Real-Time Disk Scheduling using Preemptive and Non-Preemptive Approach: A Survey

SALIM Y. AMDANI, SOHEL A. BHURA, ANKITA A. MAHAMUNE

***Abstract*— Real-Time System (RTS) is defined as a system in which the time where the outputs are produced is significant. In general, data in a real-time system is managed on individual basis by every task within the system. However, with the advancement of technology, many applications are requiring large amounts of information to be handled and managed in a timely manner. Therefore, in various application domains, data can no longer be treated and managed on individual basis, rather it is becoming a vital resource requiring an efficient data management mechanism. In an attempt to achieve the advantages of both systems, real-time and database, continuous efforts are directed towards the integration of the two technologies. Such an integration of the two technologies resulted in combined systems known as *Real-Time Database Systems* [16]. Many of these database systems are disk-resident so disk accesses often dominate the execution time of a real-time transaction. An effective disk scheduling algorithm is thus very crucial for the system to attain a high performance. Since the invention of movable head disks, several algorithms have been developed to improve the disk I/O performance using intelligent scheduling of disk accesses. Traditionally, disk IOs have been thought of as nonpreemptible operations. However, nonpreemptible IOs can be a stumbling block when designing applications requiring short, interactive responses. One such domain is that of real-time disk scheduling [12]. Blocking is undesirable since it degrades the schedulability of real-time tasks. Making disk IOs preemptible would reduce blocking and improve the schedulability of real-time disk IOs. This paper gives the survey of existing preemptive approaches that can be used for real-time disk scheduling.**

***Index Terms*— Deadline, Preemption, Real-time disk scheduling, Scheduling algorithms**

I. INTRODUCTION

The term **real-time system** has been used extensively in many applications of computing and control systems. A Real-Time System (RTS) is defined as a system in which the time where the outputs are produced is significant. The outputs must be produced within specified time bounds referred to as deadlines. The correctness of a RTS depends not only on the logical results produced, but also on the times at which such results were produced.

In general, data in a real-time system is managed on individual basis by every task within the system. However, with the advancement of technology, many applications are requiring large amounts of information to be handled and managed in a timely manner. Thus, a substantial number of real-time applications are becoming more data-intensive. Such lager amounts of information had produced an interdependency relationship among real-time applications.

Therefore, in various application domains, data can no longer be treated and managed on individual basis, rather it is becoming a vital resource requiring an efficient **data management** mechanism. Meanwhile, database management systems are designed around such a concept; that is, with the sole goal of managing data as a resource. Hence, the principles and techniques of transaction management in Database Management Systems need to be applied to real-time applications for efficient storage and manipulation of information [2].

In an attempt to achieve the advantages of both systems, real-time and database, continuous efforts are directed towards the integration of the two technologies. Such an integration of the two technologies resulted in combined systems known as ***Real-Time Database Systems*** [16].

Many real-time applications handle large amounts of data and require the support of a real-time database system. Examples include telephone switching, radar tracking, media servers and computer-aided manufacturing. Many of these database systems are disk-resident because the amount of data they store is too large (and is too expensive) to be stored in nonvolatile main memory. In these applications, **disk accesses** often dominate the execution time of a real-time transaction. An **effective disk scheduling algorithm** is thus very crucial for the system to attain a high performance. Comparing with CPU scheduling, disk scheduling is even more difficult. The main reason is that disk seek time, which accounts for a very significant fraction of disk access latency, depends on the disk head movement. Hence, the servicing order of disk requests and their service times exhibit an intricate dependency.

In real-time system, the most important objective is to satisfy the timing constraints (deadlines) of the transactions that issue the disk I/O requests. Many real-time disk scheduling algorithms are proposed for servicing such transactions. The **existing conventional algorithms** are FCFS, SSTF, SCAN, CSCAN, LOOK, CLOOK, etc. These algorithms perform better while guaranteeing optimized throughput but they do not consider the request's deadline. So, they cannot be applied to the real time applications [20].

[3]On the other hand **the real-time scheduling algorithms** like EDF, SCAN-EDF, SSEDO, SSEDV,P-SCAN,D-SCAN,FD-SCAN,DM-SCAN,RG-SCA

Manuscript received December 20, 2013.

