Constant Bandwidth Server

Muhammad Amjad Bin Abdul Malik
Hochschule Hamm-Lippstadt
B. Eng. Electronic Engineering
Lippstadt, Germany
muhammad-amjad-bin.abdul-malik@stud.hshl.de

***Abstract*—Several techniques and methodologies have been proposed in recent years to enhance the predictability of real-time systems for the better. Constant Bandwidth Server, usually known as CBS, is one of the scheduling systems discussed in this paper. In real-time systems with variable execution requirements, this scheduling strategy is widely used to manage overruns by implementing a strategy to reserve the resources. Various service mechanisms are established and provided in particular to lower the average response time of aperiodic requests while maintaining the scheduling sequence of hard periodic jobs. To illustrate these findings, we must first examine and expound on several fundamental notions that will be employed throughout this study.**

***Index Terms*—real-time system, aperiodic, scheduling, overruns**

I. Introduction

Real-time systems are computing systems that must react within precise time constraints to events in the environment [2]. In respect of this matter, the correct behaviour of the systems does not only be dependent upon the logical value of the computation, but also influenced by the time which the outcomes are produced. A response that happens too late or simply within the specific designed time or period may harm or produce a pointless outcome. Therefore, there is no doubt that real-time computing plays a big and important part in the real world. As we can see clearly throughout the years, the number of complex systems that lean on in either partially or wholly on computer control is expanding widely. Some examples of the applications that uses real-time computing are telecommunication system, nuclear and chemical plant, flight control system, industrial automation, modern medical system, railway system and many more [2].

This topic of real time systems is broad and vast but, in this paper, we will only be focusing on certain algorithms included in the dynamic scheduling, but other important elements will also be explained briefly. The main objective behind this research studies is to learn some efficient algorithms for the joint scheduling of aperiodic requests and hard periodic tasks under the Earliest Deadline First (EDF) policy. Besides, we can also learn and understand the importance of all those stated algorithms which establish a helpful foundation and framework in accommodating a Hard Real Time (HRT) system designer in specifying the most suitable method in accordance with his or her needs, by balancing the efficiency of the algorithms with implementation overhead [8]. This research studies touches on three optimal algorithms that possess different implementation overheads and performances based upon the desired usage. The algorithms that will be included are the Total Bandwidth Server (TBS), Earliest Deadline Latest (EDL) server and not to forget, Constant Bandwidth Server (CBS).

The algorithms stated before did intensify the preceding servers introduced before them greatly in terms of the performance and cost ratio. Not only that, they also can be easily implemented with very little overheads. But before that, it is good to have a grasp on the basic idea on the real time systems. For a better understanding on this topic, the flow of this paper will start with the explanation of the type of tasks that used in the system which have been categorized by the engineers and designers throughout the development, the problems and type of real-time scheduling, the insights for the three servers stated before, the simulation and comparison between servers and finally the implementation of the CBS in a software called UPPAAL.

II. Type Of Tasks

As we all know, in real-time computing systems, there are numerous of complex control applications and tasks that need to be finished off within a certain time constraints. This is what we called as deadlines. If complying the deadline is critical or strict for the operation of system which later may result in a catastrophic ramification if the deadline is not met, then the deadline is regarded to be hard. On the other hand, if meeting the deadline is desirable but missing it would not cause any critical damage, then the deadline is said to be soft. One easy example for soft real time would be multimedia applications such as audio or video streaming as for these applications, missing a deadline only leads to the degradation of the system performance and has no catastrophic consequences. In addition to their criticalness, the tasks are divided into three which are periodic, aperiodic and sporadic. Basically, tasks that are activated or initiated regularly are called periodic tasks meanwhile, tasks which have irregular arrival times are called aperiodic and aperiodic tasks with hard deadlines are called sporadic tasks.

We now contemplate that a real-time system consists of a firm set of periodic tasks and sporadic tasks that may occur dynamically. Each periodic task will generate service requests periodically or simply at a predictable time. The attributes of every periodic task, T_i of a static timing includes its worst execution time q, its period P_i and its critical delay R_i which presume to be equal or less than the period. To the contrary, sporadic tasks arise at the node at unpredictable times. Every sporadic task will be ready to be processed once the request is being granted or accepted on the node [5].

