# Optimizing the Fault Tolerance Capabilities of Distributed Real-Time Systems

Abhilash Thekilakkattil, Radu Dobrin, Sasikumar Punnekkat, and Huseyin Aysan
Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden
atl09001@student.mdh.se, {radu.dobrin, sasikumar.punnekkat, huseyin.aysan}@mdh.se

## Abstract

*Industrial real-time systems typically have to satisfy complex requirements, mapped to the task attributes, eventually guaranteed by a fixed priority scheduler in a distributed environment. These systems consist of a mix of hard and soft tasks with varying criticality, as well as associated fault tolerance requirements. Time redundancy techniques are often preferred in industrial applications and, hence, it is extremely important to devise resource efficient methodologies for scheduling real-time tasks under failure assumptions.*

*In this paper, we propose a methodology to provide a priori guarantees in distributed real-time systems with redundancy requirements. We do so by identifying temporal feasibility windows for all task executions and re-executions, as well as allocating them on different processing nodes. We then use optimization theory to derive the optimal feasibility windows that maximize the utilization on each node, while avoiding overloads. Finally on each node, we use Integer Linear Programming (ILP) to derive fixed priority task attributes that guarantee the task executions within the derived feasibility windows, while keeping the associated costs minimized.*

## 1. Introduction

Most industrial real-time applications typically have to satisfy complex requirements, mapped to task attributes and further used by the underlying scheduler in the scheduling decision. These systems are often distributed and 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. However, in majority of modern embedded applications, due to space, weight and cost considerations, it may not be feasible to provide high levels of space redundancy. Such systems often have to exploit time redundancy techniques as well together with affordable levels of space redundancy.

These systems consist of a mix of hard and soft tasks with varying criticality where the relative criticality of tasks could undergo changes during the system evolution. While the redundancy requirements typically specify the number of re-executions required per critical task, the distribution constraints resulting from e.g., zonal analysis, typically require the re-execution of a number of replicas on different computing nodes. Hence, the task allocation and the joint scheduling of critical and non-critical tasks in such systems is an extremely challenging task.

Fault tolerance is one of the fundamental means to ensure dependability [3] of computer systems. The goal of our research is to ensure better levels of fault tolerance to critical tasks in terms of tolerating greater frequency of faults and to provide better schedulability of non-critical tasks. Fault tolerance by time redundancy of critical tasks have been addressed previously and scheduling techniques [6, 7] have been developed to tolerate errors of different types. These are however inadequate when such systems are deployed in more error prone conditions due to their physical environments or operational constraints. It has been shown that distributed scheduling is effective even for hard real time systems and that the performance of such systems is a function of the current state of the system [12]. In [8] the authors addressed the issue of space redundancy by using an objective function based on a utility model which can be incorporated to the search algorithm and serve as a heuristic for guaranteeing fault tolerance requirements. Task allocation to multiple nodes have been studied by many researchers. Bannister and Trivedi [4] proposed a simple heuristic algorithm that evenly distributes the computational load of the tasks over the nodes. Ramamritham [11] presented a more complex heuristic algorithm to deal with precedence and fault tolerance constraints. More recently, Islam et.al. [9] proposed a heuristic approach to perform allocation by considering dependability and real-time constraints as well as communication efficiency. Yu and Prasanna [14] formulated the problem of power-aware task allocation on heterogeneous multiprocessor systems as an integer linear programming problem and used linearization heuristics for solving it. AlEnawy and Aydin [1] studied the power-aware task allocation problem for rate monotonic scheduling policy and compared the performances of different heuristics for solving this problem. Burns et.al. [5, 13] applied simulated annealing (SA) technique for solving allocation problems, which is a global optimization technique developed by Kirkpatrick et.al. [10]. Later, Attiya and Hamam [2] used SA to perform task allocation while trying to maximize the system reliability.

In this paper we extend our previous works [7] to a distributed environment. We aim to find optimum task allocation and FPS attribute derivation on the minimum number of nodes, such that the reliability constraints are satisfied while the system schedulability is guaranteed. We use the concept of processor utilization demand during the task allocation to ensure overload avoidance. While our target is dependable industrial systems that employ the fixed priority scheduling paradigm, the results produced by our methodology are applicable to any major real-time scheduling policy. The rest of the paper is organized as follows: in section 2 we present the task and error model. The problem statement is given in section 3 and the methodology is described in section 4 and illustrated by an example in section 5. Section 6 concludes the paper outlining on-going efforts.

