Tolerating Node Failures in Multi-Processor Real-Time Systems with Data Dependencies

Amin Naghavi
Interdiscip. Cent. for Security, Reliability and Trust
University of Luxembourg
Esch-sur-Alzette, Luxembourg
amin.naghavi@uni.lu

Tingting Hu
Department of Computer Science
University of Luxembourg
Esch-sur-Alzette, Luxembourg
tingting.hu@uni.lu

Nicolas Navet
Department of Computer Science
University of Luxembourg
Esch-sur-Alzette, Luxembourg
nicolas.navet@uni.lu

***Abstract*—In real-time systems with data-dependent tasks, ensuring both correctness and timeliness is critical not only for individual task executions but also for data processing across cause-effect chains. When these systems are deployed on multi-processor platforms, tasks belonging to the same chain might be distributed over multiple nodes. In such situations, data received from other nodes may be unreliable, as those nodes could be compromised by faults or malicious attacks. Prior research on fault-tolerant cause-effect chains has largely focused on crash faults and often fails to ensure that task deadlines are met during fault recovery. This paper presents a method that tolerates node failures caused by both faults and malicious intrusions, while ensuring task deadlines in multi-processor (or multi-core) real-time systems through active replication. Our approach leverages majority voting on outputs from replicated tasks across different nodes, enabling each task to validate incoming data before processing it further along the chain. Additionally, for systems using active replication, we present a formal job-level end-to-end latency analysis for cause-effect chains. To reduce the end-to-end latency of task chains, we propose a replica-to-node mapping strategy that enables improved worst-case response times. Experimental evaluations demonstrate that our latency-aware mapping reduces end-to-end latency compared to the commonly used worst-fit decreasing heuristic, although it may slightly reduce task acceptance ratios at high total utilizations.**

***Index Terms*—Real-Time Systems, Fault Tolerance, Cause-Effect Chain, Data Dependency, Intrusion Tolerance.**

I. Introduction

In industrial and automotive systems, actuation based on sensor data often involves multiple interdependent tasks which are forming a cause-effect chain. This concept, introduced in the AUTOSAR standard [1] and discussed in industrial contexts [2], has motivated cause-effect chain latency analysis in real-time systems. In such systems, it is sometimes important to analyze the time between when a sensor sends data and when the first corresponding response is generated by the chain. This duration is known as the Maximum Reaction Time, which is often used in electronics to analyze the button-to-action delay. Alternatively, end-to-end latency can be defined using the Maximum Data Age, which measures the time from when the first task in the chain samples the data to when the last task still produces a response based on it. This metric is used in control systems to evaluate data freshness [3, 4].

In real-time systems, faults can affect both the correctness and timing of task execution. This is critical in multiprocessor systems with cause-effect chains, where a compromised node may propagate incorrect or delayed data that leads to incorrect responses or violations of end-to-end latency constraints. Faults may be accidental, due to radiation or design errors, or malicious, introduced by an attacker who has compromised system components. In such cases, faulty outputs may go undetected unless the receiving nodes evaluates the data.

Fault-tolerant methods that require receiving nodes to verify data correctness by re-executing (replaying) tasks of the sender nodes may be infeasible due to the current load on the receiving nodes. Additionally, reconfiguration methods that reassign and execute tasks from faulty nodes on healthy ones can introduce periods of unpredictability and result in missed deadlines [5]. Active replication, however, can address these problems by guaranteeing the correctness of the outputs through majority voting and ensuring that faults on some nodes do not disrupt the schedule on other healthy ones, thereby allowing enough replicas of tasks to still meet their deadlines and produce timely outputs. While active replication has been studied in real-time systems for handling malicious faults [6], its impact on task data dependencies and end-to-end latency has not yet been explored.

This paper investigates active replication in multi-processor (or multi-core) systems with cause-effect dependencies. Each task performs majority voting on the outputs of all replicas from the preceding task before using them as input. In the presence of up to f faulty nodes, data is accepted once $f+1$ matching outputs are received. Data can only be overwritten by more recent data if it is received from at least $f+1$ nodes. This requirement introduces new challenges in end-to-end latency analysis. We propose a novel job-level end-to-end latency analysis for real-time systems with periodic tasks and active replication, focusing on accurately computing the maximum data age.

Mapping replicas to nodes in real-time systems is often done using existing heuristics, such as Best Fit Decreasing (BFD), which aims to optimize processor utilization, or Worst Fit Decreasing (WFD), which balances the load by distributing tasks across nodes. However, these approaches typically overlook the impact that the placement of one task can have on the Worst-Case Response Time (WCRT) of another task. In real-time systems with cause-effect chains, the worst-case

response times of tasks within a chain directly influence the chain's end-to-end latency. In safety-critical domains, such as vehicle braking control systems [7], satisfying end-to-end latency deadlines is critical. Recognizing the impact of mapping techniques on end-to-end latency, this paper introduces a latency-aware mapping method. The proposed approach heuristically prioritizes reducing the response times of tasks within cause-effect chains to improve end-to-end delays.

We evaluate our mapping approach against WFD under varying node counts and load conditions. The results demonstrate that our latency-aware mapping consistently finds an allocation within just a few milliseconds, which reduces the end-to-end latency of task chains compared to WFD. However, this improvement may come at the cost of a slightly lower acceptance ratio (i.e., the proportion of schedulable task sets) at high utilization levels. This trade-off occurs because optimizing the response times of tasks within cause-effect chains can increase the response times of tasks that are not part of any chain, which, at high utilization levels, may lead to unschedulable task sets.

The structure of this paper is as follows. Section II discusses related work. Section III explains the task, system, replication, and threat model. Section IV presents our job-level end-to-end latency analysis for active replication. In Section V, we detail our latency-aware mapping algorithm. Section VI presents our experimental results, and Section VII concludes the paper.

II. Related Work

Extensive research has addressed the analysis of maximum end-to-end latency for cause-effect chains under various communication models [8] in real-time systems. However, these studies typically do not consider fault tolerance. Feiertag et al. [3] defined various latency types, including maximum data age and reaction time. Günzel et al. [9] showed that the maximum reaction time serves as an upper bound for the maximum data age. Later in [10], they demonstrated that the two become equivalent after a warm-up period, which is the time required for data to traverse the entire chain once. Dürr et al. [4] proposed a task-level end-to-end latency estimation method for sporadic tasks that provides tighter bounds than the earlier approach by Davare et al. [11]. Abdullah et al. [12] provided a tight upper bound for reaction latency in systems with periodic tasks and non-blocking inter-task communication. Becker et al. [13] presented a job-level exact method to compute end-to-end latency in systems with periodic tasks and later proposed a heuristic to constrain data propagation and meet latency requirements [14]. Köhler et al. [15] introduced the robustness margin, quantifying allowable execution time variation from software extensions without violating end-to-end timing constraints.