A set of concurrent real-time tasks must be executed by hard real-time system to the extent that all time-critical tasks satisfy the deadlines that have been set for them. Each task requires some data on the computational and other resources like the input or output of a device to carry on the process. The scheduling problem is burdened by the allocation of these resources to satisfy all timing requirements [3]. Now that we already been informed about the basic knowledge of tasks, we will now go through the common scheduling problems and also the type or fractions of scheduling used in real-time scheduling.

III. Scheduling

A real-time scheduling system embodies the clock, the elements used in hardware processing and not to forget, the scheduler itself. Each task has their own schedulability in the system. This means that, depending on the characteristic of the scheduling algorithm, the tasks will first be accepted by the real-time system then fulfilled by the given or specified deadline of the task. This explains that a scheduling algorithm designates how tasks are being processed by the scheduling system. Each task will also be assigned its own description, deadline and an identifier which denotes their priorities. The selected scheduling algorithm determines how a particular task is assigned to its own priorities [9].

A. Static Versus Dynamic Scheduling

Next, we come to the classification of real-time scheduling. A real-time scheduling can be sorted or divided into two parts. We can either have a static or a dynamic scheduling algorithm. Basically, their main difference is that a static scheduler (which also called pre-runtime scheduler) will dictate the priorities of the task before the system runs, meanwhile a dynamic scheduler (which also called on-line scheduler), will be determining the task priorities as the system runs by choosing one from the contemporaneous sets of ready tasks. The tasks are then accepted from the computing environment by the hardware parts in a real-time scheduling system and processed in real-time. The processing state is indicated by an output signal. A task deadline is the amount of time allotted to finish each assignment [9].

This indicates that a static scheduler constructs its decision for the scheduling literally at the time of compilation or technically at the point which a program is converted from source code to machine code. Thus, it requires a complete advance foreknowledge with respect to attributes of the task, for example, the maximum execution times, the precedence constraints, the mutual exclusion constraints, and not to forget the deadlines. On the other hand, for a dynamic scheduler, as the consequence for making its decision for scheduling at running time, this makes it adaptable to a changing task circumstance. This is due to their behaviour of only considering the current task requests. However, it is also worth noting that the amount of time and work required to find a schedule can be tremendous [3]. “Fig. 1” is provided to have a clearer view between the distinctions stated.

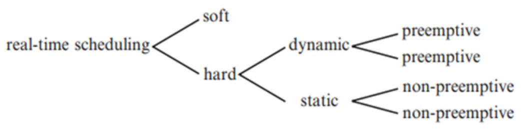

Fig. 1. Fractions of real-time scheduling with comparison between dynamic and static scheduling [3].

Predominantly, the behaviour of the system is non-deterministic. As we can see on the figure shown before, the schedulers used in dynamic scheduler are preemptive while the schedulers used in static scheduler are non-preemptive. In non-preemptive scheduling, the task that is currently being executed will not gong to be interfered until it allows the allocation of the resources to be released or used. Hence, this makes non-preemptive scheduling very reasonable in a scenario where many short tasks are needed to be executed. Contradictorily, in preemptive scheduling, it is possible to have an interrupt during the process. As for example, the currently executing task could be preempted or interrupted if there exist a more urgent task which requests for a service [3].

B. Scheduling Problems

Nowadays, most industrial control applications with real-time demands are made up of tasks from a variety types and constraints. These constraints require both types of processes, periodic and aperiodic. This may also differ for their criticality. The main objective of the kernel is to guarantee the schedulability of all critical tasks in worst-case conditions and give reasonable response times on the average for not only soft but also non-real-time tasks or process when dealing with hybrid task sets. As what is stated in the Murphy’s Law, ”Anything that can go wrong will go wrong”, meeting the requisite deadline will not always work possible to be achieved. On that account, extensive research of the scheduling algorithm is necessary to be organized for further verification. Dynamic scheduling algorithm can be implemented using two different techniques. It is either by assigning the deadline of the task based on their priority (earliest deadline) or assigning the completion time for each task by subtracting the processing time from the deadline (least laxity). As the consequence, we must understand the deadlines and the required task execution time in advance to ensure the processing elements execution times are being utilized efficiently [9].