## 2. Task and error model

We consider a distributed real-time system that consists of processing nodes which communicate over a reliable communication media and are synchronized by relatively loose synchronization algorithms implemented in the software. We denote the set of tasks by $\Gamma_i = \{\tau_1, \tau_2, .., \tau_n\}$, where each task represents a real-time thread of execution. Each task $\tau_i$ has a period $T(\tau_i)$, a known best case execution time $BCET(\tau_i)$ and a known worst case execution time $WCET(\tau_i)$. The replication requirements on $\Gamma$ are specified by $R = \{r_1, r_2, \dots r_n\}$, where $r_i \in [0, \lfloor \frac{T_i}{WCET(\tau_i)} \rfloor - 1]$. Additionally, $M = \{m_1, m_2, \dots m_n\}$ specifies the number of different nodes that need to be used for the task replications. For instance, $m_1$=3 indicates that the 3 replicas of task $\tau_1$ needs to be allocated in different processors. Consequently, we denote the set of critical tasks, primaries and alternates by $\Gamma_c = \Gamma_c^{pri} \cup \Gamma_c^{alt}$, and the set of noncritical tasks by $\Gamma_{nc}$. We assume that the utilization of the non critical tasks does not exceed the utilization requirement of the critical alternates: $U_{nc} \leq U_c^{alt}$.

We assume that the tasks have deadlines equal to their periods. Tasks have three main operational stages. First stage is the input stage where the input data is received from sensors or other tasks. Second stage is the computation stage and the third stage is the output stage where the output is delivered to the next task in the task chain or to the environment as a system output. 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.

We mainly target the transient faults that can be potentially tolerated by simple re-execution of the task. We also try to overcome effects of node failures by scheduling the alternates of certain critical tasks in different processors.

## 3. Problem statement

We assume a task set $\Gamma$ consisting of critical and non critical periodic tasks $\Gamma_c \cup \Gamma_{nc} = \{\tau_1, \dots, \tau_n\}$ with associated replication requirements $R = \{r_1, \dots, r_n\}$, $r_i \in [0, \lfloor \frac{T_i}{WCET(\tau_i)} \rfloor - 1]$. We want to find the scheduling parameters for the task set $\Gamma$ such that:

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

## 4. Methodology

Our goal is to, first, derive feasibility windows for each task instance $\tau_i^j \in \Gamma$, and allocate them on nodes, to reflect the FT requirements. Then, we assign FPS attributes that ensure task executions within their new feasibility windows, thus, fulfilling the FT requirements.

While executing non-critical tasks in the background can be a safe and straightforward solution, in our approach we aim to provide non-critical tasks a better service than background scheduling. Hence, depending on the criticality of the original tasks, the new feasibility windows we are looking for differ as:

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

While critical task instances need to complete within their FT feasibility windows to be able to re-execute feasibly upon an error, the derivation of FA feasibility windows has two purposes: 1) to prevent non-critical task instances from interfering with critical ones, i.e., to cause a critical task instance to miss its deadline, while 2) enabling the non-critical task execution at high priority levels. Since the size of the FA feasibility windows depend on the size of the FT feasibility windows, in our approach we first derive FT-feasibility windows and then FA feasibility windows. Then, we assign fixed priorities to ensure the task executions within their newly derived feasibility windows.

**Deriving the minimum number of nodes** The necessary condition for scheduling the task set $\Gamma$ on a minimum number of processing nodes is that the utilization requirement on each node does not exceed the processing capability of that particular node. Hence, the minimum number of processors required to schedule the critical tasks (primaries and alternates) is $N = \lceil U(\Gamma_c) \rceil =$

$\lceil (U(\Gamma_c^{pri}) + U(\Gamma_c^{alt})) \rceil$. If we take the replication requirements $m_i$ into account, the number of nodes required to achieve our goals is

$$N = max(\lceil U(\Gamma_c) \rceil, max(m_i) + 1)$$

We assume that in each node there exist a table M which indicates the node where the next replica of the task is scheduled. The $M[i][j]^{th}$ entry represent the processor in which the $j^{th}$ replica of the task $\tau_i$ has been scheduled. If no fault occurs on the $j^{th}$ replica of task $\tau_i$, a message is passed to the $M[i][j+1]^{th}$ node not to execute the next replica.

**Derivation and allocation of Fault Tolerant feasibility windows** The constraints for the derivation and allocation of the FT feasibility windows are:

$\forall \tau_i \in \Gamma_c,\ \forall j \in [0, \frac{LCM}{T_i}]$

C1: $end(FT\_FW(\tau_i^j)) - start(FT\_FW(\tau_i^j)) \geq C_i$

C2: $end(FT\_FW(\tau_i^j)) \leq start(FT\_FW(\tau_i^{j+1})$

C3: the processor utilization demand during any interval $[t_n, t_{n+1})\ where\ n = \{0, 1, \ldots\}$, is not greater $t_{n+1} - t_n$

where LCM represent the least common multiple of task periods. Also in C3, $t_n$ and $t_{n+1}$ are defined by either a start or an end of a FT feasibility window.

