Towards a Contract-based Fault-tolerant Scheduling Framework for Distributed Real-time Systems

Abhilash Thekkilakattil, Huseyin Aysan, Sasikumar Punnekkat
Mälardalen Real-Time Research Center, Mälardalen University, Sweden
{abhilash.thekkilakattil, huseyin.aysan, sasikumar.punnekkat}@mdh.se

***Abstract*—The increasing complexity of real-time systems has lead to the adaptation of component based methods for their development which has a promising potential for faster and more cost effective development of complex real-time systems by facilitating reuse of the real-time components. This is enabled by the components' composition using contracts, which ensures 'correctness by construction'. Modern real-time systems typically consist of mixed criticality components, and scheduling them in a fault-tolerant as well as efficient way, on a distributed platform, is a challenging task. In this paper, we propose a contract-based approach to fault tolerant scheduling of mixed criticality real-time components on a distributed platform, by providing guarantees for the hard real-time components through offline negotiated contracts, as well as flexibility for the soft real-time components through online (re-)negotiated contracts. The proposed approach uses optimization techniques, that uses timing requirements and the recommendations of studies like Fault Hazard Analysis and Zonal Analysis, to provide the contractual parameters for the mixed-criticality components.**

***Keywords*-Contracts, Fault-tolerance, Components, Real-time systems**

I. Introduction

Most industrial real-time applications typically have to satisfy complex requirements, mapped to task attributes which are further used by the underlying scheduler in the scheduling decision. These systems are often distributed and are characterized by high dependability requirements, where fault tolerance techniques play a crucial role towards achieving them. Traditionally, such systems found in, e.g., aerospace, avionics or nuclear domains, were built with high replication and redundancy, with the objective to maintain the properties of correctness and timeliness even under error occurrences. Complex real-time systems operate in highly unpredictable conditions. Even though such systems are built with utmost care, experience has shown that no amount of safety can be called extra. While time redundancy and space redundancy techniques improve the overall dependability of the system, unexpected interactions of unrelated systems can make them ineffective.

Hardware reliabilty studies like Functional Hazard Analysis (FHA) and Zonal Hazard Analysis (ZHA) are done for safety critical systems to ensure that the proposed redundancies on the hardware components e.g., wires and communication sub-systems, indeed exist. Functional Hazard Analysis (FHA)is first carried out on such systems which identifies various function related events that are of interest to the system and proposes a design criteria based on these events. Zonal Hazard Analysis (ZHA) is then used to analyze the physical locations of the components and interconnections, the possible system to system interactions and severity of potential hazards based on these factors. It basically ensures that unrelated redundant subsystems are not affected by common causes.

Functional Hazard Analysis and Zonal Hazard analysis, when applied to real-time systems, would have a direct impact on the allocation and scheduling of the real-time tasks, introducing extra constraints in addition to the normal replication and precedence constraints. Such constraints may mandate allocation of particular jobs on particular nodes, allocation of replicas on different nodes and minimum physical separation between replicas. Majority of the systems mentioned above are composed of real-time tasks of mixed criticality where the relative criticality of the tasks could undergo changes during the system evolution. The key objectives in these systems are to guarantee the critical tasks and provide a maximum possible level of service to the non-critical tasks. While a lot of research has been done on allocation and scheduling of tasks in distributed systems in a fault tolerant manner, none them take into the consideration, the impact of task criticalities along with the recommendations of functional and Zonal Hazard Analysis. Taking them into consideration while designing the system is crucial for guaranteeing the critical processes.

On the other hand, component Based Software Engineering (CBSE) is becoming a fast emerging trend in the field of software development. Software components used in real-time embedded systems differ from the components built for classical enterprise/business application software as they have to satisfy extra-functional properties in addition to the functional properties. These extra-functional properties arise due to the emphasis that real-time systems place on attributes like timing and dependability [1] constraints. The use of contracts in component based real-time systems aim to ensure the predictability by construction of the resulting

This work was partially supported by the Swedish Research Council project CONTESSE (2010-4276) and ARIES.

software. While the field of real-time uni-processor scheduling has fairly well matured since the pioneering work of Liu and Layland [2], research on building dependable real-time distributed systems using components is still an active area of research and contracts play a vital role in enabling the reuse of real-time components. A task or a group of tasks in an application form a basic unit of composition in many component models [3][4][5]. This allows us to directly apply many of the results in the classical real-time system theory to analyze a component based real-time system and define rules for composing it. Henceforth, we use the terms 'task' and 'component' interchangeably. Whenever a software engineering perspective is considered, we use the term 'component' and whenever a real-time perspective is considered we use the term 'task'.