Prof. S. Y. Amdani, Assoc. Professor and Head, Dept. of CSE B. N. C. O. E, Pusad (India) 9764996786
Prof. S. A. Bhura, Assoc. Professor Dept. of CSE B. N. C. O. E, Pusad (India) 9764996766
Miss. Ankita A. Mahamune M.E. Student, Dept. of CSE, B. N.C. O. E, Pusad (India) 9552280398

N,G-EDF, GSR, etc. schedule the requests while guaranteeing their respective deadlines. So, they can be used in real-time applications.

Some of these existing techniques may not have considered the preemption into consideration. However, nonpreemptible IOs can be a stumbling block when designing applications requiring short, interactive responses. Blocking is undesirable since it degrades the schedulability of real-time tasks. Making disk IOs preemptible would reduce blocking and improve the schedulability of real-time disk IOs. This paper gives the survey of existing preemptive approaches that can be used for real-time disk scheduling.

II. BACKGROUND

A. *Real-Time Database System*

A real-time database is a database system which uses real-time processing to handle workloads whose state is constantly changing. This differs from traditional databases containing persistent data, mostly unaffected by time. For example, a stock market changes very rapidly and is dynamic. The graphs of the different markets appear to be very unstable and yet a database has to keep track of current values for all of the markets. Real-time processing means that a transaction is processed fast enough for the result to come back and be acted on right away. Real-time databases are useful for accounting, banking, law, medical records, multi-media, process control, reservation systems, and scientific data analysis.

B. *Disk Scheduling and Disk Scheduling Problem*

The processor is much faster than the disk, so it's highly likely that there will be multiple outstanding disk requests before the disk is ready to handle them. Because disks have non-uniform access times, re-ordering disk requests can greatly reduce the latency of disk requests. The disk scheduling problem involves reordering the disk requests in the disk queue so that the disk requests will be serviced with the minimum mechanical motion by employing seek optimization and latency optimization.

C. *Real-Time Scheduling*

A hard real-time system must execute a set of concurrent real-time tasks in a such a way that all time-critical tasks meet their specified deadlines. Every task needs computational and data resources to complete the job. The scheduling problem is concerned with the allocation of the resources to satisfy the timing constraints. Figure 1 given below represents a taxonomy of real-time scheduling algorithms.

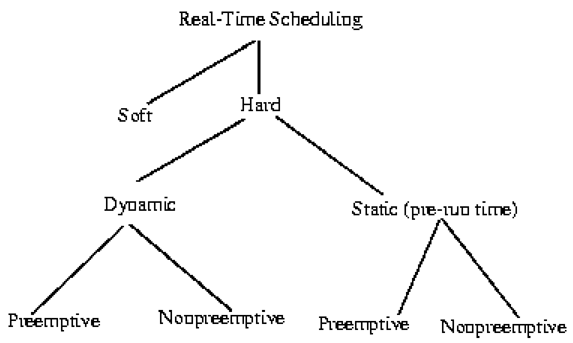

Fig.1 A taxonomy of real-time scheduling algorithms

Real-time database systems (RTDBS) have timing constraints imposed on their transactional activities. [14].Although significant research efforts have been made in the real-time database area in last several years, they are mainly focused on scheduling transactions in soft or firm real-time applications. As their applications do not require stringent timing constraints, most of their protocols adopt the best-effort approach to scheduling transactions without a guarantee of meeting transaction's deadlines. On the other hand, not much work has pursued to guarantee stringent timing constraints of transactions in hard real-time applications, i.e., their deadlines must be met. Guaranteeing all hard deadlines of transactions is one of the most important issues in hard RTDBS.

III. OVERVIEW OF EXISTING ALGORITHMS

Since the invention of movable head disks, several algorithms have been developed to improve the disk I/O performance using intelligent scheduling of disk accesses. These algorithms can be broadly divided into two classes [20]:

A. *Conventional Scheduling algorithms*

-Disk scheduling algorithms optimized to service best-effort requests:
The simplest among these is the FCFS (First Come First Serve) algorithm that schedules the disk requests in the order of their arrival time. Thus, FCFS can incur significant seek time and rotational latency overhead, since the schedule that will be derived using FCFS will be independent of the relative positions of the requested data on the disks. This limitation has been addressed by several disk scheduling algorithms, such as shortest seek time first (SSTF), SCAN, LOOK, etc. that schedules the requests to minimize the seek time and rotational latency.

