A Bi-Criteria Scheduling Heuristic for Distributed Embedded Systems under Reliability and Real-Time Constraints

Ismail Assayad
VERIMAG, 2 av. de Vignate,
38610 Gières, France.
Ismail.Assayad@imag.fr

Alain Girault
INRIA, 655 av. de l'Europe
3833 Saint-Ismier, Cedex - France
Alain.Girault@inrialpes.fr

Hamoudi Kalla
INRIA, 655 av. de l'Europe
3833 Saint-Ismier, Cedex - France
Hamoudi.Kalla@inrialpes.fr

Abstract

Multi-criteria scheduling problems, involving optimization of more than one criterion, are subject to a growing interest. In this paper, we present a new bi-criteria scheduling heuristic for scheduling data-flow graphs of operations onto parallel heterogeneous architectures according to two criteria: first the minimization of the schedule length, and second the maximization of the system reliability. Reliability is defined as the probability that none of the system components will fail while processing. The proposed algorithm is a list scheduling heuristics, based on a bi-criteria compromise function that introduces priority between the operations to be scheduled, and that chooses on what subset of processors they should be scheduled. It uses the active replication of operations to improve the reliability. If the system reliability or the schedule length requirements are not met, then a parameter of the compromise function can be changed and the algorithm re-executed. This process is iterated until both requirements are met.

Keywords: *Distributed real-time systems, safety-critical systems, reliability, multi-criteria scheduling, heterogeneous systems, active software replication.*

1 Introduction

Distributed systems are being increasingly used in critical real-time applications, such as avionics, air traffic control, autopilot systems, and nuclear plant control, in which the consequences of missing a tasks deadline may cause catastrophic loss of money, time, or even human life. This is why such systems require a high reliability. Here, reliability is defined as the probability that none of the system components will fail while processing. For example, a commercial flight-control system requires the probability of a system failure to be approximately 10^{-10}/hour, that is, the system reliability should be approximately 0.999999999 [21].

Our goal is to produce automatically a *reliable distributed static schedule* of a given algorithm onto a given distributed architecture, which satisfies two criteria: maximize the system's reliability and minimize the system's run-time. Concretely, we are given as input a specification of the algorithm to be distributed ($\mathcal{A}lg$), a specification of the target distributed architecture ($\mathcal{A}rc$), some distribution constraints ($\mathcal{D}is$), some information about the execution times of the algorithm blocks on the architecture processors and the communication times of the algorithm data-dependencies on the architecture communication links ($\mathcal{E}xe$), some information about the reliability characteristics of each component of the architecture ($\mathcal{R}el$), a reliability objective ($\mathcal{R}el_{obj}$), and a run-time objective ($\mathcal{R}t_{obj}$). The goal is to build a static schedule of $\mathcal{A}lg$ on $\mathcal{A}rc$, satisfying both objectives $\mathcal{R}el_{obj}$ and $\mathcal{R}t_{obj}$, with respect to $\mathcal{E}xe$, $\mathcal{D}is$, and $\mathcal{R}el$ (see Figure1).

This problem is difficult to solve because the two criteria are *antagonistic*: indeed, the reliability is usually improved by replicating the operations, which has a negative impact on the schedule length, and hence on the system's run-time.

The majority of hard real-time distributed systems in the literature do not attempt to introduce reliability; rather, they concentrate on the problems that arise from tasks deadline assuming a reliable hardware. For example, the heuristics proposed in [2, 15, 8, 17] are based on static or dynamic allocation and scheduling of tasks to minimize the schedule length. But none of these scheduling heuristics attempt to improve the system's reliability.

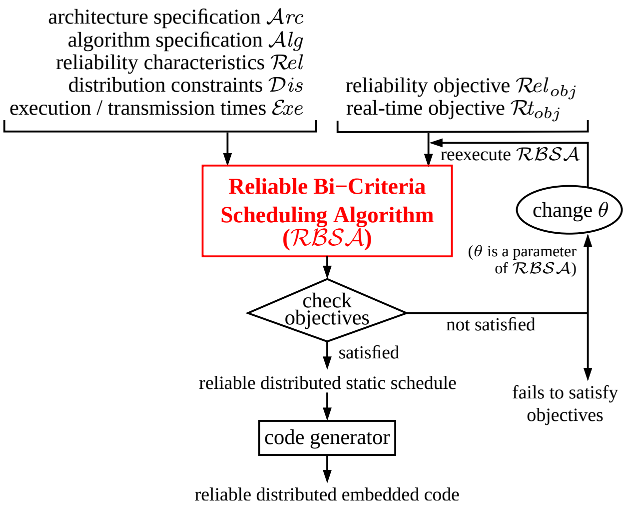

Figure 1. Our methodology to generate reliable distributed code.

To maximize the system's reliability in the task allocation problem, the authors of [21, 14, 22, 13] give an explicit reliability expression in terms of system parameters. This expression is used to drive theirs algorithms in search for an allocation that maximizes the reliability. In [21], Shatz et al. present a task allocation model for reliability; failures from processors and communication links are considered to measure the system's reliability of the proposed algorithm. In [14], Kartik et al. present an improved version of Shatz et al. algorithm, which improves the system's reliability. In [22], Srinivasan et al. present a cluster-based allocation technique to maximize the reliability in heterogeneous systems. However, none of these heuristics attempts to minimize the length of the generated schedule.

In the literature on bi-criteria scheduling problems, only a few articles consider the reliability property [6, 5, 19, 18]. Taking both reliability and tasks deadline into account, Xiao et al. [19] propose a scheduling algorithm, called eFRCD (efficient Fault-tolerant Reliability Cost Driven Algorithm), based on the reliability model of Shatz et al. [21]. Their algorithm uses a primary-backup copy scheme that enables the system to tolerate the permanent failure of any single processor. However, the tasks deadline criterion has advantage over the reliability criterion. Dogan et al. have proposed a bi-criteria list scheduling heuristics with two objectives, minimizing the schedule length and maximizing the reliability of the obtained schedule [6]. Their cost function considers the reliability of different system-components when making decisions to schedule tasks.

