2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium

# Task Mapping and Bandwidth Reservation for Mixed Hard/Soft Fault-Tolerant Embedded Systems

Prabhat Kumar Saraswat, Paul Pop, Jan Madsen
*Department of Informatics and Mathematical Modelling*
*Technical University of Denmark*
*DK-2800 Kgs. Lyngby*
{*pksa*|*pop*|*jan*}*@imm.dtu.dk*

***Abstract*—In this paper we are interested in mixed hard/soft real-time fault-tolerant applications mapped on distributed heterogeneous architectures. We use the Earliest Deadline First (EDF) scheduling for the hard real-time tasks and the Constant Bandwidth Server (CBS) for the soft tasks. The bandwidth reserved for the servers determines the quality of service (QoS) for soft tasks. CBS enforces temporal isolation, such that soft task overruns do not affect the timing guarantees of hard tasks. Transient faults in hard tasks are tolerated using checkpointing with rollback recovery. We have proposed a Tabu Search-based approach for task mapping and CBS bandwidth reservation, such that the deadlines for the hard tasks are satisfied, even in the case of transient faults, and the QoS for the soft tasks is maximized. Researchers have used fixed execution time models, such as the worst-case execution times for hard tasks and average execution times for soft tasks. However, we show that by using stochastic execution times for soft tasks, significant improvements can be obtained. The proposed strategy has been evaluated using an extensive set of benchmarks.**

## I. Introduction

Traditionally, hard and soft real-time systems have been implemented using very different techniques [1], [2], [3]. However, many applications have both hard and soft constraints [4], hence a unified approach is required. Moreover, economic pressures and multi-core architectures are driving the integration of hard and soft real-time applications onto the same platform.

In hard real-time systems, missing a deadline can lead to a catastrophic failure. Design methodologies for these systems are based on their worst-case execution times (WCET). There is a large amount of research on hard real-time systems [1], [3], including task mapping to heterogeneous architectures [5].

In soft real-time systems, missing a deadline does not cause catastrophic failures in the system but leads to a certain performance degradation. Researchers have proposed techniques for designing soft real-time systems, which use fixed execution time models such as the WCET and the average execution time (AET). However, soft real-time tasks have highly variable execution times, and using WCETs for taking design decisions would lead to overdesign (costly implementations), while using the AETs leads to an unpredictably large number of deadline misses [2].

To overcome these problems, researchers have used time models that can capture the variability of soft real-time tasks [2]. Using these time models, analysis techniques provide probabilistic guarantees for soft real-time tasks to meet their timing constraints [2]. Zamora et al. [6] have used stochastic automata networks (SAN) to model multimedia applications and to perform early design-phase trade-offs between performance and energy consumption. Network calculus has been used by Chakraborty and Thiele [7] to develop a new task model for streaming applications. In [4], time/utility functions are used to capture the decreasing "utility" of a soft task after missing its deadline. Manolache et al. [8] have used the probability density function (PDF) of the task execution time to accurately characterize the execution of soft tasks, and have shown that design decisions can be significantly improved if PDFs are used instead of fixed execution time models. In [9], Hu et al. have proposed a metric that can capture the overall system probabilistic behavior, not only individual tasks. In this paper we use the WCET for hard real-time tasks and the PDFs for soft tasks.

All of the previous approaches address hard and soft real-time tasks separately. Little research work has addressed the integration of hard and soft tasks on the same platform [10], [11], [4]. Abeni and Buttazzo [10] have proposed the Constant Bandwidth Server (CBS) for integrating hard and soft tasks on the same processor. CBS is used in conjunction with a scheduling technique such as Earliest Deadline First (EDF) or Rate Monotonic (RM), which guarantees the deadlines of hard tasks and schedules the servers. The soft tasks are scheduled by the servers, and the server parameters, i.e., bandwidth $Q_i$ and the period $T_i$, determine the probability of meeting the deadline of a particular soft task. The probability of meeting the deadline of a soft task can be considered as its quality of service (QoS). Abeni et al. have later shown [12] how the server parameters can be adaptively adjusted at runtime using a PID controller in order to maximize the QoS of soft tasks. Offline and online techniques for the derivation of CBS parameters have been proposed in [13], aiming at increasing the "benefit" associated to soft tasks.

As embedded applications are being used in various safety-critical applications, they have to perform correctly

1080-1812/10 $26.00 © 2010 IEEE
DOI 10.1109/RTAS.2010.31

even in the presence of faults. Fault-tolerance has been addressed separately for hard [14], [15] and soft [16] real-time systems. In [17] we have shown how to handle permanent faults in mixed hard/soft systems through task migration, such that hard deadlines are still satisfied and the soft tasks degrade gracefully. However, transient faults are more common and their number is increasing due to greater complexity, higher frequency and smaller transistor sizes [15]. Recently, Izosimov et al. [4] have considered mixed hard/soft real-time applications, and shown how quasi-static schedules can adapt at runtime to the actual execution times of tasks and to transient faults.

In this paper we propose a Tabu Search-based optimization algorithm which performs task mapping and processor bandwidth reservation. We are considering mixed hard/soft real-time applications that have to tolerate transient faults. We use EDF for the hard tasks and CBS for the soft tasks. Transient faults are tolerated using checkpointing with rollback recovery [15]. We have used the PDFs instead of the WCET or AET for the soft tasks, thus improving the mapping and processor bandwidth allocation decisions, leading to implementations where the deadlines for hard tasks are satisfied (even in case of faults) and better QoS is obtained for the soft tasks.

## II. Application Model

We model an application as a set $\mathcal{A}$ of interacting tasks $\tau_i \in \mathcal{A}$. The tasks are mapped on a distributed heterogeneous architecture denoted by the set $\mathcal{N}$ of processing elements. The mapping is denoted by the function $M : \mathcal{A} \rightarrow \mathcal{N}$. This mapping is not yet known but would be decided by our approach described in section V. A function $R : \mathcal{A} \rightarrow \{Hard, Soft\}$ determines if the task is hard or soft real-time, respectively. The requirement to tolerate transient faults by hard tasks is captured by the function: $F : \mathcal{A} \rightarrow \{Transient, \phi\}$, which determines if the task has to tolerate transient faults, or does not have any safety-critical requirements. We are assuming that soft real-time tasks do not have to tolerate faults, but our model and algorithms can take them into account, if necessary.

Tasks are periodic: each $\tau_i \in \mathcal{A}$ has a period $T_i$. If needed, traffic shapers [18] can be used to ensure periodic behavior. Tasks communicate asynchronously through buffers, i.e., a reader task will block if the buffer is empty and a writer task will block if the buffer is full. The buffer sizes have been determined such that there is no overflow or underflow [19], [20]. We assume that the message sizes are known, and in this paper we consider that processing elements are interconnected using a broadcast bus that uses non-preemptive EDF. We have shown how realistic bus protocols can be taken into account during the analysis [21].

