Adaptive Resource Synchronization In Hierarchical Real-Time Systems

Tom Springer, Steffen Peter, and Tony Givargis
Center for Embedded Computer Systems
University of California, Irvine, USA
{tspringe, st.peter, givargis}@uci.edu

ABSTRACT

In this paper we outline the Adaptive Resource Allocation Protocol (ARAP) as an improved resource synchronization algorithm for hierarchically scheduled real-time systems. ARAP exploits knowledge about task utilization, using a proportional-integral-derivative (PID) controller, to estimate required resource bandwidth and improve scheduling decisions. Our analysis and experiments with RTSIM show that ARAP provides better temporal isolation and resource utilization during periods of transient overload compared to state-of-the-art resource synchronization algorithms. Implemented as part of *VxWorks*, the results are confirmed using an avionic system, for which ARAP substantially reduced the number of hard real-time deadline misses.[1]

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-Time and embedded systems.

General Terms

Algorithms, Performance, Reliability.

Keywords

Real-time systems, hierarchical scheduling, resource sharing, operating systems.

1. INTRODUCTION

Modern embedded and Cyber Physical Systems such as smart cities, autonomous automotive systems or the smart electrical grid are composed of various subsystems which are often developed independently. A common requirement for such systems is temporal isolation, meaning that the temporal behavior of one subsystem should never adversely affect the temporal behavior of another subsystem.

This requirement for temporal isolation is challenged by two major properties we address in this paper. First, the amount of data and resulting computation times may vary significantly. Such variations are not easy to predict neither in their magnitude nor their duration. The second challenge is that the temporal behavior of the subsystems is often influenced by resources which are shared among tasks. For instance, if an interface to a sensor device is locked by a task in one subsystem the temporal isolation of other subsystems requiring access to the sensor could be violated.

To address these challenges a variety of solutions have been proposed. The most common approach is "over-engineering." Historically, systems were designed and considered "safe" with a utilization factor U of less than 70% [18], while systems with a higher utilization were deemed "unsafe". Traditional resource access protocols, such as Priority Inheritance Protocol (PIP) [4], the Priority Ceiling Protocol (PCP) [4] or the Stack Resource Policy (SRP) [5] are not a viable solution either because there is no system wide visibility across subsystems. To cope with these issues, the Hierarchical Scheduling Framework (HSF) has been introduced to simplify the development and integration of embedded systems and shown to be particularly useful in the area of open systems [1].

The primary goal of hierarchical scheduling is to bind the temporal behavior of those applications whose execution times deviate considerably, allowing for the predictable integration of the various subsystems. However, in order to provide this temporal isolation the basic HSF model assumes the absence of shared resources. The problem with this assumption is that most embedded systems share global resources and need to be synchronized. While traditional resource access protocols can be used to synchronize resources locally, within a subsystem, global resource access presents added challenges such as the unpredictable holding times between globally shared resources.

In this paper we present the Adaptive Resource Allocation Protocol (ARAP) which is a new resource synchronization policy for globally shared resources in an HSF, focused in a single processor environment. ARAP utilizes knowledge about previous task utilization, by applying the concept of control theory using a proportional-integral-derivative (PID) controller, to adapt to changes in the system. The primary benefit of this new strategy is that the system can now adapt to computational changes dynamically. The result being that temporal isolation is maintained between subsystems even during periods of overload.

We applied ARAP in an actual embedded system that was experiencing overload conditions resulting in missed deadlines due to soft and hard real-time tasks sharing the same resource. Our experiments demonstrated the benefit of ARAP specifically during overload conditions and as a result deadline misses for hard real-time tasks were eliminated. Other contributions of this paper include: the performance analysis of ARAP as compared to other synchronization protocols, the ARAP protocol implemented as part of the resource manager class in an open source real-time scheduling simulator, a simulation model of transient overload conditions for performance analysis and the first time implementation (to the best of our knowledge) of an HSF as part of the *VxWorks* real-time operating system.

The remainder of the paper is structured as follows: In Section 2 we provide an overview of hierarchical scheduling. Related work is reviewed in Section 3 while Section 4 presents the adaptive resource access mechanism used in our protocol. Performance analysis and results are provided in Section 5. Section 6 describes the simulation environment and Section 7 presents the implementation in an actual embedded system application.

2. PRELIMINARIES

This section presents the overall architecture of a hierarchical scheduled system as well as the definition of a system overload condition.

[1] EWiLi'14, November 2014, Lisbon, Portugal. Copyright retained by the authors.