In this paper we extend our previous works [6][7] to a distributed environment using contracts. We aim to find an optimum allocation scheme that derives execution windows for components to meet reliability requirements. Critical components are guaranteed fault-tolerant execution by using the derived contractual parameters that enable their feasible execution within the derived windows. The service provided to the non-critical components is maximized by using online (re-)negotiated contracts, based on the actual error occurrences.

The rest of the paper is organized as follows: in Sections II and III we present the related work and the system model respectively. The problem statement is given in Section IV and the methodology is described in Section V. The optimization problem formulation is presented in Section VI and the method is illustrated by an example in Section VII. Section VIII concludes the paper outlining on-going efforts.

II. Related Work

There exist numerous works in the field of fault tolerant scheduling of real-time systems. An initial approach to the proposed framework was first presented in [7]. In this paper we build on our previous work [6], which proposes an optimal scheduling of mixed criticality periodic tasks in a fault tolerant manner on a uniprocessor system, and extend it to a distributed environment. Kopetz [8] has detailed the requirements of a fault tolerant real-time distributed system, outlining the need of a solid case of a fault hypothesis and its impact on the various facets of the system design.

Ramamritham et.al. [9] showed that distributed scheduling is effective for hard real-time systems, and the performance of such systems is a function of the current state of the system . They presented a set of four algorithms to schedule tasks with deadlines and resource requirements on a distributed system. In [10], the authors have proposed a fault tolerant scheduling algorithm for aperiodic tasks in multi processor systems.

The optimal allocation and scheduling of real-time tasks on multi processor systems is a well known problem and many works exist in literature that deal with it. Some of them uses heuristics like Simulated Annealing to solve the task allocation problem. Simulated Annealing [11] is a global optimization technique that was derived from the slow cooling of molten metal to for regular crystalline structure. Attiya and Hamam [12] used Simulated Annealing to allocate tasks in a heterogenous real-time system, maximizing the reliability of the system. Bannister and Trivedi [13] proposed a simple heuristic algorithm that evenly distributes the computational load of the tasks over the nodes. Ramamritham [14] presented a more complex heuristic algorithm to allocate tasks in a distributed system. More recently, Islam et.al.[15] proposed a heuristic approach to perform allocation by considering dependability and real-time constraints as well as communication efficiency. In [16], Simulated Annealing is used to allocate tasks with complex constraints on a distributed architecture. The authors have showed that Simulated Annealing finds near optimal solutions for the task allocation problem. They have compared the result with the optimal solutions found by the brute force methods. In [17], the authors presented an exact schedulability tests for fault tolerant real-time task sets. They considered time redundancy as the fault tolerance strategy while deriving these tests.

Baruah et. al [18] showed that the problem of feasibility of preemptive periodic task system on uni-processors is NP-complete in the strong sense. However, they showed that if the utilization of a synchronous system of tasks is bounded by a constant less than 1, there exists a pseudo-polynomial time algorithm and presented the algorithm for it. While solving optimization problems involving allocation of tasks, where the normal strategy involved is generating a solution and checking whether the solution is optimal, the complexity of the feasibility of the tasks is important. Uni-processor scheduling is a fairly matured area of research with good results [2], [18]. Zhang and Burns [19] proposed a heuristic to check the feasibility of real-time tasks on a uni-processor that runs in linear time for most cases.

In [20], the authors underline the need for a comprehensive zonal analysis on critical systems. They have taken the example of the jet aircraft MD-11 of Douglas Aircraft company, showing how zonal analysis changed the initial design of the aircraft after it went into production. Among the various topics discussed in the paper, the interesting recommendations that authors made are concerning the separation and segregation of components. Separation is achieved by physically separating components, e.g., using sheathed wires to prevent a short. Segregation is achieved by placing redundant systems at strategic locations such that they are not affected by common causes e.g, routing redundant power cables through different sides of an aircraft. Such kinds of analysis on software systems could significantly improve the safety of software systems, especially real-time systems which are frequently used in safety critical applications.

Fenelon et. al [21] proposes a design criterion based on such a kind of study when undertaken on software systems. They have discussed the use of safety analysis techniques in providing inputs to the design assessment phase to adopt a suitable design strategy.

Many experimental component models exist that has been successful in persuading the academic and industrial community to explore component based approaches for real-time systems [3][22][4][5]. In many such systems [5][4], contracts are used for achieving flexibility in resource usage, at the same time maintaining the predictability of the hard real-time components. Thus, predictable flexibility [23] can be achieved using contracts.