Hard real-time tasks are characterized by their worst-case execution times $C_i$ and a deadline $D_i$. For a hard task $\tau_i$ we know the WCET $C_i^{N_j}$ for each processing element $N_j$ where $\tau_i$ is considered for mapping. WCETs can be obtained with a tool such as aiT [22]. Soft tasks are characterized by the PDF $U_i^{N_j}$ of their execution times and a soft deadline $\delta_i$. $U_i^{N_j}(h)$ is the probability that the job $J_{i,k}$ of $\tau_i$ has an execution time of $h$ on the processing element $N_j$. The PDFs can be obtained by measuring the execution times of the soft tasks by running them on actual hardware or on various hardware simulators for different inputs. There are also some hybrid techniques which incorporate both measurement and analysis to obtain such PDFs [23]. Two PDFs are depicted in Fig. 1. The PDFs depend on the mapping.

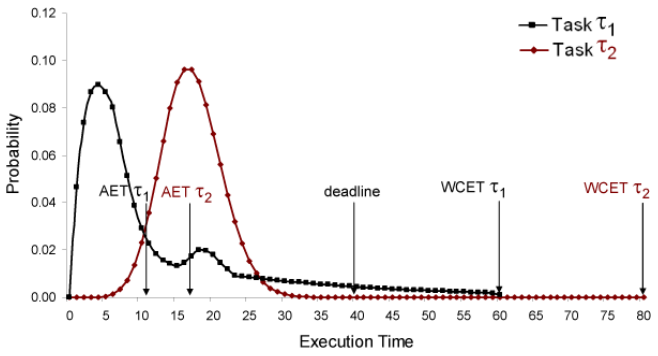



Figure 1. PDFs of the execution time of tasks $\tau_1$ and $\tau_2$ on processing element $N_1$

The QoS of a soft task $\tau_i$ is defined as the probability of meeting the deadline $\delta_i$, i.e., $QoS(\delta_i) = P\{f_{i,k} \leq r_{i,k} + \delta_i\}$, where $f_{i,k}$ and $r_{i,k}$ are the finishing and the arrival time, respectively, of the $k^{th}$ job of task $\tau_i$.

### A. Bandwidth Allocation for Soft Tasks

The QoS of a soft task $\tau_i$ depends on the CB server bandwidth $Q_i$ allocated to it. The design decision of bandwidth allocation for the soft tasks is not trivial. Fig. 1 shows the PDFs of the execution times of two soft tasks $\tau_1$ and $\tau_2$ on a processing element $N_1$. $\tau_1$ and $\tau_2$ have large variability in their execution times. $\tau_1$ has an AET and a WCET of 11 and 60 ms, respectively, while $\tau_2$ has an AET and a WCET of 17 and 80 ms, respectively. Both tasks have the same deadline and period of 40 and 100 ms, respectively. $N_1$ also contains some hard tasks. For simplicity, we have not shown these hard tasks. Let us assume that the processor utilization allocated to the hard tasks to guarantee their deadlines is 0.4 (i.e., 40% utilization). Allocating a bandwidth $Q_i$ to the CBS of a soft task $\tau_i$ with a period $T_i$ will consume $\frac{Q_i}{T_i}$ processor utilization. The spare utilization of 0.6 is available for the soft tasks, depicted visually in Fig. 3(a), hence, $\frac{Q_1}{T_1} + \frac{Q_2}{T_2} \leq 0.6$.

Fig. 2 shows the probability of satisfying the deadline (QoS) of $\tau_1$ and $\tau_2$ for different allocated $Q_1$ and $Q_2$ values, respectively. This dependence is not linear and is influenced by the shape of the PDFs. These graphs are obtained by using the stochastic analysis presented in section V-B. The

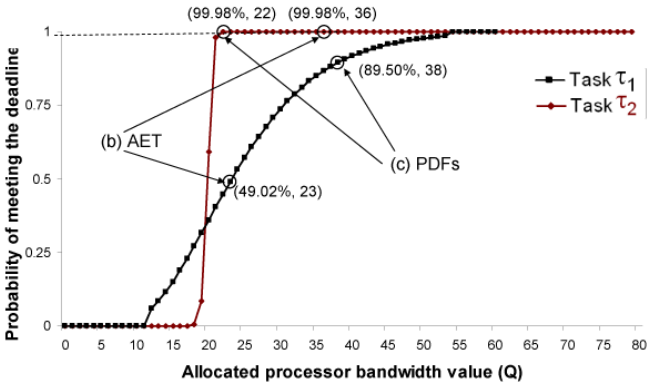



Figure 2. Probability of meeting the deadline for tasks $\tau_1$ and $\tau_2$ on $N_1$

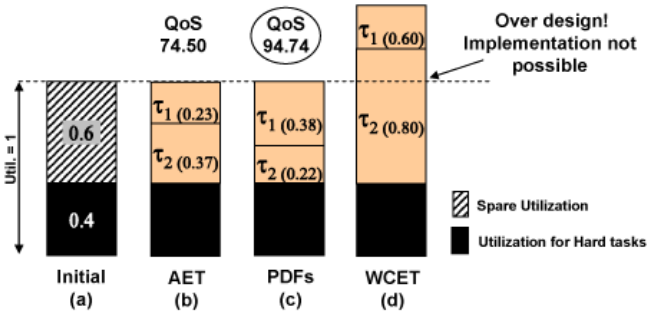



Figure 3. Allocation of Q values using (b) AET, (c) PDFs and (d) WCET.

probability of satisfying the deadline of both tasks decreases drastically when the Q value is less than the AET.

Let us consider the first approach ( see Fig. 3(b) ) where $Q_i$ values are allocated *in proportion* to the tasks' AETs (11 and 17, respectively), i.e., processor utilization for $\tau_1$ is $\frac{0.6\times11}{11+17} = 0.235$ and processor utilization for $\tau_2$ is $\frac{0.6\times17}{11+17} = 0.365$. These processor utilizations corresponds to Q values of 23 and 36 for $\tau_1$ and $\tau_2$, respectively, which, according to Fig. 2 lead to a system QoS[1] of $\frac{49.02+99.98}{2} = 74.50\%$, shown in Fig. 3(b).

However, using the stochastic execution times (captured by the PDFs), the Q value can be intelligently chosen as 38 for task $\tau_1$ and 22 for $\tau_2$ corresponding to a QoS of $\frac{89.50+99.98}{2} = 94.74\%$, shown in Fig. 3(c), much better than the previous case. Therefore, using stochastic execution times, better design decisions can be taken regarding bandwidth allocation.

Note that another problem with using the AET is that no guarantees can be given for the soft tasks, i.e., the probability of meeting the deadline cannot be derived from $Q_i$ and AET. The probability of meeting the deadline $\delta_i$, given a certain $Q_i$, can only be determined by using the PDF $U_i$, as explained in section V-B. The WCETs can also be used to allocate the processor bandwidths but corresponds to total utilization greater than 1, see Fig. 3(d).

## III. Platform Model

We consider hardware architectures consisting of a set $\mathcal{N}$ of heterogeneous processing elements (PEs), interconnected by a communication channel. The communication channel is a bus on which messages are exchanged between tasks. Each PE $N_i \in \mathcal{N}$ consists of a communication controller and a processing unit. Communicating tasks mapped on different PEs exchange messages on the communication channel. Messages are scheduled on the bus using non-preemptive EDF. We consider that the bus is fault-tolerant.

[1]Considering equal weights for the QoS of $\tau_1$ and $\tau_2$. Different weights can be assigned, if necessary.