2.1 Hierarchical Scheduling

The basic framework of a hierarchically scheduled system [2] [3] is composed of multiple applications (subsystems) where each application could be composed of multiple tasks (see Figure 1). A *global* scheduler controls which application can use the processor while the *local* scheduler determines which application's task should actually execute. Every application is allocated a separate service manager, known as the *server*. Each server is allocated a CPU capacity reserve, which is assigned as a pair (Q_i, P_i) where Q_i is defined as the time quantum and P_i is defined as the period.

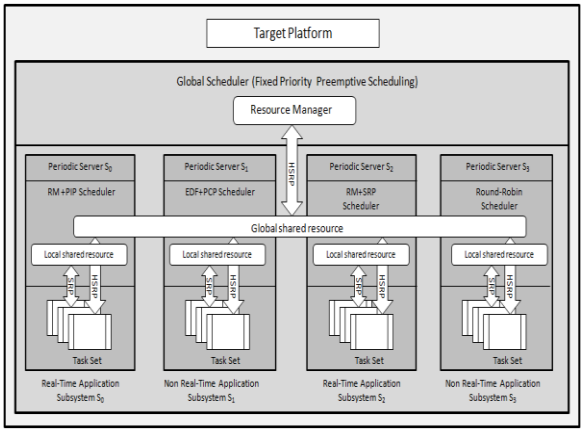

Figure 1: Hierarchical Scheduling Framework

Each task gets to execute for its assigned time quantum Q_i, when the task's time quantum Q_i is exhausted the task is blocked until its next period. In effect, the server functions as an independent processor virtually limiting the bandwidth of each application.

2.2 Resource Sharing

Resource sharing in an HSF can be classified as either local or global. Tasks that share resources within the same subsystem are considered local resource sharing. Tasks that share resource across applications are classified as global resource sharing which requires the resource be protected at the local as well as global level. Therefore a task that locks a global resource will also cause its server to lock the resource. This creates an added complication of increased resource holding times due to the server budget exhaustion.

Researchers have proposed several solutions [8][6] to the problem of this added delay in critical sections due to server budget exhaustion. One such approach called budget check checks to see if there is sufficient server budget before allowing a task to enter a critical section. If the budget is insufficient the task is not granted access to the resource until the next budget replenishment. In another approach the task is allowed to enter a critical section without checking for a sufficient budget. As a result, if the budget is exhausted while still inside the critical section the task is just allowed to continue and consume extra budget until the end of the critical section.

2.3 Overload Conditions

The problem of server budget exhaustion while a task is inside a critical section can be amplified during periods of overload because it could unnecessarily increase the time a critical task would have to wait for the resource. As a consequence, task overruns can result because tasks execute longer than expected which can have an increasingly negative impact on resource holding times. Thus, before we proceed any further, we will take the time to provide a brief definition of an overload condition. Overall system load is defined in [18] as the equivalent to the processor utilization factor *U*:

$$\rho = U = \sum_{i=1}^{n} \frac{C_i}{T_i} \quad (1)$$

where C_i is the computational time and T_i is the task period. A load value of $\rho > 1$ means the system is overloaded (requested computation time exceeds the available processor time). Consequently, there are two broad classifications of overload conditions also defined by the authors in [18]:

1. Transient overload is defined for a limited time where the average load is $\rho^{avg} \leq 1$ but the maximum load is $\rho^{max} \geq 1$.
2. Permanent overload is defined for an unpredictable duration where the average load is $\rho^{avg} \geq 1$.

It is important to note that in this paper our approach was to only consider transient overload conditions. As a result, the primary benchmark of our work is to practically eliminate hard real-time task deadline misses and manage soft real-time task misses so that the system can recover gracefully from an occasional transient overload condition.

3. RELATED WORK

Initially an HSF was proposed by authors in [3][9] as a means to perform composability analysis for open systems development. The motivation being that it can quickly become intractable to accurately verify the timing behavior of the embedded system as the complexity increases. The approach was to verify the timing behavior of each individual subsystem independently then compose each subsystem into the overall system. However, typical embedded systems are not entirely independent because subsystems may need to share resources which are why previous research on HSFs was extended to include schedulability analysis of semi-independent real-time components [8] [5].

The SIRAP [8] protocol was developed for fixed-priority preemptive scheduling while the BROE [10] protocol was developed for dynamic-priority scheduling. Both protocols use a form of budget check to determine if there was enough budget left to enter the critical section. If the remaining budget was deficient to complete the critical section the task was blocked from locking the resource until the next budget replenishment. The limitation with the budget check approach is that the critical section execution time is based upon worst-case analysis. This could lead to resource under utilization due to conservative WCET estimations. Additionally, *a priori* knowledge of the WCET for a critical section is required which is often difficult to evaluate in applications with variable execution times.