The algorithm that we propose to generate a reliable distributed static schedule, called Reliable Bi-Criteria Scheduling Algorithm ($\mathcal{RBSA}$), is different than the ones proposed in [6] and [19] in the sense that we use the *active replication of operations* [10] to improve *both* the system's reliability and the schedule length (and hence the system's run-time). Indeed, even though these two objectives are antagonistic, there are situations where replicating some operations actually *reduces* the schedule length, by improving the *locality* of computations [3].

The paper is organized as follows. Section 2 gives the system models and assumptions. The bi-criteria scheduling problem is presented in Section 3. Section 4 presents the proposed bi-criteria algorithm $\mathcal{RBSA}$. Section 6 details the performances of $\mathcal{RBSA}$. Finally, Section 7 concludes the paper and proposes future research directions.

2 System models and assumptions

2.1 Architecture model

The architecture is modeled by a graph, where each vertex is a processor, and each edge is a communication link. Classically, a processor is made of one computation unit, one local memory, and one or more communication units, each connected to one communication link. Communication units execute data transfers, called `comms`. The chosen communication mechanism is the send/receive [11], where the send operation is non-blocking and the receive operation blocks in the absence of data. Figure 2(b) is an example of architecture graph, with four processors P1, P2, P3, and P4, and four point-to-point communications links L12, L23, L24 and L34.

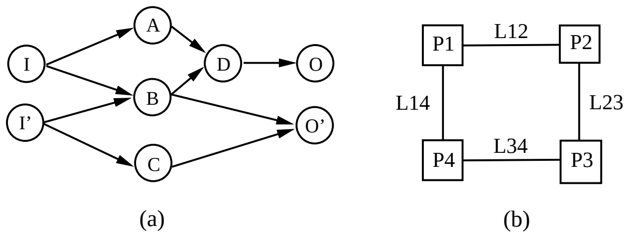

Figure 2. Example of (a) an algorithm graph $\mathcal{A}lg$ and (b) an architecture graph $\mathcal{A}rc$.

2.2 Algorithm model

The algorithm to be distributed is modeled by a *data-flow graph*. Each vertex is an *operation* and each edge is a *data-dependency*. The algorithm is executed repeatedly for each input event from the sensors (operations without predecessors) in order to compute the output events for actuators (operations without successors). This periodic sampled model is commonly used for embedded systems and automatic control systems.

Figure 2(a) is an example of algorithm graph, with eight operations: (I,I') are sensor operations, (O,O') are actuator operations, while (A,B,C,D) are computation operations. The data-dependencies between operations are depicted by arrows. For instance, the data-dependency A $\triangleright$ D corresponds to the sending of some arithmetic result computed by A and needed by D.

2.3 Execution characteristics and distribution constraints

To each operation o of $\mathcal{A}lg$, we associate in a table $\mathcal{E}xe$ its execution time on each processor: each pair $\langle o, p \rangle$ of $\mathcal{E}xe$ is the worst case execution time (WCET) of the operation o on the processor p, expressed in time units. Since the target architecture is heterogeneous, the WCET for a given operation can be distinct on each processor. Similarly, to each data-dependency of $\mathcal{A}lg$, we associate in a table of execution times $\mathcal{E}xe$, its communication times on each communication links: each pair $\langle d, l \rangle$ of $\mathcal{E}xe$ is the worst case transmission time (WCTT) of the data dependency d on the communication link l, again expressed in time units. Since the target architecture is heterogeneous, the WCTT for a given data-dependency can be distinct on each communication link.

For instance, $\mathcal{E}xe$ for $\mathcal{A}lg$ and $\mathcal{A}rc$ of Figure 2 is given in Table 1. The point-to-point links L12, L23, L24 and L34 are heterogeneous. The table only gives the WCTT for *inter-processor* communications. For an *intra-processor* communication, the WCTT is always 0 time unit.

		operation							
	time	I	I'	A	B	C	D	O	O'
proc.	P1	2.5	∞	2.5	3.0	2.0	1.5	3.0	3.0
	P2	1.5	1.5	1.5	2.0	1.0	0.5	2.0	∞
	P3	2.5	∞	2.5	3.0	2.0	1.5	3.0	3.0
	P4	1.5	1.5	1.5	2.0	1.0	0.5	2.0	∞

		data-dependency					
	time	I $\triangleright$ A B $\triangleright$ O'	I $\triangleright$ B B $\triangleright$ D	I' $\triangleright$ B	I' $\triangleright$ C	A $\triangleright$ D	C $\triangleright$ O' D $\triangleright$ O
link	L12	1.0	2.0	1.5	2.0	1.5	1.5
	L23	2.0	4.0	3.0	3.0	4.0	3.0
	L14	1.0	2.0	1.5	2.0	1.5	1.5
	L34	2.0	4.0	3.0	3.0	4.0	3.0

Table 1. Distributed constraints $\mathcal{D}is$ and execution/transmission times $\mathcal{E}xe$ for operations and data-dependencies.

Finally, specifying the distribution constraints $\mathcal{D}is$ amounts to associating the value "∞" to some pairs $\langle o, p \rangle$ of $\mathcal{E}xe$, meaning that o cannot be executed on p (see Table 1).

2.4 Reliability model

We consider only hardware components (processors and communication links) failures and we assume that the algorithm is correct w.r.t. its specification, i.e., it has been formally validated, for instance with model checking and/or theorem proving tools. We assume that the failure of a component has an exponential distribution [21], i.e., it follows a Poisson law with a constant failure rate λ. Furthermore, components failures are assumed to be independent. For instance, Table 2 gives the failure rates of the processors and communication links of the architecture of Figure 2(b).

	processors			communication links	
	P1, P4	P2	P3	L12, L34	L23, L24
λ	$2 * 10^{-6}$	10^{-6}	$3 * 10^{-6}$	$2 * 10^{-5}$	$4 * 10^{-5}$

Table 2. Failure rates for system components

Finally, none of the figures from Tables 1 and 2 derive from an existing real-life example. They are just meant for the sake of the example, but are nontheless realistic w.r.t. current real-time systems.