PEs have access to a shared memory where the code of tasks and the *checkpoints* (last non-faulty state of a task) are stored. The software architecture on each PE is composed of an EDF scheduler and a middleware implementing the CBS scheduling mechanism. The soft tasks are scheduled using CBS. The hard tasks and CB servers are scheduled using EDF. CBS enforces temporal isolation between hard and soft real-time tasks, thus guaranteeing the schedulability of hard tasks.

Each soft real-time task $\tau_i$ is assigned a CBS, characterized by the tuple $(Q_i, T_i)$, where $Q_i$ (also called as bandwidth) is the time that the soft task $\tau_i$ is allowed to use the PE every period $T_i$[2]. Each time $\tau_i$ demands more than its allocated $Q_i$, the CB server postpones the deadline $\delta_i$ with $T_i$. The EDF scheduler will schedule $\tau_i$ using this new deadline thus ensuring that a soft task will never demand more than its assigned bandwidth [10].

### A. Fault Model and Checkpointing

We assume that there can be at most $k$ transient faults within an execution cycle of the application (the least common multiple (LCM), of all the tasks' periods). Transient faults are tolerated using equidistant checkpointing with rollback recovery [15], which uses time redundancy to tolerate transient faults. The principle of checkpointing is to restore the last non-faulty state (checkpoint) of the failing task, i.e., to *recover* from fault. The checkpoint has to be saved in advance into a stable storage (in our case, the shared memory), and will be restored if the task fails. The part of the task between two checkpoints is called *execution segment*. The number of checkpoints for a hard real-time task $\tau_i$ tolerating transient faults is denoted with $n_i$. The overhead in establishing the checkpoint is captured by $O_i$. We assume that the inputs for a task are stored in its input buffers and, thus, no checkpoint is needed at the beginning of the task. The length of the execution segment is $\lceil \frac{C_i}{n_i} \rceil$. After the execution of each execution segment, an error detection mechanism is used to detect if a fault occurred during its execution. This *error detection overhead* is represented as $\alpha_i$. If

[2]The period of the server is equal to the period of the soft task.

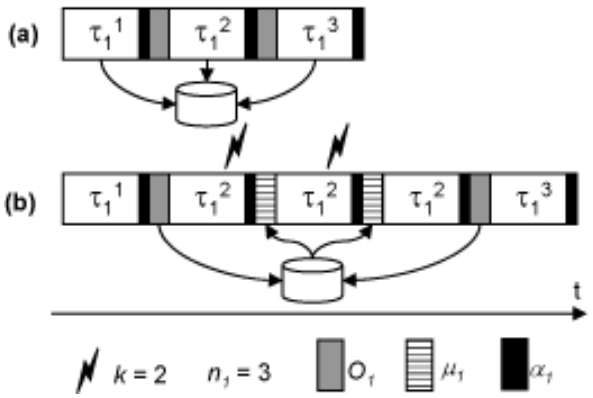


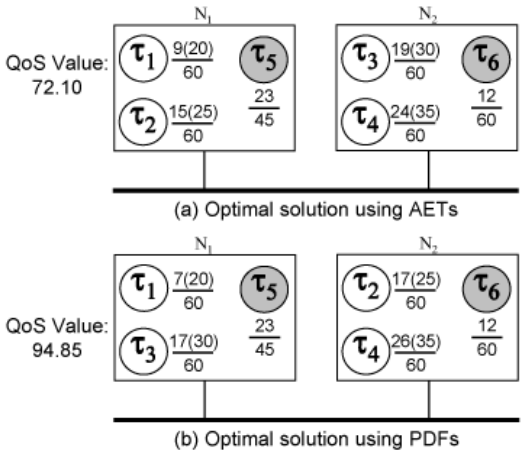

Figure 4. Task execution with checkpointing and fault recovery. (a) execution without faults, (b) with faults

a transient error is detected, the affected execution segment will be recovered (re-executed) from the last checkpoint on the same processor. This *recovery overhead* is represented as $\mu_i$.

Let us consider the example in Fig. 4, where we have the task $\tau_1$ and a fault scenario consisting of $k = 2$ transient faults that can happen during one cycle of operation. We are considering $n_1 = 3$ checkpoints for $\tau_1$. The three execution segments of $\tau_1$ are depicted as $\tau_1^1$, $\tau_1^2$ and $\tau_1^3$. After the execution of initial segment $\tau_1^1$, the error detection mechanism detects whether a fault happened or not. The overhead corresponding to error detection is shown as a black rectangle. The state information of the first execution segment is saved in the shared memory, the overhead corresponding to establishing this checkpoint is shown as a grey rectangle. Fig. 4(a) shows the execution of the task with checkpointing but without encountering any fault. It can be seen in Fig. 4(b) that the first fault happens during the execution of segment $\tau_1^2$. After the error has been detected, $\tau_1^2$ is recovered based on the information in the saved checkpoint. After a recovery overhead (shown as a striped rectangle), segment $\tau_1^2$ is executed again. Since the second fault is also detected during the execution of $\tau_1^2$, it is executed again after recovering the state information from the last saved checkpoint. After a checkpointing overhead, the third execution segment $\tau_1^3$ is executed. These overheads are taken into consideration while performing schedulability analysis of the system, as proposed in section V-A.

## IV. Problem Formulation

The problem we are addressing in this paper can be formulated as follows: Given a mixed hard/soft fault tolerant application $\mathcal{A}$ and a distributed architecture $\mathcal{N}$ and the maximum number $k$ of transient faults, we are interested to determine an implementation $\mathcal{S}$ consisting of (1) a mapping $M(\tau_i) \in \mathcal{N}$ for each task $\tau_i \in \mathcal{A}$ and (2) a set $Q$ containing the bandwidth $Q_j$, for each soft task $\tau_j$, such that the deadlines for hard tasks are satisfied, even in the case of transient faults, and the probability of meeting the deadlines of soft tasks is maximized.




| Task | AET $N_1$ | AET $N_2$ | Period | Deadline |
|---|---|---|---|---|
| $\tau_1$ | 5 | 8 | 60 | 20 |
| $\tau_2$ | 10 | 12 | 60 | 25 |
| $\tau_3$ | 14 | 15 | 60 | 30 |
| $\tau_4$ | 17 | 20 | 60 | 35 |

| Task | WCET $N_1$ | WCET $N_2$ | Period | Deadline |
|---|---|---|---|---|
| $\tau_5$ | 23 | 25 | 45 | 45 |
| $\tau_6$ | 8 | 12 | 60 | 60 |

(c) Task set for soft tasks and hard tasks

Figure 5. Optimal solutions using (a) AETs and (b) PDFs

### A. Mapping and Bandwidth Allocation Example

To illustrate the problem, let us consider a system with two PEs, $N_1$ and $N_2$, four soft real-time tasks, $\tau_1$–$\tau_4$, and two hard tasks, $\tau_5$ and $\tau_6$. In this example, we are ignoring the communication between the tasks. The fixed-execution time values, periods and deadlines on $N_1$ and $N_2$ are presented in Fig. 5(c). $N_1$ is faster than $N_2$.

Fig. 6 shows how the probability of meeting the deadline increases with $Q_i$ for the soft tasks $\tau_1$–$\tau_4$, considering the two processing elements $N_1$ and $N_2$. Both hard tasks, $\tau_5$ and $\tau_6$, have to tolerate $k = 1$ transient fault per execution cycle of the application.

