Overload Identification for Multiprocessor in Real Time System

JESTIN RAJAMONY
IT Department
CSIIT, Thovalai
INDIA

Dr. K.RAMAR
Principal
SVCET, Virudhunager
INDIA

K.P.AJITHA GLADIS
IT Department
CSIIT, Thovalai
INDIA

Abstract

In spite of many real time scheduling algorithms available it is not clear that these scheduling algorithms support fully the problems in the real time system in a local area network. There are certain "open loop" algorithm that can support only some set of characteristics such as the deadlines, precedence constraints, shared resources and future release time etc. Open loop are being referred as once the schedules are fixed there is no alterations. Open loop is fine for the static or dynamic models where the job is perfectly modeled and assigned. But when it executed for unpredictable dynamic systems the open loop does not offer its full performance due the problem of overloading in the processor. In this paper, the overloading of the processor is detected and rectified to give full performance of the processors in the network for the real time system. Here the case is studied from the worst case to the best case.

1. Introduction

There are many real time scheduling algorithms available but it is not clear that these scheduling algorithms support fully the problems in the real time system in a local area network in the onboard computers of the space shuttle. There are certain "open loop" algorithms that can support only some set of characteristics such as the deadlines, precedence constraints, and shared resources. Static scheduling algorithms have the complete knowledge of the task set and its constraints. For example the Rate

Monotonic Algorithm (RMA). But RMA does not give the full performance in the dynamic environment that is the essential in the real time systems. The dynamic scheduling algorithm does not have a complete knowledge of the task set and its constraints. For example, if a new task is in urgent and wants to be inserted in the middle of the scheduling then the RMA scheduler will not be knowing of the current task and its timing even thou the task is a predictable one.

Many real world complex problems occur in the network of computers. For example, a system in the node of the network may meet the different variation in the overload of the execution as in the case of network of computers controlling the spacecraft the workload parameters that differ due to different input from the space sensors and their interpretation. Despite there are many real time scheduling algorithms available it is not clear that these scheduling algorithms support fully the problems in the real time system in a network. There are certain "open loop" algorithms that can support only some set of characteristics such as the deadlines, precedence constraints, shared resources etc.

In the case of networks the workload may differ from one way or the other which tends to be unpredictable for the dynamic systems. Even though the open loop scheduler spring scheduling algorithm is designed for the worst-case workload parameter they are underutilized system for workload models that are not available. The problem here is that scheduling paradigms all assume the timing requirements are to be known and also to be fixed. If there is a fixed time range for the scheduling then in the case of networks it will be more tedious because of the varying inputs. So it is better to fix to a range of dead lines for the job to be finished but that becomes too complex. Due to these problems in this paper a new paradigm for detecting the overload in a network of systems is introduced.

Open loop scheduling is fair for both static and dynamic systems if there is sufficient resource in a stand-alone system. Earliest Deadline First Algorithm (EDFA) is dynamic scheduling algorithm that has the complete knowledge of the task set or timing constraints for the resource sufficient environment. If the resource is insufficient in the environment then the EDFA performance will rapidly degrade in overload situations. But EDFA will rapidly degrade its performance in overload situations because the entire task is moved to the limited available resource that doesn't suit the real time environment

Dynamic scheduling can be classified into resource sufficient environment & resource insufficient environment. Resource sufficient environment is one where the systems have the system resources in prior to the task that arrives dynamically at any time and are subjected to the scheduling. But resource insufficient environment is one where the systems have the system resources in limited to the task that arrives.

2. Motivation

Operating System is the core portion between the application program and the hardware and provides the abstract view between each other. Operating system is designed to maximize resource utilization to assure that all available CPU time, memory and I/O are used efficiently. It enhances its utility by eliminating duplicate efforts of hundreds of programmers in developing tedious and complicated routines. It provides the provision of security and confidentiality of information to users. The primary goal of operating system is efficient operation of the computer system [1]. Multi programming and time sharing system improve performance by overlapping CPU and I/O operations on a single machine. But they need to send the task in a sequence for the CPU to make the CPU to work for the maximum to give full utility.