In this paper, we aim to address the problem of allocation and scheduling of mixed criticality hard real-time components on a distributed system, while taking into consideration the recommendations of reliability studies like Functional and Zonal Hazard Analysis, when done on software systems [21]. The main differences/advantages of our approach over the approaches mentioned so far are:

1) Efficient handling of component criticalities using contracts
2) Improved processor utilization and hence cost reduction through optimization.
3) Fault tolerance strategies that cover multiple types of faults.
4) Graceful degradation of the system under faults

In addition, we also present a new processor utilization demand analysis that can handle offsets in synchronous real-time task sets.

III. System Model

A. Error Model

In this paper, we assume that a component represents a unit of failure. This failure may be due to a value or a timing error that compromises the correctness of the system. We also assume another class of errors, *zonal errors*, which arises due to the unpredictable interactions of unrelated components. This is assumed to be identifiable at the design time by mechanisms such as Zonal Hazard Analysis [21]. We also assume that any node can fail due to hardware errors, which are considered as a part of *zonal errors* and have the same recovery mechanism. We assume that each critical task has a known error frequency which can be determined with a high degree of confidence. We assume that the error detection is implemented in the underlying system and its worst case overhead is known. This error detection mechanism flags the errors as soon as it detects one. We assume that all the nodes have a consistent view of the errors flagged in the system. The recovery action is performed by the execution of a replica either in the same processor in case of a value/timing errors or in a different processor in case of a zonal error. It is assumed that during a recovery, the non-critical components are shed to execute the alternates.

B. Computational Model

We consider a distributed real-time system with uniform multi-processors which communicate over a reliable communication media and are synchronized by relatively loose synchronization algorithms implemented in the software. We denote the set of n components by $\Gamma = \{\tau_1, \tau_2, .., \tau_n\}$, where each component represents a real-time thread of execution. Each component τ_i has a period T_i and a known worst case execution time C_i such that $\frac{C_i}{T_i} \leq 1 \ \forall i \in [1, n]$. The replication requirements on Γ are specified by $R = \{r_1, r_2, \ldots r_n\}$, where $r_i \in [0, \lfloor \frac{T_i}{C_i} \rfloor - 1]$. Additionally, $M = \{m_1, m_2, \ldots m_n\}$, where $m_i \in [0, r_i]$, specifies the distribution requirement of τ_i, i.e., the number of different nodes that need to be used for its replication. For instance, m_1=3 indicates that the three replicas of the component τ_1 needs to be allocated in three different processors. We also use a binary variable β_i to denote the criticality of a task τ_i, $\beta_i = 1$ if τ_i is a critical task and $\beta_i = 0$ if τ_i is a non-critical task. The main component is called the primary component and its recovery is called a replica or an alternate.

The execution time of a replica of τ_i is denoted by $\overline{C_i}$, where $\overline{C_i} \leq C_i$. Consequently, we denote the set of critical components, primaries and alternates by $\Gamma_c = \Gamma_c^{pri} \cup \Gamma_c^{alt}$, and the set of non-critical components by Γ_{nc}. Additionally, we use τ_j^k to denote the k^{th} replica of τ_j. When $k = 0$, τ_j^k represents the primary if τ_j is a critical component and the component itself in case it is a non-critical task. LCM represents the least common multiple of all the time periods of the components in Γ. The release time of τ_j^k is given by rel_j_k and its deadline is given by dl_j_k. A contract for a task can be seen as a set of task parameters that guarantee its schedulability according to the scheduling algorithm used e.g., priorities in case a Fixed Priority Scheduling (FPS) scheme is used.

We assume that the components have deadlines equal to their periods. Each component has three main operational stages. First stage is the input stage where the input data is received from sensors or other components. Second stage is the computation stage and the third stage is the output stage where the output is delivered to the next component in the component chain or to the environment as a system output. We also assume that a component τ_i can execute only on one processor at a time instant. Execution of error detection or error handling mechanisms such as sanity checks and re-execution of failed computations are considered as a part of the computation stage.

IV. Problem statement

We assume a set of components Γ consisting of critical and non critical components $\Gamma_c \cup \Gamma_{nc} = \{\tau_1, \ldots, \tau_n\}$ with associated replication requirements $R = \{r_1, \ldots, r_n\}$,

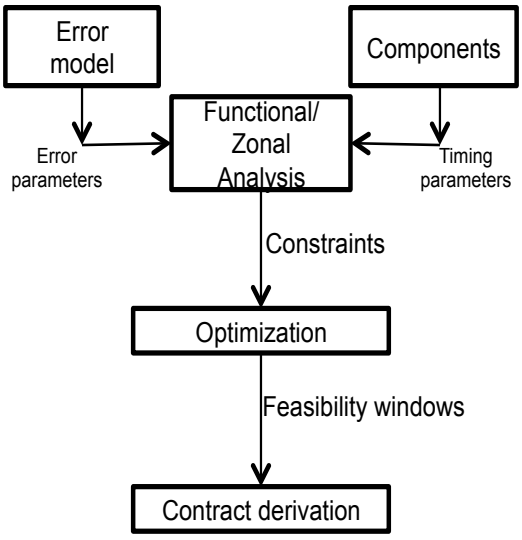

Figure 1. Methodology overview

$r_i \in [0, \lfloor \frac{T_i}{C_i} \rfloor - 1]$ and distribution requirement $M = \{m_1, m_2, \ldots m_n\}$, $m_i \in [0, r_i]$. We want to find the contractual scheduling parameters for the components Γ such that:

