GRec: automatic computation of reconfiguration graphs for multi-core platforms

Guy Durrieu, Claire Pagetti

GRec: automatic computation of reconfiguration graphs for multi-core platforms

Guy Durrieu and Claire Pagetti
ONERA France

November 26, 2019

1 Introduction

The DREAMS [17] (Distributed REal-Time Architecture for Mixed Criticality Systems – 2013-2017) European project addressed the design of a cross-domain architecture for executing applications of different criticality levels in networked multi-core embedded systems. One of the outcome of the project was the development of a fault-tolerant distributed middleware that supports some failures.

1.1 Fault model and recovery mechanism

A DREAMS-compliant platform is composed of several multi-core chips connected through a TTEthernet network [53]. Two types of failures were considered during the project:

- permanent core failures. Intensive integration of small devices on chip increases the permanent failure occurrence due to various phenomena such as aging, wear-out or infant mortality [6];
- temporal overload situations for critical applications. This occurs when non-critical (also referred to as best-effort) applications have a too consuming access to shared resources leading to a non acceptable slow down.

The DREAMS resource management [20, 25] was developed in order to maintain the applications execution even in the presence of these two types of failure. In case of temporal overload, an adaptation strategy based on interrupting the best-effort applications at run-time was applied [36]. For permanent core failure, the resource management reconfigures the platform by selecting an adequate configuration in a set of pre-defined configurations.

1.2 Objective of the paper

We have presented the DREAMS fault-tolerant resource management [20, 25] in former papers, all of them assuming that the set of pre-defined configurations was available. The configurations are organized in reconfiguration graphs such that any decision is always deterministic and any combination of core failures is anticipated. The first purpose of this paper is to formally define the notion of reconfiguration graphs for a system composed of a set of applications hosted on a distributed platform in presence of permanent core failures impacting one or more nodes. The second objective is to provide a constraint programming-based approach to automatically compute the reconfiguration graphs. The last objective is to introduce the tool, GRec (standing for **G**raph of **Rec**onfigurations tool), developed during the DREAMS project. We show on a series of benchmarks its capacity to deal with large and realistic models.

1.3 Outline

The rest of the paper is organized as follows. Section 2 details the system model and the ressource management principles designed in the DREAMS project. Section 3 presents the principles of the automatic computation of reconfiguration graphs and introduces the GRec tool which implements the off-line search strategy. The idea is to iterate the search of configurations according to several parameters. For a fixed set of parameters, a configuration is computed with a constraint programming solver. Section 4

details the modeling of a configuration computation with constraints. The reconfiguration graphs once computed are used in the compilation tool chain and embedded in the platform. Section 5 gives some hints about configuration code generation. Section 6 exhibits some experimental results obtained from case studies used during the DREAMS project.

2 System model

2.1 Platform and application model

Definition 1 (Platform) *A DREAMS-compliant platform is composed of several multi-core processors (named* nodes*)* $\mathcal{N} = \{M_1, \ldots, M_n\}$ *connected via a TTEthernet network. Each multi-core processor* M_k *is composed of* n_k *identical cores. We uniformly identify all the cores of the platform using integers ranging from* 1 *to* $n_{\mathcal{N}} = \sum_{k \leq n} n_k$.

Each multi-core can have a specific speed and we consider the WCET (Worst-Case Execution Time) of a software as the maximal value of its WCET on each multi-core. This assumption can be easily leveraged by extending the WCET as a list of WCET, one per node. However, in the sequel, we assume a unique WCET to simplify the notations and formulas.

Definition 2 (Applications) *The platform hosts a series of applications* $\mathcal{A} = \{a_1, \cdots, a_r\}$. *An application* a_k *is composed of* r_k *periodic tasks. We uniformly identify all the tasks of the application set using integers ranging from* 1 *to* $r_{\mathcal{A}} = \sum_{k \leq r} r_k$. *In particular,* $a_k = \{\tau_i = (C_i, T_i)\}_{e_{k-1} < i \leq e_{k-1} + r_k}$ *where for all* $j < r$, $e_j = \sum_{q \leq j} r_q$, C_i *is the WCET (Worst Case Execution Time) and* T_i *is the period. A task* τ_i *can be unrolled as a set of jobs denoted* $\tau_{i.j}$ *– the j-th job of* τ_i. *Moreover, each application is typed with its criticality* $\text{type}(a_k) \in \{\text{critical}, \text{best-effort}\}$.

Applications are allocated (or distributed) to the nodes. At any time, a full application is allocated to a unique node (or none if too many failures have occurred).

Definition 3 (Allocation) *For the application set* $\mathcal{A} = \{a_i\}_{i \leq r}$ *and the platform* $\mathcal{N} = \{M_j\}_{j \leq n}$, *an allocation is defined by the function* $\text{alloc}_p(a_i) \in \{M_j\}_{j \leq n} \cup \{\perp\}$.

Moreover, the DREAMS resource management relies on TSP (time and space partitioning) principles compliant with IMA (Integrated Modular Avionics) [26, 2] which is the de-facto standard for current aircraft design. TSP is ensured by the XtratuM hypervisor [42], which was a technology involved in the project. An application is mapped to a set of *partitions* where a partition is defined by one or multiple *slots*. A partition cannot be shared by two different applications. To alleviate the notations, we will assume in the sequel that a partition is composed of a unique slot. This assumption does not impact the upcoming results, as regular partitions are simply a wrapping of slots. A *slot* is temporal access to a CPU with a start time and a length. Inside a slot, several tasks can be executed. Both the partitions slots and the schedule of tasks inside a slot are computed off-line, for instance with Xoncrete [7], the mapping and scheduling tool provided with XtratuM. This off-line piece of information is called *plan* in the XtratuM terminology.

Definition 4 (Plan on a node) *For a platform* $\mathcal{N} = \{M_i\}_{i \leq n}$, *an application set* $\mathcal{A} = \{a_i\}_{i \leq r}$ *and an allocation* alloc_p, *a plan on node* M_i *consists of:*

- *a* major frame *(MAF), the length of the schedule pattern repetition;*
- *a set of* m_i *slots* $\mathcal{S}_i$ *distributed over the cores and the MAF. A slot is defined as* $sl = (b, l, c)$ *where* b *is the start time,* l *is the length and* c *is the number of the core where the slot is allocated;*
- *a mapping of the jobs of the applications allocated to the node* M_i *in the slots. Jobs are unrolled on the MAF and we know for all job in which slot it belongs to. We know moreover in which order the jobs inside a slot are executed.*

Definition 5 (Configuration) *A configuration is a full description of the mapping and the scheduling of applications. Thus a configuration is given by* $\text{Conf} = \langle \text{alloc}_p, \text{plan}_{M_1}, \ldots, \text{plan}_{M_n} \rangle$.

As for the set of cores and application tasks, the set of slots $\mathcal{S} = \cup_{i \leq n} \mathcal{S}_i$ of a given configuration *Conf* is uniformly identified using integers ranging from 1 to $m_{\mathcal{S}} = \sum_{k \leq n} m_k$.

2.2 DREAMS executive layer overview

The fault-tolerant services are implemented on top of XtratuM and consist of:

- the GRM (Global Resource Manager) which is the centralized manager;
- MON (MONitoring) service which monitors the health of cores;
- LRM (Local Resource Manager) service which manages the local configuration of a multi-core.

DREAMS choices 1 *The design choices made during DREAMS were to:*

1. *allocate permanently the GRM on a specific multi-core;*
2. *suppose that the core hosting the GRM would never fail;*
3. *assign one LRM per core and execute them synchronously at the end of the MAF;*
4. *execute one MON per core and execute them at disjoint instant;*
5. *implement all of the services as* XtratuM *partitions.*

Example 1 (Configuration) *Let us consider an architecture $\mathcal{N}_1 = \{M_1, M_2\}$, see Figure 1, composed of 2 dual-core chips ($M_1 = \{arm_1, arm_2\}$, $M_2 = \{arm_3, arm_4\}$) connected via a TTEthernet switch. The application set is composed of 3 applications $\mathcal{A} = \{a_1, a_2, a_3\}$ such that:*

application	*tasks*	*type*	$alloc_p$
a_1	$\{\tau_1 = (2, 10), \tau_2 = (4, 100)\}$	*critical*	M_1
a_2	$\{\tau_3 = (20, 100)\}$	*best-effort*	M_1
a_3	$\{\tau_4 = (2, 20), \tau_5 = (4, 100)\}$	*critical*	M_2

Figure 1: Architecture $\mathcal{N}_1$

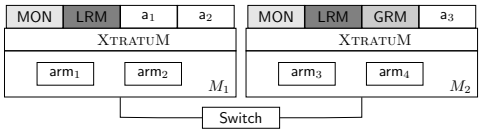
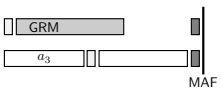


Figure 2: $plan_1$ on M_1

Figure 3: $plan_1$ on M_2



The services MON, LRM and GRM run on top of XtratuM*, as partition. GRM runs permanently on the arm_3 of M_2. The configuration consists of the 2* XtratuM *plans, the one defined on M_1 and the one defined on M_2. Both plans are named* plan_1*. For each plan, the MAF is 10 units of time. The set of slots $\mathcal{S}$ is composed of 15 slots and is shown in Figures 2 and 3. We have represented in white boxes the application slots and the DREAMS services with different shades of gray boxes. The white boxes without a name are spare partitions that would be used later in case of failure. In the sequel, we will not draw an empty slot. Among the 15 slots, 4 are dedicated to the MON (one per core), 4 to the LRM (one per core) and 1 to the GRM. Figure 2 shows the mapping and scheduling of the partition slots in* plan_1 *during a MAF. 2 LRMs (in dark gray) and 2 MON (in light gray) execute. We also observe that a_1 executes on arm_1 and a_2 on arm_2. Figure 3 shows the plan on the second multi-core. Each application is wrapped in a unique slot (or partition).*

Figure 4: Job schedule

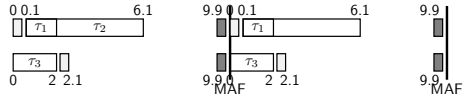
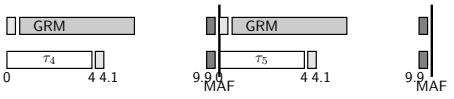


Figure 5: Job schedule



The scheduling inside the partition slots are not illustrated in the former figures. This is done in Figure 4 for M_1. a_1's slot has a length of 6 where τ_1 and τ_2 are called in sequence every 10 MAF and τ_1 is called alone 9 times over 10 MAFs. a_2's slot has a length of 2 and τ_3 is preempted every MAF. The scheduling for M_2 is given in Figure 5. a_3's slot has a length of 4 where τ_4 executes every even MAF and τ_5 executes on odd MAF every 10 MAFs. The complete repetition pattern is defined over the hyper-period, that is the lcm *(least common multiple) of all task periods and which is equal in our case to 10 MAFs.*

Usually, the mapping and scheduling need to be performed on the hyper-period. In the use cases studied during the DREAMS project, some tasks had very short periods (e.g. 10ms) while others had long ones (e.g. 1s). This leads to a long hyper-period and a schedule made of short time intervals. Making failure detection and reconfiguration at those very long hyper-periods may be inefficient. Moreover, finding such a schedule and a mapping over the hyper-period is combinatorial in the number of jobs and thus very costly. This has lead to the DREAMS design choice 2 which consists in taking a *MAF* that could either divide or multiply each period.