Let us consider first the situation in which the PDFs are not available to the designer, and AETs have to be used. A straightforward way to perform mapping and bandwidth allocation in this situation would be to assign $Q_i$ values such that the difference between the AET and the assigned $Q_i$ value is maximized for all the tasks. The formal definition of such a cost function is presented in section VI. Lets take as an example, task $\tau_3$. How would a designer maximize QoS for $\tau_3$ using only the AETs, which are 14 and 15 ms

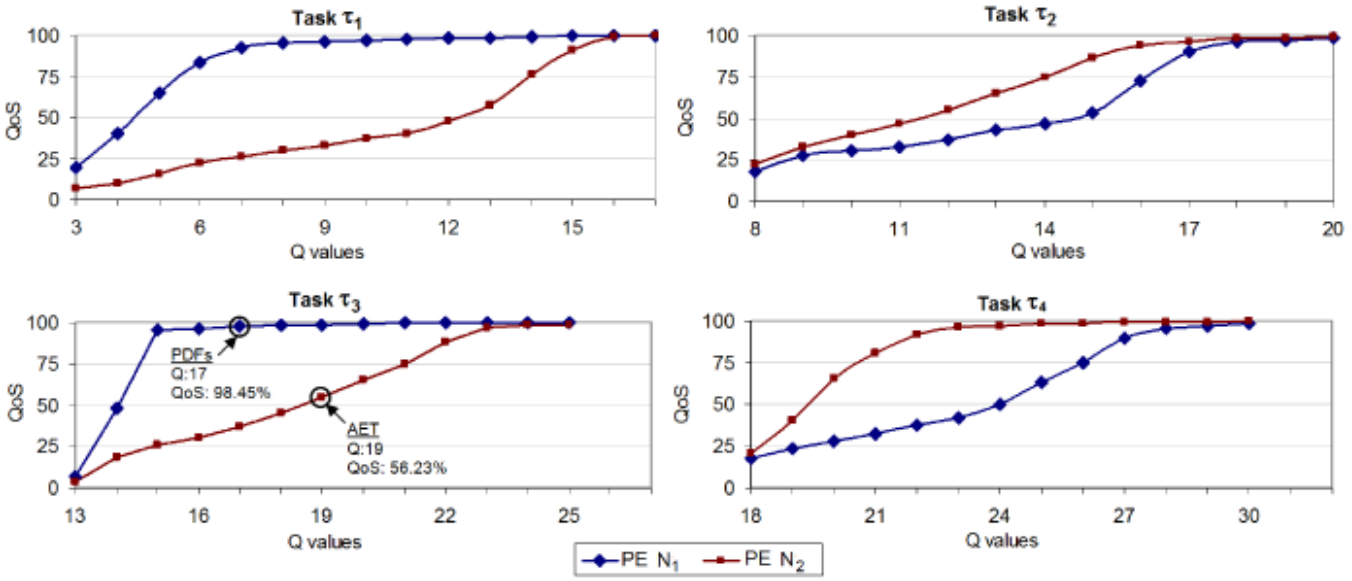



Figure 6. *QoS* vs Q values for soft tasks $\tau_1$–$\tau_4$

on $N_1$ and $N_2$, respectively? Considering that the deadlines for hard tasks have to be satisfied and the QoS for the other soft tasks have also to be maximized, the best solution is to map $\tau_3$ to $N_2$, considering a $Q_3$=19, which corresponds to a difference of $Q_3 - AET_3$ = 19 – 15 = 4. If $\tau_3$ is mapped on $N_1$, $Q_3$ can only be set to 17, which leads to a difference of only 17 – 14 = 3. Thus, by performing such a straightforward allocation for all soft tasks, we get the solution in Fig. 5(a) where the system QoS is 72.10 %. The mapping is depicted in the figure by placing the tasks inside the PEs. The grey tasks are hard, whereas the white tasks are soft. The task parameters are presented next to each task, i.e., the WCETs and periods for hard tasks are denoted as a fraction $\frac{C_i}{T_i}$, and the allocated bandwidth $Q_i$, soft deadline $\delta_i$ and the period $T_i$ of the servers associated to each soft task are depicted as $\frac{Q_i(\delta_i)}{T_i}$. The probability of meeting a deadline (QoS) for a soft task cannot be determined using fixed execution time models such as AET. The QoS value in Fig. 5(a) has been determined using the PDFs (as presented in section V-B) in order to compare it with the case in Fig. 5(b). However, using stochastic execution times (captured by the PDFs) instead of the AETs can lead to better solutions. From Fig. 6 (task $\tau_3$) we can see that a $Q_3$ of 17 on $N_1$ is actually preferred to a $Q_3$ = 19 on $N_2$, since it leads to a QoS of 98.45 % instead of 56.23 %. The optimal mapping and Q values, found out by taking the PDFs into consideration, are shown in Fig. 5(b). The system QoS value resulting from this mapping and bandwidth allocation decision is 94.85 %, significantly better than the previous case.

## V. Tabu Search-based Optimization Strategy

To solve the problem presented in the previous section, we use a Tabu Search-based strategy which decides the mapping and the allocated bandwidth. Tabu Search [24] is an optimization metaheuristic which iteratively explores the solutions in the vicinity (neighborhood) of the current solution, selecting the ones which minimize the *cost function*.

Our proposed cost function captures the schedulability of hard tasks and the QoS of soft tasks in the application and is formally defined in section V-B. By minimizing the cost function we improve the schedulability of hard tasks and maximize the QoS for the soft tasks.

Algorithm 1 presents our Tabu Search Mapping and Bandwidth Allocation (TSMBA) optimization strategy. TSMBA takes as input the application $\mathcal{A}$, the architecture $\mathcal{N}$ and produces the solution $\mathcal{S}$ consisting of the mapping $\mathcal{M}$ for all tasks and the set of bandwidth values $\mathcal{Q}$ for all soft tasks.

The algorithm starts from an initial solution $\mathcal{S}^\circ$ (line 1). The initial solution can either be schedulable or unschedulable. An schedulable solution is the one which satisfies the schedulability criteria for the hard tasks given in section V-A (i.e., all hard tasks are schedulable and all safety critical tasks can tolerate transient faults). In the initial solution, the tasks are mapped such that their utilization is evenly distributed among the processing elements (for soft tasks we consider the average utilization $\frac{AET}{T_i}$). The bandwidth for the soft tasks is allocated to a value equal to their AET.