By switching the CPU among processes, the operating system can make the computer more productive. To maximize the CPU utilization some of the process runs at all times. Process alternate between two states like CPU burst and Input Output burst and so on. The duration of the CPU burst has to be measured [2]. Whenever the CPU becomes idle the operating must select one of the processes in the ready queues to be executed. The selection process is carried out by the short term scheduler or the CPU scheduler. The scheduler selects from the processes in memory that are ready to execute and allocate the CPU to one of them. Dispatcher then distributes the process to the particular CPU that is free. Time taken by the dispatcher to stop one process and start another process for running is known as the dispatch latency.

From the single core to dual core and today multicore CPU have become a commodity items in major researches. The expectation in the next decades is that the number of cores in a CPU will increase to as many as hundreds [3]. All cores in a homogenous multiprocessor have same performance with a same instruction set, but however they differ in terms of performance characteristics with changes in clock frequency, cache size etc. Recent researches have advocated the need for a class of heterogeneous multicore processors. Here even with same instruction set, there is a possibility of lot of changes in the performance characteristics [4]. The architecture available for performance symmetric (homogenous) multi core processor is effective compared to the performance asymmetric (heterogeneous) multicore processor [5]. For example, when workload characteristics are matched to heterogeneous cores, performance gains up to 40% are observed [6].

Environments are classified into real time system and non real time system. Non real time system does not have to finish any task particular time but can take its own time and can also be used as a test work. Real time systems require that the task be performed within a particular time frame. Real time systems are classified into hard real time system and soft real time system. The dead line in the hard real time system has to be met else it leads to terrific problem. But the soft real time system is not so lenient to deadline and this leads to the minor problems. So real time scheduling is to be dealt with cautious in order to avoid disaster. Static scheduling and dynamic scheduling algorithms are the two types of algorithm that fall in the real time scheduling algorithm.

If the available resources are sufficient and the scheduling algorithm has complete knowledge of the task set and its constraints then static scheduling is preferred. E.g. Rate Montonic Algorithm [7]. When a new task arrives it is not known to the scheduling algorithm, so the algorithm does not have the complete knowledge of the task set then it is called dynamic scheduling. Dynamic scheduling is hard to predict, since it does not know about the task and its future release time. Earlier deadline first (EDF) algorithm is a dynamic scheduling algorithm that works fine with resource sufficient environment. Resource sufficient environment means all the resources are available in priority before the job arrives.

But when the system gets overloaded then the EDF algorithm degrades rapidly than any other schedulers. This is due to the fact that it gives the highest priority to transaction that is close to the deadline misses. So Adaptive earliest deadline (AED), is used that detects overload condition and then modifies transaction with priority [8] using the feed back control mechanism. In an unpredictable environment, it is very difficult for the real time system designer to meet the required deadline. Spring scheduling algorithm [9] using the online admission control algorithm can guarantee partly in resource insufficient environment that is unpredictable.

Now the distributed soft real time systems are becoming increasingly unpredictable where the execution parameters vary with the input data. So the traditional real time scheduling algorithms used in systems are less useful. So it is necessary to specify the convergence speed to the set desired performance upon load or resource changes [10]. This reveals that the performance is quite excellent in steady state behavior and also meets stability, overshoot and settling time requirements.

There are many other scheduling algorithms [11] that support the real time scheduling environment with sufficient resources. Despite many real time algorithms available, none of the algorithm supports fully the real world problems. Operating systems goals are efficiency, robustness, scalability, extensibility, portability, security, interactivity and usability.

Most of the scheduling algorithms are open loop scheduling algorithm. Open loop means once the schedules are created by the scheduler they are not adjusted based on continuous feedback. If the work load is well known in prior and is predictable, then open loop is the best algorithm for static and dynamic environment. EDF, AED are the best algorithms for open loop while the resources are sufficient and are predictable. Even with automatic technique, the problem is the resource required by the code and the resources available provides by the multicore processor are to be determined [12].

The research of this paper is towards hard real time application in the aircraft takeoff and landing. It uses the feedback control theory and its framework in an unpredictable real time system. Failure to meet these leads to deadly problems and finally disaster. Almost every research papers related to this is concerned with deadline misses and overload, but they are not fulfilling even the deadline misses and the overload problems. And while computing in the multiprocessor, they do not directly involve on the timing constraints that is the key features of the real time system. An event triggered sampling [13] has been proposed with an idea of sampling, communicating and controlling only if something significant has occurred in the system.