We thus compute a schedule on the *MAF* and then, there are two possibilities: either unroll the schedule on the hyper-period, or wrap the tasks in order to not be executed every MAF if their period is a multiple of the MAF. This is very similar to the notion of MIF (Minor Frame) and MAF on the Integrated Modular Avionics (IMA) systems [39]. When computing over the hyper-period, the schedule might be very long and the associated configuration files would have a large size. When computing over a sub-window of the hyper-period, the solutions are sub-optimal in the sense that there may exist a solution on the hyper-period but not on that sub-window. However, the schedule may be computed more easily as it is shorter and the configuration files much smaller. This is the reason why, we choose that option for GREC.

DREAMS choices 2 *Thus, it was decided to:*

- *choose a MAF such that any task period is either a multiple or a divider of the MAF*

$$\forall \tau \in \cup_{i \leq r} a_i,\ T_\tau | \text{MAF or MAF} | T_\tau$$

- *consider non preemptive schedule of tasks;*
- *the application tasks must be wrapped in order to not be executed every MAF if their period is a multiple of the MAF.*

The different acronyms are summarized in Table 1, the terminology in Table 2 and the number of elements in Table 3.

Acronym	Signification
GRM	Global Resource Management
LRM	Local Resource Management
MAF	MAjor Frame
MON	MONitoring
TSP	Time and Space Partitioning

Table 1: Table of acronyms

Terminology	Signification
node	a multi-core processor
partition	a pre-defined slot
plan	full mapping and scheduling on a node (see definition 4)
configuration	full mapping and scheduling on the platform

Table 2: Table of terminology

Set	Number of elements	Number of Sub-elements in k-th Element	Total Number of Sub-elements
$\mathcal{N}$	n nodes	n_k cores in the k-th node	$n_\mathcal{N} = \sum_{k \leq n} n_k$ cores in $\mathcal{N}$
$\mathcal{A}$	r applications	r_k tasks in the k-th application	$r_\mathcal{A} = \sum_{k<r} r_k$ tasks in $\mathcal{A}$
$\mathcal{S}$	n local sets of slots	m_k slots in the k-th node	$m_\mathcal{S} = \sum_{k \leq n} m_k$ slots in $\mathcal{S}$

Table 3: Reminder of notations

2.3 Reconfiguration in case of failure

The purpose of the DREAMS resource management is in particular to maintain a safe execution even in the presence of permanent core failures. When such a failure occurs, the LRM of the node where the core has failed selects a new plan to re-assign all its local applications or let the GRM find a new global configuration if it fails to host all them. Thus the LRM either adopts the initial configuration, or a configuration requested by the GRM, or selects a new configuration from the ones available.

DREAMS choices 3 *The reconfiguration strategy follows two rules in case not all applications could be locally hosted after some failure(s):*

1. *critical applications are locally reconfigured first,*

2. *complete applications must be moved, i.e. an application cannot run on two multi-core chips at the same time.*

Example 2 (Local reconfiguration) *Let us consider again example 1 and let us assume that* arm_1 *of* M_1 *fails during the second MAF. The failure is detected by MON and the LRM takes a local reconfiguration, that consists in moving to* plan_2 *at the next MAF as shown in Figure 6.*

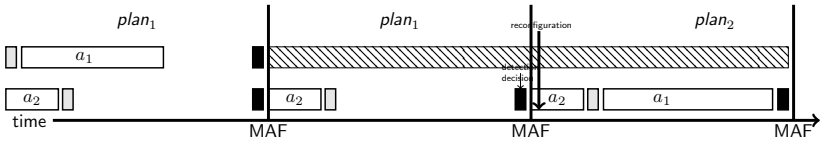


Figure 6: M_1 execution in case of permanent core failure.

At any time, the GRM knows the current configuration because it is periodically informed by the LRMs of their plan. Indeed, at every MAF, each LRM sends its current plan via an *update* message. When an LRM is not able to reconfigure all its local applications, it will reach a plan where not all applications were re-allocated. This type of configuration is pre-defined and the GRM can decide whether or not a global reconfiguration is to be performed.

Example 3 (Global reconfiguration) *Let us consider again the architecture* $\mathcal{N}_1$ *of example 1 and let us assume that* arm_4 *of* M_2 *fails during the MAF 1 while there is no failure on node* M_1*. There is no possible local reconfiguration for* a_3 *because no spare slot is available on* arm_3 *(see Figure 3). The LRM detects the failure and switches to a configuration where* a_3 *does not execute in MAF 2. The GRM receives the* update *message with the current plan of* M_2 *and deduces that* a_3 *is off platform. The left hand side of Figure 7 shows the scenario on node* M_2 *during these 2 MAF whereas the right hand side shows the scenario on node* M_1 *during the next MAF. In that case, the GRM sends an* order *to the LRM of* M_1 *to switch to* plan_4 *where it hosts* a_3*.*

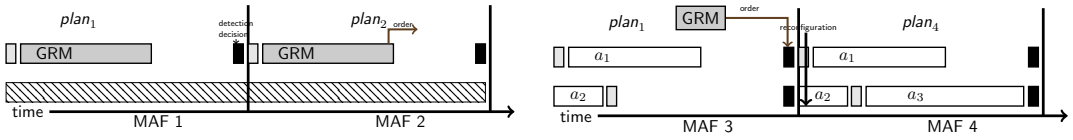


Figure 7: Global reconfiguration on M_1

The DREAMS design choices imply that the LRM takes local reconfiguration decisions at the end of the MAF by collecting all failed cores. This entails that several failures may happen during a MAF and decisions could consider multiple failures. Supporting multiple failures requires careful management of non determinism.

DREAMS choices 4 *The approach followed in the DREAMS project was to only define* symmetric behaviors*, that is for a given combination of core failures, the order in which the failures occurred has no impact on the reached configuration.*

Example 4 (Symmetric behavior) *Let us consider again the architecture* $\mathcal{N}_1$ *of example 1. Whether* arm_1 *of node* M_1 *fails followed later by* arm_4 *of node* M_2 *or* arm_4 *fails first followed later by* arm_1*, the reached configuration will be the same (*plan_2 *on* M_1 *and on* M_2*). The critical application* a_4 *is not executed any longer because there is not enough CPU time for both critical applications.*

2.4 Network reconfiguration

Reconfiguring the applications on different nodes has an impact on the network traffic. There are 3 classes of traffic on TTEthernet [53]:

– rate constraint (RC) traffic. Messages are wrapped in virtual links (VL) that are defined with their Bandwidth Allocation Gap (BAG – minimum time interval between two successive frames) and maximal packet size. Therefore, if an application is reconfigured on the same multi-core by a local reconfiguration then it has no impact on the routing table. If the reconfiguration is global, then several routing tables must be pre-defined;

– time triggered (TT) traffic. Messages are also wrapped in VL but those are emitted and transmitted at given instants. Therefore, if an application is reconfigured on the same multi-core by a local reconfiguration then it may require to update the instants. In case of global reconfiguration, new paths and new instants must be pre-computed;

– For best-effort (BE) traffic, the same reasoning as for RC traffic applies.

DREAMS choices 5 *The TT traffic is uniquely used for the resource management partitions which stay on their initial core forever. For the BE and RC traffic, the reachable configurations are provided to the tool that computes the network routing and scheduling. This provides a* super schedule.

As a consequence TTPlan [51] was updated during the project and is now able to compute a *super schedule* for the TTEthernet that supports all the possible computed configurations. A *super schedule* can be seen as the concatenation of each network schedule per configuration. Since 2 configurations cannot be activated at the same time, conflicting traffic are avoided.

3 General approach and GRec overview

The DREAMS resource management allows to continue a safe execution even in the presence of permanent core failures by switching from pre-defined configurations to pre-defined configurations. The purpose of this section is to present the principles in order to pre-compute all these configurations. The ideas were implemented in GRec.

3.1 Local and global reconfiguration graphs formalization

The LRM has been implemented as follows: it first checks that all MON have filled a dedicated shared memory area to confirm they are alive. If no new failure occurs since the previous MAF, then nothing is done. On the opposite, if some failures have occurred, it looks at an internal array that stores the *local reconfiguration graph* and takes the transitions one by one (one per core failure). Thus, GRec must generate those local reconfiguration graphs for each multi-core and the global reconfiguration for the GRM.