Real-time systems may be subject to faults and errors, and due to the strict timing constraints of tasks, it is essential to have predictable mechanisms to handle such faults [16]. However, only a few studies have addressed fault tolerance together with end-to-end latency requirements. Klobedanz et al. [7] proposed a reconfiguration-based approach to tolerate permanent faults, introducing a task-to-node mapping that also considers bus communication slot allocation to reduce end-to-end latency. Their method ensures system functionality through alternative mappings in the event of faults. Day [17] employed a standby sparing technique for tasks with varying criticality, selecting hot or cold standby based on reliability, and accounting for recovery times that must be met. Mahmud et al. [18] proposed an ILP-based method for software-to-hardware mapping in heterogeneous systems to minimize power and ensure reliability using standby sparing. They later extended their method with a meta-heuristic approach [19]. Gandhi et al. [20] addressed fault recovery by migrating tasks to other nodes, during which the system may briefly produce incorrect outputs while transitioning to a different mode. Their work focuses on minimizing transition costs to bound recovery time. Mallareddy et al. [21] introduced data age analysis for task chains using checkpointing to tolerate transient faults. Gohari et al. [22] use re-execution to tolerate transient faults by leveraging inherent task redundancies in periodic task models. A task instance is re-executed only if the next instance, released at the task's period, would produce data too late to meet the end-to-end deadline.

Although systems with cause-effect data dependencies are vulnerable to malicious attacks [23], none of the discussed methods address targeted malicious faults. Recent work by Naghavi et al. [6] provides fault tolerance against such faults. However, while their method can maintain consistent outputs across replicas in systems with data dependencies, it does not explicitly consider end-to-end latency requirements. Gandhi et al. [5], extending their earlier work [20], propose a method for tolerating malicious faults using f auditing nodes that replay task execution in the presence of f concurrent faults. When a fault is detected, it is reported to other nodes, after which the faulty node's tasks are migrated to healthy ones, and lower-criticality tasks are dropped to preserve higher-criticality tasks. However, this approach introduces a time window of instability during which task deadlines may be missed.

This paper employs active replication via N-modular redundancy (NMR) and majority voting to tolerate malicious or faulty nodes without compromising the timing guarantees of healthy ones. While some methods, such as [24], coordinate majority voting in systems with data dependencies, our approach assumes uncoordinated voting. We assume that each node votes independently at the start of each task and propose a comprehensive job-level end-to-end latency analysis for this model, along with a replica-to-node mapping strategy aimed at reducing end-to-end latency.

III. System Model

A. System and Tasks.

We assume a system with q processing elements $P_1, \ldots, P_q$, which we call nodes. Nodes can refer to processors in multi-processor systems or cores in multi-core systems. We assume n periodic tasks $\mathbb{T} = \{\tau_1, \ldots, \tau_n\}$, where each task τ_i is defined by a quintet $(B_i, C_i, D_i, \phi_i, T_i)$. Here, B_i and C_i are the Best-Case Execution Times (BCET) and Worst-Case

Execution Time (WCET) of the task τ_i, respectively. D_i is the deadline of the task, ϕ_i is the release offset of the task, and T_i (where $D_i \leq T_i$) is the period of the task. The utilization (u_i) of the task τ_i is defined based on its WCET and period as $u_i = \frac{C_i}{T_i}$.

In this paper, we assume each task is replicated on $m \geq 2f + 1$ nodes, where $m \leq q$ and f is the number of nodes that can be faulty or malicious. However, our approach also supports cases where only tasks on critical cause-effect chains are replicated, while other tasks, either from non-critical chains or not part of any chain, may remain non-replicated. Each node P_k executes a set of task replicas, denoted by Γ_k. We refer to the r-th replica of a task τ_i as τ_i^r. The system is assumed to be homogeneous, meaning all replicas of a task share the same specifications (including WCET and BCET) as the original task τ_i, regardless of the node to which they are assigned. However, the best-case and worst-case response times of a task replica τ_i^r may vary across nodes, as each node can host different sets of task replicas. Once a replica τ_i^r is allocated to a node, the function $PE(\tau_i^r)$ returns the node to which it is mapped. The best-case and worst-case response times of this replica on its assigned node are then denoted by $BCRT_i^r$ and $WCRT_i^r$, respectively.

Each task releases an indefinite number of jobs, with each job released at the task's period on every node where the task has a replica. We denote the j-th job released by task τ_i as $\tau_{i,j}$, where this job is released at time $(j-1)T_i + \phi_i$. The r-th replica of job $\tau_{i,j}$ is denoted by $\tau_{i,j}^r$.

B. Cause-Effect Chain.

In this paper, we consider multi-rate tasks with data dependencies. We assume there are v cause-effect chains, each represented as a linear, directed acyclic graph (DAG). In such a graph, tasks correspond to the graph's nodes, and edges indicate data dependencies between them. The output of each task is influenced by the data produced by the final job of its immediate predecessor. For each $k \leq v$, the sequence of tasks in the k-th task chain is denoted by $TC_k = (\tau_a \rightarrow \tau_b \rightarrow \cdots \rightarrow \tau_x)$. We define L_k as the length of the task chain TC_k, representing the number of tasks in this chain. Additionally, we introduce the function $\zeta[l]$ for $l \leq L_k$, which returns the ID of the l-th task in the chain. For example, if the first task in the chain is τ_a, then $\zeta[1] = a$.

A job chain of a task chain TC_k is a sequence of jobs, each corresponding to a task in TC_k, where data communication between these jobs may occur at runtime. If such a chain forms during execution, each job in the sequence receives data from its preceding job. If E_k denotes the number of possible job chains of the task chain TC_k that can occur at runtime, the e-th possible job chain ($e \leq E_k$) is denoted as $JC_e^k = (\tau_{a,\alpha} \rightarrow \tau_{b,\beta} \rightarrow \cdots \rightarrow \tau_{x,\chi})$, where each job in this chain represents an instance of a task in the task chain TC_k. We define the function $J[l]$ for $l \leq L_k$ to return the instance number of the job (job ID) in the l-th position in JC_e^k. For example, if the first job in JC_e^k is $\tau_{a,\alpha}$, then $J[1] = \alpha$.

The end-to-end latency (also referred to as data age) of a job chain is defined as the time interval from when the first job in the chain reads its input to the latest time when the last job in the chain writes its output. The end-to-end latency of a task chain is the maximum end-to-end latency among all its job chains within the observation window OW. The observation window is defined as [14]:

$$OW = \max\{2H, \max_{k \leq v} \lceil \frac{WCL(TC_k)}{H} \rceil . H\} \tag{1}$$

Where H is the hyperperiod of the task set (the least common multiple (LCM) of task periods), and WCL is an upper bound on the maximum data age of task chain TC_k, calculated as:

$$WCL(TC_k) = \sum_{\tau_i \in TC_k} 2.T_i \tag{2}$$

C. Communication Model

We assume implicit communication, where jobs of tasks read input data before execution, and send output data to their successors in the task chain after finishing their execution [8]. When two tasks of a chain are on different nodes, they must communicate through an inter-node communication medium. This medium introduces a delay, which can range from a best-case delay of B_{com} to a worst-case delay of C_{com}.