3 The bi-criteria problem

As said in the introduction, our goal is to find a static schedule of $\mathcal{A}lg$ on $\mathcal{A}rc$, satisfying two criteria: the run-time objective $\mathcal{R}t_{obj}$ and the reliability objective $\mathcal{R}el_{obj}$. In this section, we present in details these two criteria.

3.1 Real-time criterion

As we are targeting distributed real-time systems, we want to obtain a schedule $\mathcal{R}t_{sched}$ that satisfies the run-time objective $\mathcal{R}t_{obj}$, which means that the obtained static distributed schedule $\mathcal{R}t_{sched}$ must complete in less than $\mathcal{R}t_{obj}$ time units. The schedule length $\mathcal{R}t_{sched}$ is computed as follows:

$$\mathcal{R}t_{sched} = \max_{p_j} \left\{ \max_{o_i \ on \ p_j} E(o_i, p_j) \right\}$$

where $E(o_i, p_j)$ is the time at which operation o_i terminates its execution on processor p_j.

For instance, the length of the temporary schedule diagram of Figure 3(b) is 9 time units. In this diagram, each replica o_i^j of an operation o_i is represented by a white box, whose height is proportional to its WCET. Each communication operation $o_i^j o_k^l$ is represented by a gray box, whose height is proportional to its WCTT, and whose ends are bound by two arrows: one from the source operation and one to the destination operation.

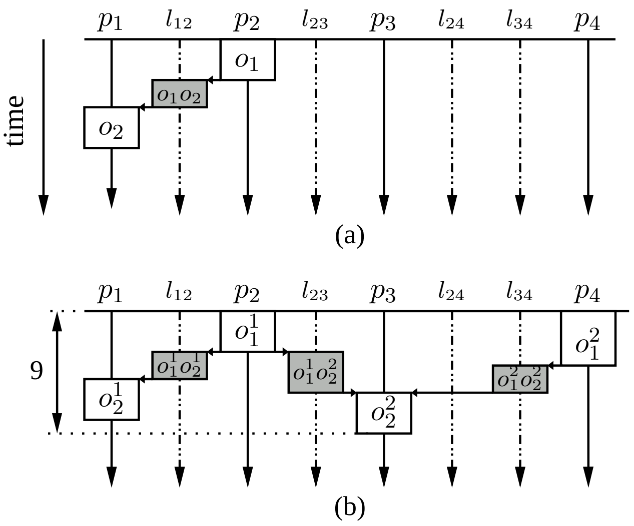

Figure 3. Temporary schedule diagram: (a) without and (b) with replication.

3.2 Reliability criterion

Our second objective is to generate a *reliable* schedule, that is, the system reliability $\mathcal{R}el_{sched}$ must be greater than $\mathcal{R}el_{obj}$. In order to evaluate the overall reliability of a such systems, we propose to use the Reliability Block Diagrams (RBDs) [1, 7], which are well suited for representing and analyzing the reliability of systems with redundancy. An RBD depicts the components in a system and their connections in terms of functioning requirements.

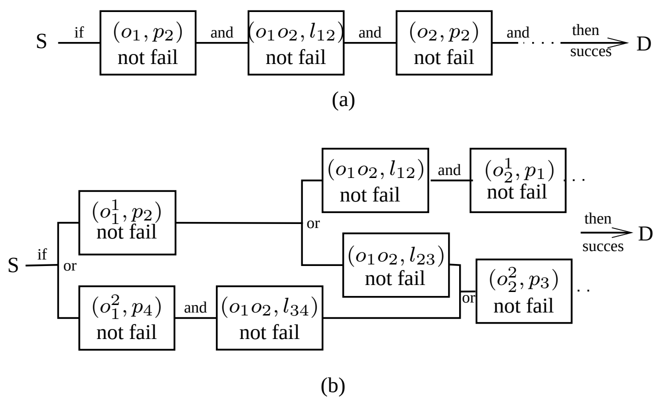

Figure 4. The reliability block diagram: (a) without and (b) with replication.

Figures 4(a) and 4(b) show respectively the RBD corresponding to the schedules of Figures 3(a) and 3(b), with appropriate links and terminals. A system is operational if there is a path from the source S to the destination D in its RBD. In our model, each component $\{o_i, c_j\}$ of an RBD is assigned the reliability cost of executing the operation/communication o_i on the processor/link c_j.

The system reliability of an RBD is based on the reliability of each of its components. The computation of the component reliability $\mathcal{R}el_{sched}(o_i, c_j)$ value is given by the following equation [21]:

$$\mathcal{R}el_{sched}(o_i, c_j) = e^{-\lambda_{c_j}\, \mathcal{E}xe(o_i, c_j)} \tag{1}$$

To compute the overall reliability $\mathcal{R}el^*_{sched}$ of a system, we start by drawing the RBD of its final distributed schedule. Then, using Equation (1), we compute the reliability of the overall system as follows:

- In systems without replication, the RBD of the schedule has a serial structure (see Figure 4(a)); its reliability can therefore be obtained in linear time by multiplying the reliability of each component of the RBD.