B. *Real Time Scheduling algorithms*

- Disk scheduling algorithms optimized to service requests with real-time deadlines:

Disk scheduling has been studied since 1960's. However, the conventional disk scheduling algorithms aim to optimize the disk throughput. To meet time constraints,some conventional real-time algorithms. The simplest of these algorithms is EDF (Earliest Deadline First). EDF schedules the requests in the order of their deadlines but ignores the relative positions of the requested data on disk. This incurs significant seek time and rotational latency overhead, thereby throughput is relatively low. To keep a good tradeoff between optimizing throughput and meeting time constraints, several hybrid real-time scheduling algorithms were proposed including PSCAN (Priority SCAN), DSCAN (Earliest Deadline SCAN), FDSCAN (Feasible Deadline SCAN), SCAN-EDF, Shortest Seek Earliest Deadline by Order/Value (SSEDO/SSEDV), etc. These algorithms start from an EDF schedule and reorder the requests so as to reduce the seek time and rotational latency overhead without violating the request deadlines.

C. *Necessity of Preemptive Scheduling*

Traditionally, disk IOs have been thought of as nonpreemptible operations. Once initiated, they cannot be stopped until completed. However, nonpreemptible IOs can be a stumbling block when designing applications requiring short, interactive responses. Different mechanisms have been presented that can be used to enable IO preemption at the disk-firmware level. The firmware-based implementation would provide stronger real-time guarantees for higher-priority requests compared to our software-based prototype. In addition to high-throughput, short response time is desirable and even required in certain application domains. One such domain is that of real-time disk scheduling[12].In real-time scheduling theory, blocking, or priority inversion, is defined as the time spent when a higher-priority task is prevented from running due to the nonpreemptibility of a low-priority task (in this paper, we refer to blocking as the waiting time). Blocking is undesirable since it degrades the schedulability of real-time tasks. Making disk IOs preemptible would reduce blocking and improve the schedulability of real-time disk IOs**.**

D. Approaches related to Preemptive Scheduling

[21]Preemptive scheduling problems are those in which the processing of a job can be temporarily interrupted, and is restarted at a later time [17] although a lot of researchers studied preemptive scheduling, but a few of them considered it in the context of JIT-scheduling problems [10]. However, preemption seems to be disregarded in JIT-scheduling problems with due date assignment.

In Preemptive RAID Scheduling [11] investigated the effectiveness of preemptive disk-scheduling algorithms to achieve better quality of service (QoS) in RAID systems. It showed when and how to preempt IOs to improve the overall performance of the RAID system.

In 2012, SukumarBabu Bandarupalli1, NeelimaPriyankaNutulapati and Prof. Dr. P. Suresh Varmaintroduced a new CPU algorithm called "A Novel CPU Scheduling Algorithm–Preemptive & Non-Preemptive" in Dec 2012. A Novel CPU Scheduling Algorithm acts as both preemptive and non-preemptive in nature based on the arrival time[5].

This **Novel CPU Scheduling algorithm** is both preemptive and non-preemptive in nature. In this algorithm a new factor called condition factor (F) is calculated by the addition of burst time and arrival time i.e., **F = Burst Time + Arrival Time.** This factor F is assigned to each process and on the basis of this factor processes are arranged in ascending order in the ready queue. Process having shortest condition factor (F) are executed first and process with next shortest factor (F) value is executed next. By considering the arrival time the new algorithms acts as preemptive or non-preemptive. Proposed CPU scheduling algorithm reduces waiting time, turnaround time and response time and also increases CPU utilizationn and throughput.

The working procedure of *A Novel CPU Scheduling Algorithm: Preemptive and Non Preemptive* is as given below:

1. Take the list of processes, their burst time and arrival time.
2. Find the condition factor F by adding arrival time and burst time of processes.
3. First arrange the processes, burst time, condition factor based on arrival time ascending order.
4. Iterate step a, b until burst time becomes zero.
a. If arrival time of first and second process are equal the arrange them based on their condition factor f.
b. Decrement the burst time and increment arrival time by 1
5. When burst time becomes zero find the waiting time and turnaround time of that process.
6. Average waiting time is calculated by dividing total waiting time with total number of processes.
7. Average turnaround time is calculated by dividing total turnaround time by total number of processes.