Hierarchical scheduling with resource sharing HSRP [2] and later extended to OPEN-HSRP [10] utilized a budget overrun approach to reduce the resource holding times during budget expiration. While this approach does provide better flexibility for applications with variable execution times there are some disadvantages. One such drawback is that even though a task is allowed to overrun its budget there still has to be a limit placed upon the maximum overrun time. In order to prevent unbounded blocking a task is forcefully preempted if it is still holding the resource during the next budget replenishment. This leads to limitations being placed upon the types of shared resources used to those that can safely be aborted to relatively short critical section execution times. Another consideration is because a task can overrun its budget the strict temporal isolation between subsystems could be violated. It is for these reasons that HSRP based systems are typically only used for soft real-time systems.

Other recently published work, known as RRP [11], took a different approach to resource locking before budget exhaustion. Instead of performing a budget check the task is allowed to enter the critical section and unlike HSRP if the budget has expired the task is simply preempted and rolled back. The RRP protocol improved the average case response times and task schedulability compared to SIRAP and the OPEN-HSRP protocols. However, the limitation with RRP is that it can only be used with shared resources that can be safely rolled back (e.g. databases).

Our approach does not rely on WCET analysis of critical section executions times but instead uses feedback which is more flexible since it represents the actual operating environment. Previous research has proposed using feedback to manage CPU scheduling reservations [12] [13] then extended it to include application defined Quality of Service (QoS) parameters. However, ARAP is the first to apply a feedback mechanism to resources that are shared across subsystems in a hierarchically scheduled system. Past research has demonstrated [14][15] that previous task behavior is a valid indicator for future task behavior and because our method incorporates feedback ARAP is better positioned to respond to transient overload conditions. Consequently, SIRAP is too conservative by using static WCET analysis while OPEN-HSRP overrun mechanisms could affect the deadlines of higher priority tasks specifically during periods of transient overload.

4. ADAPTIVE RESOURCE ACCESS PROTOCOL (ARAP)

This section provides an overview of ARAP which is a resource access protocol that synchronizes access to shared resources in a hierarchically scheduled system. In this paper we only consider shared memory as the mutually exclusive resource. However, ARAP could also be extended to include other shared resources (e.g. memory-mapped areas, device registers, and peripheral devices) as well. The access to these resources are performed as part of a critical section and protected by a semaphore.

4.1 Protocol Description

Similar to the SIRAP and OPEN-HSRP protocols ARAP utilizes a two-level hierarchy for resource management. Resources that are shared within a subsystem are managed with SRP and resources that are shared across subsystems are managed with an extended version of HSRP. The overall sequence of actions for ARAP is provided by the flowchart depicted in Figure 2.

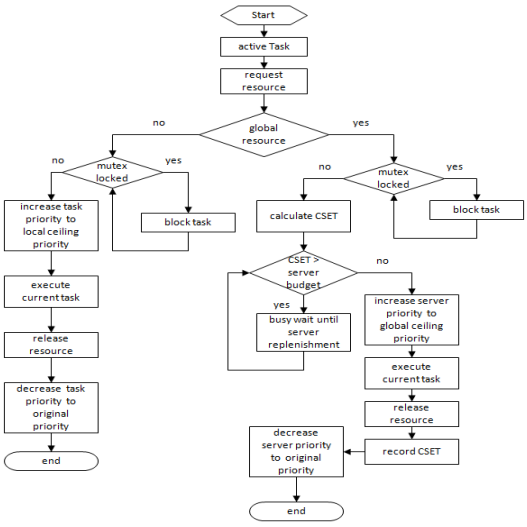

Figure 2: ARAP architecture flowchart

4.1.1 Budget Exhaustion

The primary difference between the various resource access protocols in an HSF is how the budget exhaustion of a subsystem's server is handled. As mentioned in Section 3 the SIRAP protocol performs a budget check while OPEN-HSRP permits budget overflow. Similar to the SIRAP protocol our method also performs a form of budget check. However, instead of using a static *a-priori* calculation of the Critical Section Execution Time (CSET) ARAP incorporates feedback to estimate the next resource holding time $h_{i,j}$ for that critical section execution instance.