- In systems with replication, the RBD of the schedule does not have a serial/parallel structure (see Figure 4(b)); its exact reliability can only be obtained in *exponential time*. However, we can compute an *upper bound* of the reliability $\mathcal{R}el_{sched}$ in *polynomial time*, thanks to the Minimal Cut Sets ($\mathcal{M}cs$) method [4]. The $\mathcal{M}cs$ is the minimum combination of failures that might cause a system to fail. When processors/links failures are assumed to be independent, the reliability of an $\mathcal{M}cs$ M_i is computed as follows:

$$\mathcal{R}el_{sched}(M_i) = 1 - \prod_{(o,c)\,\in\, M_i} (1 - \mathcal{R}el_{sched}(o, c))$$

  Since cut structures operate in series and components in a cut set operate in parallel, the $\mathcal{M}cs$ allows us to compute the upper bound of the system's reliability in a linear time, as follows:

$$\mathcal{R}el^*_{sched} \leq \prod_{i=1}^{k} \Big(1 - \prod_{(o,c)\,\in\, M_i} (1 - \mathcal{R}el_{sched}(o, c))\Big)$$

4 The reliable bi-criteria scheduling algorithm $\mathcal{RBSA}$

We now present our scheduling algorithm $\mathcal{RBSA}$ for maximizing the system's reliability ($\mathcal{R}el_{sched}$) and minimizing the system's run-time ($\mathcal{R}t_{sched}$). We present the algorithm in macro-steps; the superscript number in parentheses refers to the step of the algorithm, e.g., $O^{(n)}_{sched}$. First, we introduce the following notations:

- $O_{cand}^{(n)}$ is the list of *candidate* operations, built from the algorithm graph vertices. An operation is candidate if all its predecessors are already scheduled.
- $O_{sched}^{(n)}$ is the list of already *scheduled* operations.
- $pred(o_i)$ is the set of predecessors of operation o_i.
- $succ(o_i)$ is the set of successors of operation o_i.
- $\mathcal{P}$ is the set of all processors of $\mathcal{A}rc$.
- $2^{\mathcal{P}}$ is the set of combinations of processors of $\mathcal{P}$.
- $\mathcal{R}t_{sched}^{(n-1)}$ is the length of the temporary schedule at step $n-1$.
- $\mathcal{R}t_{sched}^{(n)}(o_i, \{p_1, ..., p_j\})$ is the length of the temporary schedule at step n where the j replicas $o_i^1, \ldots, o_i^j$ of o_i are scheduled respectively on the j processors $p_1, \ldots, p_j$.
- $\mathcal{R}el_{sched}^{(n-1)}$ is the reliability of the temporary schedule at step $n-1$.
- $\mathcal{R}el_{sched}^{(n)}(o_i, \{p_1, ..., p_j\})$ is the reliability of the temporary schedule at step n where the j replicas $o_i^1, \ldots, o_i^j$ of o_i are scheduled respectively on the j processors $p_1, \ldots, p_j$.

4.1 Algorithm principles

The algorithm that we propose is a greedy list scheduling heuristic [23], called $\mathcal{RBSA}$ (Reliable Bi-Criteria Scheduling Algorithm), which uses a *bi-criteria compromise function* ($\mathcal{B}cf$) as a cost function to introduce priority between operations to be scheduled. It is based on two functions: the *reliability loss* ($\mathcal{L}$) and the *schedule length gain* ($\mathcal{G}$). The first function $\mathcal{L}^{(n)}(o_i, \{p_1, \ldots, p_j\})$ computes, at each step n of the algorithm, the loss on reliability resulting from the scheduling of the j replicas $o_i^1, \ldots, o_i^j$ of o_i respectively on the j processors $p_1, \ldots, p_j$ (Figure 5):

$$\mathcal{L}^{(n)} = \frac{\mathcal{R}el_{sched}^{(n)}(o_i, \{p_1, \ldots, p_j\}) - \mathcal{R}el_{sched}^{(n-1)}}{\mathcal{R}el_{obj} - \mathcal{R}el_{sched}^{(n-1)}} \quad (2)$$

The second function $\mathcal{G}^{(n)}(o_i, \{p_1, \ldots, p_j\})$ computes, at each step n of the algorithm, the gain on the schedule length resulting from the scheduling of the j replicas $o_i^1, \ldots, o_i^j$ of o_i respectively on the j processors $p_1, \ldots, p_j$ (Figure 5):

$$\mathcal{G}^{(n)} = \frac{\mathcal{R}t_{sched}^{(n)}(o_i, \{p_1, \ldots, p_j\}) - \mathcal{R}t_{sched}^{(n-1)}}{\mathcal{R}t_{obj} - \mathcal{R}t_{sched}^{(n-1)}} \quad (3)$$

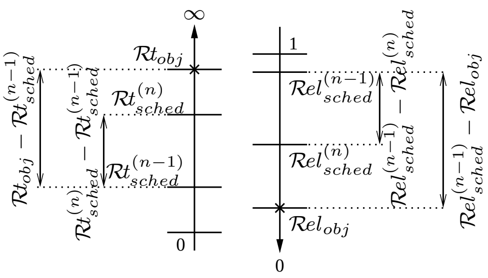

Figure 5. Reliability and run-time objectives

The cost function $\mathcal{B}cf$ computes the bi-criteria compromise value between $\mathcal{L}$ and $\mathcal{G}$; it tries to minimize the loss on reliability and maximize the gain on schedule length by replicating each operation o_i on a subset of P. It selects, for each operation o_i, the best subset $\{p_1, \ldots, p_j\}$ which gives the smallest compromise value $\mathcal{B}cf^{(n)}(o_i, \{p_1, \ldots, p_j\})$ between $\mathcal{L}^{(n)}(o_i, \{p_1, \ldots, p_j\})$ and $\mathcal{G}^{(n)}(o_i, \{p_1, \ldots, p_j\})$. To compute $\mathcal{B}cf$, we introduce a parameter θ (provided by the user and set to 45° by default):

$$\mathcal{B}cf^{(n)} = \cos(\theta)\mathcal{L}^{(n)} + \sin(\theta)\mathcal{G}^{(n)} \quad (4)$$

Here lies the advantage of having normalized both objectives within their respective functions $\mathcal{L}$ and $\mathcal{G}$: we can combine them inside the compromise function $\mathcal{B}cf$. Otherwise, the reliability being intrinsically in the interval [0,1], while the schedule length can be several orders of magnitude greater, it would have been meaningless to compare them. Actually, this would have resulted in giving much more weight to $\mathcal{R}t_{obj}$ than to $\mathcal{R}el_{obj}$.

