Resource Scheduling in Mobile Distributed Real Time Database Systems: A New Perception For Operating Systems

Gyanendra Kumar Gupta1, Shubha Jain2, Vishnu Swaroop3 and A K Sharma4
1Assistant Professor, Computer Science Department, Kanpur Institute of Technology, Kanpur.
2Associate Professor Dept. of CSE, Kanpur Institute of Technology, Kanpur
3Research Scholar, 4Assistant Professor, Dept. of Comp. Science & Engg., M.M.M. Engineering College, Gorakhpur
Email: 1gyanendrag@gmail.com, 2shubhj@rediffmail.com, 3rsvsgkp@rediffmail.com, 4akscse@rediffmail.com

ABSTRACT
Mobile Distributed Real Time Databases have all of the requirements of conventional databases with ACID properties and also require the management of mobility with time constrained data as well as transactions in distributed environment without failure that make the consistency in the database which is more important than anything. These entire additional requirements do not fulfill by the general operating system which supports the functioning of the MDRTDS systems. The database designers have to implement their own utilities like resource scheduling. a great deal it is just a replication of some common services. As a result a combined, devoted system, which will remove the duplication and improve the performance, is required instead of MDRTS System on top of conventional or a new Operating system for MDRTDS. The important property of the data in a mobile computing system is that their values can be highly dynamic and sensitive as they are used to maintain the useful and important information. To maintain the freshness and validity of the data items, Data items should be updated often by the generation of update transactions or by other means. Update transactions consist of one to several write operations and its arrival rate can be very high. It is assumed that a well-formed concurrency control protocol are used Mobile clients issue mobile transactions to access data items at the database, it has a sequence of transactions with a dead line. More over mobile transaction will become totally useless after its deadline and has to be aborted or restarted. The available operating system does not meet these trouble causes by mobility in the database system. Middle ware software in the form of system software is required to support the conventional operating system used by database management system especially for MDR TDS.

KEYWORDS

1. INTRODUCTION
The very complex and massive collection of data which is complied under some well known basic properties of database e.g. ACID; called conventional database management system. Usually application dependent designs were used in Mobile Distributed Real- Time Systems for data management but for the complicated applications and manage to huge amount of data, the algorithms, which deals with data management, grow to be very complex to build up and preserve. Mobile Distributed Real-Time Database Systems (MDRTDBS) have immerged as an substitute to manage the data with a structured and systematic approach [14]. In Mobile Distributed Real-Time Database System the transactions are associated as mobility in distributed data with explicit timing constraints, such as deadlines and the maximum temporal distance requirements between the accessed data objects. The correctness of a Real-Time Databases System depends upon the logical results and also upon the time at which the results are produced. Transactions in the system must be scheduled in such a way that they can be completed before their corresponding deadlines expire as well as satisfy database consistency constraints [13]. Mobile Distributed Real- Time Database Systems have different performance goals, failure & recovery of system, data consistency, correctness criteria, and assumptions about the applications. The conventional database system’s main objective is to provide fast response time, whereas a Real-Time Database System may be evaluated based on how often transactions miss their deadlines, the average “tardiness” of late transactions, the cost incurred in transactions missing their deadlines, data external consistency and data temporal consistency. Data in a Real-Time System is managed on individual basis by every task within the system. 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. [11]. It has been identified that for the improvement of the performance of Real-time Database Systems all the resources should be utilized efficiently avoiding duplication of work at the OS and Database System Level. This paper concentrates on the operating system perspective of scheduling of the physical resources. The organization of this paper is as follows: we first give a brief overview of Resource Scheduling in Mobile Distributed Real-Time Database Systems concentrating on the three physical resources CPU, Disk and Main Memory, then the operating system support for Real-Time Database Systems is discussed and finally the need for a Mobile Distributed Real-Time Database Operating System is justified.
2. WHAT IS SCHEDULING?
Scheduling is define in terms of Resource scheduling or Resource leveling.
Resource scheduling is a way of determining schedule periods on which activities should be performed; to smooth demand for resources or avoid exceeding given limits or availability depending on the arrangement of activities on the project. It may be able to do this without jeopardizing the end period and more often it means moving the end period. Whereas resource leveling aims to examine resource requirements during specific periods of the project minimize the variations in resource demand To improve efficiency or reflect reality by modifying activities within accessible drift. In other words modify resource loading for each unit of time, i.e. day, week or month. In practical considerations resource scheduling is not too easy, it can only distribute the same resource, substitute a different resource unless it has the same handiness set. The job new end period, may exceed the directive end period. It is requiring a re-cancelation of the schedule network. Spreading work over more time periods is not a one for one relationship, because of the "startup time" experienced each daytime. Which is why in due course is gorgeous, but only within limits or for short bursts. Doubling a resource may save only 25% of the time. The last 10% may take 20% of the total effort - or more. The Effect of constraints is if the job is time limited the total cost may increase. So if the job is cost limited it may simply have to "find" more resources when needed or even cut back on the job scope. And if the job is resource limited it may have to negotiate more time, more money, or both. Thus for improvement it look to improve the schedule applying minimum late start time, minimum late finish time, look for highest resource demand, redistribute to achieve best resource usage, look for opportunities for parallel activities and to provide more drift. Try to cascade activities calling for similar skills, to improve efficiency and avoid "downtime".