The neighborhood of the current solution is generated using design transformations (moves) that change the current system implementation (line 6). The neighborhood can be very large, thus we consider a limited number of neighboring solutions, called *candidate set*. Let us consider the example

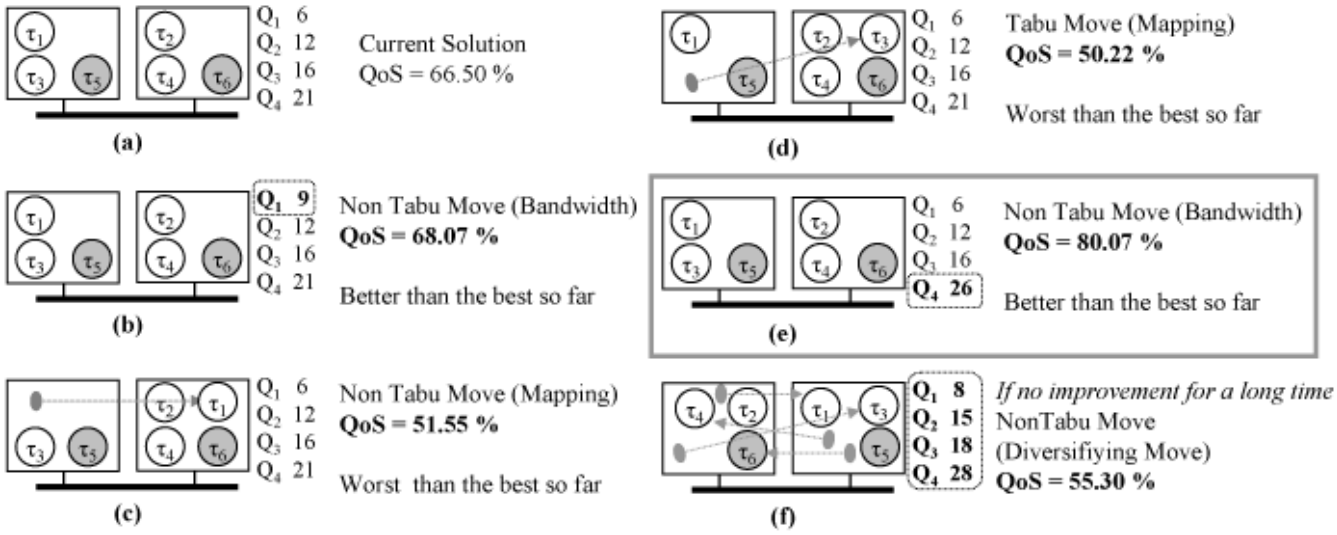



Figure 7. Tabu Search Moves for the system in Fig. 5(c)

in Fig. 7 where the current solution is in Fig. 7(a) and part of the neighborhood is depicted in Fig. 7(b)–(e). We consider the same system as shown in Fig. 5(c).

The mapping changes are denoted with an arrow and the bandwidths are also listed in the figure. We use two kinds of moves: *Mapping Move (MM)*, which changes the mapping for a task; and *Bandwidth Move (BM)*, where the bandwidth for a soft task is changed.

In MM, we randomly select a task from a randomly selected PE and map it to another PE (also randomly selected). The cost corresponding to this is calculated and then used by TSMBA to evaluate the solution. For example, solutions shown in Fig. 7(c) and 7(d) have been generated by performing MM on the current solution in Fig. 7(a), where the mapping of tasks $\tau_1$ and $\tau_3$ is changed to $N_2$ and $N_1$, respectively. In BM, a random soft task is selected from a randomly selected PE. The bandwidth of this task is changed with a random integer value in the interval [-5, 5]. For example, solutions shown in Fig. 7(b) and 7(e) have been generated from Fig. 7(a) by performing BM where Q values of $\tau_1$ and $\tau_4$ are increased by 3 and 5, respectively.

**Algorithm 1** TSMBA $(\mathcal{A}, \mathcal{N})$

1: $\mathcal{S}^{\circ}$ = InitialSolution($\mathcal{A}, \mathcal{N}$)
2: $\mathcal{S}^{current} = \mathcal{S}^{best} = \mathcal{S}^{\circ}$
3: $Cost^{best}$ = CostFunction($\mathcal{S}^{\circ}$)
4: $TabuList = \phi$
5: **for** *max_iter* iterations **do**
6: $NS$ = GenerateNeighborhood($\mathcal{S}^{current}$)
7: $\mathcal{S}^{current}$ = SelectSolution($NS$)
8: **if** CostFunction($\mathcal{S}^{current}$) $< Cost^{best}$ **then**
9: $Cost^{best}$ = CostFunction($\mathcal{S}^{current}$)
10: $\mathcal{S}^{best} = \mathcal{S}^{current}$
11: **end if**
12: $TabuList = TabuList \bigcup \mathcal{S}^{current}$
13: **end for**
14: return $\mathcal{S}^{best}$

Tabu Search maintains a selective history of the solutions visited (called *tabu list* and denoted with *TabuList*) to avoid revisiting already explored solutions. The *TabuList* is initialized in line 4 of algorithm 1 and updated with the currently visited solution in line 12. For example, in Fig. 7, the solution shown in Fig. 7(d) has already been visited by the heuristic, and it is marked as *tabu* and is put in the *TabuList*. From the given neighborhood *NS*, the *non-tabu* solution with the minimum cost is chosen, and the exploration continues. A tabu solution can also be chosen if the cost of this solution is better than the best solution encountered so far.

It can be seen that for the given *NS* in Fig. 7 the solution in Fig. 7(e) would be chosen as the current best solution $\mathcal{S}^{current}$ (line 7) because its cost (corresponding to all the hard deadlines satisfied and a QoS of 80.07%), is better than the cost of other solutions in *NS*. This solution is also marked as the best solution so far ($\mathcal{S}^{best}$) since its cost is better than the best so far (line 10). Using this solution as the current solution, *TSMBA* iterates again until a bound on the number of iterations (*max_iter*), given by the designer, is reached (line 5). In the end, $\mathcal{S}^{best}$ is reported (line 14).

During the iterative search in TSMBA, it may happen that the heuristic is stuck in a region containing sub-optimal solutions and no improvement is seen for several iterations. In this case, a *diversification move* (DM) is performed in which several mappings and bandwidth allocations are changed, resulting in a very different solution. An example

of a solution resulting from performing a DM can be seen in Fig. 7(f), where mapping and bandwidths of all tasks are changed. The hope is that a DM move will lead the search to unexplored areas, where better solutions can be found.

### A. Schedulability analysis for hard tasks

We assume that the deadlines for hard real-time tasks are equal to the periods[3], and for each PE $N_j$ we use the utilization-based test [3] to determine if the task set composed of hard tasks and CBS servers is schedulable:

$$\sum_{\substack{\forall \tau_i : R(\tau_i)=Hard \\ \wedge M(\tau_i)=N_j}} \frac{C_i'}{T_i} + \sum_{\substack{\forall \tau_i : R(\tau_i)=Soft \\ \wedge M(\tau_i)=N_j}} \frac{Q_i}{T_i} + U_R^{N_j} \leq 1,$$

where $C_i'$ is the WCET of $\tau_i$ considering the fault-tolerance overheads and $U_R^{N_j}$ is the worst-case utilization needed to recover from faults.

In our task model, the checkpointing related parameters of safety-critical hard tasks are represented by $n_i$ and $O_i$, where $n_i$ is the number of checkpoints and $O_i$ is the computation time overhead to establish one checkpoint in task $\tau_i$. The effective WCET inclusive of the checkpoint overhead $O_i$ and error detection overhead $\alpha_i$ for each task is given by $C_i' = C_i + (n_i - 1)(O_i + \alpha_i) + \alpha_i$, where $C_i$ is the WCET of the task. The last term $\alpha_i$ corresponds to the execution of last execution segment where only the error detection is done and no checkpoint is stored.

We have assumed at most $k$ transient faults during an execution cycle of the application. When a fault happens in $\tau_i$, the task has to be restored from its previously saved checkpoint. The length of the execution segment that has to be executed is $\lceil \frac{C_i}{n_i} \rceil$.