1) the schedulability of the critical components and alternates, $\Gamma_c = \{\tau_i \mid r_i \neq 0\}$ is *guaranteed*
2) the schedulability of non critical components, $\Gamma_{nc} = \{\tau_i \mid r_i = 0\}$, is *maximized*
3) the utilization on processing nodes is maximized
4) the number of processing nodes is minimized

V. Methodology

Our goal is to, first, derive feasibility windows that reflect the criticality requirements. Hence, depending on the criticality of the original components, the new feasibility windows we are looking for differ as:

1) *Fault Tolerant* (FT) feasibility windows for critical components
2) *Fault Aware* (FA) feasibility windows for non-critical components

Then we derive contractual parameters for each critical and non-critical components such that the critical components and their replicas are guaranteed and the non-critical component receive a service level proportional to the resource usage in the system. Critical components are associated with offline negotiated contracts that map scheduling parameters to feasibility windows. Non-critical components are allocated resources at runtime, online depending on the actual error occurrences.

A. Derivation and allocation of feasibility windows

A Fault Tolerant Feasibility Window (FT_FW) is a temporal window in which a critical component has to execute and complete, such that it can feasibly re-execute (i.e., before its original deadline) upon an error. A Fault Aware Feasibility Window (FA_FW) is a temporal window allocated to non-critical components, in order to control their interference with the critical ones, i.e., the execution of a non-critical component may not jeopardize the fault tolerant execution of any critical one. In our method, the derivation of feasibility windows is performed jointly with the component allocation to processors. A feasibility window is considered to be valid only if it satisfies certain temporal properties, e.g., the size of a feasibility window of a component should be at least equal to the WCET of the component or its replica. We use optimization theory to derive the feasibility windows in a resource efficient manner.

An important point to be noted here is that, in this paper, we allocate the last m_i of the r_i replicas in different processors. This is to minimize the overhead involved during recovery. A re-execution is tried first on the same processor as the primary is scheduled, only if this fails an attempt is made to re-execute it on a different processor. Re-execution on a different processor is costly in terms of the network bandwidth since data has to be send back and forth for this purpose. Thus it is most desirable to perform a recovery on the same processor and only if this fails should it be attempted on a different processor. However, this does not affect the generality of the method as more complex criterion could be used to allocate the replicas.

B. Offline negotiated contracts for critical components

The fault-tolerant execution of the critical components is guaranteed using offline negotiated contracts. Offline negotiated contracts represent the new component parameters which are derived in order to ensure the component execution within their respective feasibility windows, previously derived in step V-A.

C. Online negotiation for non-critical components

Non-critical components are allocated to Fault Aware Feasibility Windows during the feasibility window derivation phase i.e., in the remaining slack after the allocation and scheduling of critical components. However, they are executed only if they can successfully negotiate a contract with the system at runtime. The success of the negotiation depends on several runtime parameters e.g., resource usage and error occurrence.

VI. Optimization Formulation

In this section we present the formulation of the non-linear optimization problem of allocating the components in the minimum number of processors such that the replication requirements are satisfied. Let P_i be the variable that indicate whether the i^{th} processor is used. Our goal function is to minimize the number of processors:

$$G = \sum_{i=1}^{MAX} P_i$$

$$MAX = n + \sum_{j=1}^{n} m_j$$

Here, n is the number of tasks, m_j represent the distribution requirement of τ_j and MAX represents the maximum number of processors that will be required to schedule the given components. This is the case when the component and its last m_i replicas (that has to be scheduled on a different processor) are each allocated exclusively to a processor.

Let Pi_j_k be the binary variable that represents whether the k^{th} replica of the component τ_j (i.e., τ_j^k) is allocated to the i^{th} processor. Pi_j_k can be either one or zero with one indicating that τ_j^k is allocated to processor P_i and zero otherwise. Since non-critical components do not have any replicas, $k = 0$ for all non-critical components. This however is implicitly taken care by the system of equations.

$$\forall i \in [1, MAX],\ \forall j \in [1, n]\ and\ \forall k \in [0, r_j]$$
$$Pi\_j\_k \leq 1$$

Note that $r_j = 0$ for a non-critical components and the equation for the constraint remains valid.