If tasks are on different nodes, a majority vote must be taken between the outputs of the replicas of a task before this output is accepted as input by a successive task τ_i. This majority vote is performed by each replica of the jobs of task τ_i. We assume that when a replica of a job of τ_i's predecessor task produces output, its output is stored in the buffers on the nodes where replicas of task τ_i are located. Before using a buffered input, a job of task τ_i performs majority voting among the latest outputs of the replicas of its preceding task. As soon as $f + 1$ matching outputs are found, this output is used as the input by τ_i. We assume that information regarding the cause is embedded in the received messages from other nodes. In this case, the voter selects the $f + 1$ matching outputs generated based on the most recent cause.

We assume that communication between two tasks on the same node happens with negligible delay. If a replica of τ_i and its predecessor are located on the same node, τ_i uses the data produced by the predecessor task replica without performing a majority vote. In this case, any data received from other nodes is simply ignored. This is because we consider only node failures in this paper. However, the method can be adapted for systems where tasks may also experience transient faults, in which case majority voting would be required even for data from replicas on the same node.

D. Threat Model

As mentioned before, we assume that up to f nodes can be faulty or malicious. Such nodes may fail silently, produce incorrect outputs, or introduce delays due to being compromised by an attacker. A node that loses communication with other nodes is also considered faulty. We assume that a malicious node cannot affect healthy nodes in any way

other than by sending incorrect or delayed outputs. To prevent malicious nodes from overloading the network, we assume that mechanisms such as ingress traffic shaping on Ethernet switch ports, or bus guardians and active stars in shared bus systems, are employed. As a result, a malicious node cannot increase the communication delay between two healthy nodes beyond the bound C_{com}. It is also assumed that nodes sign messages using unique cryptographic signatures, preventing malicious nodes from impersonating healthy ones and sending messages on their behalf. Finally, we assume that tasks on healthy nodes always execute correctly and meet their deadlines.

IV. Calculating End-to-End Latency

In this section, we describe how to calculate the maximum end-to-end latency (data age) of a task chain in fault-tolerant, multiprocessor real-time systems with data dependencies. This involves identifying all job chains associated with a task chain and determining the maximum latency among them, which represents the task chain's maximum end-to-end latency.

A. End-to-End Latency

Based on [13], in a non-replicated setting, the end-to-end latency of a job chain is defined as the interval from the release of the first job in the chain to the latest time that the last job in the chain produces data. The release time of the first job in a job chain can be calculated as $(J[1]-1)\cdot T_{\zeta[1]}+\phi_{\zeta[1]}$, where $J[1]$ and $\zeta[1]$ are the job ID and task ID of the first job in the chain, respectively. The latest time that the last job in a job chain of length L_k produces data is given by the sum of this job's release time (computed similarly for $J[L_k]$ and $\zeta[L_k]$ as described above) and the worst-case response time of its respective task, $WCRT^r_{\zeta[L_k]}$.

In a system with task replication, data generated by a task chain is only accepted if at least $2f+1$ replicas of the task chain produce data related to the same cause. This requirement ensures that, even if up to f nodes are faulty, the remaining replicas on $f+1$ correct nodes can provide a majority for the voter to decide the output. Therefore, the end-to-end latency of a replicated job chain is calculated as:

$$\begin{aligned} Lat(JC^k_e) = \min_{\substack{\forall S\subset\{1,2,\ldots,m\}\\ |S|=2f+1}} \max_{r\in S}((J[L_k]-1).T_{\zeta[L_k]}+\phi_{\zeta[L_k]} \\ +WCRT^r_{\zeta[L_k]}) - ((J[1]-1).T_{\zeta[1]}+\phi_{\zeta[1]}) \end{aligned} \tag{3}$$

Assuming that there are E_k job chains associated with a task chain TC_k within the observation window, the end-to-end latency of the task chain is calculated as [13]:

$$Lat(TC_k) = \max_{e\le E_k} Lat(JC^k_e) \tag{4}$$

In the following, we discuss how to identify all job chains associated with a task chain that may occur at runtime, which is necessary for calculating the task chain's end-to-end latency.

B. Job Chains

A job chain is a sequence of jobs, each corresponding to a task in a task chain, where each job reads data produced by the job of the preceding task in the sequence. To identify job chains, the *Read Interval* and *Data Interval* of each job in the task chain are calculated. If, for two consecutive tasks, the read interval of a job of the successive task overlaps with the data interval of a job of the preceding task, these jobs may be part of a job chain. In the following, we formally define and calculate these intervals for job replicas, building on the approach in [13], which provides these values for jobs in non-replicated systems. We then extend this method to determine data intervals for jobs in fault-tolerant real-time systems and discuss how to construct corresponding job chains.

1) Replicas' Read and Data Intervals: The read interval $RI(\tau^r_{\zeta[l],J[l]}) = [\underline{Re}(\tau^r_{\zeta[l],J[l]}), \overline{Re}(\tau^r_{\zeta[l],J[l]})]$ of the r-th replica of a job $\tau_{\zeta[l],J[l]}$, which is a job of a task positioned at the l-th location in a cause-effect chain, is the time interval from the earliest possible moment it can read the data to the latest possible moment it can do so before starting to process the data. The upper bound and lower bound of the read interval of the r-th replica of $\tau_{\zeta[l],J[l]}$ are calculated as:

$$\begin{aligned} \underline{Re}(\tau^r_{\zeta[l],J[l]}) &= (J[l]-1).T_{\zeta[l]}+\phi_{\zeta[l]} \\ \overline{Re}(\tau^r_{\zeta[l],J[l]}) &= (J[l]-1).T_{\zeta[l]}+\phi_{\zeta[l]}+WCRT^r_{\zeta[l]}-C_{\zeta[l]} \end{aligned} \tag{5}$$

In this case, the data interval $DI(\tau^r_{\zeta[l],J[l]}) = [\underline{Da}(\tau^r_{\zeta[l],J[l]}), \overline{Da}(\tau^r_{\zeta[l],J[l]})]$ for the r-th replica of a job $\tau_{\zeta[l],J[l]}$ is calculated according to the earliest time that the replica produces an output, and the latest time that the next job of the same task will rewrite this output. The upper and lower bounds of the data interval for the r-th replica of $\tau_{\zeta[l],J[l]}$ are calculated as:

$$\begin{aligned} \underline{Da}(\tau^r_{\zeta[l],J[l]}) &= (J[l]-1).T_{\zeta[l]}+\phi_{\zeta[l]}+BCRT^r_{\zeta[l]} \\ \overline{Da}(\tau^r_{\zeta[l],J[l]}) &= J[l].T_{\zeta[l]}+\phi_{\zeta[l]}+WCRT^r_{\zeta[l]} \end{aligned} \tag{6}$$