**Derivation and allocation of Fault Aware feasibility windows** We aim to identify FA feasibility windows for non-critical task instances to protect critical ones from being adversely affected. According to our assumptions, the total utilization of critical tasks and alternates, together with non-critical tasks, exceeds the available resources. Hence, as a part of recovery action upon errors, the underlying fault tolerant on-line mechanism checks if there is enough time left for the non-critical task instances to complete before their new deadlines. If not, these instances are not executed.

To derive and allocate the FA feasibility windows, we use the same approach as for the FT_FW derivation and allocation, on the set of non-critical tasks, $\Gamma_{nc}$, but *in the remaining slack* after the critical task primaries are scheduled to execute as late as possible. We do so due to two reasons: we want to prevent non-critical tasks from delaying the execution of critical primaries beyond their FT deadlines, and to allow non-critical tasks to be executed at high priority levels.

In some cases, we may fail finding valid FA_FW for some non-critical task instances. This scenario could occur if the optimization solver fails due to violations of C3, i.e., it can not find an interval in which the utilization demand is less than the length of the interval. In these cases, we keep the original feasibility windows, e.g.,periods, and we make sure that the priority assignment mechanism will assign these non-critical tasks a background priority, i.e., lower than any other critical task, and any other non-critical task with a *valid FA_FW*.

**Derivation of FPS attributes** At this point, we can use the method earlier developed in [7] to derive feasible FPS attributes to the tasks. The basic idea is to analyze the overlapping between feasibility windows on each node, and to use ILP to derive priorities that ensure the task executions within the derived FT/FA feasibility windows. Please note, however, that EDF can be used as well, as the derivation of deadlines has been performed by using the processor utilization demand function.

## 5. Example

We illustrate the proposed methodology by a simple example. Consider a task set $\Gamma$={A,B,C,D,E} with parameters specified in Table 1. Hence, the minimum number of nodes required to feasibly schedule the task set and to fulfill the reliability constraints is:

**Table 1. Task Set- example**

| $\tau_i$ | $T_i$ | $C_i$ | $R_i$ | $M_i$ | $U_i(pri + alt)$ | Criticality |
|---|---|---|---|---|---|---|
| A | 10 | 2 | 1 | 1 | 0.4 | C |
| B | 5 | 1 | 2 | 0 | 0.6 | C |
| C | 4 | 1 | 3 | 2 | 1 | C |
| D | 10 | 3 | 0 | 0 | 0.3 | NC |
| E | 10 | 4 | 0 | 0 | 0.4 | NC |

$$N = max(\lceil U(\Gamma_c) \rceil, max(m_i) + 1) = max(2, 3) = 3$$

We denote the $k^{th}$ alternate of the task A, B, and C by $A_k$, $B_k$ ,$C_k$. In our example, task A has $T_i$=10, $C_i$=2, $R_i$=1, $M_i$=1, which means that at least one of A's alternate needs to be executed in a different node than its primary. The allocation of the FT/FA feasibility windows and the task executions in the worst case error occurrence scenario, is presented in figure 1. In this scenario, $A$, $B$, $B_1$, $C$, $C_1$ and $C_2$ are hit by errors. Hence, the non-critical task E is shed due to the temporary overload, while D can still feasibly execute. At runtime, however, it is unlikely that the worst case scenario will occur. Consider that only $A$, $B$, $B_1$ and $C$ are hit by errors. In this case, $B_2$ and $C_1$ successfully execute as a result of which the execution of $C_2$ is no longer required. This creates the sufficient slack for the execution of task E, as illustrated in figure 2.

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

## 6 Conclusions and ongoing work

In this paper we have presented preliminary results towards providing a priori guarantees in distributed real-