Resource Scheduling in Mobile Distributed Real-time Database Systems. There are three types of major physical resources: the processors, the disks and main memory buffers, that should be managed effectively and efficiently and even at the database level with support from underlying operating systems. The main criteria in assessing the success of any scheduling policy is the success ratio i.e. the number of transactions completed successfully before their deadline during the mobility of the resource in distributed environment.

CPU Scheduling
CPU scheduling is the basis of multi-programmed operating systems. By switching the CPU among processes, the operating system can make the computer more productive. In this paper, we introduce the basic scheduling concepts and discuss in great length CPU scheduling, FCFS, SJF, Round-Robin, Priority, and the other scheduling algorithms should be familiar to the researchers. This is their first exposure to the idea of resource allocation and scheduling, so it is important that they understand how it is done. A simple way is to write several different CPU schedulers and compare their performance by simulation. The source of CPU and I/O bursts may be generated by random number generators or by a trace tape. More recent systems (Windows NT) are back to having sophisticated CPU scheduling algorithms. What troop the change, and what will happen in the future? Most OSs today use very similar cpu time scheduling algorithms, all based on the same basic ideas, but with OS-specific adaptations and extensions. These algorithms are not the best algorithm that we can imagine, but it is, proven mathematically and by experience in the early days of OS programming, but the recent implemented algorithm shortest job first, round robin, priority based, multiple priority queue based that is the closest to the 'best' algorithm. Perhaps when computers get more powerfully some day, which we might implement the ideal cpu time scheduler. Another way is that this algorithm is designed for general-purpose computers. Special-purpose OSs or Mobile & Distributed systems and some real-time systems will use a very different algorithm. An Inheritance based scheduling algorithm is suitable for new systems in which processes can give their cpu time to "child" processes and as such act as schedulers themselves. In this way, multiple types of scheduling can be implemented; a scheduler for real-time processes, a scheduler for interactive processes, one for batch processing etc... The "root" scheduler is the basic scheduler implemented by the OS itself, which schedules processes and scheduler processes. While the individual scheduler processes might provide each another way of scheduling, the "root" scheduler still uses some Multiple Priority queue (MPQ-form).