This calculation assumes that the data is written on the same node with negligible delay. If data is sent from one node to another, the communication delay must be included in the data interval. In this case, the upper and lower bounds of the data interval, denoted as $DI'(\tau^r_{\zeta[l],J[l]}) = [\underline{Da'}(\tau^r_{\zeta[l],J[l]}), \overline{Da'}(\tau^r_{\zeta[l],J[l]})]$, can be calculated as:

$$\begin{aligned} \underline{Da'}(\tau^r_{\zeta[l],J[l]}) &= (J[l]-1).T_{\zeta[l]}+\phi_{\zeta[l]}+BCRT^r_{\zeta[l]}+B_{com} \\ \overline{Da'}(\tau^r_{\zeta[l],J[l]}) &= J[l].T_{\zeta[l]}+\phi_{\zeta[l]}+WCRT^r_{\zeta[l]}+C_{com} \end{aligned} \tag{7}$$

2) Jobs' Data Interval: If replicas of two consecutive jobs in a chain are on the same node, the replica of the successive job can trust the data from its predecessor and begin processing the data. However, if the replicas are on different nodes, the reader cannot trust the data it receives from another node. In this case, each replica of a job must perform majority voting over the outputs of the preceding job's replicas. Since voting is done independently by each replica, the read interval must be considered separately for each job replica. A job replica only accepts data as input if at least $f+1$ replicas of the preceding

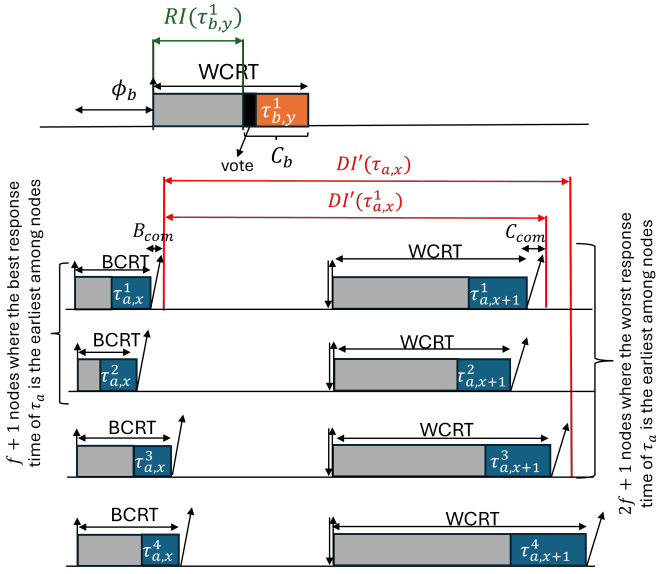


Fig. 1: Overlap between the data interval of job $\tau_{a,x}$ and the read interval of a replica of job $\tau_{b,y}$, indicating they may be part of the same job chain, if τ_a precedes τ_b in the task chain.

job produce the same data. Therefore, the data interval is determined per job rather than per job replica.

We denote the data interval of the job $\tau_{\zeta[l],J[l]}$ as $DI'(\tau_{\zeta[l],J[l]}) = [\underline{Da'}(\tau_{\zeta[l],J[l]}), \overline{Da'}(\tau_{\zeta[l],J[l]})]$. The earliest time at which $f+1$ replicas can produce enough outputs for the voter to make a decision determines the lower bound of the data interval, which is calculated as:

$$\underline{Da'}(\tau_{\zeta[l],J[l]}) = \min_{\substack{\forall S \subset \{1,2,\ldots,m\} \\ |S|=f+1}} (\max_{r \in S} \underline{Da'}(\tau^r_{\zeta[l],J[l]})) \tag{8}$$

The data is guaranteed to be overwritten once $2f+1$ replicas produce outputs related to a more recent cause. This is because, even if up to f nodes fail or behave maliciously, at least $f+1$ replicas on healthy nodes will still produce matching outputs that the replicas of the successor job can vote on and accept as their input. Therefore, the upper bound of the data interval can be defined as the earliest time at which it is guaranteed that at least $2f+1$ replicas have rewritten the data, which is calculated as follows:

$$\overline{Da'}(\tau_{\zeta[l],J[l]}) = \min_{\substack{\forall S \subset \{1,2,\ldots,m\} \\ |S|=2f+1}} \max_{r \in S} \left(\overline{Da'}(\tau^r_{\zeta[l],J[l]}) \right) \tag{9}$$

Figure 1 illustrates the data interval of job $\tau_{a,x}$ and the read interval of job replica $\tau^1_{b,y}$, where tasks τ_a and τ_b are part of a task chain in which τ_a sends data to τ_b, and their replicas are located on different nodes. In this example, we assume that each task has four replicas, with at most one faulty node ($f=1$). For simplicity, only one replica of τ_b is shown.

In this figure, the lower bound of the data interval of $\tau_{a,x}$ is determined by the earliest time at which $f+1$ replicas of $\tau_{a,x}$ might respond; this marks the earliest moment at which voted data might be provided. The upper bound of the data interval is based on the earliest time by which it is guaranteed that at least $2f+1$ replicas of $\tau_{a,x+1}$ respond. By the time this upper bound is reached, the data written by $\tau_{a,x}$ is guaranteed to be overwritten by $\tau_{a,x+1}$, even if up to f nodes are faulty.

As mentioned, the read interval is calculated individually for each job replica. Figure 1 shows the read interval of the job replica $\tau^1_{b,y}$, which spans from the earliest to the latest time that $\tau^1_{b,y}$ might begin execution.

3) Generating Job Chains: To identify job chains associated with a task chain, we examine pairs of jobs from consecutive tasks where the data interval of the job from the preceding task (e.g., $\tau_{\zeta[l-1]}$) intersects with the read interval of any replica of the job from the succeeding task (e.g., $\tau_{\zeta[l]}$).