The effect of control system performance degrades when a periodic task is implemented. This generates the problem in sampling and produces latency jitter [14]. Two major problems have been identified in the control application that reduces the performance of the system.

- ❖ Allocation of resources to control applications in order to maximize control performance.
- ❖ Novel computational models for implementing control algorithms using real time technology.

The second problem was removed by a one shot task model [15]. In this paper the allocation of resources to the multicore processor using proper control algorithm in order to maximize the control performance.

The major work of the paper is selecting the desired algorithm and fixing to the desired processor in the heterogeneous multicore processor. So if a deadline is not met by any one of the processor, then the scheduler submits it to the next processor of high or low end speed using the feed back control framework [16]. The aim of this paper is based on the deadline based metrics from the worst case to the best case, where there is a major shift of total load in the system. The system CPU should be fully utilized when there is job waiting in the queue. Also the processor utilization can be dynamically obtained by assigning priorities on the basis of the current deadlines [17]. Most of today's job works with the threads, where a processor holds the specified resources. The thread are given a specified percentage of CPU cycle over a period of time and uses a feedback back scheduler to assign automatically both proportion and periods [18].

Based on this a QoS optimization algorithm and a communication subsystem architecture was developed [19]. The actuator depends on the QoS algorithm that meets both predictability and graceful degradation requirements. The QoS negotiation guarantees the required QoS and rejects the service request, by outperforming binary admission control schemes [20]. Our Scheduler architecture includes the following elements

- ❖ Feed back control schedulers architecture that maps the feed back control structure.
- ❖ Resource as multicore processor in a shared network with a on shot model selective approach.
- ❖ A set of performance metrics and a speed analyzer for the digital controller.

In contrast, our frame work enables system designers to systematically design adaptive real time systems with established analytical methods to achieve desired performance guarantee in an unpredictable environment

3. Stability

Stability of the system is an important part of the scheduling. It is a necessary condition to prevent miss ratio and utilization from staying at the undesirable limit 100%. If there is no input, the internal stability will automatically settle to the nearby set point within a specified amount of time. But here for the sake of simplicity the stability is now kept still for future research.

4. Schedulers Architecture

The real time scheduler for the network of computers consists of a scheduler, a Dispatcher and 5 CPU's as shown in the figure 1.

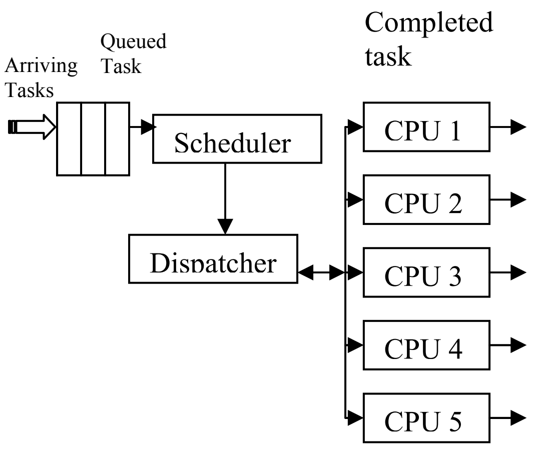

Figure 1: Real Time Scheduler

These components are been explained in brief. The scheduler receives the tasks in the queue and is based on the algorithm, the task are rescheduled to the dispatcher. The dispatcher finds out the idle Central Processing Unit (CPU) in the Network and the task is executed there. But most of the works are research oriented on the task that is fixed and is well known. Here in this paper the task are unpredictable and is dynamic and time varying. In the network, the systems could get workload where it is shared with initial start as the nominal assumption.

5. Processor specialization

Assume that each system in the network has different workload and each task is independent. Several different forms such as milestone method, sieve function method or multiple version method are imprecise computation [21].