The question whether **preemptive algorithms are better than nonpreemptive** ones for scheduling a set of real-time tasks has been debated for a long time in the research community. In fact, especially under fixed priority systems, each approach has advantages and disadvantages. Recently, limited preemption models have been proposed as a viable alternative between the two extreme cases of fully preemptive and nonpreemptive scheduling [8].

Often, preemption is considered a prerequisite to meet timing requirement in real-time system design; however, in most cases, a fully preemptive scheduler produces many unnecessary preemptions. Arbitrary preemptions can introduce a significant runtime overhead and may cause high fluctuations in task execution times, so degrading system predictability.

In particular, at least four different types of costs need to be taken into account at each preemption. ***Scheduling cost*** (for inserting the running task into the ready queue, switch the context, and dispatch the new incoming task); ***Pipeline cost***(for flushing the processor pipeline when the task is interrupted and refilling it when the task is resumed); ***Cache-related cost***(for reloading the cache lines evicted by the preempting task); ***Bus-related cost***(due to the extra bus interference for accessing the RAM because of the additional cache misses caused by preemption).

To reduce the runtime overhead due to preemptions and still preserve the schedulability of the task set, **the following approaches have been proposed in the literature.**

• ***Preemption Thresholds Scheduling (PTS):*** This approach, proposed by Wang and Saksena[22], allows a task to disable preemption up to a specified priority level, called *preemptionthreshold*. Thus, each task is assigned a regular priority and a preemption threshold, and the preemption is allowed to take place only when the priority of the arriving task is higher than the threshold of the running task.
• ***Deferred Preemptions Scheduling (DPS):*** According to this method, first introduced by Baruah[6] under Earliest Deadline First (EDF), each task Ti specifies the longest interval q_i that can be executed nonpreemptively. Preemption is **postponed** for a given amount of time, rather than moved to a specific position in the code.
• ***Fixed Preemption Points (FPP)*:**According to this approach, proposed by Burns [7], a task implicitly executes in nonpreemptive mode and preemption is allowed only at predefined locations inside the task code, called *preemptionpoints*.

EXAMPLE USED:
To better appreciate the importance of limited preemptive scheduling and to better understand the difference among the

limited preemptive approaches they have used the following Table I that reports a sample task set as a common example:

I: PARAMETERS OF A SAMPLE TASK SET

	C_i	T_i	D_i
τ_1	1	6	4
τ_2	3	10	9
τ_3	6	18	12

1] FULLY PREEMPTIVE MODE

Following Fig. 2 illustrates the schedule produced by Deadline Monotonic in fully preemptive mode.

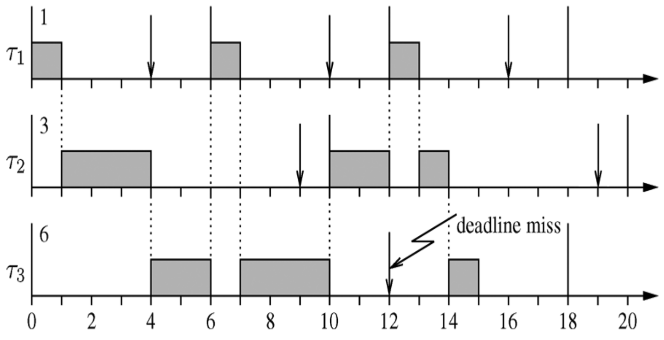

Fig. 2. Schedule produced by Deadline Monotonic (in fully preemptive mode) on the task set of Table I.

As clear from the figure, the task set is not feasible, since task T3 misses its deadline.

2] NONPREEMPTIVE MODE

Fig. 3 illustrates the schedule generated by Deadline Monotonic on the task set of Table I when preemptions are disabled.