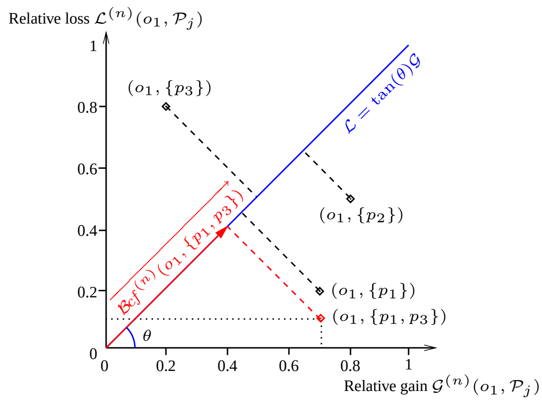

Figure 6. Selection of the best choice with $\mathcal{B}cf$.

Figure 6 illustrates the $\mathcal{B}cf$ computation for operation o_1. We first compute the set $2^{\mathcal{P}}$ of all combinations of all processors of $\mathcal{P}$. We then compute, for each set $P_i \in 2^{\mathcal{P}}$, the compromise value $\mathcal{B}cf^{(n)}(o_1, P_i)$. Graphically, it is the length of the $\overrightarrow{\mathcal{B}cf^{(n)}(o_1, P_i)}$ vector, whose end is the orthogonal projection of the point $(\mathcal{G}(o_1, P_i), \mathcal{L}(o_1, P_i))$ onto the line $\mathcal{L} = \tan(\theta)\mathcal{G}$ (with $\theta = 45°$ here). In the present case, the best compromise value is reached for $P_i = \{p_1, p_3\}$. As a consequence, o_1 is replicated onto two processors, p_1 and p_3. In general, replicating an operation maximizes the system reliability [20] and minimizes the schedule length [3].

4.2 Our scheduling algorithm $\mathcal{RBSA}$

The $\mathcal{RBSA}$ scheduling algorithm is shown in Figure 7.

Algorithm $\mathcal{RBSA}$**:**
input: $\mathcal{A}lg$, $\mathcal{A}rc$, $\mathcal{E}xe$, $\mathcal{D}is$, $\mathcal{R}el$, $\mathcal{R}el_{obj}$, $\mathcal{R}t_{obj}$, and θ;
output: a reliable distributed static schedule of $\mathcal{A}lg$ on $\mathcal{A}rc$ satisfying $\mathcal{R}el_{obj}$ and $\mathcal{R}t_{obj}$, or a fails message;
begin
Compute the set $2^{\mathcal{P}}$ of all combinations of processors of $\mathcal{P}$;
/* the user can limit the degree k of processor combinations */
Initialize the lists of candidate and scheduled operations:
n := 0;
$O^{(0)}_{cand} := \{o \in O \mid pred(o) = \emptyset\}$;
$O^{(0)}_{sched} := \emptyset$;
while $O^{(n)}_{cand} \neq \emptyset$ **do**

① For each candidate operation o_{cand}, compute $\mathcal{B}cf^{(n)}$ on each set P_k of $2^{\mathcal{P}}$:
$$\mathcal{B}cf^{(n)}(o_{cand}, P_k) := \cos(\theta)\mathcal{L}^{(n)}(o_{cand}, P_k) + \sin(\theta)\mathcal{G}^{(n)}(o_{cand}, P_k)$$

② For each candidate operation o_{cand}, select the best set $P^{o_{cand}}_{best}$ such that:
$$\mathcal{B}cf^{(n)}(o_{cand}, P^{o_{cand}}_{best}) := \min_k \mathcal{B}cf^{(n)}(o_{cand}, P_k)$$

③ Select the most urgent candidate operation o_{urgent} between all o^i_{cand} of $O^{(n)}_{cand}$ such that:
$$\mathcal{B}cf^{(n)}(o_{urgent}, P^{o_{urgent}}_{best}) := \max_i \mathcal{B}cf^{(n)}(o^i_{cand}, P^{o^i_{cand}}_{best})$$

④ Schedule actively each replica of o_{urgent} on each processor of $P^{o_{urgent}}_{best}$; the implied communications are also implemented actively on the communications links;

⑤ Compute the new values $\mathcal{R}el_{sched}$ and $\mathcal{R}t_{sched}$;

⑥ **if** $(\mathcal{R}el_{sched} < \mathcal{R}el_{obj})$ **or** $(\mathcal{R}t_{sched} > \mathcal{R}t_{obj})$
then return "fails to satisfy objectives" /* the user can re-execute the algorithm by changing θ or the objectives */

⑦ Update the lists of candidate and scheduled operations:
$O^{(n)}_{sched} := O^{(n-1)}_{sched} \cup \{o_{urgent}\}$;
$O^{(n+1)}_{cand} := O^{(n)}_{cand} - \{o_{urgent}\} \cup$
$\{o' \in succ(o_{urgent}) \mid pred(o') \subseteq O^{(n)}_{sched}\}$;

⑧ n := n + 1;

end while
end

Figure 7. The $\mathcal{RBSA}$ scheduling algorithm.

Initially, $O^{(0)}_{sched}$ is empty and $O^{(0)}_{cand}$ is the list of operations without any predecessors. At the n-th step, these lists are updated according to the data-dependencies of $\mathcal{A}lg$.

At each step n, one operation o_{cand} of the list $O^{(n)}_{cand}$ is selected to be scheduled on at least one processor. To select an operation, we select at the micro-steps ① and ②, for each candidate operation o_{cand}, the set $P^{o_{cand}}_{best}$ of processors having the smallest bi-criteria compromise value. Then, among those best pairs $\langle o_{cand}, P^{o_{cand}}_{best}\rangle$, we select at the micro-step ③ the one having the biggest $\mathcal{B}cf$ value, i.e., the most urgent pair $\langle o_{urgent}, P^{o_{urgent}}_{best}\rangle$.

The selected operation o_{urgent} is replicated and implemented actively at the micro-step ④ on each processor of $P^{o_{urgent}}_{best}$ computed at micro-step ②, and the communications implied by these implementations are also implemented actively on communications links. When a communication operation is generated, it is assigned to the set of communication units bound to the fastest communication medium connecting the processors executing the source and destination operations.