There are a wide range of algorithms for scheduling the CPU in traditional database systems. These algorithms generally stress justice and attempt to balance CPU and I/O bound transactions. These scheduling algorithms are not adequate for mobile distributed real-time transactions. In mobile distributed real-time environments, transactions should get access to the CPU based on range area of resource, criticalness and deadline, not only fairness. If the complete data access requirements without any failure and timing constraints are known in advance, then scheduling can be done through transaction pre-analysis. On the other hand, in many cases complete knowledge may not be available. Then a priority based scheduling algorithm may be used, where the priority is set based on deadline, criticalness, length of the transaction, or some combination of these factors [15]. There are some simple preempt-able CPU scheduling algorithms like: Most Critical transaction First (MCF), Earliest Deadline First (EDF) and Criticalness and Deadline First (CDF) [12]. Criticalness and deadline distributions strongly affect transaction performance. Under the value-weighting scheme, criticalness is a more important factor than the deadline with respect to the performance goal of maximizing the deadline guarantee ratio for high critical transactions and maximizing the value imparted by real time transactions. Overheads such as locking and message communication are to be non-negligible and cannot be ignored in real-time transaction analysis [15].

Disk Scheduling
The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth. Access time has two major components. Seek time is the time for the disk are to move the
heads to the cylinder containing the desired sector. Rotational latency is the additional time waiting for the disk to rotate the desired sector to the disk head. Minimize seek time i.e. Seek time \( \approx \) seek distance. Disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

SSTF is common and has a natural request SCAN and C-SCAN perform better for systems that place a heavy load on the disk. Performance depends on the number and types of requests. Requests for disk service can be influenced by the file-allocation method. The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary. Either SSTF or LOOK is a reasonable choice for the default algorithm. The performance of these algorithms depends heavily on the workload (number of requests). Under light load all algorithms perform the same. If the queue seldom has more than one outstanding request, then all algorithms are effectively the same. Their performance also depends upon the file organization and the type of generated requests. Multilevel Feedback Scheduling Algorithm before multilevel feedback scheduling algorithm, we used only multilevel scheduling algorithm which was not so flexible, but this one uses separate queue for handling the processes, it automatically adjust the priority of the process. If the priority is high and the process will be allotted to lower priority queue, then automatically it can switch to the higher priority queue. For example, one queue may implement round robin algorithm the other may use first come first serve algorithm, the allocation is based on the type of process i.e the process is either of cpu bound or i/o bound etc it is now used by generally all the operating system only because of its flexibility but it is some complex to understand.

In a disk-based database system, disk I/O occupies a major portion of transaction execution time. There are two objectives for any disk-scheduling algorithm:

**Minimize the throughput** - the average number of requests satisfied per time unit.

**Maximize the response time** - the average time that a request must wait before it is satisfied. [18]

Some of the general disk scheduling algorithms are: First In First Out (FIFO), Shortest Seek Time First (SSTF), SCAN, Circular SCAN (C-SCAN), LOOK, Circular LOOK (C-LOOK), Round- Robin, Priority Scheduling, Fair Share, Shortest Job First, Exponential Queue and Elevator Algorithm.

There are real-time disk scheduling algorithms: Earliest Deadline (ED), Priority SCAN (P-SCAN), Feasible Deadline SCAN (FD-SCAN), Shortest Seek and Earliest Deadline by Ordering (SSEDO), Shortest Seek and Earliest Deadline by Value (SSEDV) (Chen et al, 1991). The two algorithms called SSEDO and SSEDV combine deadline information and disk service time information in different ways. [15]. A real-time disk scheduling algorithm, Concurrent DS-SCAN (CDS-SCAN), which maximizes throughput for modern storage devices by allowing concurrent I/O requests at the device whenever possible. Past real-time disk scheduling algorithms allowed a single request at a time to go to the storage device, which dramatically reduces the utilization and throughput for modern storage devices, such as RAID arrays and disks with efficient positional-aware scheduling algorithms. We extended the DS-SCAN algorithm so that it can properly account for multiple outstanding I/O requests and guarantee real-time constraints for both outstanding and pending real-time requests. We demonstrate CDS-SCAN's performance on a storage array. [20]

### 3. PROCESS MANAGEMENT

A process is an instance of a computer program that is being sequentially executed. Every program running on a computer, be it background services or applications, is a process. One CPU can run one process at a time. Process management is an operating system's way of dealing with running multiple processes on a single CPU. Since most computers contain one processor with one core, multitasking is done by simply switching processes quickly (known as context switching). Process management involves computing and distributing CPU time as well as other resources.