If any one of $Pi\_j\_k = 1$, then the corresponding P_i is also 1, i.e., $\forall i \in [1, MAX]$,

$$P_i = \left\lceil \frac{\sum_{j=1}^{n} \sum_{k=0}^{r_j} Pi\_j\_k}{1 + \sum_{j=1}^{n} \sum_{k=0}^{r_j} Pi\_j\_k} \right\rceil$$

Here, we sum up the Pi_j_k's and divide the sum by one more than this sum and find the smallest integer greater than or equal to the result. In case the Pi_j_k's sum up to zero, the one in the denominator will make sure that a divide by zero error does not occur. If they do not sum up to zero, we get one since any number when divided by a larger number will result in a value between zero and one, the ceil of which will give a one. Thus, P_i will either be zero or one.

The release time of the primary of a component τ_j, given by rel_j_0, is set to the start of its period.

$$\forall j \in [1, n]$$
$$rel\_j\_0 = 0$$

Similarly the release time of τ_j^k, given by rel_j_k, is set to the deadline of the previous replica i.e., τ_j^{k-1} for each critical component τ_j:

$$\forall j \in [1, n],\ \forall k \in [1, r_j]\ and\ \beta_j = 1$$
$$rel\_j\_k = dl\_j\_(k-1)$$

The minimum size of a window of the primary of a component or its replica, defined by a release time and deadline should be at least equal to the WCET of the component or its replica. We write this constraint as two constraints; one for the primary and the other for the replicas since the primary and the replicas can have different execution times and also because non-critical components do not have replicas.

$$\forall j \in [1, n]$$
$$dl\_j\_0 \geq rel\_j\_0 + C_j$$

and,

$$\forall j \in [1, n],\ \forall k \in [1, r_j],\ and\ \beta_j = 1$$
$$dl\_j\_k \geq rel\_j\_k + \overline{C_j}$$

The deadline of the replica k of τ_j should not lead to an infeasibility in the derivation of a valid execution window for the replica $k + 1$ of the same component.

$$\forall j \in [1, n],\ \forall k \in [0, r_j],\ and\ \beta_j = 1$$
$$dl\_j\_k \leq T_j - ((r_j - k) \times \overline{C_j})$$

Similarly, the deadline of a non-critical component must be less than or equal to the end of its period.

$$\forall j \in [1, n],\ and\ \beta_j = 0$$
$$dl\_j\_0 \leq T_j$$

The two constraints given above specifies an upper bound on the deadlines of a feasibility window. The first constraint bounds the deadline of a replica k such that there is provision to derive valid feasibility windows for replica $k + 1$, of at least the minimum size which is equal to the execution time of the replica.

The next constraint ensure that the last m_j replicas of τ_j^k, are allocated to different processors.

$$\forall i \in [1, MAX],\ \forall j \in [1, n],\ and\ \beta_j = 1$$
$$Pi\_j\_0 + \sum_{k=r_j - m_j + 1}^{r_j} Pi\_j\_k \leq 1$$

Here either Pi_j_0 or one of the last Pi_j_k's can be equal to 1.

The next constraint ensure that the primary and the replicas of a component are indeed allocated to one of the processors i.e., none of them remain unallocated.

$$\forall k \in\ [0, r_j],\ and\ \forall j \in\ [1, n]$$
$$\sum_{i=1}^{MAX} Pi\_j\_k\ =\ 1$$

The above constraint mandates that at least and only one of the Pi_j_k of τ_j^k is equal to one for $i \in [1, MAX]$.

The last set of constraints ensure that the processor utilization demand during any time interval does not exceed the length of the interval. This ensures the schedulability of the components allocated to each processor. The important problem here is to allocate the components in such a way that the non-critical components ($\beta_j = 0$) and the replicas

of the critical components ($\beta_j = 1$) are allocated in a mutually exclusive manner. This is because non-critical components can be scheduled in overlapping windows with that of the replicas since the non-critical components are shed upon errors. This is achieved by two sets of constraints on the utilization of each processor: one ensuring that the total utilization of all critical components and its replicas scheduled on a processor is less than 100% for any time interval and the total utilization of critical and non-critical components scheduled on a processor is less than 100% in any time interval on the same processor.

The constraint while allocating the critical component primaries and their replicas is:

$$\forall i \in [1, MAX], t_1 < t_2$$

$$\sum_{\tau_x^z \in \Gamma_c} \eta_{x,z}(t_1, t_2) C'_x \leq t_2 - t_1, \quad \forall t_1, t_2$$

Where, $\eta_{x,z}(t_1, t_2)$ gives the number of instances of τ_x^z released in the interval $[t_1, t_2]$ and $C'_x = C_x$ when the primary of τ_x is considered and $C'_x = \overline{C_x}$ when its alternate is considered.