One of the major problems that we may encounter in the analysis of sporadic task systems is the concerns of unpredictability of the task requests. For periodic task systems, the set of task requests to be encountered is known a priori. However, there is by no means to determine which set of task requests would be encountered in irregular task systems. The key scheduling issue thus is on-line feasibility testing. The system must decide if a new sporadic task can be accepted when it arrives at the node. Acceptance signifies that the job can be completed in the time between its arrival and its deadline without jeopardizing the deadlines of periodic tasks or previously accepted sporadic activities. The accepted task is then queued until it is selected to be processed by the local scheduler. Otherwise, another module of the scheduler in responsibility of dispatching this task to another computer in the distributed system can take refusal into account [5] and this is where we will study and observe the importance of server which closely related to the term of 'Temporal Isolation'.

IV. Server

The demand for temporal isolation is undeniably becoming a main important discussed topics as the aim of this method of scheduling was to combine and merge not only all real-time tasks together, but also including the non-real-time tasks as well. Resource Reservation represents the most powerful technique of scheduling which is used to achieve such a desired property [1]. Therefore, it is certainly useful in preserving hard tasks from overruns that are caused by the soft tasks if we are dealing with a composed set of both soft and hard tasks at one time [4]. An essential property that must be ensured intending to run multiple concurrent tasks in computing system is temporal protection. This can prevent unexpected overruns that happens in a task from giving a bad impact on the execution of other tasks. In terms of hardware part, this is achieved by having each task or set of tasks assigning a fragment of the CPU and scheduling in a manner that it would acquire more bandwidth than it has been allotted. By using this approach, the capacity of processor is viewed as a finite resource that can be reserved, just as disk blocks and physical memory [2].

As for the implementation of resource reservation, a real-time server, also known as reservation server, will be assigned to each of the application in a system. In a server, there must be a budget which denotes by Q, period which denotes by P and α which indicates the ratio of Q/P, and it is called the bandwidth of the server. Later, we will come across again with those terms stated before and also some equations related to them when we dive more into each type of servers' insights. It is known that when we schedule soft aperiodic tasks and hard aperiodic tasks, we may encounter some issues while managing them under dynamic priority assignments [4].

Therefore, several different service methods or servers have been introduced to solve these problems by implementing different approaches but with the same purpose which is to reduce the average response time of aperiodic requests without jeopardizing or make concessions on the schedulability of the existing hard periodic tasks in the system. Some of the popular dynamic servers proposed are Constant Bandwidth Server (CBS) alongside with other two predecessor algorithms which are the Total Bandwidth Server (TBS) and Earliest Deadline Late Server (EDL). We will be discussing on all three of algorithms stated because they are quite interconnected with each other, but we will only be emphasizing on CBS. In general, they are somehow quite similar to each other for the reason that all of the periodic tasks in the system will be scheduled or handled by using Earliest Deadline First (EDF) algorithm. Concerning the fixed-priority assignments, dynamic scheduling are characterized by higher schedulability bounds, since it can increase the size of aperiodic servers, enhance aperiodic responsiveness, thus making the processor to be better utilized [6].

A. Total Bandwidth Server (TBS)

Total Bandwidth Server or TBS uses an approach which enhancing the response time of an aperiodic task by assigning a possible short deadline to each of the aperiodic request. The assignment needs to be completed in the manner that the whole processor utilization of the aperiodic load would never go beyond the predefined maximum value which denotes by U_s. In other words, the server basically works as if each time an aperiodic task requesting to enter the system or if any aperiodic request is initiated to system, the total bandwidth of the server is immediately assigned to it whenever possible [6].