Definition 6 (Local reconfiguration graphs) *A local reconfiguration graph is a tuple* $\langle Q, \rightarrow, \dashrightarrow, q_0 \rangle$ *where:*

– Q *is a finite set of plans;*

– $q_0 \in Q$ *is the initial plan;*

– $\rightarrow \subseteq Q \times \mathbb{N} \times Q$ *is the set of local transitions from one plan to another. The integer represents the failed core id;*

– $\dashrightarrow \subseteq Q \times 2^{\mathbb{N}} \times Q$ *is the set of transitions from one plan to another requested by the GRM. The set of integers represents the failed cores id that have failed elsewhere.*

Example 5 (Local reconfiguration graph) *Let us consider again the system of example 1 with the different possible reconfiguration described in the former examples. The local reconfiguration graph of* M_1 *is given in Figure 8. We recognize the transition from* plan_1 *to* plan_2 *in case* arm_1 *fails. There is another local decision in case* arm_2 *fails leading to* plan_3 *that consists in moving* a_1 *on* arm_2 *and there is also the global reconfiguration leading to* plan_4 *when* arm_4 *fails. From* plan_4*, the plans reached after any core failure are specified.*

The local reconfiguration graph associated to the node M_2 *is shown in Figure 9. Whenever* arm_4 *fails, the reached plan is* $plan_2$*. A global reconfiguration in case both cores* arm_1 *and* arm_2 *have failed leads to* $plan_3$ *where* a_1 *is not executed any longer due to the lack of CPU. Since we do not know in which order they fail, it could be either* 1 *or* 2 *that triggers the global reconfiguration. The label of the transition is thus* $1, 2$*.*

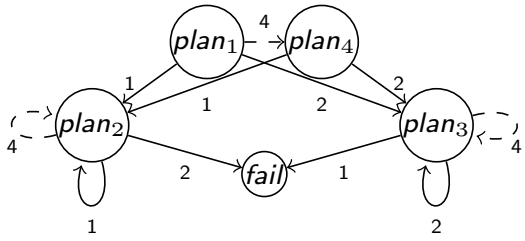


Figure 8: Reconfiguration graph of M_1

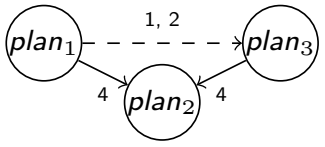


Figure 9: Local reconfiguration graph M_2

In order to deal with multiple failures, the solution we chose was to impose the local reconfiguration graphs to be *symmetric* (see DREAMS choices 4).

Definition 7 (Path, word and symmetry) *Let $\mathcal{G} = \langle Q, \rightarrow, \dashrightarrow, q_0 \rangle$ a local reconfiguration graph. A* finite path *$p = q_0.q_1 \dots q_n$ in $\mathcal{G}$ is such that for all $i < n$, $\exists l_i \in \mathbb{N}$, $(q_i, l_i, q_{i+1}) \in \rightarrow \cup \dashrightarrow$. The* word *$w_p$ associated to the path p is the sequence of integers provided by transitions, i.e. $w_p = l_0 \dots . l_{n-1}$. The graph is* symmetric *if for any two words w_{p_1} and w_{p_2} such that w_{p_2} is a permutation of w_{p_1}, the final state reached by the paths associated to w_{p_1} and w_{p_2} is the same.*

Example 6 *The reconfiguration graph of Figure 8 is symmetric. For instance, the words $w_1 = 1.4$ and $w_2 = 4.1$ leads to the state* plan_2.

In order to deal with any combination of failures, we chose to exhaustively enumerate all combinations and this entails the graph to be *complete*.

Definition 8 (Complete graphs) *Let $\mathcal{G} = \langle Q, \rightarrow, \dashrightarrow, q_0 \rangle$ a local reconfiguration graph. Let $[l, k]$ the set of core* ids *of the multi-core. The graph is complete if*

$$\forall q \in Q, \forall i \in [l, k], \exists q' \in Q, (q, i, q') \in \rightarrow$$

Example 7 *The reconfiguration graph of Figure 8 is complete. The core* ids *associated to this graph are* 1 *and* 2. *From any plan, there exist two outgoing transitions labeled by* 1 *and* 2.

Definition 9 (Global reconfiguration graph) *A global reconfiguration graph is a tuple $\langle Q, \rightarrow, \dashrightarrow, q_0 \rangle$ which consists of the product of all local reconfiguration graphs $\langle Q^i, \rightarrow^i, \dashrightarrow^i, q_0^i \rangle$ from the LRMs. More precisely:*

- $Q = Q^1 \times \ldots \times Q^n$,
- $q_0 = (q_0^1, \ldots, q_0^n)$,
- $\dashrightarrow \subseteq Q \times \mathbb{N} \times Q$ *is defined as*

$$((q^1, \ldots, q^n), l, (p^1, \ldots, p^n)) \in \dashrightarrow \iff \exists i \leq n, (q^i, l, p^i) \in \rightarrow^i \wedge \forall j \neq i, q^j = p^j$$

  Thus a dashed transition represents a unique local reconfiguration transition taken by an LRM.

- $\rightarrow \subseteq Q \times 2^{\mathbb{N}} \times Q$ *is defined as*

$$((q^1, \ldots, q^n), \{l_1, \ldots, l_m\}, (p^1, \ldots, p^n)) \in \rightarrow \iff \begin{cases} \exists k \leq n, \exists i_1, \ldots, i_k, \forall l \leq k \\ \qquad (q^{i_l}, \{l_1, \ldots, l_m\}, p^{i_l}) \in \dashrightarrow^i \\ \qquad \wedge \forall j \neq i_1, \ldots, i_k, q^j = p^j \\ \wedge \quad \forall l_k, \exists j, \exists r^j \ (r^j, l_k, q^j) \in \rightarrow^j \end{cases}$$

  A plain transition represents a global reconfiguration requested by the GRM to some LRMs. Such a transition can only occur after a local failure such that some application is not any longer executed. To detect such a situation, the GRM knows the plans executed by the LRM and those requiring a global decision.

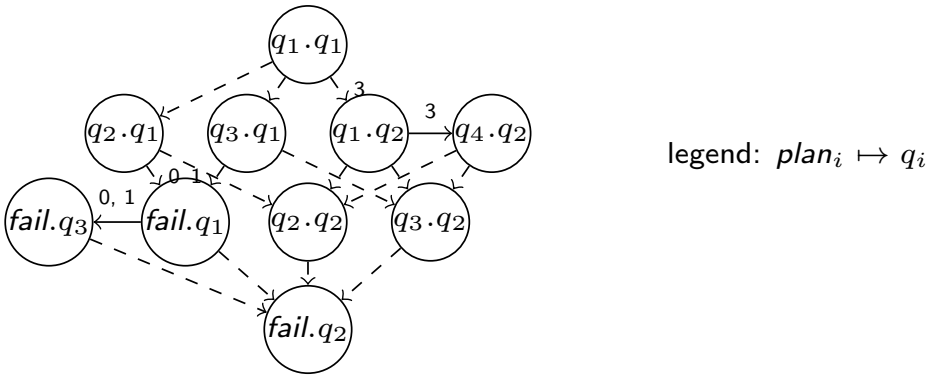


Figure 10: Global reconfiguration graph

Example 8 *Let us consider again the architecture* $\mathcal{N}_1 = \{M_1, M_2\}$ *of Figure 1. The global reconfiguration graph is drawn in Figure 10. We did not draw the label on the transition except those that target a global transition.*

Back to the reconfiguration graph of node M_1 *given Figure 8, once in the* plan_2 *the GRM will not send a global reconfiguration. But due to the distributed implementation and the temporal delays on the processors and the network, the GRM can still believe the node is* plan_1 *when sending the request. Such a behavior is shown in Figure 11. However, the LRM of* M_1 *will remain in its local plan.*

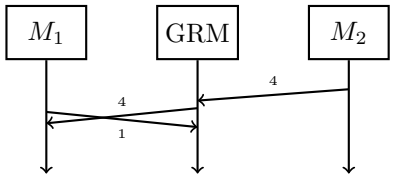


Figure 11: Sequence diagram illustrating an ignored global reconfiguration

3.2 Search strategy

The work was done in a collaborative context offered by the DREAMS project and our approach was to re-use existing tools as much as possible. The overall DREAMS toolchain has been presented in [4]. In particular XONCRETE [7] is capable of computing local plans for multi-core.

DREAMS choices 6 *Thus, it was decided to:*

- *define manually an initial* allocation *of applications (*alloc_p*);*
- *compute for each multi-core an initial plan with* XONCRETE *in order to permanently fix the slots on each node (*$\mathcal{S}$*).*

We could easily leverage these limitations in a future work, at the cost of an increased complexity. GREC must compute an *optimal* configuration for any combination of permanent core failures. We have stated in Section 2.2 that GREC searches solutions in a sub-optimal space as schedules are computed with a length of MAF instead of lcm. Thus in our context, *optimal* configuration stands for executing as many critical applications as possible. The exploration has a double dimension: a given combination of core failures and a more classical configuration (mapping and scheduling). Our strategy for this double exploration is shown in Figure 12.

The GREC steps are highlighted in bold on the arrows while the boxes represent the handled objects. The inputs of the flows are the platform and application descriptions, together with the initial inputs mentioned above. GREC then follows four main steps that are detailed below.

3.3 Step 1: failure combination generation

First GREC computes all the combinations of core failures with a PYTHON script. Such a combination is represented by an array of size $n_{\mathcal{N}} = \sum_{k \leq n} n_k$ (the overall number of cores on the platform). The value

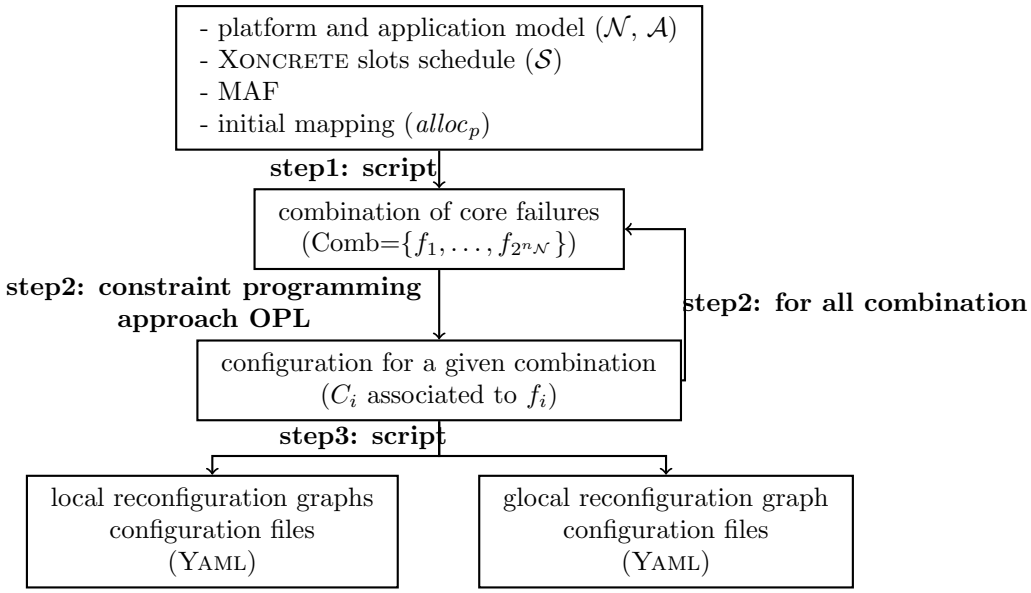


Figure 12: Overview of GREC.