The utilization needed to recover from a fault in a segment is $\frac{\lceil \frac{C_i}{n_i} \rceil + \alpha_i + \mu_i}{T_i'}$, where $T_i'$ is the time interval within which the task should recover. $T_i'$ is given by the designer and is application specific. For example, if task $\tau_i$ should recover within its deadline $D_i = T_i$, then $T_i' = T_i$. For automative applications, for example, if it is allowed for the task to recover before the end of the next period, then $T_i' = 2 \times T_i$. Given a processing element $N_j$, we are interested to determine the utilization $U_R^{N_j}$ needed to recover the hard tasks in case of faults in the worst-case fault scenario. The worst-case fault scenario is that fault occurrence of $k$ faults which leads to the largest $U_R^{N_j}$. This means that all the $k$ faults happen in that execution segment which corresponds to the largest recovery utilization. Thus, for the PE where $\tau_i$ is mapped, i.e., $N_j = M(\tau_i)$ we have:

$$U_R^{N_j} = \max_{\substack{\tau_i : R(\tau_i)=Hard \\ \wedge F(\tau_i) \neq \phi}} k \times \frac{\lceil \frac{C_i}{n_i} \rceil + \alpha_i + \mu_i}{T_i'}$$

[3]Arbitrary deadlines can be handled using the Processor Demand Criterion [1].

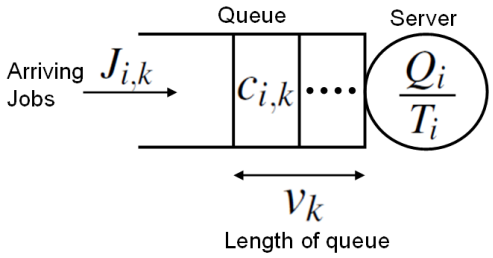



Figure 8. Queuing model of CBS

For the messages on the bus we use the analysis for the preemptive EDF to check that the bus utilization is below 1, considering the current mapping.

### B. Schedulability analysis for soft tasks

The schedulability of a soft task $\tau_i$ (i.e., the probability of meeting its deadline $\delta_i$) depends on the bandwidth $Q_i$. Due to the temporal isolation property of CBS, each soft task can be analyzed individually. This property is particularly useful for the exploration strategy. Since the calculation of $QoS$ is computationally expensive, the $QoS$ for all $Q_i$ values considered is computed beforehand and stored for later use. We consider integer $Q_i$ values in the interval $[AET, WCET]$. If $Q_i \geq WCET$ the soft deadline $\delta_i$ is met in 100% of the cases, whereas if $Q_i < AET$, the probability of meeting soft deadline $\delta_i$ is very small.

For a given soft real-time task $\tau_i$, $QoS(\tau_i)$ is defined as the probability of meeting the deadline $\delta_i$, i.e., $QoS(\delta_i) = P\{f_{i,k} \leq r_{i,k} + \delta_i\}$, where $f_{i,k}$ and $r_{i,k}$ are the finishing and the arrival time, respectively, of the $k^{th}$ job of task $\tau_i$. This probability is calculated by modeling the CBS (serving the soft tasks $\tau_i$) as a queuing system [25]. For every task $\tau_i$, its arriving jobs $J_{i,k}$ are seen as tokens to be served by the server having the capacity $Q_i$, Fig. 8. Each arriving job $J_{i,k}$ has a computation time of $c_{i,k}$ units and arrives every $T_i$ units of time. The system can be described with a random process defined as follows:

$$\begin{cases} v_1 = c_{i,1} \\ v_k = max\{0, v_{k-1} - Q_i\} + c_{i,k} \end{cases}$$

The state variable $v_k$ indicates the length of the queue (in time units) immediately after the job $J_{i,k}$ having a computation time $c_{i,k}$ arrives. Since the server can serve only $Q_i$ units every $T_i$ units of time, the finishing time of the served job $J_{i,k}$ is $f_{i,k} = r_{i,k} + \lceil \frac{v_k}{Q_i} \rceil T_i$.

Thus, the probability that a job would finish before $\lceil \frac{v_k}{Q_i} \rceil T_i$, has a lower bound which is the probability that the queue length is $v_k$ immediately after the job arrives. This probability can be calculated by considering $\pi_m^{(k)} = P\{v_k = m\}$ as the state probability of the process $v_k$, and then calculating its stationary solution. We already have the PDF of the request times of the arriving jobs, $U_i(h)$. Since we know that $c_{i,k}$ is

time invariant, as $U_i(h)$ does not depend on $k$, the value of $\pi_m^{(k)}$ can be calculated as follows:

$$\pi_m^{(k)} = P\{v_k = m\} = P\{max\{v_{k-1} - Q_i, 0\} + c_k = m\}$$

i.e., the probability that $v_k$ has a length $m$ is equal to the probability that the sum of the execution time $c_k$ of the immediately arrived job and the old queue length $max\{v_{k-1} - Q_i, 0\}$ is equal to $m$. Using the PDF of the incoming requests, the solution can be derived as:

$$\begin{aligned}\pi_m^{(k)} &= \sum_{h=0}^{Q_i} (U_i(m) \times \pi_h^{(k-1)}) \\ &+ \sum_{h=Q_i+1}^{\infty} (U_i(m-h+Q_i) \times \pi_h^{(k-1)})\end{aligned}$$

where the first term corresponds to the cases when the arriving jobs have their computation time less than or equal to $Q_i$, thus can be completely served by the server and the second term corresponds to the cases when their computation times are greater than $Q_i$, thus cannot be completely served by the server leading to an increase in the queue length.

One can use a Markov matrix $M$, describing the aforementioned random process, to calculate the stationary solution of $\pi_m^{(k)}$. An element $n_{i,j}$ of Markov matrix $M$ gives the probability of queue length changing from $i$ to $j$. If the matrix describing the state probability vector of $\pi_m^{(k)}$ is denoted by $\Pi^k$, the stationary solution can be calculated by using the equation $\Pi^k = M\Pi^{k-1}$. The detailed derivation and the matrices can be seen in [25]. By solving for the stationary solution of this state probability vector, the probability of meeting a certain deadline can be calculated. Our heuristic will decide on the appropriate $Q$ values that minimize the cost function, thus maximizing the total $QoS$.

### C. Cost Function

In the Tabu Search-metaheuristic the solutions are evaluated on the basis of a cost function. The cost function has to be minimized. The cost function for a solution $\mathcal{S}$ is given as follows:

$$\begin{aligned}&\sum_{\forall N_i \in \mathcal{N}} \max(0, U_{N_i} - 1) \times w_{penalty} \\ &+ \sum_{\forall \tau_i : R(\tau_i) = Soft} (1 - QoS(\tau_i)) \times w_i\end{aligned}$$

where $w_{penalty}$ corresponds to a very large penalty added to the cost of a solution in case the hard tasks are not schedulable, i.e., if the utilization of a processor is greater than 1. In case the hard tasks are schedulable, the first term is 0, and the second term of the cost function corresponds to maximizing the QoS of the soft tasks. Individual weights $w_i$ can be assigned to soft tasks to differentiate them in terms of their importance.