### 4. CONTEXT SWITCHING

The state of the first process must be saved somehow, so that, when the scheduler gets back to the execution of the first process, it can restore this state and continue. This is accomplished as follows -

All the data required to define the state of the process - all the registers that the process may be using, especially the program counter, plus any other operating system specific data that may be necessary is stored in one data structure, called a switch frame or a process control block (PCB). The PCB for the first process is created and saved. The OS then loads the PCB and context of the second process. In doing so, the program counter from the PCB is loaded, and thus execution can continue in the new process. In a context switch new processes are chosen from a queue or queues. Process and thread priority can influence which process continues execution, with processes of the highest priority checked first for ready threads to execute. When a process is first created, it occupies the "created" or "new" state. In this state, the process awaits admission to the "ready" state. This admission will be approved or delayed by process scheduler. Typically in most desktop computer systems, this admission will be approved automatically, however for real time operating systems this admission may be delayed.

A "ready" or "waiting" process has been loaded into main memory and is awaiting execution on a CPU (to be context switched onto the CPU). There may be many "ready" processes at any one point of the systems execution - for example, in a one processor system, only one process can be executing at any one time, and all other "concurrently executing" processes will be waiting for execution. A ready queue is used in computer scheduling. Modern computers are capable of running many different programs or processes at the same time. However, the CPU is only capable of handling one process at a time. Processes that are ready for the CPU are kept in a queue for
"ready" processes. Other processes that are waiting for an event to occur, such as loading information from a hard drive or waiting on an internet connection, are not in the ready queue. A "running", "executing" or "active" process is a process which is currently executing on a CPU. From this state the process may exceed its allocated time slice and be context switched out and back to "ready" by the operating system, it may indicate that it has finished and be terminated or it may block on some needed resource (such as an input / output resource) and be moved to a "blocked" state. Should a process "block" on a resource (such as a file or a device), it will be removed from the CPU (as a blocked process cannot continue execution) and will be in the blocked state. The process will remain "blocked" until its resource becomes available, which can unfortunately lead to deadlock. From the blocked state, the operating system may notify the process of the availability of the resource it is blocking on (the operating system itself may be alerted to the resource availability by an interrupt). Once the operating system is aware that a process is no longer blocking, the process is again "ready" and can from there be dispatched to its "running" state, and from there the process may make use of its newly available resource. A process may be terminated, either from the "running" state by completing its execution or by explicitly being killed. In either of these cases, the process moves to the "terminated" state.

5. PROCESS SCHEDULING

Scheduling is a key concept in computer multitasking and multiprocessing operating system design. It refers to the way processes are assigned priorities in a priority queue. This assignment is carried out by software known as a scheduler. The scheduler is concerned mainly with –

CPU utilization - to keep the CPU as busy as possible.

Throughput - number of processes that complete their execution per time unit.

Turnaround - amount of time to execute a particular process.

Waiting time - amount of time a process has been waiting in the ready queue.

Response time - amount of time it takes from when a request was submitted until the first response is produced

Non-preemptive scheduling - In non-preemptive scheduling, a job is completed before making another scheduling decision. Since the OS waits till a job is completed, one poorly designed program can cause the whole system to hang. Non-preemptive scheduling is generally used in real time operating systems.

Preemptive scheduling - In preemptive scheduling a scheduling decision can be made while the current job is executing. The OS does not wait for a job to compete to schedule another job but provides each process with a fixed amount of processor time (also known as time slice). It is more reliable and makes the OS more responsive.

The concept of priority is important as many processes are competing with each other for the CPU time and memory. The priority can be external or internal.

External priority is specified by the user at the time of initiating a process. This priority can be changed during the time of execution. If the user does not assign a priority then the OS assigns a default priority. Internal priority is based on the calculation of the current state of the process. This can be based on the estimated time it would take for a process to compete. Based on this, the OS can which process it should execute first.