0 at index i means the i-th core has not failed while 1 means that the core has failed.

Example 9 *For instance, for the architecture $\mathcal{N}_1$ of Figure 1, no failure will be represented by* $[0,0,0,0]$ *whereas the failures of arm_1 and arm_3 would be represented by* $[1,0,1,0]$.

The set of generated failure cases is *exhaustive*: all possible failure combinations are taken into account. There is a combinatorial explosion in that dimension as the number of combinations is $2^{n_\mathcal{N}}$. We denote by *Comb* the set of all failure combinations.

3.4 Step 2: configurations generation

Practically, an iterative loop calls a constraint programming solver to compute a configuration for a given combination of core failures. As there are many rules to follow, this step is decomposed as a series of sub-steps. Figure 13 summarizes the process guiding the computation of the set of configurations. Practically, the new configurations are computed with the constraint solver IBM OPL Studio framework [32] and the iteration is done with a C code.

Initially, the set of configuration is empty and the set of failures *Comb* to deal with is full (with all possible combinations of failures computed by the PYTHON script). We denote by $\mathcal{S}_{conf}$ the set of all reachable configurations.

- GREC selects a combination of failures in the set *Comb* computed by the step 1. The scan is ordered from the minimal number of failures to the highest (thus the first iteration will compute the initial configuration where no failure occurred and where the initial mapping is respected);
- GREC first tries to find an already computed configuration that is not impacted by the failed cores. For a configuration *Conf*, used_core(*Conf*) is the set of cores that are used by the configuration, i.e. the set of cores that execute at least a slot with an application. used_core(*Conf*) $\cap f = \emptyset$ means that the configuration is not impacted by the additional core failures. In that case, GREC keeps the configuration *Conf* unchanged as it supports the core failures f;
- if no already computed configuration can accept the combination of failure f, then the solver must find a new configuration *Conf*$'$. The strategy is to try first local reconfiguration. Thus GREC searches among $Conf \in \mathcal{S}_{Conf}$, if the application allocations of *Conf* can be preserved. $alloc_p(Conf) = alloc_p(Conf')$ means that the spatial allocation of application is preserved. However the scheduling and the local mapping can be changed;

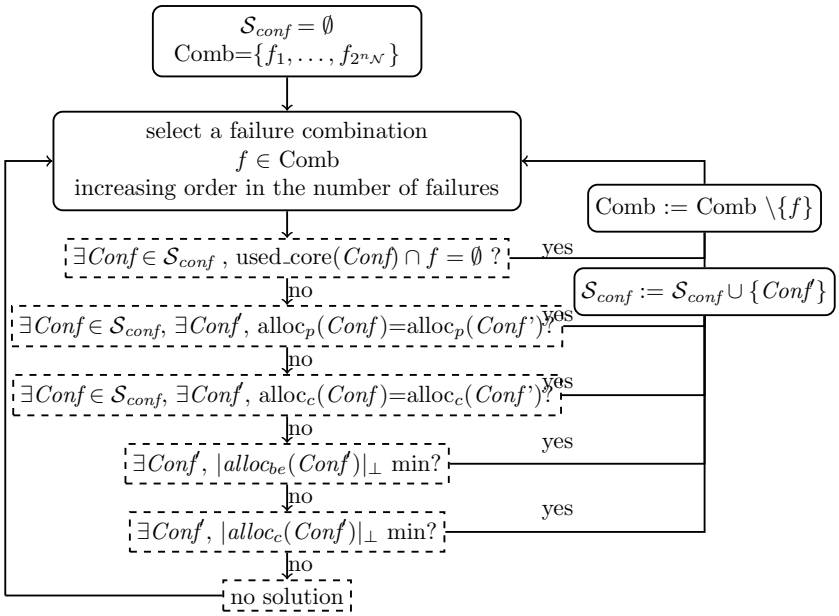


Figure 13: Configurations generation process.

- if no local solution can be found, GREC searches for a global reconfiguration, only moving the spatial allocation of best-effort applications. $alloc_c(Conf) = alloc_c(Conf')$ means that the spatial allocation of critical applications is preserved;
- if no such solution can be found, GREC tries to find a new spatial assignment of all applications, while removing the least possible number of best-effort applications. For any configuration *Conf*, $|alloc_{be}(Conf)|_\perp$ is the number of best-effort applications that have been removed from the platform;
- if no such solution can be found, GREC tries to find a new configuration while removing the least possible number of critical applications;
- when no application can be mapped, the search reaches no solution.

Example 10 *Let us illustrate the algorithm on the architecture* $\mathcal{N}_1 = \{M_1, M_2\}$*. The exhaustive enumeration of failures combinations is the set Comb=*$\{[0,0,0,0], [1,0,0,0]$, $[0,1,0,0]$, $[0,0,1,0]$, $[0,0,0,1]$, $[1,1,0,0]$, $[1,0,1,0]$, $[1,0,0,1]$, $[0,1,1,0]$, $[0,1,0,1]$, $[0,0,1,1]$, $[1,1,1,0]$, $[1,1,0,1]$, $[0,1,1,\ 1]$, $[1,1,1,1]\}$. GREC *computes the initial configuration shown in the example 1. This configuration is defined by*

$$\text{Conf}_1 = \langle \text{alloc}_{p.1} = \begin{cases} a_1 \mapsto M_1 \\ a_2 \mapsto M_1 \\ a_3 \mapsto M_2 \end{cases}, M_1.\text{plan}_1, M_2.\text{plan}_1 \rangle$$

By definition, alloc_p *describes a unique mapping of applications to nodes. Due to the computation of all possible reconfiguration, the applications may be mapped differently on the platform. This particularly occurs in case of global reconfiguration. We thus need to enumerate (or identify) all mappings. This is done by numbering the occurrences of the function* alloc_p *and we use the notation* $\text{alloc}_{p.k}$ *to define the k-th* alloc_p*. As a consequence* $\text{alloc}_{p.1}$ *is the initial mapping given as an input.*

For the failure combination $[1,0,0,0]$*, we have seen in example 2 that a local reconfiguration can be applied. Thus the new configuration is* $\text{Conf}_2 = \langle \text{alloc}_{p.1}, M_1.\text{plan}_2, M_2.\text{plan}_1 \rangle$*.*

For the failure combination $[0,1,0,0]$*, a local reconfiguration can also be applied that consists in moving* P_1 *on core* arm_1*. Thus the new configuration is* $\text{Conf}_3 = \langle \text{alloc}_{p.1}, M_1.\text{plan}_3, M_2.\text{plan}_1 \rangle$*.*

The failure combination $[0,0,1,0]$ *is impossible because it was supposed in the DREAMS project that the GRM could never fail. Thus all combinations* $[X,X,1,X]$ *are also impossible. This limitation could be leveraged, but first the design of the resource management would have to be slightly modified.*

The failure combination $[0,0,0,1]$*, as shown in example 3, leads to a global reconfiguration* $\text{Conf}_4 = \langle \text{alloc}_{p.2}, M_1.\text{plan}_4, M_2.\text{plan}_2 \rangle$*. The* $M_2.\text{plan}_2$ *simply executes the GRM.*

The failure combination $[1,1,0,0]$ *leads to a global reconfiguration where* a_1 *is lost.* $\text{Conf}_5 = \langle \text{alloc}_{p.3}, M_1.\_, M_2.\text{plan}_3 \rangle$.

The failure combination $[1,0,0,1]$ *leads to a global reconfiguration where* a_3 *is lost.* $\text{Conf}_6 = \langle \text{alloc}_{p.4}, M_1.\text{plan}_2, M_2.\text{plan}_2 \rangle$.

The failure combination $[0,1,0,1]$ *leads to a global reconfiguration where* a_3 *is lost.* $\text{Conf}_7 = \langle \text{alloc}_{p.4}, M_1.\text{plan}_3, M_2.\text{plan}_2 \rangle$.

The failure combination $[1,1,0,1]$ *has no solution. All applications are lost. The computed configurations correspond exactly to the states of the reconfiguration graph of Figure 10.*

3.5 Step 3: configuration files generation

The DREAMS resource management compilation tool chain requires several configuration files, among which some YAML files that describe the reconfiguration graphs. Once all possible reconfigurations have been computed, GREC generates one configuration file for each node. These YAML files are textual and are compiled into XML files that are used by the XTRATUM hypervisor.

4 Constraint programming

Let us detail in this section how each configuration is computed with OPL STUDIO (this corresponds to step 2 of Figure 12).