As a matter of fact, TBS is one of the simplest servers to implement in dynamic priority server. This is due to the fact that it only needs to monitor the deadline which has been assigned to the last aperiodic request, so as to apply the appropriate deadline to the newly issued request. The request then can be sequenced into the ready queue and this is when the EDF will process it just like any other periodic task. Hence, the overhead is only due to the increased length of the ready queue if several aperiodic requests are pending concurrently. This problem can be overcome by organising a separate First In First Out (FIFO) queue for aperiodic requests, and inserting only the first one into the ready queue and so this extent, the overall overhead is practically negligible [2].

B. Earliest Deadline Late Server (EDL)

We can see that just by implementing a simple design of algorithm, TBS still capable of achieving great response times for aperiodic tasks. Nevertheless, we could still acquire more fulfillment by improving some other important factors. EDL server is introduced to improve TBS server and the improvement of TBS that has been applied into EDL server is made by taking the advantage of the laxity or what we could say as leniency. As we know, as request arrives at the node of the system, the current executing periodic task have enough effective laxity, as in the interval between the completion time and the deadline to be safely preempted. Then, the idle times of an EDL scheduler are used to schedule aperiodic requests as soon as possible, postponing the execution of periodic tasks. Generally, when there are no aperiodic activities in the system, the periodic tasks are scheduled according to the EDF algorithm [8].

As it has been pointed out, the EDL server schedulability's synthesis is quite direct and straight. However, EDL server cannot be classified as a practical approach due to the complexity of computing the idle times at which each new aperiodic task arrives. This computation must be done each time considering whether the periodic task has been partially executed or maybe already done and completed when the aperiodic task arrive [8].

C. Constant Bandwidth Server (CBS) Insights

Although the algorithms described in the previous sections are optimal, but they have too much overhead to be considered as a practical technique. Another TBS algorithm's key flaw is that it did not employ a server budget to manage the execution of aperiodic tasks. Instead, it only relies based on the information gained from the worst-case computation time enumerated by each task that arrives in the system. If those details are unavailable or unknown, which most likely happen because of highly variable execution times, then hard tasks are vulnerable of transient overruns that occurs in the soft tasks and will probably fail to meet their deadlines. Thus, in these instances, CBS algorithm may come in handy as it has an equivalent performance with TBS and it also has temporal isolation.

In this section we will be exploiting the idea behind CBS server. This server uses a different strategy called the bandwidth reservation strategy and thus making it more efficient compared to other servers. It guarantees that, if U_s is the fraction of processor time assigned to a server (namely its bandwidth), its contribution to the total utilization factor is no greater than U_s, even in the presence of overloads. This feature is unavailable in TBS implementation, whose actual contribution is limited to only U_s, in contingent of all the served jobs execute no more than the worst-case execution time that has been proclaimed.

The method used in CBS can be described briefly using an analogy of as new job enters the system, it is assigned to an appropriate scheduling deadline to make sure the demand does not exceed the allocated bandwidth. It is then enqueued in the ready queue of EDF. Whenever a task attempts to execute more than it is expected to execute, its deadline will be postponed and hence making its priority decreases. This is done to reduce the interference on the other tasks. Bear in mind that the task will still remain eligible for execution even though the deadline has been shifted back. This is why CBS is said to be a conserving algorithm because it basically accomplishes the available slack in an efficient way. Hence this will provide a better responsiveness not only towards non-work conserving algorithms but also towards other reservation approaches that schedule the extra portions of jobs in background [1].

There will be no isolation between the tasks they are being managed by only a single server. This would also indicate that all the tasks in the subset will have to share the same bandwidth. In spite of that, other tasks outside the subset will be protected against overruns if such thing happens in the subset. That being the case, the server should be designed cautiously in terms of rules used for the deadline assignments to prevent any hard deadline to be missed when the system runs.

Just like what has been mentioned in the system server earlier in paper, CBS has quite the same basic concept, but we use different annotations or letters to denotes the budget and the period of the server. In CBS, server budget is denoted by c_s whereas the maximum budget and period of the server are denoted by Q_s and T_s respectively. The bandwidth of the server, U_s is the ratio between Q_s and T_s. As an additional detail, a fixed deadline $d_{s,k}$ is included in the server in which the dynamic deadline $d_{i,j}$ will be equal $d_{s,k}$, whenever a job $J_{i,j}$ is served. Concurrently, the value of c_s will be decreases each time it is used when there exist a served job taking into account that the value of c_s must be greater than zero. If the server budget, c_s is finished or empty, it will be recharged at its maximum value Q_s and a new server deadline will be created as $d_{s,k} + 1 = d_{s,k} + T_s$ [2].