Memory Management - Here we concentrate on the system resource main memory. There are two issues whether memory space is limited or is ample. If a Real-Time System has only limited amount of memory, buffer management, which concerns the allocation of memory space among concurrent transactions, has to be specially designed. The goal here is to ensure that the execution of high priority transactions is not delayed by limited memory. On the other hand, if there is plenty of memory, much of the data can reside in main memory; the resultant database system is called a Memory Resident Database System (MRDBS). A MRDBS is fast and predictable access time, which make it suitable for real-time applications. We will be discussing buffer management.

The availability of memory affects transaction response time in two ways. First, before a transaction starts its execution, buffers have to be allocated to the transaction. These buffers are used to store the execution code, copies of files and data paged in from disk, and any temporary objects produced. Depending on the transaction, a certain number of buffers have to be allocated in order to prevent the transaction from thrashing. When memory is running low, a transaction may be blocked from execution. The amount of memory available in a system thus limits the number of concurrently executable transactions. Some applications, such as image processing, have high demands on memory. Their executions will be significantly slowed down if memory is limited and frequent memory swapping is done. The job of a buffer manager is to allocate memory buffers to transactions intelligently such that high priority transactions enjoy shorter response times.

Data buffering plays an important role in database systems where part of the database is retained in a main memory space so as to reduce disk I/O and, in turn, to reduce the transaction response time. The principle of buffer management is based on transaction reference behaviors [12]. In terms of locality, there are basically three kinds of reference strings in database systems:

Intra-transaction locality, where each transaction has its own reference locality, i.e., the probability of reference for recently referenced pages is higher than the average reference probability.

Inter-transaction locality, where concurrent transactions access a set of shared pages.

Restart-transaction locality, where restarted transactions repeat their previous reference behavior [15].

Buffer management policies should capitalize on one or more of these three types of locality. Buffer allocation and buffer replacement are considered to be two basic components of database buffer management [7]. Buffer allocation strategies attempt to distribute the available buffer frames among concurrent database transactions, while buffer replacement strategies attempt to minimize the buffer fault rate for a given
buffer size and allocation. The two schemes are closely related to each other and are usually integrated as a buffer management component in database systems. There are many buffer replacement policies like the Global Least Recently (G-LRU), Priority Memory Management (PMM), the Priority DBMIN etc [4]. Operating System Support for Scheduling In Real-Time Database Systems. Most of the research on Real-time Database Systems has been based on developing and investigating real-time transaction scheduling algorithms, disk and I/O scheduling with a goal to minimize the deadline miss ratio of transactions. However, less attention has been paid to the operating system aspects of the system, which support the predictable behavior of a real-time transaction [6].

Scheduling theory in Real-Time Operating System area is usually based on task that has a time constraint. A task is a software module and is the scheduling entity in an operating system where as the basic unit of scheduling in Real-Time Database System is a real-time transaction [11]. Some researcher [16] has examined various operating system services that are commonly used in DBMS’s. He has evaluated several designs and implementations of each service and their effectiveness in meeting DBMS’s goals. He has pointed out that operating system services in many systems are not appropriate for supporting the database systems. Either those services are poorly implemented or are too slow for the database system tasks, which result in degradation of performance of the system significantly. As a result, many DBMS’s designers have to re-implement similar functionality in the user space.

The database should implement its own buffer policy on top of Windows NT [8]. Also, if the DBMS implements its own buffer management system, the crash recovery process would be impossible by the operating system. It seems as if it’s not possible to get the best performance by using the operating system’s buffering, concurrency control and crash recovery. It is likely that the Real-Time Database System designers will continue implementing their own scheduling and management systems but without adequate support from the underlying subsystems, none of the scheduling algorithms can guarantee predictable transaction performance. The same is true with Real-time Operating Systems and Real-Time Database Systems. There is lack of cooperation between them which results in a duplication of some common services, which leads to a degradation in system performance, and also make the system unpredictable. Clearly, this is intolerable for real-time applications. Real-Time Database building blocks must be integrated with the Real-Time Operating System kernel and other runtime environment building blocks in order to avoid wasteful duplication and provide predictable services. Making transaction execution times predictable through an adequate architecture and OS support does not guarantee that the deadline of a transaction will be met. It is the scheduling mechanism of a Real-Time Database System that utilizes such information and guarantees both consistency and timing constraints [13]. Necessitate to Mobile Distributed Real-Time Database Operating System (MDRTDOS)