4.1 Input data for constraint solving

The input for GREC is the system model as described in section 2. More precisely, the following elements are specified: the components of the platform hardware architecture (nodes, cores, etc.); the applications hosted on the platform (tasks, periods, WCETs, etc.) and the temporal partitioning. Because of the DREAMS choices 6, we also incorporate the XONCRETE partition slots schedule and the initial $alloc_p$ imposed by the designer. More formally, the inputs are:

- the MAF;
- the task set defined as $\mathcal{T} = \{\tau_1, \ldots, \tau_{r_\mathcal{A}}\}$;
- the task properties defined as $\forall \tau \in \mathcal{T}, prop_t[\tau] = [a, T, C, type]$ where a is the application it belongs to, T period, C WCET and $type \in \{critical, best\text{-}effort\}$;
- the set of jobs defined as $\mathcal{J} = \{\tau_{1.1}, \cdots, \tau_{1.j_1}, \cdots, \tau_{r_\mathcal{A}.j_{r_\mathcal{A}}}\}$ where the tasks are unrolled on the MAF. In particular, the number of jobs associated to task τ_k over the MAF is $j_k = \lceil MAF/T_k \rceil - 1$. This unrolling is done automatically by OPL STUDIO;
- the job properties defined as $\forall \tau_{i.j} \in \mathcal{J}, prop_j[\tau_{i.j}] = [\tau_i, r_{i.j}, d_{i.j}]$, where $r_{i.j} = j \times T_i$ is the release time of job $\tau_{i.j}$ and $d_{i.j} = (j+1) \times T_i$ is its deadline;
- the partition slots set $\mathcal{S} = \{sl_1, \ldots, sl_{m_\mathcal{S}}\}$;
- the slot properties $\forall sl \in \mathcal{S}, prop_{sl}[sl] =< b, l, \nu, c, avail >$ where b is the start time, l is the slot duration, ν is the node to which it belongs, c is the core on which the slot is allocated, $avail \in \{\top, \bot, L, M, G\}$ defines the slot availability with $\top$ equals available, $\bot$ non available (e.g. because the associated core has failed), L dedicated to LRM, M dedicated to MON and G dedicated to GRM. The last slot is special as it represents a virtual slot where canceled applications will be allocated.

4.2 Decision variables

Constraint solving consists in giving a value satisfying a set of constraints to a set of special variables called *decision variables*. In our case, there are two decision variables:

- $p_{i.j} \in \mathcal{S}$ specifies the allocation of each job $\tau_{i.j}$ to an available slot;
- $s_{i.j} \in [0, MAF - 1]$ specifies the start time of each job $\tau_{i.j}$ within the MAF.

4.3 Constraints

The set of constraints specified in GREC covers different aspects that are detailed below in a mathematical fashion, very close to their actual expression in the OPL STUDIO language [32].

Basic rules Only one job is active at a given time, expressed as: given two different jobs allocated to the same slot, either the first ends before the second starts or the first starts after the second ends.

$$\forall \tau_{i.j}, \tau_{k.l} \in \mathcal{J}, \tau_{i.j} \neq \tau_{k.l} \wedge p_{i.j} = p_{k.l} \Rightarrow s_{i.j} \geq s_{k.l} + C_k \vee s_{k.l} \geq s_{i.j} + C_i$$

Each job must start after its release time and must end before its deadline.

$$\forall \tau_{i.j} \in \mathcal{J}, \mathrm{r}_{i.j} \leq s_{i.j} \wedge s_{i.j} + C_i \leq \mathrm{d}_{i.j}$$

Partition allocation rules There is only one application within a partition slot, expressed as: given two different jobs, if they are allocated to the same slot then they belong to the same application. As a short cut, we write $a_{\tau_{i.j}} = prop_t[prop_j[\tau_{i.j}][0]][0]$ (the application the job $\tau_{i.j}$ belongs to).

$$\forall \tau_{i.j}, \tau_{k.l} \in \mathcal{J}, \tau_{i.j} \neq \tau_{k.l} \wedge p_{i.j} = p_{k.l} \Rightarrow a_{\tau_{i.j}} = a_{\tau_{k.l}}$$

A job can only be allocated to an *available* slot and must start and end within the slot. As short cut, we write $b_{i.j} = prop_{sl}[p_{i.j}][0]$ the beginning of the slot $p_{i.j}$, $l_{i.j} = prop_{sl}[p_{i.j}][1]$ its length and $avail_{i.j} = prop_{sl}[p_{i.j}][2]$ its availability.

$$\forall \tau_{i.j} \in \mathcal{J}, b_{i.j} \leq s_{i.j} \wedge s_{i.j} + C_i \leq b_{i.j} + l_{i.j} \wedge avail_{i.j} = \top$$

We also try to minimize the dispersion of jobs of a given task among slots. Meaning that if a slot is long enough, greater than the period of a task, we try to allocate several jobs of the same task to it. This constraint is not mandatory and can be removed.

$$\forall \tau_{i.j} \in \mathcal{J}, s_{i.j} + C_i \leq b_{i.j-1} + l_{i.j-1} \Rightarrow p_{i.j} = p_{i.j-1}$$

Strategy related rules The strategy-related rules can be either activated or deactivated depending on the current strategic phase. The strategies described in section 3.4 first try to allocate all applications. Meaning that none is canceled *status(a)= active*. When several cores have failed and not all applications could be allocated, then some are not executed anymore to search for a solution. In that case, those applications are tagged as *status(a)= cancel* and are allocated to the slot m, which can be seen as a *trash* slot. Implicitly, all constraints detailed before are only valid for *active* applications.

$$\forall \tau_{i.j} \in \mathcal{J}, p_{i.j} = m \iff status(a_{\tau_{i.j}}) = cancel$$

Then OPL STUDIO is called in sequence as long as it does not find a solution with the following constraint:

1. in case of local reconfiguration (with short cut $c_{i.j} = prop_{sl}[p_{i.j}][2]$)

$$\forall \tau_{i.j} \in \mathcal{J}, c_{i.j} = alloc_p(a_{\tau_{i.j}})$$

2. in case of global reconfiguration where critical applications remain unmoved

$$\forall \tau_{i.j} \in \mathcal{J}, type(a_{\tau_{i.j}}) = critical \implies c_{i.j} = alloc_p(a_{\tau_{i.j}})$$

3. in case of global reconfiguration, no constraint;

4. in case of canceling best-effort application, some of the best-effort applications are tagged with *cancel* in the input file;

5. in case the previous strategies failed, some of the critical applications are tagged with *cancel* in the input file.

5 Code generation

The configuration files are expressed in YAML which is a much less verbose description than an xml. From those files, the XTRATUM xml configuration file is computed and some C code as well. Each YAML file includes several parts.

Description of applications The first part (`apps`) specifies the set of applications which can be hosted on the considered node in some configuration computed by GREC. For each application, tasks composing the application are listed.

Example 11 *The code below shows the application description for the example 1 and node* M_1.

```
apps:
  - &A1
    name: a1
    tasks:
      - { name: tau1, func: tau1 }
      - { name: tau2, func: tau2 }
  - &A2
    name: a2
    tasks:
      - { name: tau3, func: tau3 }
   - &A3
    name: a3
    tasks:
      - { name: tau4, func: tau4 }
      - { name: tau5, func: tau5 }
```

Description of the time splitting The second part (`hw_desc`) describes the partition slots on the cores. For each configuration (`plan`), slot start time and duration are specified as well as the name of the application that are allocated to the slot.

Example 12 *The code below shows the configurations for the example 1, node* M_1 *and core* arm_1. XTRATUM *does not support real numbers, therefore the times are given in* μs *instead of* ms.

```
hw_desc:
  num_cores: 2
  processor_table:
    - id: 0 # core arm1
      plan:
       - id: 1 # plan 1 on  arm1
          major_frame: 10000
          slots:
            - { id: 0, start: 0, duration: 100, dlrm: mon_cf }
            - { id: 1, start: 100, duration: 6000, part: a1 }
            - { id: 2, start: 9900, duration: 100, dlrm: lrm_cf }
       - id: 2 # plan 2 on arm1,  arm1 failed
          major_frame: 10000
          slots:
       - id: 3 # plan 3 on core  arm1,  arm2 failed
          major_frame: 10000
          slots:
            - { id: 0, start: 0, duration: 100, dlrm: mon_cf }
            - { id: 1, start: 100, duration: 6000, part: a1 }
            - { id: 2, start: 6100, duration: 2000, part: a2 }
            - { id: 3, start: 9900, duration: 100, dlrm: lrm_cf }
       - id: 4 # plan 4 on  arm1, arm3 failed
          major_frame: 10000
          slots:
            - { id: 0, start: 0, duration: 100, dlrm: mon_cf }
            - { id: 1, start: 100, duration: 6000, part: a1 }
            - { id: 2, start: 9900, duration: 100, dlrm: lrm_cf }
     - id: 1 # core arm2
       ....
```

Description of job allocation The next part is devoted to specifying the jobs schedule inside the slot for each configuration (`plan`).

Example 13 *The code below shows the job schedule for* a_1 *of example 1 on node* M_1.

```
part_desc:
  - id: a1
    plans:
      - id: 1
        slots:
          - id: 1
            tasks: [ tau1, tau2]
    ...
```

Local reconfiguration graph The last part of the YAML file is the reconfiguration table.

Example 14 *The code below shows the reconfiguration table of example 1 on node M_1. It consists of an array of* size = nb_cores × nb_plans, *i.e.* size= 2×4.

```
reconfiguration_table:
  - [ 2, 3 ]   // plan 1: if core 1 fails go to plan 2, if core 2 fails go to plan 3
  - [ -1, -1 ] // plan 2: if core 1 or 2 fails, nothing to be done
  - [ -1, -1 ] // plan 3: if core 1 or 2 fails, nothing to be done
  - [ 2, 3 ]   // plan 4: if core 1 fails go to plan 2, if core 2 fails go to plan 3
```

6 Experiments

In this section, we highlight how GREC was used in the project and its capabilities. GREC was run on the processor Intel® Xeon® CPU 2.40GHz including eight cores and 36GB of main memory. A *time limit parameter* (the default value of which is 900 seconds) is set to stop OPL STUDIO. When there is a timeout, it is supposed that no solution exists.

6.1 First DREAMS use case

The platform was composed of two Freescale QorIQ T4240QDS (Freescale QorIQ T4240) [27] and a DREAMS Harmonized Platform [18], an heterogeneous platform based on the Xilinx ZC-706C FPGA platform (see Figure 15).

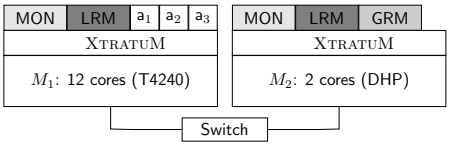


Figure 14: First DREAMS use case – variant 1

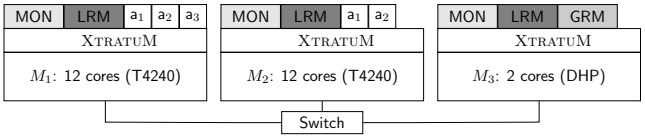


Figure 15: First DREAMS use case – variant 2 and more

The use case is composed of three applications:

- the MPEG server [33] developed by University of Kaiserslautern;
- the ROSACE [45] case study. ROSACE has been modified during DREAMS in order to communicate with the MPEG server. Thus, one task has been added to send data to MPEG server and an existing one has been modified to receive data from it;
- a new application *Order generator* that has been developed during the project and which aims at sending command to ROSACE.