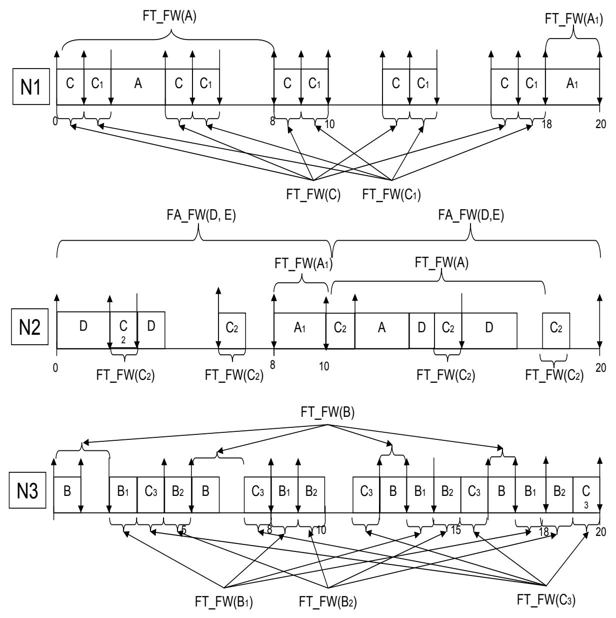

**Figure 1. Allocation and execution under worst case error scenario**

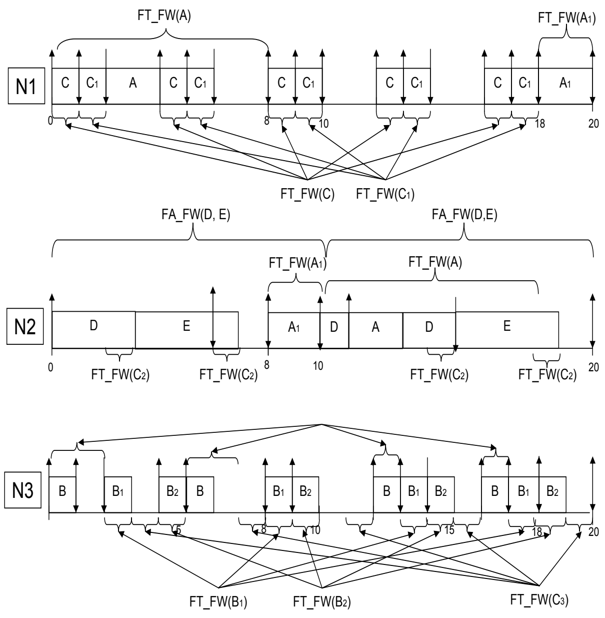

**Figure 2. Allocation and execution under average case error scenario**

time systems with redundancy requirements, while maximizing the resources utilization. We built on our previous works while extending it to a distributed environment with stringent reliability constraints. We derive fault tolerant feasibility windows and allocate them on nodes while using the concept of processor utilization demand to avoid overloads. We proposed the use of optimization techniques to find the optimum window sizes and allocations that fully utilizes the minimum number of processors. Future works will focus on the formalization and evaluation of the proposed approach, as well as extensions to incorporate energy aware mechanisms.

## References

[1] T. AlEnawy and H. Aydin. Energy-aware task allocation for rate monotonic scheduling. *Real Time and Embedded Technology and Applications Symposium, 2005. RTAS 2005. 11th IEEE*, pages 213–223, 2005.

[2] G. Attiya and Y. Hamam. Task allocation for maximizing reliability of distributed systems: A simulated annealing approach. *Journal of Parallel and Distributed Computing*, 66(10):1259–1266, 2006.

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

[4] J. A. Bannister and K. S. Trivedi. Task Allocation in Fault-Tolerant Distributed Systems. *Acta Informatica*, 20:261–281, 1983.

[5] A. Burns, M. Nicholson, K. Tindell, and N. Zhang. Allocating and scheduling hard real-time tasks on a parallel processing platform. 1994.

[6] A. Burns, S. Punnekkat, and R. Davis. Analysis of checkpointing for real-time systems. *Real-Time Systems*, 20(1):83–102, 2001.

[7] R. Dobrin, H. Aysan, and S. Punnekkat. Maximizing the fault tolerance capability of fixed priority schedules. In *The Fourteenth IEEE Internationl Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2008, Kaohisung, Taiwan, 25-27 August 2008, Proceedings*, pages 337–346, 2008.

[8] P. Emberson and I. Bate. Extending a task allocation algorithm for graceful degradation of real-time distributed embedded systems. In *RTSS '08: Proceedings of the 2008 Real-Time Systems Symposium*, pages 270–279, Washington, DC, USA, 2008. IEEE Computer Society.

[9] S. Islam, R. Lindstrom, and N. Suri. Dependability driven integration of mixed criticality sw components. *Object and Component-Oriented Real-Time Distributed Computing, 2006. ISORC 2006. Ninth IEEE International Symposium on*, pages 485–495, 2006.

[10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. 2002.

[11] K. Ramamritham. Allocation and scheduling of complex periodic tasks. *Distributed Computing Systems, 1990. Proceedings., 10th International Conference on*, pages 108–115, 1990.

[12] K. Ramamritham, J. Stankovic, and W. Zhao. Distributed scheduling of tasks with deadlines and resource requirements. *Computers, IEEE Transactions on*, 38(8):1110–1123, 1989.

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

[14] Y. Yu and V.K.Prasanna. Power-aware resource allocation for independent tasks in heterogeneous real-time systems. *Parallel and Distributed Systems, 2002. Proceedings. Ninth International Conference on*, pages 341–348, 2002.