The primary benefit of incorporating feedback is that the system can dynamically adapt to changes in *CSET* specifically during periods of transient overload. In this situation the SIRAP protocol tends to be too conservative because with SIRAP the overload condition would have to be factored into the *CSET* calculation resulting in task under utilization. While the OPEN-HSRP protocol can better adapt to overload conditions the overrun mechanism could adversely affect the response times of higher priority tasks. Therefore, our approach leads to a more robust and better utilized system with a higher degree of determinism.

4.1.2 Feedback Mechanism

Our feedback architecture is implemented as part of the kernel (see Figure 3) and consists of a PID which is used to estimate the execution time for a task executing inside a critical section. The output or observed value of the PID is the *estimated error ratio (ER)* which is defined as the ratio between the actual measured critical section execution time and the previous window of past error ratios. The *semaphore* request mechanism is used as the actuator of the system determining whether a task is granted access to the critical section. The control action is performed by either allowing the task to acquire the semaphore or to block the task waiting on the semaphore. If a task τ_i requests a semaphore (e.g. *srp_wait ()*) it has to pass the budget check test to acquire the semaphore. The budget check test uses information from the PID controller as well as information from the scheduler to determine if there is enough remaining budget to complete the critical section. To apply the feedback control ARAP uses the PID controller to compute $\Delta CSET$ which is used to project the next critical section execution time based upon the *ER*. Using the basic form a general PID controller [19] defined as:

$$\Delta CSET(t) = -C_P ER(t) - C_I \sum_{IW} ER(t) - C_D \frac{ER(t-DW)-ER(t)}{DW} \quad (2)$$

where C_p, C_I and C_D are the PID control parameters, *IW* is the integration window and *DW* is the size of the differentiation interval.

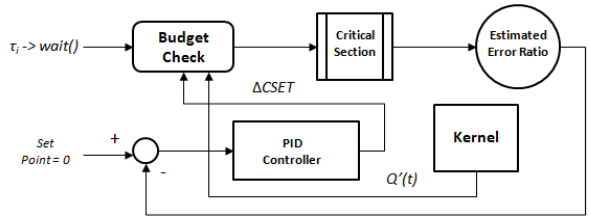

Figure 3: Structure of feedback architecture

4.1.3 Implementation Considerations

The implementation of ARAP is similar to SIRAP but the locking operations (*srp_wait*, and *srp_signal*) are modified to utilize the feedback mechanism. The locking operation is performed by the *srp_wait* function so when a task tries to acquire a resource the

local scheduler performs a check to determine if there is enough budget to complete the critical section. At the semaphore request time t let the function *calcRHT()*, which utilizes the PID controller, calculate the projected resource holding time so that $h_{i,j} = calcRHT()$. At the same time let the function *getCurBudget()* get the subsystem server's remaining budget such that $Q'_s(t) = getCurBudget(t)$. If the task's projected resource holding time $h_{i,j} < Q'_s(t)$ then the task is allowed to lock the resource and execution continues according to the rules of HSRP. On the other hand, if $h_{i,j} > Q'_s$ then the task is not permitted to lock the resource until the next subsystem budget replenishment, this is known as *self-blocking*. The release operation is performed by the *srp_signal* function which signifies the completion of the critical section. The time that is spent in the critical section is used as the feedback to the PID controller. At the semaphore release time t' the function *recordRHT(* $h_{i,j}(t)$*)* records the actual time spent in the critical section.

5. PERFORMANCE ANALYSIS

This section provides the background for the performance analysis of ARAP as part of a hierarchically scheduled system (equations are applied in subsection 5.3). Given that both ARAP and SIRAP perform a budget check the analysis performed for SIRAP [8] can be applied to ARAP as well. The difference is that ARAP uses a projected CSET based upon feedback whereas SIRAP uses static *a-priori* CSET.

5.1 Local Performance Analysis

According to the authors in [7] each subsystem $S_s \in S$ is schedulable if

$$\forall_{\tau_i}\ 0 < \exists t \leq D_i\ rbf(i,t) \leq sbf_{\mathrm{T}}(t) \tag{3}$$

where $sbf_{\mathrm{T}}(t)$ is the supply bound function used by authors [8] to calculate the minimum CPU allocations required during an interval of time. Authors in [7] presented a periodic processor model to characterize the allocations defined by what they called the *virtual processor model* represented as $\mathrm{T}(P,Q)$. The supply bound function (see Figure 4) of the virtual processor model is defined as:

$$sbf_{\mathrm{T}}(t) = \begin{cases} t-(k+1)(P-Q), if\ t \in W^{(k)} \\ (k-1)Q, \qquad\qquad otherwise \end{cases} \tag{4}$$

where $k = max(\lceil t-(P-Q)/P \rceil, 1)$ and $W^{(k)}$ is defined as the interval $\lceil (k+1)P-2Q, (k+1)P-Q \rceil$.