Thus the application set is defined as:

application	number of tasks	criticality	alloc_p
ROSACE	12	critical	M_1
MPEG Server	1 (Mpeg2Server =(100, 1000))	best-effort	M_1
Order generator	1 (VacGenerator=(1, 10000))	best-effort	M_1

The task set for ROSACE is { *elevator*= (1, 50), *q_filter*= (1, 100), *vz_filter*= (1, 100), *vz_control*= (1, 200), *engine*= (1, 50), *va_control*= (1, 200), *aircraft_dynamics*= (10, 50), *az_filter*= (1, 100), *h_filter*= (1, 100), *va_filter*= (1, 100), *altitude_hold*= (1, 200), *ToMPEG*= (1, 10000)}. Time values are here expressed in milliseconds, but the time unit may also be parameterized.

Variant 1: MAF = 1000ms, 95 time slots, 1 Freescale QorIQ T4240QDS. In this first variant, we consider a subset of the demonstrator with only one Freescale QorIQ T4240QDS (see Figure 14). Since the DREAMS Harmonized Platform cannot fail, the number of failure cases is 4096. GREC computes 79 valid reconfiguration plans. The table on the left hand side below summarizes the time and space needed by GREC to make the computation. The table on the right hand side details the number of reconfiguration and memory footprint.

steps	1, 2	3
time	4m 41s	2m 20s
memory utilization	391,6 MB	251,7 MB

	M_1	M_2
number of configurations	78	13
size of file	578.6kB	9.3kB

Variant 2: MAF = 1000ms, 3 applications, 14 cores. In this variant, we fix the number of cores at 14 and we consider two architectures. The same as in variant 1 (as in Figure 14) and the architecture of Figure 15 with only 6 active cores per Freescale QorIQ T4240 node. The number of failures is still 4096. In the table below we compare the results for different number of slots (where the slots are identical whether there are 2 or 3 nodes). The value $< X, Y, 2 >$ represents the number X of active cores in the first node and the number Y of active cores in the second node. For each configuration, we look at 2 sub-cases: when there is some available slot on the DREAMS Harmonized Platform which never fails or none.

number of slots	53		53		119		119	
active cores	$< 12, 0, 2 >$		$< 6, 6, 2 >$		$< 12, 0, 2 >$		$< 6, 6, 2 >$	
active slots on M_3	2 slots	0 slot	2 slots	0 slot	2 slots	0 slot	2 slots	0 slot
overall time	15s	1m 48s	46s	3m 11s	1m 3s	2m 39s	17s	5m 4s
number of plans	77	94	26	70	73	97	14	60
size of file	485.8kB	646.2kB	118.9kB	482kB	459.7kB	533.2kB	90.6kB	530.3kB

Variant 3: MAF = 1000ms, 2 Freescale QorIQ T4240QDS, 5 applications. In this variant, we consider the platform of Figure 15 with all cores activated. The number of failure cases is 16 777 216. The ROSACE and MPEG server applications are duplicated, to increase the load on the platform. That makes 255 jobs. We vary the number of slots.

number of slots	97	181	229
overall time	12h 59m	22h 22m	27h 33m
number of plans	3844	7722	5824
size of file	5.3Mb	6.45Mb	5.9Mb

Variant 4: MAF = 1000ms, 2 Freescale QorIQ T4240QDS, 6 applications. We keep the same architecture as before and in the application set we add the simplified version of the FAS (Flight Application Software) [44], the control system of the Automated Transfer Vehicle developed by EADS Astrium Space Transportation. This application is composed of 19 tasks leading to 58 jobs, with a great variety of periods and some very large WCET. Thus, it is more *hard* to schedule it. We made two experiments: the first by reducing the large WCET, so that $\forall \tau$, $C_\tau = min(C_\tau, 30)$ and the second with the real WCET.

number of slots	97	229	97	229
WCET	reduced		regular	
overall time	2j 15h 15m	4j 5h 31m	stop at 8j	stop at 8j
number of plans	20711	110882	73823 for 1085220 configurations	68164 for 170967 configurations
size of file	395.78Mb	1.05Gb	-	-

6.2 DREAMS demonstrator

For the DREAMS avionic demonstrator, we used the architecture of Figure 15. The use case composed of four applications including 40 tasks has been deployed:

- the Flight Management System (FMS case study) developed by Thales Research & Technology [19];
- other proprietary applications.

Due to confidentiality issue, we will not present more details on the use case. As usual, we have varied the number of slots and active cores.

number of slots	43		98		181
types	all critical	one best-effort	all critical	one best-effort	all critical
active cores	$< 4, 4, 2 >$		$< 4, 4, 2 >$		$< 12, 12, 2 >$
overall time	51s	51s	55s	57s	6h 12m
number of plans	197	201	195	216	649
size of file	68kB	64.1kB	73.6kB	73.7kB	539.1kB

6.3 Discussion on the results

Thanks to the decoupling between the calls to OPL Studio and the iteration with efficient scripts, the overall principles are very efficient (for an exhaustive search) and not prohibitive in terms of memory consumption.

Problem complexity Determining a priori the number of configuration or the exploration time is hardly feasible. Indeed, the experimental results exhibit a great disparity between different case studies or different variants of the same set of applications. The exploration time depends on several contributing factors:

- time to compute one new configuration. This factor depends on the OPL Studio modeling and used heuristics. For instance, a large variance of WCET is harmful: the cohabitation of large tasks and very fast tasks leads to a huge lost of efficiency (regular variant 4);
- number of combination of core failures: this has an impact on the size of the loop on step 2;
- number of slots: the more slots, the more time. This clearly emerges from the experiments;
- number of reusable configurations: the variant 4 with FAS has very poor performances because computation of new configurations is needed too often;
- ability to compute quickly reusable configurations: some configurations computed after 10^6 configurations could have worked for many previous configurations and the overall number of configuration would have greatly been reduced if computed at the very beginning. A way to compute better configurations would be to add an optimization criterion (e.g. minimal number of cores), however minimization problems take much longer than feasibility problems;
- initial mapping: because of the DREAMS choices 1, the DREAMS Harmonized Platform cannot fail. If there are some available slots on the DREAMS Harmonized Platform, once a critical application is allocated there, it would remain there during the exploration steps and this will accelerate the search. Thus, the sooner GRec uses them, the fastest the reconfiguration graph will be computed;
- number of tasks: the variants 3 and 4 (reduced version) with 97 slots is a good illustration. Just adding the FAS leads to an extreme degradation of performance for an equivalent number of plans;
- presence of best-effort applications: non-critical applications introduce additional steps hence increasing the number of strategic rules.

Storage space and run-time efficiency The size of the configuration files could also be a limiting factor depending on the target architecture. The current configuration file format is compliant with XtratuM configuration file while is verbose and in xml. We made a comparison in two situations: first we compute the reconfiguration graph on the hyper-period and second, we have wrapped the C tasks with a counter over the hyper-period. For instance, this reduces the variant 1 from 3.2MB to 578.6kB on M_1. This is the reason why for all experiments, we use the second situation. However even in this case, the hardware storage may become a bottleneck. There are mainly three approaches to deal with this issue: the first is to store the reconfiguration graph as such in a large DDR memory as currently done; the second is to store a compressed version of the configuration file or the third is to store the files in a dedicated memory (e.g. flash). For the two last solutions, XtratuM would have to be modified. A degraded approach would be to restrain the exploration in the depth of the number of failures per node to keep the configuration file size below a certain threshold. Note that GRec search is independent of XtratuM and would be able to deal with more compact formats.

The run-time execution time depends on the read memory access time to the reconfiguration graph and on a linear contribution of the number of new failures. Indeed, let us suppose that at some instant the LRM is in a certain reconfiguration graph state s. At the next MAF, it may that k cores $i_1, \ldots, i_k$ have failed where k is bounded by the number of cores on the node. Thus, the run-time will compute the k transitions $s \rightarrow^{i_1} s_{i_1} \ldots \rightarrow^{i_k} s_{i_k}$ through an enumerate instruction. As the graph is symmetric, it will consider the i_j in ascendant way.

7 Related work

7.1 Scheduling

Since the paper by C. L. Liu and J. W. Layland [40], scheduling studies in general imply the existence of a dynamic scheduler which, according to a given policy, actually performs the scheduling of arriving tasks. Thus most papers in this domain are mainly interested in improving the *schedulability bound*, the calculus of which decides whether or not a given task set is guaranteed to always meet its deadlines [24], or in empirically testing the effectiveness of the underlying theories [9]. The approach of GREC is quite different from these ones: the certification of an avionic critical system requires the determinism of its operation, so given a hardware configuration, GREC has to find an actual static scheduling of all periodic tasks, by computing an accurate valid start time for each task; determinism implies there is no dynamic scheduler, but an hypervisor for each core, controlling the execution process by launching each task at the specified start time.

More specifically within the multi-core context, the proposed methods in general involve a partitioning process, intended to split the tasks over the different cores [11], [23], [34], [41]. The partitions are as far as possible made of "compatible" tasks from the period point of view, the objective again being to maximize the utilization bound for each core, and thus the global performance of the system. By contrast, for GREC the partitioning is an input parameter, elaborated by the system designer according to safety requirements (application containment) rather than performance or utilization ones. Additionally, some of the previously cited works ([23], [34]) try to enhance the compatibility of tasks using some "task set scaling" algorithms; these algorithms, such as [38], adapt the ratio *period/execution time* of a given task set according to a reference task (e.g. the one with the lowest period), again in order to optimize the utilization bound of the associated core. [34] claims that doing so the utilization bound for each core approaches 100%, but this kind of scaling algorithm would not be admissible for an aeronautic critical embedded system: for GREC, task periods and execution times are intangible input parameters.

7.2 Reconfiguration graph

Graphs, as computer science objects, have been widely studied and used in various domains, especially in the reconfiguration of parallel computers domain, e. g. for analyzing the validity of reconfiguration algorithms, something which is quite close to our problem of calculating valid reconfiguration for a set of multi-core nodes.