A job with longer execution time and another job with smaller execution time is called. But the task T_i will have a deadline as D_i and a start time S_i. Each task T_i in the system has the set I, ET, VAL, S, D in it where I represents logical version, ET represents the execution time, VAL represents the values of different types of implementation, S is the start time and D is the deadline. Each task can be adjusted within the range that is specified for given deadline. Each task has one or more logical versions I = ($T_{i1,}$ T_{i2} ... $T_{ik)}$. When applied in the network of systems these tasks are sent to different CPU so the task does not seams to have multiple implementations. Generally in digital control systems the task timing constraints are allowed to adjust within a specified range without affecting the system stability.

Each task in different CPU has different execution time ET = {$ET_{11,}$ $ET_{12,}$ ... ET_{kj}} of different versions and they get into different values. This ET specified here is for one CPU. But in different CPU the ET get split to $ET_{21,}$ $ET_{22,}$ ... $ET_{kj.}$ This specifies the CPU and the task that is split for that CPU. The nominal execution time is used for the requested CPU utilization. For example if the ET_{11} = 0.01, then the CPU1 utilization is 1% from the miss ratio for the first version. For the same CPU if the ET_{12} = 0.25 then the processor utilization is 2.5% for the second version and for the third ET_{13} = 0.15 means 1.5% and so on. So the total execution time results in just 0.35 that is about 3.5% CPU utilization.

6. Observation

Imagine five types of job of different sizes 50, 40, 30, 20, 10 are approaching the processor for the execution as shown in the table 1. It takes nearly 5 secs for a job of size 50 to complete its execution. So to get full processor utilization it requires 50 size jobs. But the second job is of the size just 40 which spares 20% of the processor time unused.

And the third job of the size 30 that spares 40% of the time unused. So in total the overall performance is 60% leaving 40% of the time to remain the processor in the network to be idle.

Job Size	Performance	Drop out	Loss
50	100 %	0 %	0 %
40	80 %	20 %	0 %
30	60 %	40 %	0 %
20	40 %	60 %	0 %
10	20 %	80 %	0 %
Total Performance			60 %
Total Drop out			40 %
Waste age			0 %

Table 1 Dropout Problem

At the same time consider if the job size is of the size 70 then the processor represents that the CPU utilization is of 100% (i.e. full CPU utilization) as shown in the of table 2. So the above 20 size of the job is now overloaded which is held as waiting. The scheduler does not know this because it has crossed the scheduler and is in the processor and is not shown by the processor as that it is overloaded. So a loss of nearly 40% is occurring as shown in the table 2.

Another job of the size 60 is overloaded to the processor that gives the loss of the 20%. With this it is possible only to get the CPU total utilization to 88% giving a total loss of 12% and a total wastage of 12% from the overloading of a processor. To overcome this situation of overloading in the network of the processors this paper defines the full work of how to make utilize full performance of the processor in the network without overloading.

7. Experiment

Consider 5 CPU, scheduler, and a dispatcher in the network of processor. Assume 5 jobs have arrived in the queue, the scheduler schedules the job in the queue using any of the scheduling algorithms and sends it to the dispatcher. The dispatcher then identifies the processor that is idle in the network and sends the task to it.

Job Size	Performance	Drop out	Loss
70	140 %	0 %	- 40 %
60	120 %	0 %	- 20 %
50	100 %	0 %	0 %
40	80 %	20 %	0 %
30	60 %	40 %	0 %
Total Performance			88 %
Total Drop out			12 %
Waste age			-12 %

Table 2 Overloading Problem

The processor will execute the job if it is a predictable one because here the jobs are considered as real time jobs. Since it is a real time system any inputs from the related sensors may occur. If in case the task is totally unpredictable then it is removed from the scheduler and corrective measures are taken [22]. This part is not discussed in depth in this paper.

Consider 5 jobs of varying sizes of 70, 60, 50, 40, and 30 are arriving at the scheduler from the job queue as shown in the table 3. This is then passed to the scheduler where it schedules according to any scheduling algorithm and passes it to the dispatcher that identifies the idle CPU and gives the job for execution. The processor in the network has a capacity to execute the job of 50 sizes to bring out its full performance. But the first job here is 70 in size so there is an overloading 40% that is considered as overloading and processor shows that it is executing in full performance leaving the wastage time. But this can be of nothing for a non real time system but for a real time system this wastage can lead to some disaster. Similarly the second job has a wastage time of 20%. The third job is in full utilization of the processor. But the fourth job whose maximum utilization itself is 80% leaving the CPU idle for 20% of time.