If there exist pending jobs, then a CBS is meant to be in an active state meanwhile when there are no pending jobs that need to be executed, then the state is in idle mode. Once a job $J_{i,j}$ arrives in which the server is active, the job request will be enqueued in a pending jobs queue in the manner of First In First Out (FIFO). In opposition to that, if a job $J_{i,j}$ arrives while server is in the idle state, we need to consider either the value of c_s is greater than the value of $(d_{s,k} - r_{i,j})U_s$ or not. If it is greater, the server then generates a new deadline $d_{s,k} + 1 = r_{i,j} + T_s$, or else the job will be served with the last server deadline $d_{s,k}$ using the balance of the current budget c_s and deadline. For a clearer view upon this algorithm, an example of a scenario with a diagram is provided below to help in understanding the process step by step [2].

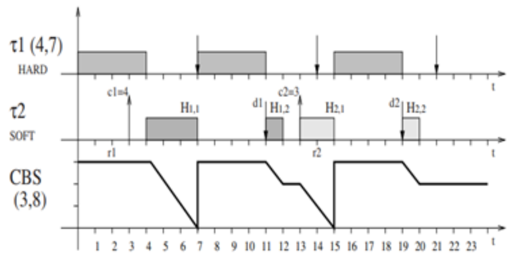

Fig. 2. CBS implementation [2].

From "Fig. 2", we can see that the tasks are divided into two which are hard periodic task which denotes by τ_1, and soft task scheduled together which denotes by τ_2. Both tasks are served by a CBS with a maximum budget, Q_s of 3 and period, T_s of 8. Hard periodic task, τ_1 is set with a computation time, c_1 of 4 and period, T_1 is equal to 4. The first job for the soft task, $\tau_2(J_{2,1})$ is initiated at (r1) = 3 while the server is inactive state, and it requires 4 units time of execution. Since $c_s \geq (d_0 - r_1)U_s$, the job is assigned a deadline $d_1 = r_1 + T_s = 11$ and c_s is recharged till in reaches the maximum budget which is 3. Since the budget, c_s is being fully utilised at time t = 7, it will later be recharged again. Therefore, the server will generate a new deadline, $d_2 = d_1 + Ts$, thus making the new deadline, $d_2 = 19$. Since the server deadline is postponed, τ_1 will be executed until it completes as it emerges as the task with the earliest deadline [2].

Subsequently, $\tau2$ is now resumed and the first job that is being postponed before is completed at time t = 12, and making the budget left equal to 2. The second job of task τ_2 arrives at time r_2 = 13 and imposes execution time of 3 units. Since c_s is smaller than $(d_2 r_2)Us$, then it is possible for the last server deadline d_2 to handle job $J_{2,1}$. However, the server budget is drained at t = 15, hence a new server deadline, d_3 is generated making the deadline being shifted backwards to t = 27 which calculated using the previous formula $d_3 = (d_2 + T_s)$. Not to forget the server budget, c_s is once again being recharged until it achieves the maximum value which is 3. One more time, τ_1 becomes the highest priority task and executes until time t = 19. As job $J_{1,3}$ finally completed, τ_2 is now able to execute the task to complete job $J_{2,1}$ at time t = 20. This means that the server budget is left to be 2 [2].

D. Constant Bandwidth Server Properties

As we can observe so far, methods introduced in CBS provide a lot of useful qualities properties. The mechanism used is appropriate and convenient in assisting applications that includes complex computation. One of the most critical properties is the isolation property. This property can rigorously be expressed according to the theorem and lemma stated below.

Theorem:

The CPU utilization of a CBS S with parameters (Q_s, T_s) is $U_s = Q_s/T_s$, independently from the computation times and the arrival pattern of the served jobs [2].