[16] advocates the use of Reconfiguration Graph Grammars (RGG) for supporting the design and analysis of reconfiguration algorithms in massively parallel computers. Within this context, fault-tolerance is achieved by including spare processors in the design. When one or more processors fail, a reconfiguration is performed which substitutes spare processors for the faulty ones by activating certain communication lines and deactivating others within the processor array. A RGG production represents a change in the communication lines which performs the reconfiguration. The addition of edges between nodes indicates the activation of communication lines, and the removal of edges indicates their deactivation.

In a similar way [22] uses formal verification techniques based on graph transformations applied to a "visual language" to reason about the independence of simulation and reconfiguration steps in a reconfigurable system (as running example, the paper models a railway system).

[30] addresses the problem of efficient processing of virtualized applications on a cluster. In contrast to the usual static resource reservation scheme, the paper describes a prototype providing mechanisms to efficiently manage a cluster-wide context switch of virtualized jobs. Again, reconfiguration graphs and reconfiguration plans are involved in the context switch mechanism.

[55] [56] respectively present a formal investigation method and a language to describe architectures and complex reconfiguration, the latter mainly focusing on the necessary constructions needed for an effective but formal reconfiguration language.

7.3 Off-line computation of configuration

Automatically partitioning and scheduling a set of tasks is equivalent to multi-dimensional bin-packing, which is NP-hard as shown by [12].

Finding off-line scheduling of real-time tasks on multi-processors is an old problem that has been tackled in several papers. The authors of [57] were among the first to present an algorithm to find a feasible

solution or prove the non feasibility of task sets. [46] defines an optimal branch and bound search algorithm that maps synchronous implicit-deadline dependent task sets on a distributed architecture, taking into account the network delays. [1] extends these results by scheduling also the messages exchanged between the allocated tasks.

Numerous approaches use constraint programming (or SAT/SMT based search) to perform off-line scheduling, e.g. [21, 49, 31, 5, 10, 48, 29, 8, 13, 47]. What has changed since the last years is the capability of the solvers to deal with very large problems. If finding a configuration is quite standard in the literature, dealing with reconfiguration graphs as we do is much more rare.

Lots of works rely on the use of heuristics as they are much more efficient than exhaustive search at the cost of being sub-optimal.

7.4 Off-line computation with reconfiguration

The terms of computation with reconfiguration are often used in a different sense than the GRec one. The reconfiguration in those cases is used in order to reach higher performance at lower cost: the hardware (in general FPGAs) is dynamically reconfigured at each step of a given computation.

[15] describes a reconfigurable parallel architecture – named Modulor–, developed at ONERA in the late eighties, whose interconnection topology could be dynamically modified in order to match the communication characteristics of a given algorithm, according to a pre-computed strategy. The objective of the MODULOR project was to design and build a massively parallel reconfigurable computer, as well as the software tools needed for developing applications that efficiently use reconfiguration potentialities of the architecture. MODULOR software included tools for partitioning and mapping parallel applications on the parallel architecture, each phase corresponding to a different configuration.

[14] addresses the problem of hardware/software co-synthesis of dynamically reconfigurable embedded systems. The goal is to generate dynamically reconfigurable heterogeneous distributed hardware and software architecture meeting real-time constraints while minimizing the system hardware cost. A programming interface allows the efficient reconfiguration of reprogrammable devices. Reconfiguration delays are taken into account such that real-time deadlines are not exceeded. Reconfiguration, in this case, is considered mainly as a way allowing to share hardware resources between functions not simultaneously active. Fault tolerance, however, is also considered, but as an extension. The architecture in this case includes fault detection mechanisms (parity, checksum, address range check...). Unlike GRec, error recovery involves spare processing elements.

In a similar manner, [35] addresses real-time periodic task scheduling on reconfigurable systems. The objective of configurability is roughly the same as above, but in the context of real-time it is necessary to guarantee that the deadline constraints are satisfied, taking into account the overhead introduced by the required reconfigurations. The real-time task scheduling problem is here formulated using Mixed Integer Linear Programming

Among the works dealing with reconfigurable systems, not so much use exhaustive search to precompute configurations. The authors of [50] have formalized the notion of feasible schedule with constraints. During his PhD, Aymen Gammoudi [28] has formalized the reconfiguration problem to compute reconfiguration of heterogeneous multi-core to deal with energy management.

In his PhD, Ali Syed [52] has extended the reconfiguration graphs to allow aperiodic tasks.

7.5 Fault tolerance

The hardware platform targeted by GRec achieves fault tolerance through specific MON tasks (see section 2.2) scheduled in such a way that each one, periodically running on a given core, is able to detect a failure on another core; a reconfiguration process is then performed. Other works, such [43][3][54], use redundant multi-threaded processes in order to detect and recover soft errors. This is done by comparing the results of the different copies of a process. This method allows for the detection of more subtle failures (for example transient failures) than the GRec one, at the cost of a huge loss of processing power, since the same code is executed in parallel several times. Additionally, the comparison requires to take into account the potential asynchronicity between different copies of the same process.

Same redundancy principles can be implemented in hardware, at different degrees depending on specific safety and performance application requirements [37].

8 Conclusion and future work

We have presented GREC, a tool developed during the DREAMS project, based on constraint solving to automatically compute reconfiguration graphs for multi-core distributed platform in order to support the occurrence of permanent core failures. GREC was able to deal with realistic case studies and produce the expected valid results within a reasonable time. We will continue exploring the limiting factors. In particular, we could add the possibility to reuse a partition slot schedule of a failed core, in which case we would have some symmetry and this would greatly improve the exploration. We will improve the tool and methodology in order to leverage some DREAMS choices.

The next perspective concerns the association of failure rate information on the core failure. We would like to:

- assign a core a *failure rate*,
- and only compute reconfigurations down to a depth such that the evaluated global probability for the whole system to be unable to be reconfigured is less than an acceptable level (for example regarding the Development Assurance Levels (DAL) of avionic applications).

The problem would be then to assign a realistic failure rate to the cores of each specific node. Moreover, it could happen that the acceptable DAL level is unreachable.

Acknowledgments

The research leading to these results has received funding from the European project DREAMS under reference n° 610640 and from the ONERA project MAUSART. The authors would like to thank Valentine Bellet for her coding in C.

References

[1] Tarek F. Abdelzaher and Kang G. Shin. Combined task and message scheduling in distributed real-time systems. *IEEE Trans. Parallel Distrib. Syst.*, 10(11):1179–1191, 1999.

[2] Aeronautical Radio Inc. *Avionics Application Software Standard Interface – ARINC Specification 653*, 1997. `https://www.aviation-ia.com/products/653p0-2-avionics-application-software-standard-interface-part-0-overview-arinc-653`.

[3] P. Ashok Kumar and B. Thanasekhar. Fault Tolerance in Multi-Core Processors Using Flexible Redundant Threading. *International Journal of Advanced Computational Engineering and Networking*, 2(7):92–96, July 2014.

[4] Simon Barner, Alexander Diewald, Jörn Migge, Ali Syed, Gerhard Fohler, Madeleine Faugère, and Daniel Gracia Pérez. DREAMS toolchain: Model-driven engineering of mixed-criticality systems. In *20th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS'17)*, pages 259–269, 2017.

[5] Frédéric Boniol, Pierre-Emmanuel Hladik, Claire Pagetti, Frédéric Aspro, and Victor Jégu. A framework for distributing real-time functions. In *Proceedings of the 6th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'08)*, pages 155–169, 2008.

[6] Shekhar Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro*, 25:10–16, 2005.

[7] Vicent Brocal, Miguel Masmano, Ismael Ripoll, Alfons Crespo, Patricia Balbastre, and Jean-Jacques Metge. Xoncrete. In *Proceedings of the 5th Conference on Embedded Real Time Software and Systems (ERTS'10)*, 2010. `http://www.fentiss.com/documents/xoncrete_overview.pdf`.

[8] Alan Burns, Tom Fleming, and Sanjoy K. Baruah. Cyclic executives, multi-core platforms and mixed criticality applications. In *27th Euromicro Conference on Real-Time Systems (ECRTS'15)*, pages 3–12, 2015.

[9] J. Calandrino, H. Leontyev, A. Block, U. Devi, and J. Anderson. LITMUS^{RT}: A Testbed for Empirically Comparing Real-Time Multiprocessor Schedulers. In *27th IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 111–126, Rio de Janeiro, Brazil, December 2006.

[10] Thomas Carle, Manel Djemal, Dumitru Potop-Butucaru, and Robert De Simone. Static mapping of real-time applications onto massively parallel processor arrays. In *14th International Conference on Application of Concurrency to System Design (ACSD'14)*, pages 112–121, 2014.

[11] C.-W. Chang, J.-J. Chen, T.-W. Guo, and H. Falk. Real-time partitioned scheduling on multi-core systems with local and global memories. In *18th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 467–472, Yokohama, Japan, January 2013. IEEE.

[12] E.G. Coffman Jr, M.R. Garey, and D.S. Johnson. Approximation algorithms for bin packing: A survey. In *Approximation algorithms for NP-hard problems*, pages 46–93. PWS Publishing Co., 1996.

[13] Silviu S. Craciunas and Ramon Serna Oliver. Combined task- and network-level scheduling for distributed time-triggered systems. *Real-Time Systems*, 52(2):161–200, 2016.

[14] B. P. Dave. CRUSADE: Hardware/Software Co-Synthesis of Dynamically Reconfigurable Heterogeneous Real-Time Distributed Embedded Systems. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, pages 97–104, Munich, Germany, March 1999. IEEE.

[15] V. David, C. Fraboul, J. Y. Rousselot, and P. Siron. Partitioning and mapping communication graphs on a modular reconfigurable parallel architecture. In L. Bougé, M. Cosnard, Y. Robert, and D. Trystram, editors, *Proceedings of the Second Joint International Conference on Vector and Parallel Processing CONPAR 92—VAPP V*, volume 634 of *Lecture Notes in Computer Science*, pages 43–48, Lyon, France, September 1992. Springer.