The above constraint ensures that the processor utilization demand during any interval does not exceed the length of the interval while allocating the critical components' primaries and its replicas.

The next set of constraints ensures the schedulability of the critical components' primaries along with the non-critical components.

$$\forall i \in [1, MAX], t_1 < t_2$$

$$\sum_{\tau_x^0 \in \Gamma_c^{pri} \cup \Gamma_{nc}} \eta_{x,0}(t_1, t_2) C_x \leq t_2 - t_1, \quad \forall t_1, t_2$$

Where, $\eta_{x,0}(t_1, t_2)$ gives the number of instances of τ_x^0 released in the interval $[t_1, t_2]$. We remind the reader that τ_x^0 represent the non-critical component in case $\beta_x = 0$ and the primary in case $\beta_x = 1$. This constraint ensures that the processor utilization demand during any interval does not exceed the available processor time during the interval while allocating the critical components' primaries and the non-critical components.

Let us now review the processor utilization demand analysis for real-time tasks proposed by Baruah et. al [18].

Processor Utilization Demand Analysis: The seminal paper by Baruah et. al [18] show that the number of task releases between any two time interval t_1 and t_2 can be efficiently computed. They have also given a polynomial time algorithm for the feasibility of task sets with utilization bounded by a constant less than 1. We now review how the authors showed that the number of instances can be efficiently computed.

The authors considered two inequalities:

$$t_1 \leq s_i + KT_i$$

and,

$$t_2 \geq s_i + KT_i + d_i$$

where d_i is the relative deadline of τ_i and K is a natural number. For synchronous task systems, $s_i = 0$.

They solved the inequalities to get the number of values of K that satisfies both the inequalities which gives the number of instances of τ_i completely scheduled in the interval $[t_1, t_2]$ (denoted by $\eta(t_1, t_2)$) i.e.,

$$\eta(t_1, t_2) = max \left\{0, \left\lfloor \frac{t_2 - d_i}{T_i} \right\rfloor - \left\lceil \frac{t_1}{T_i} \right\rceil + 1\right\}$$

This method does not suit us;we show this by an example. Consider a task τ_i having a period T_i, offset off_i and relative deadline with respect to the start of the period $d_i \leq T_i$. In our optimization strategy, we have to find the number of instances of τ_i completely scheduled between off_i and d_i. Substituting $t_1 = off_i$ and $t_2 = d_i$, we get:

$$\eta(t_1, t_2) = max \left\{0, \left\lfloor \frac{d_i - d_i}{T_i} \right\rfloor - \left\lceil \frac{off_i}{T_i} \right\rceil + 1\right\}$$

$$\Rightarrow \eta(t_1, t_2) = max \{0, 0 - 1 + 1\} = 0$$

However, the number of instances of τ_i completely scheduled in the interval between off_i and d_i is 1. Thus the method presented in [18] does not suit our need. We obtain $\eta(t_1, t_2)$ following a similar strategy as in [18].

We assume a synchronous task system with relative deadline $d_i \leq T_i$. Let $\eta(t_1, t_2)$ denote the total number of natural numbers K that satisfy *both* the following inequalities for $0 \leq t_1 < t_2$:

$$t_1 \leq KT_i + off_i \tag{1}$$

$$KT_i + d_i \leq t_2 \tag{2}$$

Thus $\eta(t_1, t_2)$ gives the number of instances of τ_i that are *completely* scheduled in the interval $[t_1, t_2]$

Lemma VI.1.

$$\eta(t_1, t_2) = max \left\{0, \left\lfloor \frac{t_2 - d_i}{T_i} \right\rfloor - \left\lceil \frac{t_1 - off_i}{T_i} \right\rceil + 1\right\}$$

Proof: The inequalities 1 and 2 transforms to:

$$K \geq \frac{t_1 - off_i}{T_i} \tag{3}$$

$$K \leq \frac{t_2 - d_i}{T_i} \tag{4}$$

Thus, the smallest K that satisfy the inequality 3 is,

$$K = \left\lceil \frac{t_1 - off_i}{T_i} \right\rceil$$

and the largest K that satisfy the inequality 4 is,

$$K = \left\lfloor \frac{t_2 - d_i}{T_i} \right\rfloor$$

Thus the total number of natural numbers that satisfy *both* the inequalities 1 and 2 is:

$$\eta(t_1, t_2) = max\left\{0, \left\lfloor \frac{t_2 - d_i}{T_i} \right\rfloor - \left\lceil \frac{t_1 - off_i}{T_i} \right\rceil + 1\right\}$$

Thus, $\eta(t_1, t_2)$ can be computed efficiently for synchronous task sets with offsets. ■

Note however that this increases (squares) the number of points to be checked for feasibility. Thus, we need to explore more efficient algorithms for schedulability analysis of task sets with offsets. The quick processor demand analysis proposed by Zhang and Burns [19], could be extended to check the feasibility of periodic tasks with offsets.