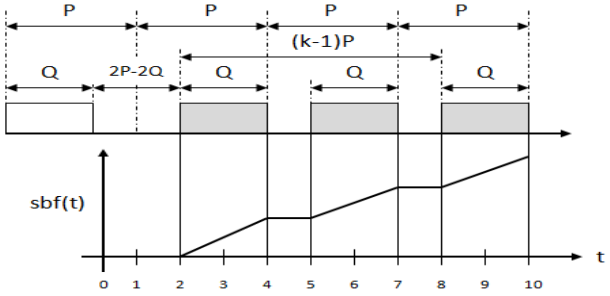

Figure 4: Supply bound function virtual processor model T(3,2), k=3

The request bound function $rbf(i,t)$ of a task τ_i is defined as:

$$rbf_{fp} = C_i + I_S(i) + I_H(i,t) + I_L(i) \tag{5}$$

where C_i is the WCET of task τ_i, $I_S(i)$ is the maximum self-blocking for task τ_i, $I_H(i,t)$ is the interference from tasks with a higher priority than τ_i and $I_l(i)$ is the maximum interference by tasks with lower priority than τ_i which share a global resource, such that:

$$I_S(i) = \sum_{k=1}^{o} h_{i,k} \tag{6}$$

$$I_H(i,t) = \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil \left(C_j + \sum_{k=1}^{o} h_{j,k}\right) \tag{7}$$

$$I_L(i) = \max_{j=i+1,\dots n}\left(2 \max_{k=1,\dots o}\left(h_{j,k}\right)\right) \tag{8}$$

The resource holding time $h_{i,j}$ for τ_i is defined as the maximum critical section execution $c_{i,j}$ plus the interference from the tasks with a higher preemption level than the ceiling of the resource during the CSET of $c_{i,j}$. Such that $h_{i,j}$ is computed using $W_{i,j}(t)$ as follows:

$$W_{i,j}(t) = c_{i,j} + \sum_{l=ceil(c_{i,j})+1}^{u} \left\lceil \frac{t}{T_l} \right\rceil C_l \tag{9}$$

where $ceil(c_{i,j})$ is the ceiling of the resource accessed within the critical section $c_{i,j}$ and C_l, T_l are the worst-case execution time and period of the task that has a higher preemption level than $ceil(c_{i,j})$. The value u represents the maximum resource rc_j within a subsystem.

5.2 Global Performance Analysis

For global scheduling analysis the virtual processor model can be extended to a global model $\mathrm{T}_s(P_s, Q_s)$ where multiple subsystems S_s can be verified according to equation (3). Therefore, the schedulability test for a fixed priority global scheduler is defined as:

$$W_{i+1} = Q_i + B_k + \sum_{j=1}^{i-1} \left\lceil \frac{W_i}{\Pi_j} \right\rceil (C_j) \tag{10}$$

where B_k of subsystem S_k is the maximum resource holding time with a preemption level less than π_k.

5.3 Performance Results

Schedulability analysis for HSRP was performed by authors in [2] which is very similar to the SIRAP analysis which excludes self-blocking but has to consider the overrun mechanisms. Similar to SIRAP as well the analysis for OPEN-HSRP is extended from equation (3) as follows:

$$\forall \tau_i\ \exists t: 0 < t \leq D_i, rbf_{fp}(i,t) + b_i \leq sbf(t) \tag{11}$$

where b_i is the maximum blocking time when τ_i is blocked by a lower priority task. The supply bound function $sbf(t)$ is defined by equation (4) and the request bound function $rbf_{fp}(i,t)$ is defined as follows:

$$rbf_{fp}(i,t) = C_i + \sum_{\tau_k \in HP(i)} \left\lceil \frac{t}{\mathrm{T}_k} \right\rceil C_k \tag{12}$$

where $HP(i)$ is the set of tasks with priorities higher than τ_i. The global schedulability is defined as:

$$\forall S_s\ \exists t: 0 < t \leq P_s, LBF_s(t) \leq t \tag{13}$$

where the load bound function $LBF_s(t)$ is defined as follows:

$$LBF_s = RBF_s + B_s \tag{14}$$

where

$$RBF_S(t) = Q_s + \sum_{S_k \in HPS(s)} \left\lceil \frac{t}{P_k} \right\rceil Q_k \tag{15}$$