Finally, we check at the micro-step ⑥ if the two objectives $\mathcal{R}el_{obj}$ and $\mathcal{R}t_{obj}$ are satisfied or not. If they are not, the user can change θ or the objectives and re-execute the algorithm.

4.3 An example

Figure 8 shows the final reliable schedule produced by $\mathcal{RBSA}$ with θ = 45° for the graphs $\mathcal{A}lg$ and $\mathcal{A}rc$ of Figure 2. The objectives taken for running $\mathcal{RBSA}$ were $\mathcal{R}t_{obj}$ = 16 and $\mathcal{R}el_{obj}$ = 0.999997. The results obtained by $\mathcal{RBSA}$ are $\mathcal{R}t_{final}$ = 13 and $\mathcal{R}el_{final}$ = 0.9999991.

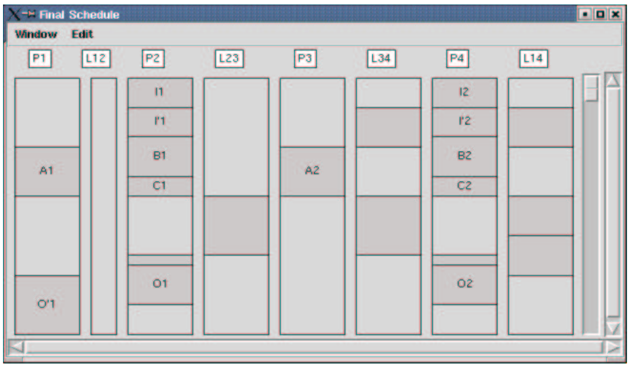

Figure 8. The final reliable schedule produced by $\mathcal{RBSA}$ with θ = 45°.

4.4 Run-time behavior

In order to give the same weight to $\mathcal{L}$ and $\mathcal{G}$ in the $\mathcal{B}cf$ computation, we first take $\theta = 45°$. If the $\mathcal{R}el_{obj}$ or $\mathcal{R}t_{obj}$

requirements are not met at the micro-step ⑤, then the user can refer to Table 3 to change θ, $\mathcal{R}el_{obj}$, or $\mathcal{R}t_{obj}$, and re-execute $\mathcal{RBSA}$ until both requirements are met.

	Reliability and run-time objectives	
$\mathcal{RBSA}$ output	$\overline{\mathcal{R}el_{obj}}$ and $\mathcal{R}t_{obj}$	$\mathcal{R}el_{obj}$ and $\overline{\mathcal{R}t_{obj}}$
user action	$\theta_{new} \in [\theta, 90°]$; re-execute $\mathcal{RBSA}$;	$\theta_{new} \in [0°, \theta]$; re-execute $\mathcal{RBSA}$;
$\mathcal{RBSA}$ output	$\overline{\mathcal{R}el_{obj}}$ and $\overline{\mathcal{R}t_{obj}}$	$\mathcal{R}el_{obj}$ and $\mathcal{R}t_{obj}$
user action	change $\mathcal{R}el_{obj}$, $\mathcal{R}t_{obj}$, and/or θ; re-execute $\mathcal{RBSA}$;	generate reliable distributed code;

$\overline{\mathcal{R}el_{obj}}$ (resp. $\overline{\mathcal{R}t_{obj}}$) means that $\mathcal{R}el_{obj}$ (resp. $\mathcal{R}t_{obj}$) is not satisfied

Table 3. Re-execution strategy for $\mathcal{RBSA}$.

Four cases can arise:

1. If $\mathcal{R}el_{obj}$ is not met, then we re-execute $\mathcal{RBSA}$ with a new θ: here, we take $\theta_{new} \in [\theta, 90°]$, meaning that $\mathcal{L}$ will have more weight than $\mathcal{G}$ in $\mathcal{B}cf$.

2. If $\mathcal{R}t_{obj}$ is not met, then we re-execute $\mathcal{RBSA}$ by changing θ. We take $\theta_{new} \in [0°, \theta]$, meaning that $\mathcal{L}$ will have less weight than $\mathcal{G}$ in $\mathcal{B}cf$ when we re-execute $\mathcal{RBSA}$. For instance, in Figure 9, we decrement θ; so $\{p_3\}$ become the best replication set for operation o_1, instead of $\{p_1, p_3\}$ previously; as a consequence, o_1 is not replicated in the new schedule.

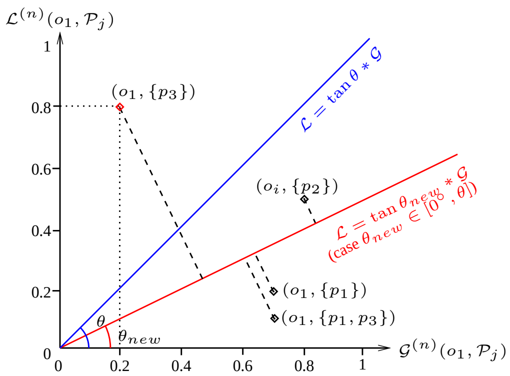

Figure 9. Changing the bi-criteria compromise parameters.

3. If none of the criteria are satisfied, then the user has to loosen at least one of the two objectives $\mathcal{R}el_{obj}$ or $\mathcal{R}t_{obj}$ before re-executing $\mathcal{RBSA}$. He/she can also change θ.

4. If both criteria are satisfied, then the user can generate executable code from the final schedule, for instance by using the SynDEx tool [16, 9].

Finally, remember that for complexity reasons, we compute in $\mathcal{B}cf$ only the upper bound of the partial schedule's reliability. Hence, once we have obtained the final schedule, we compute its *exact* reliability, which we compare to $\mathcal{R}el_{obj}$.