VII. Example

We illustrate the proposed methodology by a simple example. Consider a set of components Γ={A,B,C,D} with parameters specified in Table I.

Table I
SET OF COMPONENTS- EXAMPLE

τ_i	T_i	C_i	R_i	M_i	$U_i(pri + alt)$	β_i
A	10	2	2	1	0.6	1
B	5	2	1	1	0.8	1
C	5	1	0	0	0.2	0
D	10	6	0	0	0.6	0

We denote the k^{th} alternate of the component A and B by A_k and B_k. In our example, component A has T_i=10, C_i=2, R_i=2, M_i=1, which means that, the last 1 of A's alternate needs to be executed in a different node than its primary.

We derive the constraints presented in section VI and perform the optimization to get an allocation of tasks in the minimum number of processors. For example, constraints on the release times and deadlines of the primary and the replicas of component A are as follows:

$$rel\_A\_0 = 0$$

$$rel\_A\_1 = dl\_A\_0$$

$$rel\_A\_2 = dl\_A\_1$$

$$dl\_A\_0 \geq rel\_A\_0 + C_A$$

$$dl\_A\_1 \geq rel\_A\_1 + \overline{C_A}$$

$$dl\_A\_2 \geq rel\_A\_2 + \overline{C_A}$$

$$dl\_A\_0 \leq T_A - 2 \times C_A \Rightarrow dl\_A\_0 \leq 6$$

$$dl\_A\_1 \leq T_A - C_A \Rightarrow dl\_A\_1 \leq 8$$

$$dl\_A\_2 \leq T_A \Rightarrow dl\_A\_2 \leq 10$$

Similarly, all the contraints are derived and optimization is performed to obtain the component allocation on the minimum number of processors. The optimal allocation of the critical primaries, its alternates and the non-critical components in Fault Tolerant and Fault Aware Feasibility Windows, will result in a fault tolerant schedule in the minimum number of processors. The allocation of the FT/FA feasibility windows and the component executions in the worst case error occurrence scenario, is presented in figure 2.A. In this scenario, A_0, B_0 and $A_1$are hit by faults. Hence,

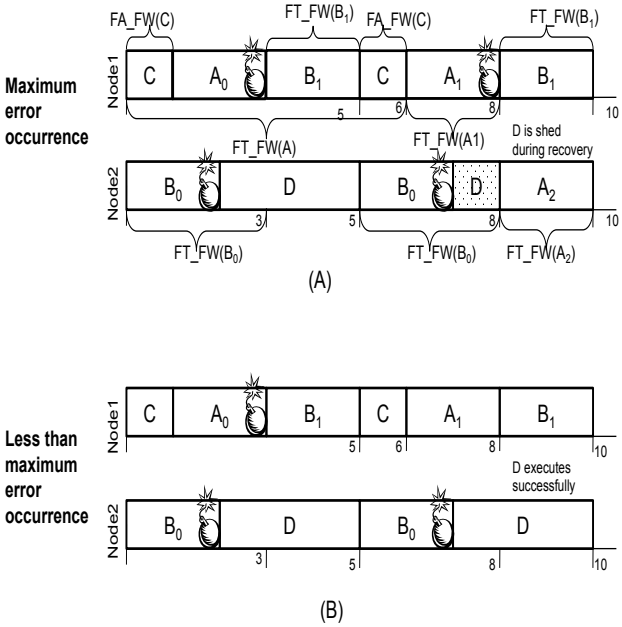

Figure 2. (A). Allocation and execution under worst case error scenario (B). Allocation and execution under average case error scenario

the non-critical component D is shed due to the temporary overload on node 2, while C can still feasibly execute. At runtime, however, it is unlikely that the worst case scenario will occur. Consider that only the primary of A, i.e., A_0 is hit by an error in addition to the error on B_0. In this case, A_1 successfully execute as a result of which the execution of A_2 is no longer required. This creates the sufficient slack for the execution of component D, as illustrated in Figure 2.B, as a result of which D can execute successfully. Note that the fault tolerant and fault aware windows of the components in Figure 2.B are the same as in Figure 2.A.

In any case, the execution of the critical primaries and alternates are guaranteed feasible execution on the minimum number of nodes.