where $HPS(s)$ is the set of subsystems with a higher priority than subsystem S_s and B_s is the maximum time that S_s is blocked by lower priority subsystems. The performance of ARAP is evaluated as the minimum value of the request bound function $rbf_{fp}(t)$ that would guarantee schedulability. For the synthetic workload we generated random variances of a hierarchical system consisting of 3 separate subsystems such that $S_3 < S_2 < S_1$. Each subsystem S_n consisted of 3 tasks with a global resource being

shared between each subsystem. Each subsystem has a total utilization of 15%. Task periods ranged between 100 and 1000. Figure 5 represents the overall task acceptance rate for the simulated task sets required for task schedulability as defined by equation (3). Notice that ARAP has an improved acceptance rate over SIRAP. The reason for this improvement is that the WCET calculation for ARAP adapts to the current workload which provides greater flexibility than SIRAP's static approach. Additionally, because ARAP is adaptable it is comparable to OPEN-HSRP but without the added complexity of managing the overruns of the OPEN-HSRP protocol.

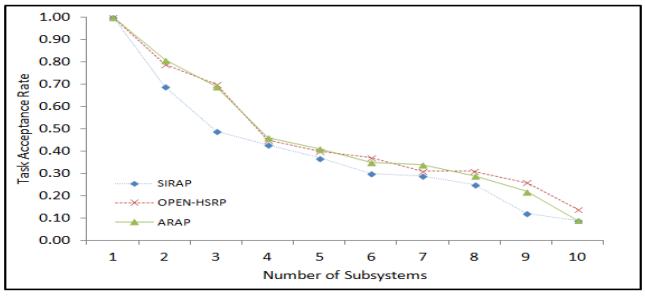

Figure 5: Task acceptance rate for simulated task sets

6. SIMULATION ENVIRONMENT

This section describes the simulation environment we used to further analyze ARAP which was implemented as a simulation component within RTSIM [16]. RTSIM (Real-Time System Simulator) is a task scheduler simulation and is used primarily for simulating real-time control systems. In order to implement ARAP in RTSIM we extended the existing resource manager class to include the feedback mechanisms.

6.1 Modeling and PID Tuning

A Heaviside step function was used to model the transient overload condition. Heaviside functions are used extensively in control theory to represent different loads. The discontinuous nature of this function maps nicely to an overload situation where we can model periods of nominal, ramp-up and ramp-down behaviors. The Ziegler-Nichols [17] tuning method was used to tune the feedback mechanism.

6.2 Example Task Set

The example task set consisted of a total of five periodic tasks, two hard real-time and three soft real-time tasks but only two tasks T_3 and T_5 shared a critical region and therefore were synchronized by a semaphore. For the two tasks (T_3 and T_5), that shared a semaphore, execution times were modeled to exceed their nominal rates. The Tasks T_4 and T_5 which represent hard real-time tasks were allowed to execute up to their predefined WCET while Task T_3 was modeled to exceed its bandwidth, thereby generating a transient overload condition.

In order to provide a comparison of ARAP we simulated both the traditional first-come-first-serve (FCFS) and OPEN-HSRP resource allocation protocols in RTSIM. One note; is that due to the simulated transient overload conditions of our scenario SIRAP was not included in the results because it was repeatedly denied access to the shared resource due to conservative nature of the protocol.

We ran simulations using the task set defined above with the Heaviside function to simulate task execution times for analysis as to how well the ARAP protocol performed against EDF scheduling with FCFS resource allocation (no PIP) as well as EDF using resource sharing with budget overrun (OPEN-HSRP). We executed sample runs modeling transient overload conditions at 0%, 5%, 10%, 15%, 20% and 25% respectively. The Figures 6 and 7 separate out the miss rates between the hard and soft real time tasks.

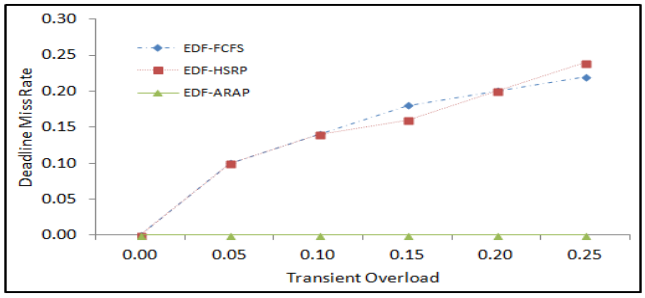

Figure 6: Hard Real-Time task miss rate (RTSIM)