[16] M. D. Derk and L. S. DeBrunner. Reconfiguration Graph Grammar for massively parallel, fault tolerant computers. In J. Cuny, H. Ehrig, G. Engels, and G. Rozenberg, editors, *Proceedings of the 5th International Workshop on Graph Grammars and Their Application to Computer Science*, volume 1073 of *Lecture Notes in Computer Science*, pages 185–195, Williamsburg, VA, USA, November 1994. Springer.

[17] DREAMS consortium. DREAMS: Distributed REal-time Architecture for Mixed Criticality Systems, 2013 – 2017. `http://dreams-project.eu`.

[18] DREAMS consortium. DREAMS final integration. `http://www.uni-siegen.de/dreams/publications/2016-12-20-newsletter.pdf`, 2016.

[19] Guy Durrieu, Madeleine Faugère, Sylvain Girbal, Daniel Gracia Pérez, Claire Pagetti, and Wolfgang Puffitsch. Predictable flight management system implementation on a multicore processor. In *Proceedings of the 7th Conference on Embedded Real Time Software and Systems (ERTS'14)*, 2014.

[20] Guy Durrieu, Gerhard Fohler, Gautam Gala, Sylvain Girbal, Daniel Gracia Pérez, Claire Pagetti, and Simara Pérez Zurita. Dreams about reconfiguration and adaptation in avionics. In *Proceedings of the 8th Conference on Embedded Real Time Software and Systems (ERTS'16)*, 2016.

[21] Cecilia Ekelin. *An Optimization Framework for Scheduling of Embedded Real-Time Systems*. PhD thesis, Chalmers University of Technology, 2004.

[22] Claudia Ermel and Karsten Ehrig. Visualization, Simulation and Analysis of Reconfigurable Systems. In A. Schürr, M. Nagl, and A. Zündorf, editors, *Third International Symposiumon Applications of Graph Transformations with Industrial Relevance, AGTIVE 2007*, volume 5088 of *Lecture Notes in Computer Science*, pages 265–280, Kassel, Germany, October 2007. Springer.

[23] M. Fan, Q. Han, G. Quan, and S. Ren. Multi-core partitioned scheduling for fixed-priority periodic real-time tasks with enhanced RBound. In *Fifteenth International Symposium on Quality Electronic Design*, pages 284–291, Santa Clara, CA, USA, March 2014. IEEE.

[24] C. J. Fidge. Real-Time Schedulability Tests for Preemptive Multitasking. *Real-Time Systems*, 14(1):61–93, January 1998.

[25] Gerhard Fohler, Gautam Gala, Daniel Gracia Pérez, and Claire Pagetti. Evaluation of DREAMS resource management solutions on a mixed-critical demonstrator. In *9th European Congress on Embedded Real Time Software and Systems (ERTS'18)*, 2018.

[26] Radio Technical Commission for Aeronautics (RTCA) and EURopean Organisation for Civil Aviation Equipment (EUROCAE). DO-297: Software, electronic, integrated modular avionics (ima) development guidance and certification considerations, 2007. `https://standards.globalspec.com/std/2018378/RTCA%20DO-297`.

[27] Freescale. T4240 QorIQ: Integrated multicore communications processor family reference manual, 2014. `https://www.nxp.com/docs/en/user-guide/T4240RDBUG.pdf`.

[28] Aymen Gammoudi, Daniel Chillet, Mohamed Khalgui, and Adel Benzina. Mapping of periodic tasks in reconfigurable heterogeneous multi-core platforms. In *Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE'18)*, pages 99–110, 2018.

[29] Raul Gorcitz, Emilien Kofman, Thomas Carle, Dumitru Potop-Butucaru, and Robert de Simone. On the scalability of constraint solving for static/off-line real-time scheduling. In *Formal Modeling and Analysis of Timed Systems - 13th International Conference (FORMATS'15)*, pages 108–123, 2015.

[30] Fabien Hermenier, Adrien Lèbre, and Jean-Marc Menaud. Cluster-Wide Context Switch of Virtualized Jobs. Research Report 6929, INRIA, April 2009.

[31] Pierre-Emmanuel Hladik, Hadrien Cambazard, Anne-Marie Déplanche, and Narendra Jussien. Solving a real-time allocation problem with constraint programming. *J. Syst. Softw.*, 81(1):132–149, January 2008.

[32] IBM ILOG CPLEX Optimization Studio. `http://www03.ibm.com/software/products/en/ibmilogcpleoptistud/`, 2017.

[33] D. Isovic and G. Fohler. Quality aware mpeg-2 stream adaptation in resource constrained systems. In *Proceedings. 16th Euromicro Conference on Real-Time Systems, 2004. ECRTS 2004.*, pages 23–32, 2004.

[34] A. Kandhalu, K. Lakshmanan, J. Kim, and R. Rajkumar. pCOMPATS: Period-Compatible Task Allocation and Splitting on Multi-core Processors. In *IEEE 18th Real Time and Embedded Technology and Applications Symposium*, pages 307–316, Beijing, China, April 2012. IEEE.

[35] H. Kooti, E. Bozorgzadeh, S. Liao, and L. Bao. Transition-aware Real-Time Task Scheduling for Reconfigurable Embedded Systems. In *Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pages 232–237, Dresden, Germany, March 2010. IEEE.

[36] Angeliki Kritikakou, Claire Pagetti, Christine Rochange, Matthieu Roy, Madeleine Faugère, Sylvain Girbal, and Daniel Gracia Pérez. Distributed run-time wcet controller for concurrent critical tasks in mixed-critical systems. In *Proceedings of the 22th International Conference on Real-Time and Network Systems (RTNS'14)*, pages 139–148, 2014.

[37] J.H. Lala and R.E. Harper. Architectural Principles for Safety-Critical Real-Time Applications. *Proceedings of the IEEE*, 82(1):25–40, January 1994.

[38] S. Lauzac, R. Melhem, and D. Mossé. An Efficient RMS Admission Control and its Application to Multiprocessor Scheduling. In *Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*, pages 511–518, Orlando, FL, USA, USA, March–April 1998. IEEE.

[39] Yann-Hang Lee, Daeyoung Kim, Mohamed Younis, and Jeff Zhou. Scheduling tool and algorithm for integrated modular avionics systems. In *Proceedings of the 19th Digital Avionics Systems Conference (DASC'00)*, 2000.

[40] C.L. Liu and J.W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *JACM*, 20(1):40–61, 1973.

[41] J. Liu and M. Yang. Task Scheduling of Real-Time Systems on Multi-Core Embedded Processor. In *IEEE International Conference on Intelligent Systems and Knowledge Engineering*, pages 580–583, Hangzhou, China, 2010. IEEE.

[42] Miguel Masmano, Ismael Ripoll, Alfons Crespo, Jean-Jacques Metge, and Paul Arberet. Xtratum: An open source hypervisor for TSP embedded systems in aerospace. In *DASIA 2009. DAta Systems In Aerospace.*, May. Istanbul 2009.

[43] H. Mushtaq, Z. Al-Ars, and K. Bertels. Fault Tolerance on Multicore Processors using Deterministic Multithreading. In *8th IEEE Design and Test Symposium*, Marrakesh, Morocco, December 2013. IEEE.

[44] Claire Pagetti, Julien Forget, Frédéric Boniol, Mikel Cordovilla, and David Lesens. Multi-task implementation of multi-periodic synchronous programs. *Discrete Event Dynamic Systems*, 21(3):307–338, 2011.

[45] Claire Pagetti, David Saussié, Romain Gratia, Eric Noulard, and Pierre Siron. The rosace case study: From simulink specification to multi/many-core execution. In *20th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'14)*, April 2014.

[46] Dar-Tzen Peng, Kang G. Shin, and Tarek F. Abdelzaher. Assignment and scheduling communicating periodic tasks in distributed real-time systems. *IEEE Trans. Software Eng.*, 23(12):745–758, 1997.

[47] Quentin Perret, Pascal Maurère, Éric Noulard, Claire Pagetti, Pascal Sainrat, and Benoît Triquet. Mapping hard real-time applications on many-core processors. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems (RTNS'16)*, pages 235–244, 2016.

[48] Wolfgang Puffitsch, Éric Noulard, and Claire Pagetti. Off-line mapping of multi-rate dependent task sets to many-core platforms. *Real-Time Systems*, 51(5):526–565, 2015.

[49] Klaus Schild and Jörg Würtz. Scheduling of time-triggered real-time systems. *Constraints*, 5(4):335–357, October 2000.

[50] Christoph Steiger, Herbert Walder, Marco Platzner, and Lothar Thiele. Online scheduling and placement of real-time tasks to partially reconfigurable devices. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium*, RTSS '03, pages 224–235, Washington, DC, USA, 2003. IEEE Computer Society.

[51] Wilfried Steiner. An evaluation of smt-based schedule synthesis for time-triggered multi-hop networks. In *Proceedings of the 31st Real-Time Systems Symposium (RTSS'10)*, pages 375–384, 2010.

[52] Ali Abbas Jaffari Syed. *Model-Based Design and Adaptive Scheduling of Distributed Real-Time Systems*. doctoralthesis, Technische Universität Kaiserslautern, 2018.

[53] TTTech. TTEthernet Time-Triggered Ethernet (SAE AS 6802), November 2011. `https://www.sae.org/standards/content/as6802/`.

[54] V. Vargas, P. Ramos, J.-F. Méhaut, and R. Velazco. NMR-MPar: A Fault-Tolerance Approach for Multi-Core and Many-Core Processors. *Applied Sciences*, 8(3), March 2018.

[55] M. Wermelinger and J. L. Fiadeiro. Algebraic software architecture reconfiguration. In *Proceedings of the 7th European software engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of software engineering ESEC/FSE-7*, volume 24 Issue 6 of *ACM SIGSOFT Software Engineering Notes*, pages 393–409, Toulouse, France, September 1999. ACM New York, NY, USA.

[56] M. Wermelinger, A. Lopes, and J. L. Fiadeiro. A graph based architectural (Re)configuration language. In *Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering ESEC/FSE-9*, volume 26 Issue 5 of *ACM SIGSOFT Software Engineering Notes*, pages 21–32, Vienna, Austria, September 2001. ACM New York, NY, USA.

[57] J. Xu. Multiprocessor scheduling of processes with release times, deadlines, precedence, and exclusion relations. *Software Engineering, IEEE Transactions on*, 19(2):139 –154, feb 1993.