broken by the earliest gap. After the task is added to the gap decided as the best, a best possible schedule is formed. Best Gap scheduling algorithm proved to give very effective results and hence was employed in a new approach BG- Earliest Deadline First [15].

X. CONCLUSION

Various existing scheduling algorithms improve different CPU scheduling factors like Turnaround time, waiting time, starvation problem, resource utilization, CPU idle time and many others. Many techniques emphasize on priorities, other on fair share while others on resource utilization. Despite the fact that there are various point of views regarding the efficiency and better performances of these algorithms, a balance is to be maintained between the overheads introduced and performance gains. Hence a few algorithms are developed keeping all these factors in mind at the same time. But, there is still a scope for further improvement. We can improve the efficiency by combining two strong algorithms having their own region of perfection. This mix and match strategy can be applied both in random fashion or a pros and cons focused manner. In our future work we will propose a new algorithm combining the concept of multi-level queue scheduling and one of the simpler algorithms in this field keeping in mind the complexity issues and the desirable performance metrics.

ACKNOWLEDGMENT

The authors would like to thank Dr. Mukesh Kumar and Er. Deepak Singla for their extensive help and constant discussions.

REFERENCES

[1] Alan Burns and Andy Wellings. Real-Time Systems and ProgrammingLanguages. Addison WesleyLongmain, April 2009.
[2] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", Journal of the ACM, Vol. 20. No. 1, pp. 46-61.
[3] Wenming Li, "Group-EDF- A New Approach and an Efficient Non-Preemptive Algorithm for Soft Real-Time Systems, in 2006
[4] Silberschatz ,Galvin and Gagne, Operating systems concepts,8th edition, Wiley, 2009
[5] David Lifka. The ANL/IBM SP scheduling system. In JSSPP. 1995
[6] A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with back_lling. *IEEE TPDS*, 12(6):529.543, 2001.
[7] D. Jackson, Q. Snell, and M. Clement, "Core algorithms of the Maui scheduler". In *Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, Lect. Notes Comput. Sci. Vol. 2221, pp 87–102, 2001.
[8] D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Parallel job scheduling - a status report, June 2004
[9] D. Lifka, "The ANL/IBMSP scheduling system". In Job Scheduling Strategies for Parallel Processing, pp. 295-303, Springer-Verlag, 1995. Lect. Notes Comput. Sci. Vol. 949.
[10] A.W.Mu'alem and D. G. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling". *IEEE Trans. on Parallel and Distributed Syst.* 12(6), pp. 529-543, Jun 2001
[11] J. P. Jones and B.Nitzberg, "Scheduling for parallel supercomputing: a historical perspective of achievable utilization". In Job Scheduling Strategies for Parallel Processing, pp. 1-16, Springer-Verlag, 1999. Lect. Notes Comput. Sci. Vol. 1659.
[12] D. Talby and D. G. Feitelson, "Supporting priorities and improving utilization of the IBMSP scheduler using slack-based backfilling". In 13th *Intl. Parallel Processing Symp.*, pp. 513-517, Apr 1999
[13] W. A. Ward, Jr., C. L. Mahood, and J. E. West, "Scheduling jobs on parallel systems using a relaxed backfill strategy". In Job Scheduling Strategies for Parallel Processing, Springer-Verlag, Lect. Notes Comput. Sci. Vol. 2537, pp. 88-102, 2002.
[14] B. G. Lawson and E. Smirni, "Multiple-queue backfilling scheduling with priorities and reservations for parallel systems". In Job Scheduling Strategies for Parallel Processing, Springer-Verlag, Lect. Notes Comput. Sci. Vol. 2537, pp. 72-87, 2002.
[15] Dalibor Klus_a_cek and Hana Rudov_a. E_cient Grid scheduling through the incremental schedule-based approach. Computational Intelligence: An International Journal, 27(1):4{22, 2011.
[16] Lingyun Yang, Jennifer M. Schopf and Ian Foster, "Conservative Scheduling: Using predictive variance to improve scheduling decisions in Dynamic Environments", SuperComputing 2003, November 15-21, Phoenix, AZ, USA.