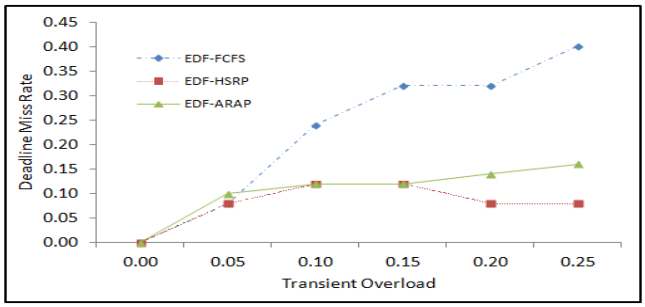

Figure 7: Soft Real-Time task miss rate (RTSIM)

In the figures above EDF-FCFS represents EDF scheduling with standard first-come-first-serve resource management (no PIP), EDF-HSRP represents EDF scheduling with OPEN-HSRP resource management and EDF-ARAP represents EDF scheduling with PID feedback control. Depicted in Figure 6 the ARAP protocol outperforms the other methods while it sacrifices some deadline misses in soft real-time tasks. Notice that in Figure 6 both scheduling mechanisms EDF-FCFS and EDF-HSRP exhibit hard real-time task misses. Even though EDF-HSRP manages the soft real-time tasks with a *periodic server* hard real-time task misses are realized because the overrun mechanism allows the task to continue, even though its server budget has been exhausted. In Figure 7 EDF-HSRP outperforms EDF-ARAP because the overrun mechanism allows the task to continue at the expense of hard real-time deadline misses.

7. PRACTICAL EVALUATION

To demonstrate the practicality of our approach we also implemented ARAP as part of a ground-based command and control test set used for satellite telemetry processing. A hardware-in-the-loop (HWIL) simulator was used to provide the workload for our system. This particular use case was chosen because telemetry processing times can vary considerably depending upon the data rate and how densely the telemetry frame is populated. In this way, we can use the HWIL simulator to generate transient overload conditions.

The main software components of the system includes a hard real-time periodic task that performs the frame processing of a telemetry stream. The other primary software component is a soft real-time task that provides the health and status monitoring for the vehicle. The telemetry processing task and the monitoring task both share a global resource which is the decommutated telemetry buffer. Similar to the simulation environment we used the HWIL

simulator to model transient overloads between 0%-25%. Two traditional resource synchronization protocols (PIP and FCFS) were used in the evaluation for comparison.

The results for the overall deadline miss rates were separated out based upon soft and real-time tasks. Soft real-time tasks were scheduled by the local subsystem scheduler which is scheduled by the global fixed priority scheduler of *VxWorks*. Hard real-time tasks were scheduled directly by the global *VxWorks* scheduler. Soft real-time tasks were modeled to allow their execution time to exceed their budget while the hard real-time tasks were designed to not exceed their pre-defined utilization budget.

Figures 8 and 9 show the miss rates of the hard real-time tasks and soft real-time tasks respectively. The feedback mechanism represented as FPPS-ARAP in the graph was compared against the priority inheritance protocol (FPPS-PIP) and the first-come-first-server (FPPS-FCFS) protocol. Notice how even when using priority inheritance a lower priority task can still cause a higher priority task to miss their deadline. The reason is that while PIP does solve the priority inversion problem it does not solve the problem of unbounded blocking.

As illustrated in Figure 8 ARAP does provide the mechanism for eliminating the extended blocking by a lower priority task however, the soft real-time task could be affected causing increased missed deadlines for the soft real-time task. Notice that in Figure 9 ARAP reports the highest number of deadline misses for soft real-time tasks. The reason for this behavior is that during a transient overload the task may be denied access to the resource and have to wait until the next budget replenishment period. As shown in the results using a feedback mechanism can directly benefit the determinism of a hard real-time at the possible expense of other soft real-time tasks that share the global resource.

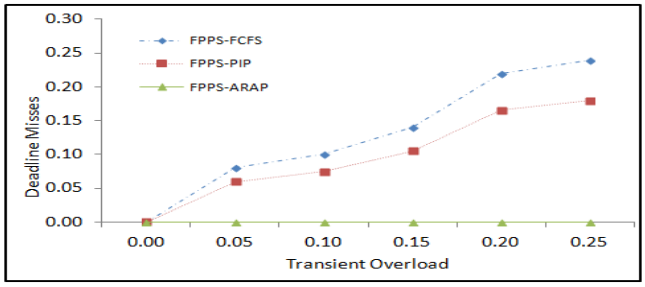

Figure 8: Hard Real-Time task miss rate (vxWorks)