Vital paradigm applications of MDRTDBS are stock trading systems, E-Banking, E-Ticketing other E-commerce applications, mobile computing applications, navigation systems, telephone management systems, and computer integrated manufacturing. Database management system provides the set of programs for creation and processing database. With growing demands of heavy data processing, as pointed out earlier, some of the operating systems do not yield good performance for database management system because of incompatibility in the configuration. The database management systems performance remains underutilized, as operating system does not provide sufficient functionality to the database implementation. Database Operating Systems are those, which are designed for supporting the function needed by Database Management System [18]. They are designed by emphasizing the database systems need. Mobile Distributed Real-Time Databases have all of the requirements of traditional databases, such as managing access to structure, shared, permanent data, but they can also require management of time constrained data and time-constrained transactions. Mobility introduces a great computing freedom to users to access Computing power as well as the need of database. Where ever the data are spread over a number of places on different servers in different operating system environment. Some applications are requiring the real-time requirements where the time constraint is primary concern to complete the job. So the Dynamic Resource Scheduling (DRS) continuously balances computing capacity in resource pools to deliver the performance, scalability and availability not possible with limited physical infrastructure. Dynamically allocate IT resources to the highest priority applications. Create rules and policies to prioritize how resources are allocated to virtual machines.

Giving IT autonomy to organizations. Provide dedicated IT infrastructure to organizations units while still achieving higher hardware utilization through resource pooling. Allow organization units to build and manage virtual machines within their resource pool while giving central IT control over hardware resources. Get better service levels for all applications. DRS continuously balance capacity will ensure that each virtual machine has access to appropriate resources at any point in time. Simply deploy new capacity. DRS will seamlessly take advantage of the additional capacity of new servers added to a resource pool by redistributing virtual machines without system disruption. Computerize planned server maintenance. DRS can automatically leave all virtual machines off physical servers to enable scheduled server maintenance with zero downtime.

7. CONCLUSION

As the mobile and distributed in real time environment become increasingly, operating systems for both mobile and stationary computers. It has a number of challenges the very first is disparity management, second is physical mobility of both user and data physical management and third is user demand in computing power and requirement of heavy data in form of test, picture and graphics. The available scheduling
management could not accommodate these challenges efficiently. In coming days the requirement again grow and grow in other forms also. To realize the user computing power requirement to become more acceptable, it must be widely available, while it must minimizing the intrusion of a mobile host within the stationary environment. So to gratifying the coming and existing challenges the database system cannot totally depend upon the conventional operating system and hardware.

The goal of this paper is to give a concise outline of Resource Scheduling in Mobile Distributed Real-Time Database Systems with an Operating System viewpoint. It has been identified that the Mobile Distributed Real-Time Database System designers still continue to implement their own scheduling and management systems but without enough support from the essential subsystems, none of the scheduling algorithms can guarantee predictable transaction performance. Mobility is one main concern in failure of transactions because of un-reliable (wireless) network where distributed environment rises an other issue to overload the database requirement. Duplicity, Fragmentation, Backup and Recovery are some major issue in Mobile Distributed Real Time Database Systems.

There is duplication of some common services, which leads to a degradation in system performance making the system more unstable. Mobility causes the system to recompense more anxiety on data consistency rather than delay in transaction or other causes. Distribution and Time constraints at the real-time requirements, the support provided by the OS and duplication of some common services a small but efficient Mobile Distributed Real-Time Database Operating System is required to accommodate the necessitate.

8. REFERENCES