## VI. Experimental Evaluation

In the first set of experiments we were interested in determining the quality of the proposed optimization strategy as the systems become larger. For this, we have used ten synthetic benchmarks of 6 to 70 tasks mapped on architectures with 2 to 20 PEs. We have used WCETs in the interval 3 to 18 ms and have generated PDFs, to match the shape of those measured on real-life benchmarks, with expectations between 15 to 60 ms. Messages sizes are in between 10 to 40 Kb. Since the bus bandwidth is assumed to be 10 Mbps, the transmission times on the bus are in between 1 to 4 ms. For architectures containing 9 or more processors the bus bandwidth is doubled to account for the increased communication traffic. We are also assuming that half of the hard tasks in an application tolerate transient faults. We are assuming that there can be at most 1 transient fault per execution segment of the application. We are assuming a hard task $\tau_i$ has to recover before the end of the next period, i.e., $2 \times T_i$. The checkpointing overhead and number of checkpoints for these tasks are between 1 to 3 ms and 2 to 8, respectively.

Table 1 presents the experimental setup details and the obtained results. To determine the quality of our approach we have compared the results obtained using TSMBA with the results obtained by applying a straightforward (SF) optimization strategy. This is a strategy that a good designer will pursue in case only fixed execution times (AET) are available, and not the full PDFs. SF is also based on a Tabu Search heuristic, but is using a slightly different cost function, which, for the soft tasks, uses the difference between allocated bandwidth and the AET. The goal of this approach is to allocate the bandwidth in such a way that the difference between the AET and the allocated bandwidth is maximized for all the soft tasks, thus increasing the probability to meet the deadlines. This is captured by the formula $\frac{\Delta_{avg}}{\Delta_{dev}}$, where $\Delta_{avg}$ is average, for all soft tasks, of the difference between the allocated bandwidth and the AET of soft tasks and $\Delta_{dev}$ is the standard deviation of these difference values.

| No. | *No. of* PE | SRT | HRT | Iter. TSMBA | Iter. SF | *QoS TSMBA* | *QoS SF* | *Improvement* |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 3 | 0 | 0 | 99.20 | 52.23 | 46.97 |
| 2 | 3 | 7 | 4 | 738 | 1052 | 99.28 | 70.24 | 29.04 |
| 3 | 4 | 9 | 6 | 0 | 0 | 99.49 | 76.35 | 23.14 |
| 4 | 5 | 11 | 8 | 894 | 3563 | 97.46 | 65.45 | 32.01 |
| 5 | 6 | 13 | 9 | 0 | 0 | 97.89 | 74.96 | 22.93 |
| 6 | 7 | 16 | 10 | 0 | 0 | 98.67 | 64.65 | 34.02 |
| 7 | 8 | 18 | 12 | 1383 | 5436 | 98.34 | 77.34 | 21.00 |
| 8 | 10 | 22 | 13 | 1457 | 6964 | 97.24 | 62.46 | 34.79 |
| 9 | 15 | 35 | 17 | 0 | 0 | 97.43 | 70.35 | 27.09 |
| 10 | 20 | 44 | 26 | 0 | 0 | 96.28 | 71.31 | 24.97 |

Table I
Synthetic Benchmarks

| Benchmark Name | *No. of* PE | SRT | HRT | Iter. TSMBA | Iter. SF | *QoS TSMBA* | *QoS SF* | *Improvement* |
|---|---|---|---|---|---|---|---|---|
| auto-indust-cords | 3 | 9 | 7 | 525 | 786 | 97.88 | 62.79 | 35.08 |
| auto-indust-mocsyn | 4 | 9 | 7 | 0 | 0 | 98.89 | 74.89 | 24.00 |
| consumer-cords | 2 | 5 | 3 | 220 | 352 | 95.26 | 72.22 | 23.03 |
| consumer-mocsyn | 3 | 5 | 3 | 0 | 0 | 98.11 | 71.01 | 27.10 |
| networking-cords | 2 | 4 | 3 | 234 | 419 | 98.15 | 62.66 | 35.49 |
| networking-mocsyn | 3 | 4 | 3 | 0 | 0 | 99.28 | 75.79 | 23.49 |
| telecom-cords | 3 | 10 | 6 | 0 | 0 | 99.42 | 68.88 | 30.55 |
| telecom-mocsyn | 4 | 10 | 6 | 432 | 968 | 99.69 | 74.20 | 25.49 |

Table II
REAL-LIFE BENCHMARKS

| No. | *No. of* PE | SRT | HRT | *Init. Util.%* | *QoS TSMBA* | *QoS SF* | *Improvement* |
|---|---|---|---|---|---|---|---|
| 1 | 7 | 16 | 10 | 50.07 | 99.34 | 75.39 | 23.95 |
| 2 | 7 | 16 | 10 | 55.09 | 99.34 | 75.39 | 23.95 |
| 3 | 7 | 16 | 10 | 60.08 | 99.34 | 75.39 | 23.95 |
| 4 | 7 | 16 | 10 | 65.08 | 99.34 | 75.34 | 24.00 |
| 5 | 7 | 16 | 10 | 70.11 | 98.75 | 74.02 | 25.16 |
| 6 | 7 | 16 | 10 | 75.02 | 98.41 | 73.59 | 24.39 |
| 7 | 7 | 16 | 10 | 80.07 | 98.85 | 71.78 | 27.08 |
| 8 | 7 | 16 | 10 | 85.01 | 98.45 | 71.89 | 26.56 |
| 9 | 7 | 16 | 10 | 90.10 | 98.78 | 68.89 | 29.89 |
| 10 | 7 | 16 | 10 | 95.38 | 98.67 | 64.65 | 34.02 |

Table III
VARYING INITIAL UTILIZATION

Let us see the results in Table I. For all the cases, the initial solution (where the utilization was balanced among the processing elements and the bandwidth was set to AET) can either be schedulable or unschedulable. The values in columns 3 and 4, respectively, indicate in which iterations of the Tabu Search (implementing SF and TSMBA) schedulable solution starting from the initial solution was found. For the cases with a schedulable initial solution, the value in column 3 and 4 is 0. We have tuned the Tabu Search parameters for both approaches such that the results are as close as possible to the optimal solution (i.e., no improvements were seen for large number of iterations). Tabu Search runs for 8,000 iterations for smaller systems (i.e., cases 1–6) and 16,000 iterations for bigger systems (i.e., cases 7–10).

We can notice that by optimizing the mapping and bandwidth allocation we can find schedulable solution, even in the cases when the initial solution is not schedulable. Both SF and TSMBA have found schedulable solutions in all the cases, i.e., the hard deadlines are satisfied even in the case of transient faults. However, TSMBA finds schedulable solution much earlier than SF, thus exploring the design space much faster, increasing the chances of finding schedulable solutions and improving the QoS of the system. This is because TSMBA uses the PDFs to take better mapping and bandwidth allocation decisions, as highlighted in the motivational examples in section IV-A.

Columns 5 and 6 present the *QoS* obtained by TSMBA and SF, respectively, for the soft tasks. It can be seen from the results that the *QoS* values obtained by TSMBA are better than those of SF on average with 29.60%, with a difference as high as 46.97%. It can also be seen that TSMBA results in solutions with *QoS* close to 96% even in the case of large systems of 20 PEs.