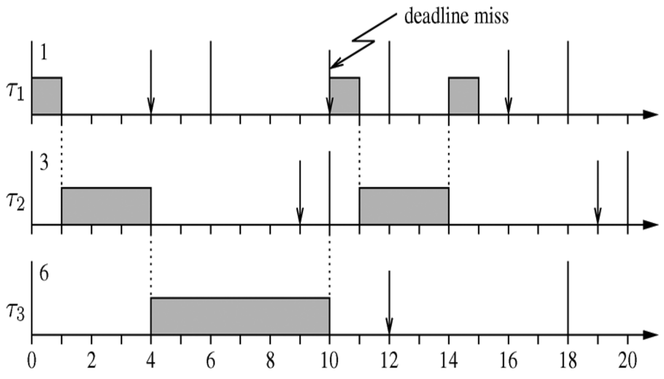

Fig. 3. Schedule produced by nonpreemptive deadline monotonic on the taskset of Table I.

3] PREEMPTION THRESHOLDS SCHEDULING (PTS)

According to this model, proposed by Wang and Saksena [15], each task is assigned a nominal priority Pi (used to enqueue the task into the ready queue and to preempt) and a *preemption threshold Ɵi>=Pi* (used for task execution). Then Ti can be preempted by Th only if Ph>*Ɵi*. At the activation time $r_{i,k}$, the priority of Ti is set to its nominal value Pi so it can preempt all the tasks Tj with threshold *Ɵj<Pi* The nominal priority is maintained as long as the task is kept in the ready queue. During this interval, Ti can be delayed by all tasks Th with priority Ph>Pi and by at most one lower priority task Tl with threshold *Ɵl>=Pi*. When all such tasks complete (at time $s_{i,k}$), Ti is dispatched for execution and its priority is raised at its threshold level *Ɵi* until the task terminates (at time $f_{i,k}$) During this interval, Ti can be preempted by all tasks Th with priority Ph>*Ɵi*. Notice that, when Ti is preempted, its priority is kept to its threshold level.

For example, if priorities are assigned as P1=3,P2=2 and P3=1,and thresholds as *Ɵ1=3*, *Ɵ2=3* and *Ɵ3=2*, the task set of Table I results to be schedulable, and the schedule produced in the synchronous periodic arrival pattern is illustrated in Fig. 4.

Notice that, at t=6, T1 can preempt T3 since P1>*Ɵ3*. However, at t=10, T2 cannot preempt T3 , being P2= *Ɵ3*. Similarly, at t=12 ,T1 cannot preempt T2, being P1= *Ɵ2*.

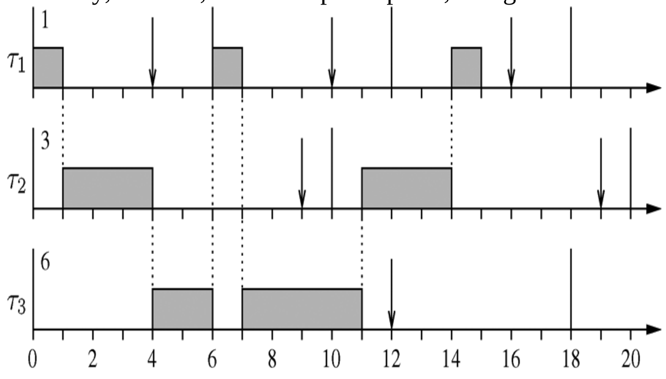

Fig. 4. Schedule produced by Deadline Monotonic on the task set in Table I with priorities P1=3, P2=2 and P3=1, and thresholds *Ɵ1=3*, *Ɵ2=3* and *Ɵ3=2*.

The example illustrated in Fig. 4 shows that a task set unfeasible under both preemptive and nonpreemptive scheduling can be feasible under preemption thresholds, for a suitable setting of threshold levels.

4] DEFERRED PREEMPTIONS SCHEDULING (DPS)

According to this method, each task Ti defines a maximum interval of time qi in which it can execute nonpreemptively. Preemption is enabled by a timer interrupt after exactly qi units (unless the task completes earlier). For example, considering the same task set of Table I, assigning q2=2 and q3=1, the schedule produced by Deadline Monotonic with deferred preemptions under the activation-triggered model is feasible, as illustrated in Fig. 5.