5 $\mathcal{RBSA}$ time complexity

We compute the time complexity of $\mathcal{RBSA}$ as follows. Among the micro-steps ① to ⑧, the dominant one is ①. The computational complexity of the combinations of processors of $\mathcal{P}$ is $\mathcal{O}(m^{k+1})$, where m is the number of processors in $\mathcal{A}rc$ and k is the degree of maximum processor combinations. Thus, the time complexity of micro-step ① is $\mathcal{O}(Nm^{k+1})$, where N is the number of operations in $\mathcal{A}lg$. Thus, for n iterations the overall time complexity is $\mathcal{O}(nNm^{k+1})$. Finally, since exactly one operation is replicated and scheduled at each iteration, $n = N$, and the total time complexity is therefore $\mathcal{O}(N^2m^{k+1})$.

6 Performance evaluation

6.1 Simulation parameters

To evaluate $\mathcal{RBSA}$, we have compared its performances with our previous algorithm proposed in [8], called FTBAR (Fault-Tolerance Based Active Replication), and with the algorithm proposed by Hashimoto in [12], called HBP (Height-Based Partitioning). HPB actively replicates all operations once, therefore producing schedules that tolerate one processor failure, while FTBAR actively replicates all operations a fixed number of times, say n, therefore producing schedules that tolerate $n - 1$ processor failures. We have implemented all three algorithms within the SynDEx tool [16, 9]. SynDEx generates automatically executable fault-tolerant distributed code, by first producing a static fault-tolerant distributed schedule of a given algorithm on a given distributed architecture (either with FTBAR, HBP, or $\mathcal{RBSA}$), and then by generating sthe real-time fault-tolerant distributed executive implementing this schedule.

The performance comparisons were done in two ways and with various parameters: first $\mathcal{RBSA}$ with $\theta = 0°$ against FTBAR and HBP without any replication of operation, then $\mathcal{RBSA}$ with $\theta = 45°$ against FTBAR and HBP with exactly one replication of each operation. At each run, the $\mathcal{R}el_{obj}$ and $\mathcal{R}t_{obj}$ objectives given to $\mathcal{RBSA}$ were computed on the final schedule produced by FTBAR.

The random algorithm graphs were generated as follows: given the number of operations N, we randomly generate a set of levels with a random number of operations. Then, operations at a given level are randomly connected to operations at a higher level. The WCET of each operation are randomly selected from a uniform distribution with the mean equal to the chosen average execution time. Similarly, the WCTT of each data dependency are randomly selected from a uniform distribution with the mean equal to the chosen average communication time. For generating the complete set of algorithm graphs, we have varied two parameters: N=25, 50, 75, 100, and the Communication-to-Computation Ratio CCR=0.1, 1, and 10, defined as the average communication time divided by the average computation time.

6.2 Performance of $\mathcal{RBSA}$ against HBP and FTBAR for $\theta = 0°$

In this simulation, the architecture graph $\mathcal{A}rc$ was a fully connected network of 4 processors, with the failure rates of processors and communications links given in Table 2. For each schedule, we have computed the normalized schedule length (NSL), obtained by dividing the output schedule length by the sum of the computation costs on the critical-path of each graph [3]. Thus, we have compared the average NSL produced by $\mathcal{RBSA}$ with those produced by FTBAR and HBP, averaged over 50 random $\mathcal{A}lg$ graphs. To make the comparison fair, FTBAR and HBP were run without any replication of operation.

In Figure 10, we have plotted the average NSL as a function of CCR, for N=100 operations. $\mathcal{RBSA}$ was run with $\theta = 0°$, meaning that only $\mathcal{R}t_{obj}$ was taken into account as objective (i.e., no reliability objective).

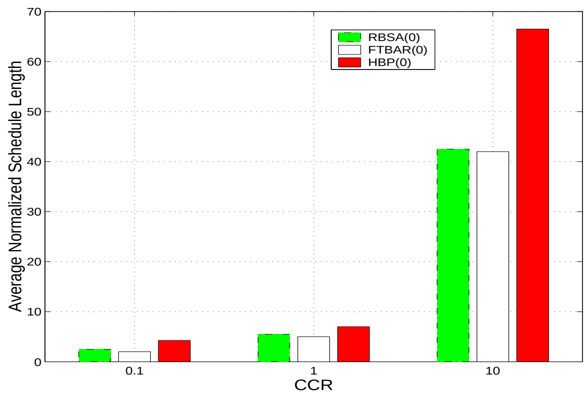

Figure 10. Average NSLs for $\theta = 0°$ and $N = 100$ operations.

We note that when CCR increases, so does the NSL, due to a greater communication cost. For small values of CCR, the three algorithms bear almost similar results, $\mathcal{RBSA}$ and FTBAR being slightly better than HBP. For CCR=10, there is little difference between the performance of $\mathcal{RBSA}$ and FTBAR, and both outperform significantly HBP. Hence, since FTBAR is only very slightly better than $\mathcal{RBSA}$, we think that the latter can be used directly for minimizing the schedule length, provided that $\theta = 0°$.

6.3 Performance of $\mathcal{RBSA}$ against HBP and FTBAR for $\theta = 45°$

In this simulation, the architecture graph was a fully connected network of 6 processors. This time, $\mathcal{RBSA}$ was run with $\theta = 45°$, meaning with an equal weight of the reliability and the schedule length. FTBAR and HBP were both required to replicate actively each operation exactly once.

In Figures 11 and 12, we have plotted respectively the average NSL and the average reliability as a function of CCR, for N=100 operations.

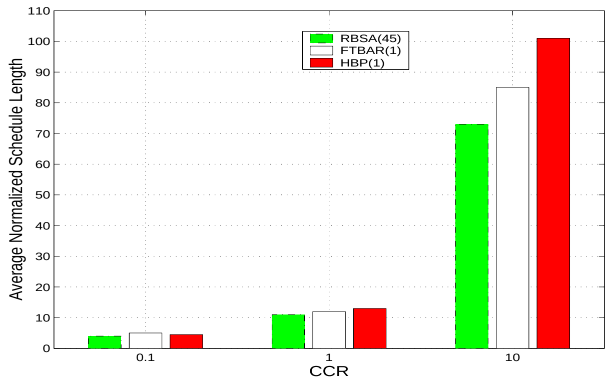

Figure 11. Average NSLs for $\theta = 45°$ and $N = 100$ operations.