If, for the r-th replica of a task, there exists an s-th replica of its preceding task in the chain that resides on the same node (i.e., $PE(\tau^s_{\zeta[l-1]}) = PE(\tau^r_{\zeta[l]})$), then the task replica reads data directly from that preceding task replica, without requiring voting or inter-node communication. In such cases, a job with ID $J[l]$ of task replica $\tau^r_{\zeta[l]}$ is considered part of a job chain with a job with ID $J[l-1]$ of task replica $\tau^s_{\zeta[l-1]}$ if:

$$DI(\tau^s_{\zeta[l-1],J[l-1]}) \cap RI(\tau^r_{\zeta[l],J[l]}) \neq \emptyset \tag{10}$$

If, for a task replica $\tau^r_{\zeta[l]}$ in a task chain, there is no replica of its preceding task located on the same node (i.e., $\nexists s \mid PE(\tau^s_{\zeta[l-1]}) = PE(\tau^r_{\zeta[l]})$), a majority vote is performed at the start of each job of $\tau^r_{\zeta[l]}$. In this case, the maximum length of the data interval for the job with ID $J[l-1]$ of the preceding task is given by $DI'(\tau_{\zeta[l-1],J[l-1]})$. A job with ID $J[l]$ from the task replica $\tau^r_{\zeta[l]}$ can only be considered part of a job chain with a job with ID $J[l-1]$ from its preceding task if there exists a task replica $\tau^r_{\zeta[l]}$ such that:

$$DI'(\tau_{\zeta[l-1],J[l-1]}) \cap RI(\tau^r_{\zeta[l],J[l]}) \neq \emptyset \tag{11}$$

For example, in Figure 1, the data interval of $\tau_{a,x}$—which is based on the time at which majority voting among the outputs of its replicas can be performed—overlaps with the read interval of at least one replica of $\tau_{b,y}$ (in this example, $\tau^1_{b,y}$). Therefore, these two jobs can be part of a job chain.

Finally, after identifying all pairs of jobs from consecutive tasks in the task chain that may belong to a job chain (using Equations 10 and 11), we can determine the job chains. A job chain JC^e_k is formed if the following condition is met:

$$\forall l \leq L_k, \exists r \quad \begin{cases} \text{Eq. 10 holds} & \text{if } \exists s | PE(\tau^s_{\zeta[l-1]}) = PE(\tau^r_i), \\ \text{Eq. 11 holds} & \text{otherwise.} \end{cases} \tag{12}$$

V. Latency-Aware Replica-to-Node Allocation

When tasks are part of cause-effect chains, the worst-case response time (WCRT) of their replicas can impact the chain's end-to-end latency. Although some tasks may not belong to any chain, assigning them to the same nodes as chain tasks can affect the WCRT of those chain tasks. To address this, we propose a heuristic, latency-aware replica-to-node mapping strategy aimed at reducing the end-to-end latency of cause-effect chains, where latency is computed using the method described in Section IV.

Algorithm 1: Latency-Aware Mapping

input : Nodes: $\{P_1, ..., P_q\}$, Task set: $\mathbb{T} = \{\tau_1, ..., \tau_n\}$, Task Replicas: $\{\tau_i^j | \forall \tau_i \in \mathbb{T}, \forall j \in \{1, ..., m\}\}$, Task Chains: $\{TC_1, ..., TC_v\}$

output: $\Gamma_1, ..., \Gamma_q$

1 $CT = \{\tau_i \in \mathbb{T} \mid \exists k \in \{1, \ldots, v\}$ such that $\tau_i \in TC_k\}$
2 **for** *each* $\tau_i \in$ *sorted(CT, "Utilization", reversed) :*
3 $CNodes = \{P_1, \ldots, P_q\}$
4 **for** *each* $j \in \{1, ..., m\}$ *:*
5 $P_s = \arg \min_{P_k \in CNodes} (\sum_{\tau_x^y \in \Gamma_k} u_x)$
6 $\Gamma_s = \Gamma_s \cup \{\tau_i^j\}$
7 $CNodes = CNodes \backslash P_s$
8 $FT = \mathbb{T} - CT$
9 **for** *each* $\tau_i \in$ *sorted(FT, "Utilization", reversed) :*
10 $CNodes = \{\forall P_k | k \in \{1, ..., q\} \wedge u_i + \sum_{\tau_x^y \in \Gamma_k} u_x \leq 1\}$
11 **if** $|CNodes| < m$*:* return "Unsuccessful"
12 **for** *each* $j \in \{1, ..., m\}$*:*
13 *min_diff* $= \infty$
14 **for** $P_k \in CNodes$*:*
15 $wcrt\_before = \sum_{\substack{\tau_x^y \in \Gamma_k \\ \tau_x \in CT}} WCRT_x^y$
16 $\Gamma_k = \Gamma_k \cup \{\tau_i^j\}$
17 $wcrt\_after = \sum_{\substack{\tau_x^y \in \Gamma_k \\ \tau_x \in CT}} WCRT_x^y$
18 $\Gamma_k = \Gamma_k \backslash \{\tau_i^j\}$
19 $wcrt\_diff = wcrt\_after - wcrt\_before$
20 **if** *(*$wcrt\_diff < min\_diff$*)*
21 $P_s \leftarrow P_k$
22 $min\_diff = wcrt\_diff$
23 $\Gamma_s = \Gamma_s \cup \{\tau_i^j\}$
24 $CNodes = CNodes \backslash P_s$
25 return "Successful"

The method begins by assigning tasks that are part of at least one chain, aiming to keep the total worst-case response time of chain-related tasks as low as possible. Once these tasks are placed, the remaining tasks—those not involved in any chain—are assigned in a way that minimizes their impact on the overall WCRT of the chain tasks. The full mapping procedure is detailed in Algorithm 1.

The algorithm uses the Worst-Fit Decreasing (WFD) strategy to map replicas of tasks that are part of at least one cause-effect chain to the nodes. This helps distribute the load of these tasks more evenly across the nodes, thereby reducing their response times. The algorithm first collects all such tasks into a set CT (line 1). It then maps tasks in CT in decreasing order of their utilizations (line 2), starting with the task that has the highest utilization. We first map all replicas of a task to nodes before proceeding to the next task. The candidate set of nodes (*CNodes*) for mapping a replica of a task consists of all nodes to which no other replica of that task has been assigned. Initially, this set includes all nodes before any replicas of the task are mapped (line 3). Each replica is mapped to the candidate node with the lowest total utilization, where total utilization is defined as the sum of the utilizations of all task replicas currently mapped to that node (lines 4–6). After each mapping, the selected node is removed from the candidate set before mapping other replicas of the task (line 7). Assigning replicas to nodes continues with the next highest-utilization task in CT, until all replicas of the tasks in CT are mapped.

After mapping all chain tasks, the algorithm proceeds to map the replicas of the remaining tasks that do not belong to any chain. These tasks, collected in the set FT, are mapped in decreasing order of their utilizations (lines 8, 9). To mitigate the interference of non-chain tasks with chain tasks, the algorithm evaluates how placing each replica of a non-chain task would affect the WCRTs of already assigned chain task replicas across all candidate nodes. Candidate nodes are initially defined as those where adding the next task does not cause the total utilization of replicas on the node to exceed 1. If there are not enough candidate nodes to map all replicas of the next task, the algorithm fails to find a schedulable mapping (line 11). Otherwise, for each candidate node, the algorithm computes the total WCRT of all chain tasks both (1) before and (2) after placing the non-chain task replica (lines 10-18). It then selects the node where the difference between these two values is smallest, i.e., the mapping that increases the total WCRT of chain tasks least (lines 19-22). After selecting the node that least affects chain task response times, the algorithm maps the replica to this node (line 23), removes the node from the candidate set (line 24), and repeats the process to assign the task's remaining replicas to other nodes. The algorithm then maps replicas of the remaining tasks to nodes, completing the process once all non-chain task replicas have been assigned.