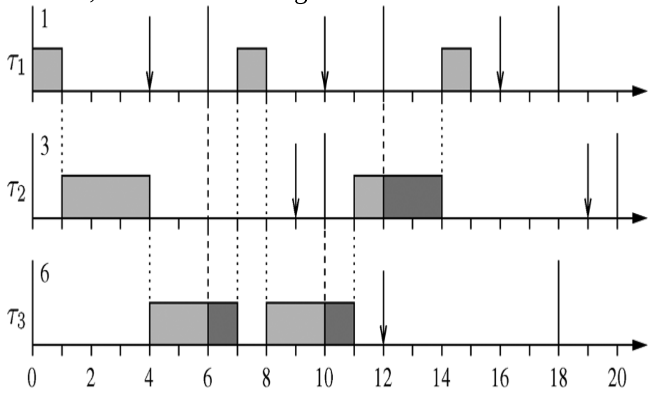

Fig. 5. Schedule produced by deadline monotonic with deferred preemptions for the task set reported in Table I, with q2=2 and q3=1

Dark regions represent nonpreemptive interval triggered by the arrival of higher priority tasks.

5] FIXED PREEMPTION POINTS (FPP)

According to this model, each task is split into m_i nonpreemptive chunks (subjobs), obtained by inserting mi-1preemption points in the code. Thus, preemptions can only occur at the subjobs boundaries. All the jobs generated by one task have the same subjob division.

For example, consider the same task set of Table I, and suppose That T2 is split into two subjobs of 2 and 1 unit, and T3 is split into two subjobs of 4 and 2 units. The schedule produced by Deadline Monotonic with such a splitting is feasible and it is illustrated in Fig. 6.

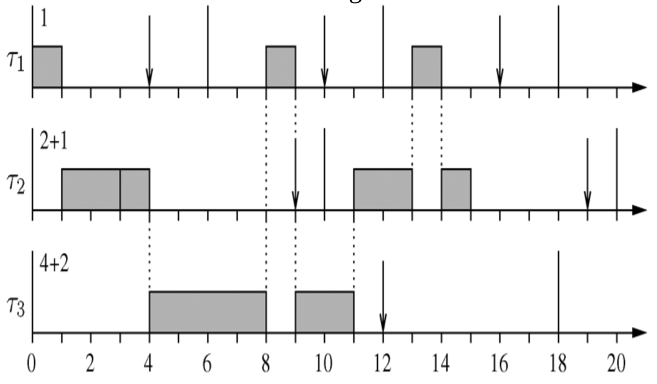

Fig. 6. Schedule produced by Deadline Monotonic for the task set reported in Table II, when T2 is split into two subjobs of 2 and 1 unit, and T3 is split into two subjobs of 4 and 2 units, respectively.

Thus [8] have presented a survey of limited preemptive schedulingalgorithms, as methods for increasing the predictability and efficiency of real-time systems. The most relevant result that clearly emerges from the experiments is that, under fixed priority scheduling, any of the considered algorithms dominates both fully preemptive and nonpreemptive scheduling, even when preemption cost is neglected.

IV. Conclusion

In real-time system, the most important objective is to satisfy the timing constraints (deadlines) of the transactions that issue the disk I/O requests. Many real-time disk scheduling algorithms are proposed for servicing such transactions.

Some of these existing techniques may not have considered the preemption into consideration. However, nonpreemptible IOs can be a stumbling block when designing applications requiring short, interactive responses. Blocking is undesirable since it degrades the schedulability of real-time tasks. Making disk IOs preemptible would reduce blocking and improve the schedulability of real-time disk IOs. Often, preemption is considered a prerequisite to meet timing requirement in real-time system design; however, in most cases, a fully preemptive scheduler produces many unnecessary preemptions. To reduce the runtime overhead due to preemptions and still preserve the schedulability of the task set different preemptive scheduling approaches have been proposed. This paper gives the survey of existing preemptive approaches that can be used for real-time disk scheduling.

References

[1] Abbott, Robert and Hector Garcia-Molina,"Scheduling Real-Time Transactions: a Performance Evaluation," *Proceedings of the 14th VLDB Conference, pp. 1-12, 1988.*

[2] Saud A. Aldarmi, "Real-Time Database Systems:Concepts and Design", *Department of Computer Science,The University of York,April 1998.*

[3] S.Y. Amdani, M.S. Ali, "An Overview of Real-Time Disk Scheduling *Algorithms", International Journal on Emerging Technologies* **2***(1): 126-130(2011).*