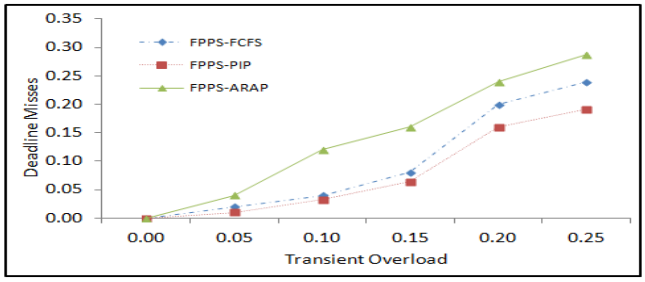

Figure 9: Soft Real-Time task miss rate (vxWorks)

8. CONCLUSION

In this paper we considered the problem of sharing global resources in a hierarchical scheduled system. Traditionally, HSF was designed for soft real-time applications, in part due to problem of unbounded resource holding times between global resources. Our approach which utilized feedback from the actual system to estimate future usage provided greater flexibility and allowed for the system to adapt to changes better than other state-of-the-art synchronization protocols. By implementing ARAP as part of an actual embedded system application we were able to effectively eliminate deadline misses for a critical high priority task. Our motivation for this work stems for the aerospace industry where systems are routinely over engineered in the interest of real-time determinism. It is a common perception that an embedded system is considered "unsafe" above 70% total utilization. As a result of this work we demonstrated that we can build more efficient embedded systems by more effectively managing the tasks within that system and in doing so reducing the total number of processing elements required.

9. ACKNOWLEDGMENT

This work was supported in part by the National Science Foundation under NSF grant number 1136146.

10. REFERENCES

[1] Z. Deng and J.W. Liu, "Scheduling real-time applications in an open environment." In Proc. of IEEE Real-Time Systems Symp. 1997.

[2] R.I. Davis and A. Burns, "Resource Sharing in Hierarchical Fixed Priority Pre-emptive Systems." In RTSS'06.

[3] P. Goyal, X. Guo and H.M. Vin, "A hierarchical CPU scheduler for multimedia operating systems." In OSDI, pp. 107-121, 1996.

[4] L. Sha, R. Rajkumar and J.P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization". IEEE trans. Comput. Vol 39, pp. 1175-1185, 1990.

[5] T.P. Baker , "Slack-Based Scheduling of Real-Time Processes". Real-Time Systems, vol. 3 pp. 67-99, 1991.

[6] L.Abeni and G.C. Buttazzo, "Resource reservations in dynamic real-time systems", Real-Time Systems, pp. 123-165, 2004.

[7] A. Mok, X.Feng and D. Chen. "Resource Partition for Real-Time Systems". In Proc. of IEEE Real-Time Technology and Appl. Symp., pp99-110, 2005.

[8] M. Behnam, T. Nolte, M Sjodin and I Shin. "SIRAP: A synchronization protocol for hierarchical resource sharing real-time open systems," In EM-SOFT 07.

[9] N. Fisher, M. Bertogna and S. Baraugh. "The Design of an EDF-Scheduled Resource-Sharing Open Environment". In RTSS '07.

[10] M. Behnam, T. Nolte, M Sjodin and I Shin. "Overrun Methods and Resource Holding Times for Semi-Independent Real-Time Systems," IEEE trans. on Indus. Informatics, 2010.

[11] M. Asberg, T. Nolte and M. Behnam, "Resource Sharing Using the Rollback Mechanism in Hierarchically Scheduled Real-Time Open Systems," In RTSA '13

[12] L. Abeni, G.C. Buttazzo , "Hierarchical qos management for time sensitive applications," In RTAS'01.

[13] L. Abeni, L. Palopoli, G. Lipari, J. Walpole, "Analysis of a Reservation-Based Feedback Scheduler," In RTSS '02.

[14] P. Phinjaroenphan, S. Beivinakoppa, P. Zeephongsekul. "A Method For Estimating the Execution Time of a Parallel Task on a Grid mode," Advances in Grid Computing – EGC 2005.

[15] Burchard, L.-O.; Altenbernd, P., "Estimating decoding times of MPEG-2 video streams," Image Processing, vol.3, 2000

[16] RTSIM, http://www.rtsim.sssupit.com

[17] Ziegler, J.G Nichols, N.B. Optimal settings for automatic controllers. Transactions of the ASME pp759-768, 1962.

[18] G. C. Buttazzo, Hard Real-Time Computing Systems, Springer, Real-Time System Series, 2011

[19] Stankovic, Jack A., et al. "The case for feedback control real-time scheduling." Real-Time Systems, 1999. Proc of the 11th Euromicro Conference on. IEEE, 1999.