VI. Evaluation

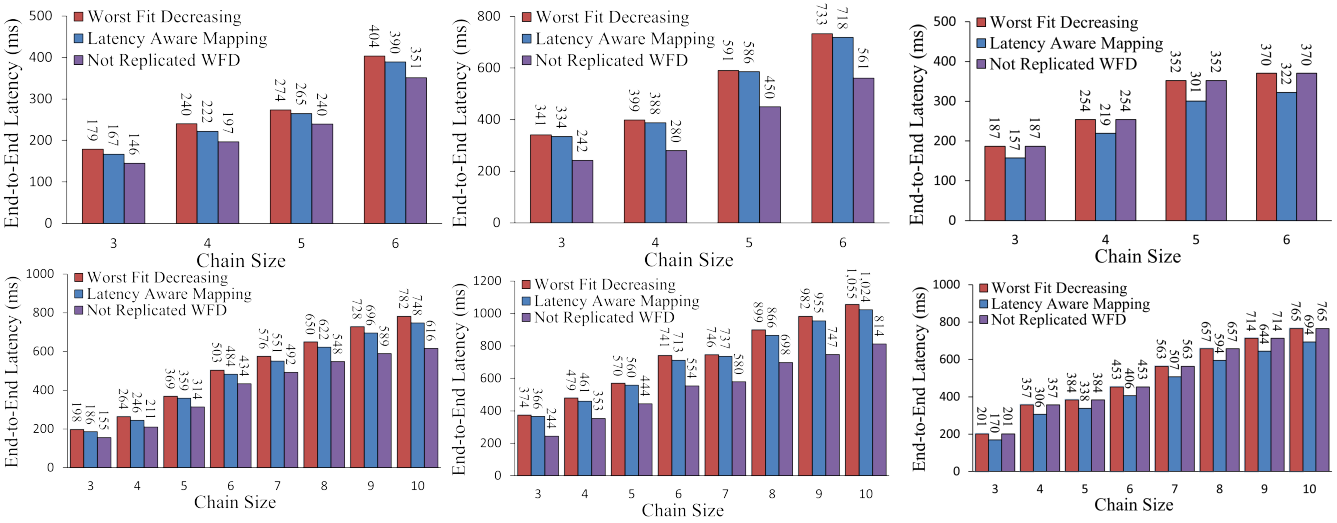
We implemented our fault-tolerant end-to-end latency calculation in Python on a Linux platform[1]. The analysis was performed on a single core of an AMD Ryzen 9 5950X CPU (3.4 GHz) with 125 GB of RAM. To evaluate the impact of different mapping approaches on end-to-end latency, we compared our latency-aware mapping method with the WFD algorithm, which maps replicas to nodes while ensuring that no two replicas of the same task are placed on the same node. For further comparison, we also considered the case where tasks are not replicated and mapped directly using WFD. In addition, we investigated the impact of different mapping methods on schedulability when there is replication.

In these experiments, we assumed best- and worst-case communication delays of 100 μs and 1000 μs, respectively.

A. Workload Generation

To conduct our experiments, we generated random task sets, each consisting of 20 tasks. We assumed that all tasks are critical; therefore, three replicas were generated per task,

[1]Available at https://github.com/aminnaghavi/FT-E2EL (released under the AGPL v3.0 license; results can be reproduced using the provided code and instructions).



(a) 5 Nodes, Total utilization of 1.2

(b) 5 Nodes, Total utilization of 1.5

(c) 6 Nodes, Total utilization of 1.5

Fig. 2: Maximum end-to-end latency of task chains with varying sizes: analysis for chain sizes 3–6 (top) and 3–10 (bottom).

resulting in a total of 60 task replicas per task set. We used the Dirichlet-Rescale algorithm [25] to generate random task utilizations that sum to a predefined utilization bound. Task periods were randomly selected from the real-world automotive benchmark dataset [26]. The worst-case execution time for each task was computed using its utilization and period, according to the formula $C_i = u_i \times T_i$. We assumed implicit deadlines, meaning the deadline D_i of each task is equal to its period T_i.

Cause-effect chains were created by randomly selecting tasks from half of the task set (i.e., 10 tasks), while the remaining 10 tasks were excluded from any chain. We allowed tasks with different periods to be part of the chains; however, to limit the number of job chains, we only considered chains where tasks are ordered by their periods, meaning that each task in a chain has a longer period than its predecessor [4]. We evaluated two scenarios: in the first, each task set includes four chains of lengths 3, 4, 5, and 6; in the second, each task set contains eight chains with lengths ranging from 3 to 10.

B. End-to-End Latency

We compare the maximum end-to-end latency of a non-replicated system (using WFD for task mapping) with that of replicated systems employing either our latency-aware mapping or WFD. The maximum end-to-end latencies of chains in replicated systems were computed using our latency analysis method designed for active replication, as described in Section IV. For non-replicated systems, the maximum end-to-end latencies were calculated using the approach from [13].

We considered a multi-processor system to schedule task sets with total utilization greater than 1. In the non-replicated system, we considered that tasks are scheduled on two nodes (the minimum number of nodes to schedule these tasks). In the replicated system, we considered configurations with 5 and 6 nodes, with each task having three replicas distributed across three different nodes. We used Rate Monotonic (RM) scheduling and included only task sets schedulable under RM and the specified mapping methods. Experiments were repeated for 100 task sets for each combination of total utilization, number of nodes, and each scenario described in §VI-A.

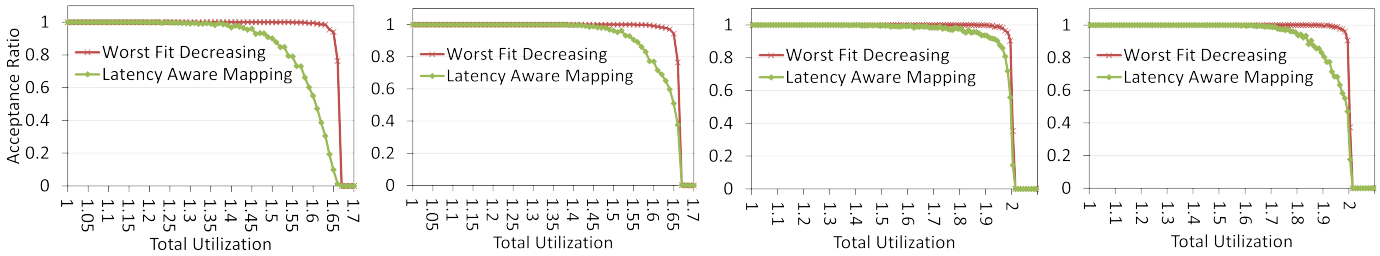
TABLE I: Total number of possible job chains across all task chains for different mappings under varying chain sizes, utilizations, and node counts.