In this paper to overcome this overloading, it is detected first for each processor and then the total utilization of the processors in the network is determined. Then the overall total utilization is raised to the maximum to get the full performance of the processor.

Job Size	Performance	Drop out	Loss
70	99.99 %	0.01 %	0 %
60	99.99 %	0.01 %	0 %
50	99.99 %	0.01 %	0 %
40	99.99 %	0.01 %	0 %
30	99.99%	0.01%	0 %
Total Performance			99.99 %
Total Drop out			0.01%
Waste age			0 %

Table 3 Rectification of Overloading

Consider now the first job of size 70 is now executed in the first processor, which has the capacity of 50 sizes to be the maximum to get the full performance. So when the job size reaches the 50th size of the 70 size job then it is equated to 99% i.e. processor utilization u(k) = 0.99. That is the processor is in full utilization. The rest 20 size that is left from the 70 size is shifted to the next processor.

So the next processor which takes 50 size from the current job and the next job (for example here 20 + 60 = 80) is equated to u(k) = 1 leaving behind for the third processor 30 size of the job. This is done for all the jobs in the queue. So the total utilization T(k) is now 99% i.e. T(k) = 1. Now the T(k) = 99% is now increased from the 99% to slight higher to 99.99% giving a marginal increase of 0.01%. This gives the T(k) = 1 when the total processor utilization is 99.99%. But the actual CPU utilization is A(K) = 100%, that is compared to the T(k) which gives $T(k) \approx A(K)$ giving the full performance of the processor.

8. Results:

From the above experiment it very clear that the CPU is in full utilization as shown in the figure 2 for a network of computers working in the real time environment. The 0.01% wastage is negligible since in a real time system if the processor didn't work for 1% of the time it might lead to disasters but not for 0.01% of negligible time. Combination of negligible wastage does not affect the efficiency of the computer in the network.

The output is about 12% more than the other schedulers when it is used in the multiprocessors. The CPU processor when it crosses the 99.99% is immediately set to the next processor that remains idle. So no processor can cross the limit of 100% that indicates the processor is overloaded.

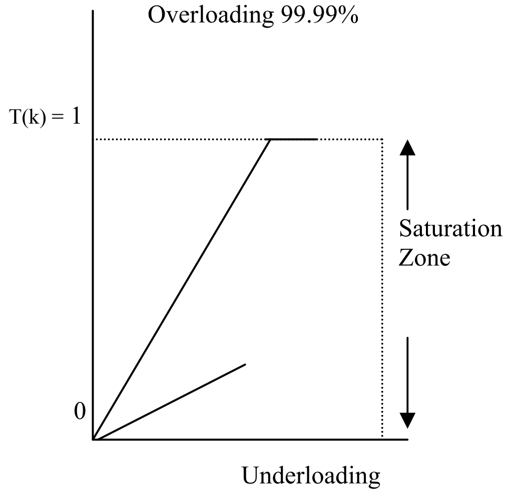

Figure 2 Grapical Representation of Overloading Rectification

9. Related Works

This closed loop information to adjust the scheduling has been in trial in the multilevel feedback queue [23]. There is no systematic study on the feedback driven scheduling. [24] presented a feedback based scheduling scheme that adjusts CPU allocation based on the application dependent progress monitors. [7] has adopted a priority assignment policy based on the EDF. [25] has given the feed back control on the CPU to be overloaded at all the time causing the CPU to be in full work. When the workload is worst then the CPU still has the starvation problem.

10. Conclusion

In this work, for a real time scheduling systems in a network of systems that communicate with one another the overloading of the processor has been explored. This would give a new idea on the real time scheduling in the network. Any algorithms used, and some experiment results that are used for the scheduling are not mentioned in this paper. But the way it operates in the real time to give full performance has been mentioned with a sample result.