[4] S.Y.Amdani, M.S.Ali and S.M.Mundada, " Mathematical Model for Real Time Disk Scheduling Problem",*Proceedings published in International Journal of Computer Applications® (IJCA),2012.*

[5] SukumarBabu Bandarupalli, NeelimaPriyankaNutulapati and Prof. Dr. P.SureshVarma," A Novel CPU Scheduling Algorithm–Preemptive & Non-Preemptive", *International Journal of Modern Engineering Research (IJMER), Vol.2, Issue.6, Nov- pp-4484-4490, Dec. 2012.*

[6] S. Baruah, "The limited-preemption uniprocessor scheduling of sporadic task systems," in *Proc. 17th Euromicro Conf. Real-Time Syst. (ECRTS'05)*, Palma de Mallorca, Balearic Islands, Spain, Jul. 6–8, 2005, pp. 137–144.

[7] A. Burns, S. Son, Ed., "Preemptive priority based scheduling. An appropriate engineering approach," *Adv. Real-Time Syst.*, pp. 225–248,1994.

[8] Giorgio C. Buttazzo, Marko Bertogna and Gang Yao," Limited Preemptive Scheduling for Real-Time Systems. A Survey", *IEEE transactions on Industrial Informatics, vol. 9, no. 1, Feb 2013.*

[9] M. J. Carey, R. Jauhari and M. Livny, "Priority in DBMS Resource Scheduling", *in Proceedings of the Fifteenth International Conference on Very Large Data Bases*, Amsterdam, Netherland, pp. 397-410, August 1989.

[10] T.C.EdwinCheng,Y.MikhailKovalyov, "Batch scheduling and common due-date assignment on a single machine", *Discrete Applied Mathematics 70(1996)231–245,1996.*

[11] ZoranDimitrijevic,RajuRangaswami and Edward Chang, "Preemptive RAID Scheduling"

[12] ZoranDimitrijevic, RajuRangaswami and Edward Y. Chang, "Systems Support for Preemptive Disk Scheduling", *IEEE TRANSACTIONS ON COMPUTERS, VOL. 54, NO. 10, OCTOBER 2005.*

[13] E.D. Jensen, C.D. Locke, and H. Toduda, "A Time-Driven Scheduling Model for Real-Time OperatingSystems", *Proceedings of Real-Time Systems Symposium*, pp. 112-122, 1985.

[14] Kwok-Wa Lam, Sang H. Son, Sheung-Lun Hung, and Zhiwei Wang, "Scheduling Transactions with Stringent real-time Constraints",*Information Systems Vol. 25, No. 6, pp. 431–452, 2000.*

[15] Shuhui Li, ShangpingRen,Yue Yu, Xing Wang, Li Wang, and Gang Quan, "Profit and Penalty Aware Scheduling for Real-Time Online Services", *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, VOL. 8, NO. 1, FEBRUARY 2012.*

[16] Lindtrom, "Real Time Database Systems", Jan Lindstrom *Solid,an IBM Company It¨alahdenkatu 22 B 00210 Helsinki, Finland March 25, 2008.*

[17] A.B Zhaohui Liu and T.C. Edwin Cheng, "Scheduling with job release dates, delivery times and preemption penalties", *Information Processing Letters 82(2002)107–111.*

[18] C. L. Liu and James W. Layland "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment" *Journal of the ACM*, volume 20, issue 1, pp. 46-61, January 1973.

[19] Reddy ALN, Wyllie J. "Disk Scheduling in Multimedia I/O system. In: *Proceedings of ACM multimedia'93, Anaheim, CA,* August 1993. p. 225–3*4*

[20] PrashantShenoy,Harrick M.Vin,"Cello:A Disk Scheduling Framework for Next Generation Operating Systems", *in Proceedings of the ACM SIGMETRICS'98,2002.*

[21] NeelamTyagi,MehdiAbedi and Ram GopalVarshney,"A preemptive scheduling and due date assignment for single-machine in batch delivery system", *Conference on Advances in Communication and Control Systems 2013 (CAC2S 2013.)*

[22] Y. Wang and M. Saksena, "Scheduling fixed-priority tasks with pre-emption threshold," in *Proc. 6th IEEE Int. Conf. Real-Time Comput. Syst. Appl. (RTCSA'99)*, Hong Kong, China, Dec. 13–15, 1999, pp. 328–335.