Method	Chain Sizes	Average number of possible job chains		
		U: 1.2, q: 5	U: 1.5, q: 5	U: 1.5, q: 6
Latency Aware Mapping	3-6	2148.66	3015.89	1181.11
	3-10	6584.62	15437.83	4053.24
Replication WFD	3-6	2325.91	2830.68	1315.02
	3-10	6870.09	14410.95	4318.57
No Replication WFD (q=2 fixed)	3-6	1236.79	1116.15	1315.02
	3-10	2817.52	3908.34	4318.57

TABLE II: Execution times of our latency-aware mapping and job-level end-to-end latency calculation of replicated task chains in different scenarios.

Algorithm	Chain Sizes	Execution time of the algorithm		
		U: 1.2, q: 5	U: 1.5, q: 5	U: 1.5, q: 6
Latency Aware Mapping	3-6	1.80 ms	2.01 ms	1.57 ms
	3-10	1.95 ms	2.13 ms	1.74 ms
End-to-End Calculation	3-6	5.002 s	4.706 s	3.530 s
	3-10	27.914 s	33.220 s	16.871 s

Figure 2a presents the end-to-end latencies for scenarios involving chains of lengths ranging from 3 to 6 and 3 to 10, using replicated task sets with a total utilization of 1.2, scheduled across 5 nodes. Figure 2b shows the results for the same configurations, but with task sets having a total utilization of 1.5. Finally, Figure 2c presents the latencies for scenarios with a total utilization of 1.5, mapped across 6 nodes. Each bar represents the average end-to-end latency for chains of the specified lengths, calculated over 100 task sets. The number of job chains created in each experiment is listed in Table I, while the execution times for our mapping method and end-to-end latency calculation are provided in Table II. As shown, the execution time of our mapping method is in the order of a few milliseconds, while the job-level end-to-



(a) 5 Nodes, Chains 3-6 (b) 5 Nodes, Chains 3-10 (c) 6 Nodes, Chains 3-6 (d) 6 Nodes, Chains 3-10

Fig. 3: Acceptance ratio for latency-aware mapping and WFD across different node counts for chain sizes of 3–6 and 3–10.

end latency calculation takes several seconds, depending on the load per resource ratio.

As shown in Figures 2a and 2b, scheduling three replicas of each task across five nodes increases the end-to-end latency for both WFD and our latency-aware method, primarily because the system load is tripled while the number of nodes is only increased by two. This is the minimum number of nodes required to schedule task sets with a total utilization of 1.5 and three replicas per task. In this case, our fault-tolerant end-to-end latency analysis considers a greater number of job chains than the analysis without replication, as detailed in Table I. This is due to the higher WCRT of tasks (resulting from increased load per resource) and the larger read and data intervals (as defined in Equations 5 and 6). Nevertheless, Figures 2a and 2b show that, on average, our latency-aware mapping method still achieves lower end-to-end latency than WFD with replication for chains of different sizes. For example, as seen in Figure 2a, scenarios with chain sizes ranging from 3 to 6 and 3 to 10 at a total utilization of 1.2 show an average improvement of 5.27% and 4.55%, respectively, in end-to-end latency using our method compared to WFD. Similarly, Figures 2b show that at a total utilization of 1.5, our method achieves an average improvement of 1.93% and 2.77% for the two scenarios, respectively, compared to WFD with replication. The comparison between Figures 2a and 2b shows that increasing total utilization reduces the effectiveness of our latency-aware mapping in decreasing end-to-end latency. This is because, under high task loads, the algorithm may be unable to assign a task to the node for which the mapping is most favorable to the response time of chain tasks, as doing so could cause the total utilization of replicas on that node to exceed 1. This is also evident from Table I, where increasing the total utilization leads to a higher number of job chains created by our method. Nevertheless, due to reduced worst-case response times, our method still manages to achieve, on average, lower end-to-end latency compared to WFD with replication.

Figure 2c shows that the maximum end-to-end latency of our latency-aware mapping is significantly reduced when scheduling task sets with a total utilization of 1.5 on 6 nodes. It is worth noting that WFD with replication produced the same end-to-end latency as WFD without replication. This is because, in this scenario, the replicated system behaves similarly to the non-replicated case, as the same task-to-node mapping is effectively repeated across each pair of nodes. As a result, the jobs exhibit identical worst- and best-case response times. In contrast, our latency-aware method achieves lower end-to-end latency than WFD, both with and without replication. This is because it maps tasks that are not part of any chain to nodes where they have minimal impact on the WCRT of chain tasks, thereby improving the response times of chain tasks. Consequently, our mapping method yielded 14.30% and 11.29% lower end-to-end latency than WFD for scenarios with cause-effect chains of sizes 3-6 and 3-10, respectively. The reduction in WCRT of tasks due to our mapping method also led to a smaller number of job chains compared to WFD, as shown in Table I.

C. Acceptance Ratio

We compared the number of schedulable task sets using our latency-aware mapping method against the WFD method when mapping three replicas per task onto 5 nodes (Figures 3a and 3b) and 6 nodes (Figures 3c and 3d). For each total utilization value from 1.0 to 2.0 in increments of 0.01, we generated 1,000 task sets per utilization level and determined how many were schedulable using RM, after mapping replicas to nodes with each method. We evaluated both scenarios described in §VI-A, and the results are summarized in Figure 3. Although our latency-aware mapping reduces the worst-case response time of chain tasks (thereby lowering the end-to-end latency of cause-effect chains), it generally achieves a lower acceptance ratio than WFD at higher total utilization levels. This occurs because improving the response times of chain tasks can lead to increased response times for non-chain tasks, which may result in some task sets becoming unschedulable. Nevertheless, our mapping method still maintained a high acceptance ratio, even under heavy task loads relative to the available computational resources. On 6 nodes, more than 60% of task sets were schedulable in both scenarios at a total utilization of 1.96, while over 90% were schedulable at a slightly lower utilization of 1.85. Similarly, on 5 nodes, over 60% of task sets were schedulable at a utilization level of 1.59, and the acceptance ratio exceeded 90% at a total utilization of 1.5. Additionally, our latency-aware mapping can guarantee end-to-end latency requirements in systems with tight end-to-end deadlines [15].