In the second set of experiments we were interested in performing the same evaluation as in the first case, but now considering real-life benchmarks. We have used the Embedded Systems Synthesis Benchmarks Suite (E3S), version 0.9 [26]. The benchmarks contain real-life applications characterized on various processor architectures. Since the benchmarks only report the mean execution time for the soft tasks, we have modified the benchmarks by generating the PDFs for soft tasks by taking into consideration the architecture and the speed of the processors. The benchmarks were also modified to annotate some hard tasks as safety-critical and checkpointing related parameters were added as well. We have considered that these benchmarks are mapped on 2 to 4 processors. Table II presents the experimental setup details and the obtained results. It can be seen from the results that for real-life benchmarks our approach gives solutions with better QoS than the straightforward approach, an average improvement of 28.04%. The deadlines for hard tasks are satisfied and the transient faults are tolerated. We can also see that TSMBA finds schedulable solutions earlier than the SF approach (column 3 and 4, respectively).

In the third set of experiments we were interested to determine the ability of TSMBA to find good quality solutions as the utilization of the system increases. We have used a synthetic benchmark with 26 tasks (16 soft tasks and 10 hard tasks) mapped on architecture with 7 PEs. Table III presents the experimental setup and obtained results. We have varied the execution times resulting in 10 cases corresponding to different initial utilizations (column 5), where for the soft tasks, the bandwidth is set to AET. The spare utilization varies from around 50% in the first case to only 5% for the tenth case. Having larger spare utilization increases the probability of finding solutions with better QoS since more bandwidth can be given to soft tasks. As it can be seen, TSMBA is able to find schedulable solutions with very good QoS even as the initial utilization of the system increases. Whereas SF is not able to find good quality solutions, and the QoS degrades much faster with increase in initial utilization.

## VII. CONCLUSION

In this paper we have addressed the issue of task mapping and processor bandwidth allocation in mixed hard/soft fault-tolerant real-time systems. We have used EDF for the hard tasks and CBS for the soft tasks, thus integrating both hard and soft tasks on the same processor. We have considered that transient faults are tolerated using checkpointing with rollback recovery.

We have proposed a Tabu Search based heuristic which performs design optimizations, resulting in implementations

where the deadlines of all hard real-time tasks are met and QoS of soft tasks in maximized. We have shown that by using stochastic execution times instead of the WCET or AET for the soft tasks, better mapping and processor bandwidth allocation decisions can be taken. The proposed algorithm has been evaluated on several synthetic and real-life benchmarks.

## REFERENCES


[1] G. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, Boston, 1997.

[2] G. Buttazzo, G. Lipari, L. Abeni, and M. Caccamo, *Soft Real-Time Systems: Predictability vs. Efficiency (Series in Computer Science)*. Plenum Publishing Co., 2005.

[3] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997.

[4] V. Izosimov, P. Pop, P. Eles, and Z. Peng, "Scheduling of fault-tolerant embedded systems with soft and hard timing constraints," in *Proceedings of Design, Automation & Test in Europe DATE '08*, 2008, pp. 915–920.

[5] R. Pop and S. Kumar, "A survey of techniques for mapping and scheduling applications to network on chip systems," School of Engineering, Jonkoping University, Research Report 04:4, 2004.

[6] N. Zamora, X. Hu, and R. Mărculescu, "System-level performance/power analysis for platform-based design of multimedia applications," *ACM Transactions on Design Automation of Electronic Systems*, vol. 12, no. 1, 2007. [Online]. Available: http://www.gigascale.org/pubs/1001.html

[7] S. Chakraborty and L. Thiele, "A new task model for streaming applications and its schedulability analysis," in *Proceedings of Design, Automation and Test in Europe DATE '05*, 2005, pp. 486–491.

[8] S. Manolache, P. Eles, and Z. Peng, "Task mapping and priority assignment for soft real-time applications under deadline miss ratio constraints," *ACM Transactions on Embed. Comput. Syst.*, vol. 7, no. 2, pp. 1–35, 2008.

[9] X. S. Hu, T. Zhou, and E. H. M. Sha, "Estimating probabilistic timing performance for real-time embedded systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 6, pp. 833–844, 2001.

[10] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *Proceedings of 19th IEEE Real-Time Systems Symposium*, 1998, pp. 4–13.

[11] L. A. Cortes, P. Eles, and Z. Peng, "Quasi-static scheduling for real-time systems with hard and soft tasks," in *Proceedings of Design, Automation and Test in Europe DATE '04*, vol. 2, 2004, pp. 1176–1181.

[12] L. Abeni, L. Palopoli, G. Lipari, and J. Walpole, "Analysis of a reservation-based feedback scheduler," in *Proceedings of 23rd IEEE Real-Time Systems Symposium*, 2002, pp. 71–80.

[13] A. Oliveira, E. Camponogara, and G. Lima, "Dynamic reconfiguration in reservation-based scheduling: An optimization approach," in *Proceedings of 15th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2009, pp. 173–182.

[14] A. Burns, R. Davis, and S. Punnekkat, "Feasibility analysis of fault-tolerant real-time task sets," *Euromicro Conference on Real-Time Systems*, pp. 29–33, 1996.

[15] P. Pop, V. Izosimov, P. Eles, and Z. Peng, "Design optimization of time and cost-constrained fault-tolerant embedded systems with checkpointing and replication," *IEEE Transactions on VLSI Systems*, vol. 17, pp. 389–402, 2009.

[16] H. Aydin, R. Melhem, and D. Mosse, "Tolerating faults while maximizing reward," in *Proceedings of 12th Euromicro Conference on Real-Time Systems*, 2000, pp. 219–226.

[17] P. K. Saraswat, P. Pop, and J. Madsen, "Task migration for fault-tolerance in mixed-criticality embedded systems," in *Proceedings of 2nd Workshop on Adaptive and Reconfigurable Embedded Systems*, 2009.

[18] E. Wandeler, A. Maxiaguine, and L. Thiele, "Performance analysis of greedy shapers in real-time systems," in *Proceedings of Design, Automation and Test in Europe DATE '06*, vol. 1, 2006, p. 6pp.

[19] J. Hu and R. Mărculescu, "Application-specific buffer space allocation for networks-on-chip router design," *Proceedings of the IEEE/ACM International conference on Computer-aided design ICCAD '04*, pp. 354–361, 2004.

[20] S. Manolache, P. Eles, and Z. Peng, "Buffer space optimisation with communication synthesis and traffic shaping for nocs," *Proceedings of Design, automation and test in Europe DATE '06*, pp. 718–723, 2006.

[21] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei, "Timing analysis of the FlexRay communication protocol," *Real-Time Systems*, vol. 39, no. 1-3, pp. 205–235, 2008.

[22] "aiT Worst-Case Execution Time Analyzer - Homepage," http://www.absint.com/ait/, 2009.

[23] G. Bernat, A. Colin, and S. M. Petters, "WCET analysis of probabilistic hard real-time systems," in *Proceedings of 23rd IEEE Real-Time Systems Symposium RTSS 2002*, 2002, pp. 279–288.

[24] F. Glover, "Tabu Search Part I," *ORSA Journal on Computing*, vol. 1, no. 3, pp. 190–206, 1989.

[25] L. Abeni and G. Buttazzo, "Stochastic analysis of a reservation based system," in *Proceedings of 15th International Parallel and Distributed Processing Symposium*, 2001, pp. 946–952.

[26] "Embedded System Synthesis Benchmarks Suite (E3S) - Homepage," http://ziyang.eecs.umich.edu/dickrp/e3s, 2009.