Lemma:

Given a set of n periodic hard tasks with processor utilization Up and a set of m CBSs with processor utilization, the whole set is schedulable by EDF if and only if $U_p + U_s \leq 1$ [2].

Due to the general isolation property, we can utilize a bandwidth reservation approach to devote a portion of the CPU time to soft tasks which have long computation times. The most important implication of this discovery is that soft tasks can be scheduled alongside hard tasks without compromising the priori guarantee, even if the soft task execution timings are unknown or the soft requests exceed the predicted load. Any time saved due to early completions is automatically reclaimed by CBS. This is because, if the budget is depleted, it is always refilled at its maximum value and the deadline of the server will be postponed. As a result, the server retains its eligibility, and the server budget can be used for the pending requests with the current deadline [2].

E. Simulation Results Comparing TBS and CBS

Next, we come to the simulation result when comparing TBS and CBS to reassure the findings that we have learned so far from this paper. Since we stated that CBS brings improvement and have better performance than TBS, thus in this segment, we will see the whether the CBS is efficient enough to handle the soft aperiodic request. In our study, we discovered that CBS service technique enhanced the responsiveness by quite a large ratio when compared to TBS.

Basically, the test that is being run is comparing the mean tardiness undergo by the soft tasks as CBS and TBS are handling and managing them. The periodic tasks utilization factor which denotes by (UHard) is set to 0.6 in this experiment. Based on the graph below, we can see that CBS surpasses TBS in handling the tasks which has high variance of execution times. From the results given in "Fig. 3", due to the worst-case supposition, TBS may cause the processor to be underutilised [2].

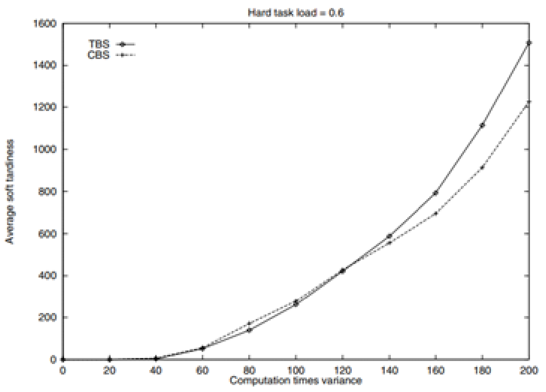

Fig. 3. Result on comparing TBS with CBS [2].

F. Simulation Results Comparing TBS and CBS

Next, a simple implementation of CBS can be simulated using a software called UPPAAL and in this section we will see the basic steps of methods used in CBS service mechanism. The steps can be divided into four different states. It is important to note that the simulation only covers the basic idea of CBS and not touching into the critical parts.

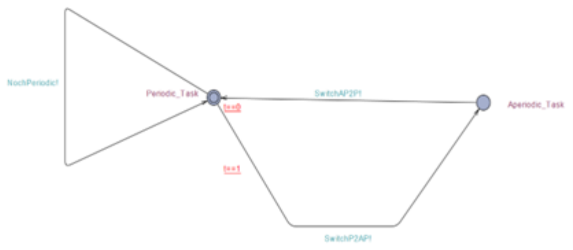

Fig. 4. Switching task processing [10].

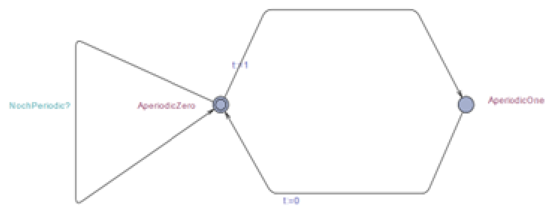

Fig. 5. Aperiodic is being handled [10].

Based on “Fig. 4” and “Fig. 5”, we can observe that the periodic task will always be chosen to be the executed if there is no request initiated by the aperiodic task to be executed. If there is no request from the aperiodic task, it will merely loop into the periodic task, as shown in the diagram. In order to switch from executing periodic task to executing aperiodic task, the mode must be changed from idle to active. The CBS mode on the other hand, must first identify whether the server budget is available or not in order to execute the aperiodic task. Theoretically, CBS will only be in active state if there exist a pending jobs. This will be explained by using the figures below.