VII. Conclusion

In this paper, we present a method for tolerating malicious faults in multiprocessor real-time systems with cause-effect chain dependencies, using active replication. To formally estimate the end-to-end latency of task chains, we introduce a design-time analysis technique that explores all possible job chains by estimating the data intervals of jobs and the read intervals of job replicas. This analysis assumes that a job replica on a node can read data from job replicas of its preceding task on other nodes only if a majority vote based on that data can be completed within its read interval. Additionally, we propose a novel latency-aware replica-to-node mapping strategy that aims to reduce the maximum end-to-end latency of task chains by lowering the worst-case response times of task replicas. Through evaluation, we compared this latency-aware mapping with the worst-fit decreasing approach. Our results show that our method significantly reduces end-to-end latency, though this improvement may come at the cost of slightly lower acceptance ratios at higher utilizations compared to WFD.

In future work, we aim to extend this method to support sporadic task models and to determine task-level upper bounds on end-to-end latency in actively replicated systems. Additionally, developing optimized mapping and priority assignment strategies to minimize end-to-end latency in systems with varying criticalities remains an open challenge.

References

[1] AUTOSAR, "Specification of timing extensions," Release 4.0.1, 2009.

[2] S. Liu, B. Yu, N. Guan, Z. Dong, and B. Akesson, "Real-time scheduling and analysis of an autonomous driving system," *Proc. RTSS Ind. Challenge Problem*, 2021.

[3] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson, "A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics," in *Real-Time Syst. Symp. (RTSS)*, 2009.

[4] M. Dürr, G. V. D. Brüggen, K.-H. Chen, and J.-J. Chen, "End-to-end timing analysis of sporadic cause-effect chains in distributed systems," *ACM Trans. on Embed. Comp. Syst. (TECS)*, vol. 18, no. 5s, pp. 1–24, 2019.

[5] N. Gandhi, E. Roth, B. Sandler, A. Haeberlen, and L. T. X. Phan, "Rebound: defending distributed systems against attacks with bounded-time recovery," in *Proc. of the European Conf. on Comp. Syst.*, 2021, p. 523–539.

[6] A. Naghavi and N. Navet, "Total execution order in fault-tolerant real-time systems," in *Proc. of Int. Conf. on Real-Time Networks and Syst. (RTNS)*, 2025, p. 12–24.

[7] K. Klobedanz, J. Jatzkowski, A. Rettberg, and W. Mueller, "Fault-tolerant deployment of real-time software in autosar ecu networks," in *Embed. Syst.: Design, Analysis and Verification*, 2013, pp. 238–249.

[8] A. Hamann, D. Dasari, S. Kramer, M. Pressler, and F. Wurst, "Communication Centric Design in Complex Automotive Embedded Systems," in *Euromicro Conf. on Real-Time Syst. (ECRTS)*, vol. 76, 2017, pp. 10:1–10:20.

[9] M. Günzel, K.-H. Chen, N. Ueter, G. v. d. Brüggen, M. Dürr, and J.-J. Chen, "Timing analysis of asynchronized distributed cause-effect chains," in *Real-Time and Embed. Tech. and Appl. Symp. (RTAS)*, 2021, pp. 40–52.

[10] M. Günzel, H. Teper, K.-H. Chen, G. von der Brüggen, and J.-J. Chen, "On the Equivalence of Maximum Reaction Time and Maximum Data Age for Cause-Effect Chains," in *Euromicro Conf. on Real-Time Syst. (ECRTS)*, vol. 262, 2023, pp. 10:1–10:22.

[11] A. Davare, Q. Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli, "Period optimization for hard real-time distributed automotive systems," in *Proc. of Annual Design Automation Conf.*, ser. DAC '07, 2007, p. 278–283.

[12] J. Abdullah, G. Dai, and W. Yi, "Worst-case cause-effect reaction latency in systems with non-blocking communication," in *Design, Automation Test in Europe Conf. Exhibition (DATE)*, 2019, pp. 1625–1630.

[13] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "End-to-end timing analysis of cause-effect chains in automotive embedded systems," *Journal of Syst. Architecture*, vol. 80, pp. 104–113, 2017.

[14] ——, "Synthesizing job-level dependencies for automotive multi-rate effect chains," in *IEEE Int. Conf. on Embed. and Real-Time Comp. Syst. and Appl. (RTCSA)*, 2016, pp. 159–169.

[15] L. Köhler, P. Hertha, M. Beckert, A. Bendrick, and R. Ernst, "Robust cause-effect chains with bounded execution time and system-level logical execution time," *ACM Trans. Embed. Comp. Syst.*, vol. 22, no. 3, Apr. 2023.

[16] A. Naghavi, S. Safari, and S. Hessabi, "Tolerating permanent faults with low-energy overhead in multicore mixed-criticality systems," *IEEE Trans. on Emerging Topics in Computing*, vol. 10, no. 2, pp. 985–996, 2022.

[17] J. Day, *An AUTOSAR-Compliant Automotive Platform for Meeting Reliability and Timing Constraints (2011-01-0448)*, 2016, pp. 33–47.

[18] N. Mahmud, G. Rodriguez-Navas, H. Faragardi, S. Mubeen, and C. Seceleanu, "Power-aware allocation of fault-tolerant multi-rate autosar applications," in *2018 25th Asia-Pacific Software Engineering Conf. (APSEC)*, 2018, pp. 199–208.

[19] N. Mahmud, G. Rodriguez-Navasa, H. Reza, S. M. Faragardib, and C. Seceleanua, "Optimized allocation of fault-tolerant embedded software with end-to-end timing constraints," Tech. Rep., 2019.

[20] N. Gandhi, E. Roth, R. Gifford, L. T. X. Phan, and A. Haeberlen, "Bounded-time recovery for distributed real-time systems," in *Real-Time and Embed. Tech. and Appl. Symp. (RTAS)*, 2020, pp. 110–123.

[21] S. Mallareddy, P. K. Kondooru, and D. Gangadharan, "Checkpointing-aware end-to-end data age analysis of task chains under transient faults," in *Int. Symp. on Real-Time Distributed Computing (ISORC)*, 2024, pp. 1–10.

[22] P. Gohari, J. Voeten, and M. Nasri, "Towards a safe and latency-aware fault-tolerant scheduling technique for multi-rate task chains," in *Proc. of Int. Conf. on Real-Time Networks and Syst. (RTNS)*, 2025, p. 25–36.

[23] P. Nasahl and N. Timmers, "Attacking autosar using software and hardware attacks," Jul. 2019, escar USA.

[24] P. Fara, G. Serra, A. Biondi, C. Donnarumma *et al.*, "Scheduling replica voting in fixed-priority real-time systems," in *Euromicro Conf. on Real-Time Syst. (ECRTS)*, vol. 1, 2021.

[25] D. Griffin, I. Bate, and R. I. Davis, "Generating utilization vectors for the systematic evaluation of schedulability tests," in *IEEE RTSS*, 2020, pp. 76–88.

[26] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmarks for free," in *Int. Wkshp. on WATERS*, vol. 130, 2015, p. 43.