VIII. Conclusions and ongoing work

In this paper we present a contract based approach to fault tolerant scheduling of mixed criticality real-time systems, by providing guarantees for the hard real-time components through offline negotiated contracts, as well as flexibility for the soft real-time components through online negotiated contracts. We built on our previous works while extending it to a distributed environment with stringent reliability

constraints. Mathematical optimization is used to derive optimal feasibility windows for component executions and re-executions while minimizing resource usage. The re-execution requirements reflect the error scenarios in the system in addition to the results of studies like Functional Hazard Analysis and Zonal Hazard Analysis to maximize reliability.

Future works will focus on the implementation of the proposed approach and comparison with other approaches, extensions to incorporate energy aware mechanisms and migration of non-critical components to gain even better levels of service.

REFERENCES

[1] A. Avizienis, J. Laprie, and B. Randell, "Fundamental concepts of dependability," *Research Report N01145, LAAS-CNRS*, 2001.

[2] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *The Journal of ACM*, 1973.

[3] T. Bures, J. Carlson, I. Crnkovic, S. Sentilles, and A. Vulgarakis, "ProCom- the progress component model reference manual, version 1.0," Mälardalen University, Technical Report ISSN 1404-3041 ISRN MDH-MRTC-230/2008-1-SE, 2008.

[4] "FRESCOR-final project report (deliverable: D-FR)." [Online]. Available: http://www.frescor.org/index.php?page=publications

[5] M. Aldea, G. Bernat, I. Broster, A. Burns, R. Dobrin, J. Drake, G. Fohler, P. Gai, M. Harbour, G. Guidi, J. Gutierrez, T. Lennvall, G. Lipari, J. Martinez, J. Medina, J. Palencia, and M. Trimarchi, "FSF: A real-time scheduling architecture framework," in *The 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2006.

[6] R. Dobrin, H. Aysan, and S. Punnekkat, "Maximizing the fault tolerance capability of fixed priority schedules," in *The 14th IEEE Internationl Conference on Embedded and Real-Time Computing Systems and Applications*, 2008.

[7] A. Thekkilakattil, R. Dobrin, S. Punnekkat, and H. Aysan, "Optimizing the fault tolerance capabilities of distributed real-time systems," in *14th International Conference on Emerging Technologies and Factory Automation, WiP*, 2009.

[8] H. Kopetz, "On the fault hypothesis for a safety-critical real-time system," in *Automotive Software Connected Services in Mobile Networks, Lecture Notes in Computer Science*, 2006.

[9] K. Ramamritham, J. Stankovic, and W. Zhao, "Distributed scheduling of tasks with deadlines and resource requirements," *IEEE Transactions on Computers*, 1989.

[10] S. Ghosh, R. Melhem, and D. Mosse, "Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems," *IEEE Transactions on Prarallel and Distributed Systems*, 1997.

[11] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," 1983.

[12] G. Attiya and Y. Hamam, "Task allocation for maximizing reliability of distributed systems: A simulated annealing approach," *Journal of Parallel and Distributed Computing*, 2006.

[13] J. A. Bannister and K. S. Trivedi, "Task Allocation in Fault-Tolerant Distributed Systems," *Acta Informatica, Springer-Verlag*, 1983.

[14] K. Ramamritham, "Allocation and scheduling of complex periodic tasks," *The 10th International Conference on Distributed Computing Systems*, 1990.

[15] S. Islam, R. Lindstrom, and N. Suri, "Dependability driven integration of mixed criticality sw components," *Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, 2006.

[16] K. Tindell, A. Burns, and A. Wellings, "Allocating hard real time tasks + (an np-hard problem made easy)," 1993.

[17] A. Burns, R. Davis, and S. Punnekkat, "Feasibility analysis of fault-tolerant real-time task sets," 1996.

[18] S. K. Baruah, R. R. Howell, and L. Rosier, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor," *Real-Time Systems*, 1990.

[19] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with edf scheduling," *IEEE Transactions on Computers*, 2009.

[20] R. Caldwell and D. Merdgen, "Zonal analysis: the final step in system safety assessment [of aircraft]," 1991.

[21] P. Fenelon, J. A. McDermid, M. Nicolson, and D. J. Pumfrey, "Towards integrated safety analysis and design," *SIGAPP Appl. Comput. Rev.*, 1994.

[22] H. Hansson, M. Åkerholm, I. Crnkovic, and M. Törngren, "SaveCCM- a component model for safety-critical real-time systems," in *Euromicro Conference, Special Session Component Models for Dependable Systems*, 2004.

[23] R. Dobrin, "Combining off-line schedule construction and fixed priority scheduling in real-time computer systems," Ph.D. dissertation, Mälardalen University, 2005.