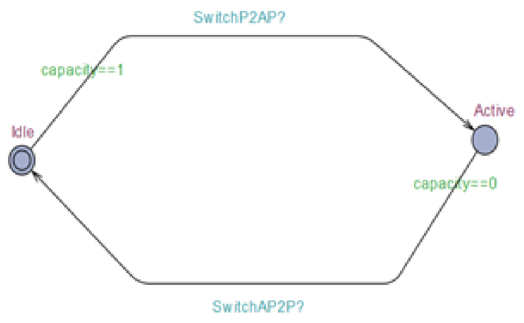

Fig. 6. CBS mode changes [10].

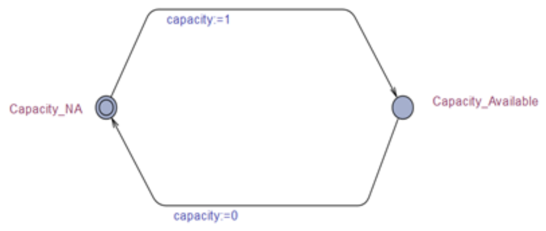

Fig. 7. State changes of CBS server budget (capacity) [10].

In “Fig. 6” and “Fig. 7”, the state and mode changes of CBS are shown to indicate that CBS mode may change if the server capacity or server budget is available which denoted by (capacity = = 1). It will then switch again to the idle state if the server budget is exhausted or already fully being utilized by the served tasks which denotes by (capacity = = 0). To run this simulation, a GitHub link is provided on the reference section.

V. Conclusion

Short to say, I have learned a lot through this research study that has been conducted. Not only I got the chance to learn on the topic that is assigned to me, but I also earned some knowledge on a few other topics, for example, EDL and also TBS which I could say the precedent algorithm in a timeline or a step of improvement before coming to CBS existence. It is regarded as an example or guide to be considered in this topic for a clearer understanding. Apart from that, this research study gives me an opportunity for me to strengthen my knowledge on Real Time System which I believe one of the most critical parts and important to comprehend in pursuing my course of study.

References

[1] C. W. Mercer, S. Savage, and H. Tokuda, “Processor capacity reserves for multimedia operating systems,” in Proceedings of IEEE international conference on Multimedia Computing and System, May 1994.

[2] Abeni, L. and Buttazzo, G., 2004. Resource Reservation in Dynamic Real-Time Systems. Real-Time Systems, 27(2), pp.123-167.

[3] Kopetz, H., 2004. Real-time systems. Boston, Mass.: Kluwer Acad.

[4] Biondi, A., Melani, A. and Bertogna, M., 2022. Hard Constant Bandwidth Server: Comprehensive formulation and critical scenarios. [online] Ieeexplore.ieee.org. Available at: ¡https://ieeexplore.ieee.org/document/6871182/¿

[5] Silly, M., 1995. A Dynamic Scheduling Algorithm for Sporadic Tasks under Resource and Timing Constraints. IFAC Proceedings Volumes, 28(5), pp.247-252.

[6] Buttazzo, G., 2011. Hard real-time computing systems. New York: Springer.

[7] Abeni, L., Lipari, G. and Lelli, J., 2015. Constant bandwidth server revisited. ACM SIGBED Review, 11(4), pp.19-24.

[8] Spuri, M. and Buttazzo, G., 1994. Efficient aperiodic service under earliest deadline scheduling. [online] Ieeexplore.ieee.org. Available at: ¡https://ieeexplore.ieee.org/document/342735/¿.

[9] Audsley, N., Burns, A., Richardson, M. and Wellings, A., 1991. Hard Real-Time Scheduling: The Deadline-Monotonic Approach. IFAC Proceedings Volumes, 24(2), pp.127-132.

[10] Malik, A., 2022. Real-Time-System/CBS.xml at main · Amjad-Malik/Real-Time-System. [online] GitHub. Available at: ¡https://github.com/Amjad-Malik/Real-Time-System/blob/main/CBS.xml¿.