References

[1] "Operating System", Deitel, Choffnes, Pearson Education, III Edition, 2007.
[2] S.Y.Borkar et al, "Platform 2015; Intel Processor and Platform evolution for the next decade Technical Report White Paper", Intel Corporation, 2005.
[3] Matt Gillespie, "Preparing for the second stage of multicore hardware: Asymmetric heterogeneous cores", Technical Report, Intel Corporation, 2008.
[4] R.Kumar et al, "Single ias hetrogenous multicore architecture for multithreaded workload performance", ISCA, Page 64, 2004.
[5] R.Kumar et al, "Core architecture optimization for heterogeneous chip multiprocessors", PACT, 2006.
[6] C. Lu, J. A. Stankovic, T. F. Abdelzaher, G. Tao, S. H. Son, and M. Marley, "Performance Specifications and Metrics for Adaptive Real-Time Systems," *21th IEEE Real-Time Systems Symposium,* Orlando, FL, December 2000.
[7] J. R. Haritsa, M. Livny and M. J. Carey, "Earliest Deadline Scheduling for Real-Time Database Systems", *IEEE RTSS*, 1991.
[8] W.Zhao, K. Ramamritham and J.A.Stankovic, "Preemptive Scheduling Under Time and Resource Constraints," IEEE Transactions on Computers 36(8), 1987.
[9] John A. Stankovic, Tian He, Tarek Abdelzaher, Mike Marley, Gang Tao, Sang Son and Chenyang Lu, "Feedback Control Scheduling in Distributed Real-Time Systems," IEEE Real-Time Systems Symposium, London, UK, December 2001.
[10] J.A.Stankovic, M.Spuri, K.Ramamritham and G.C.Buttazzo, Deadline scheduling for real-time systems EDF and related algorithms, Kluwer Academic Publishers, 1998.
[11] Chenyang Lu, John .A.Stankovic, "Feedback Control Real Time Scheduling: Framework, Modeling and algorithm," Journal of Real Time System., 2000.
[12] Tyler sondag, Hridesh Rajan, "Phase guided thread to core assignment for improved utilization of performance asymmetric multicore processors", Paper of multicore processor, USA, 2009.
[13] Anton Cervin, Toivo Henningson, "Scheduling of event triggered controllers on a shared network", Proc. Of IEEE Conference on Decision and control, Cancun, Mexico, 2008.
[14] K.E.Arzen, A.Cervin, J.Eker and L.sha, "An Introduction to control and scheduling co-design", Proc. IEEE conference, Decision and control, 2000.
[15] Camilo Lozoya, Manel Velasco, Pau Marti, "The one shot task model for robust real time embedded control systems", IEEE Transactions on Industrial informatics, vol 4, No. 3, 2008.
[16] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," Journal of ACM, Vol. 20, No. 1, pp. 46-61, 1973.
[17] D.C. Steere at.al, "A Feed back - driven proportion allocator for real rate scheduling", 2000.
[18] T.F. Abdelzaher and K.G.Shin, "End-host Architecture for QoS-Adaptive Communication," IEEE *Real-Time Technology and Applications Symposium*, Denver, Colorado, June 1998.
[19] T.F.Abdelzaber, E.M.Atkins & K.G.Shin, "QoS Negotiation in Real time systems and its application to automatic flight control," IEEE Real time Technology and Applications Symposium, June 1997.
[20] Pawet Piqtek, Wojciech Greja, "Speed Analysis of a Digital Controller In the Critical Application", Journal of Automation, Mobile Robotics & Intelligent Systems, Vol 3, 2009
[21] J.W.S. Liu.et.al "Algorithms for Scheduling Imprecise Computations", IEEE Computer, Vol 24, No.5, May 1973.
[22] Jestin Rajamony, "Closed Loop Scheduling for Real Time Network Operating System", Proceed ICCES'05 – IITM, India, 2005.
[23] P.R. Blevins and C.V.Ramamoorthy, " Aspects of a dynamically adaptive operating systems", IEEE Transactions on Computers, Vol. 25, No.7, pp. 713-725, July 1976.
[24] D.C. Steere. et. al., " A feedback-driven Proportion Allocator for Real-Rate Scheduling", Operating Systems Design and Implementation, Feb 1999.
[25] Chenyang Lu, J.A. Stankovic, Gung Tao, Sang H. Son " Design and Evaluation of a feedback Control EDF Scheduling Algorithm", Department of Computer Science, University of Virgina.