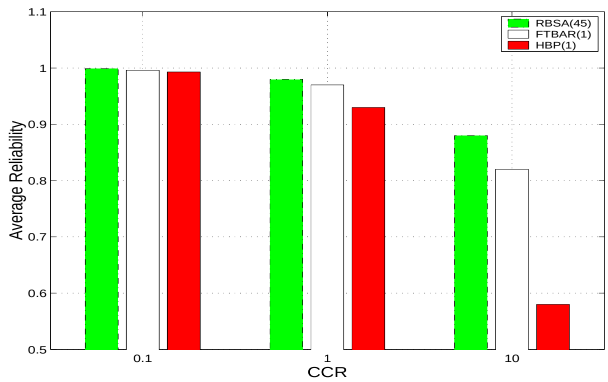

Figure 12. Average reliability for $\theta = 45°$ and $N = 100$ operations.

For CCR=0.1, all three algorithms have similar performances. For CCR=1, $\mathcal{RBSA}$, FTBAR and HBP are similar for the NSL, while HBP is slightly less efficient for the reliability. However, for CCR=10, $\mathcal{RBSA}$ outperforms significantly FTBAR and HBP both for the NSL and the reliability. This is due to the fact that we use the active replication of $\mathcal{A}lg$'s operations, not only to improve the system's reliability, but also to improve the locality of computations and hence the schedule length [3]; not surprisingly, this has more influence when CCR=10 because communications are more expensive compared to computations. Our results indicates that the bi-criteria heuristics of $\mathcal{RBSA}$ can meet its two requirements, and still outperform other existing single-criterion heuristics.

In order to study the impact of $\mathcal{A}lg$'s size on our algorithm, we have applied $\mathcal{RBSA}$, FTBAR and HBP to four sets of 60 randoms graphs, respectively with N=25, 50, 75, and 100 operations. Then, we have plotted in Figures 13 and 14 respectively the average NSL and the average reliability as a function of N, for CCR=1.

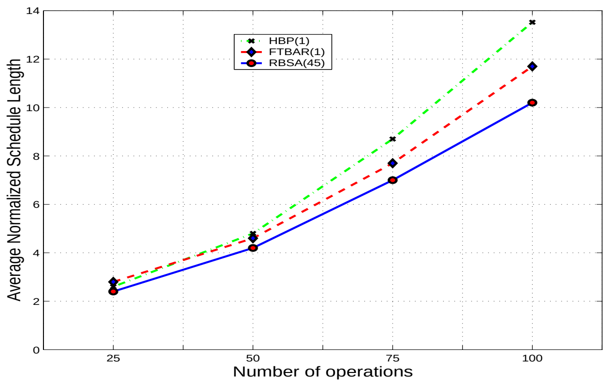

Figure 13. Average NSLs for $\theta = 45°$ and $CCR = 1$.

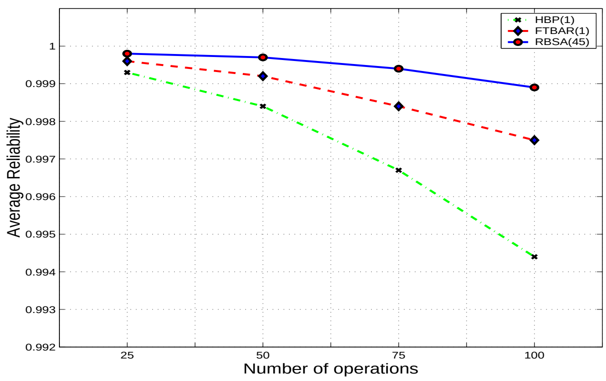

Figure 14. Average reliability for $\theta = 45°$ and $CCR = 1$.

Again, we see that $\mathcal{RBSA}$ outperforms both FTBAR and HBP, and that this effect becomes greater when N increases.

7 Conclusion and future work

We have proposed a new bi-criteria scheduling heuristic, called $\mathcal{RBSA}$ (Reliable Bi-Criteria Scheduling Algorithm), that produces automatically a reliable static distributed schedule of a given algorithm $\mathcal{A}lg$ on a given distributed architecture $\mathcal{A}rc$ according to two criteria: maximizing the system's reliability and minimizing the system's run-time. The problem is that these two criteria are antagonistic: maximizing the reliability requires to replicate the operations of $\mathcal{A}lg$ onto several processors of $\mathcal{A}rc$, but this penalizes the run-time. Conversely, scheduling each operation exactly once minimizes the run-time but does not improve the reliability.

Our solution is based a the bi-criteria compromise function, called $\mathcal{B}cf$, which normalizes both criteria w.r.t. the objectives given by the user, and chooses, for each operation of $\mathcal{A}lg$, the subset of processors of $\mathcal{A}rc$ such that replicating this operation onto the processors of this subset maximizes the reliability and minimizes the run-time. $\mathcal{B}cf$ uses a parameter $\theta \in [0, 90°]$, provided by the user, which gives more weight either to the reliability objective if it is greater than 45°, or to the run-time objective otherwise.

Our algorithm can be re-executed if the system's reliability or run-time objective is not met, by changing the θ parameter of $\mathcal{B}cf$, until both objectives are met.

The experimental results show that $\mathcal{RBSA}$ algorithm slightly outperforms other scheduling algorithms with replication on both criteria. The two algorithms taken for comparison duplicate each operation of $\mathcal{A}lg$, and schedule both replica onto two distinct processors of $\mathcal{A}rc$, therefore achieving a tolerance of exactly one processor failure in the system. Instead of replicating brutally each operation of $\mathcal{A}lg$, $\mathcal{RBSA}$ chooses the best subset of processors of $\mathcal{A}rc$ (possibly a one-element subset) onto which scheduling this operation, in order to optimize both criteria.

Currently, we are working on new solutions to introduce some backtracking into the heuristics to avoid re-executing entirely the algorithm when one objective is not met.

References

[1] A. Abd-allah. Extending reliability block diagrams to software architectures. Technical report, Center for software engineering, computer science department, university of southern california, Los Angeles, CA 90089 USA, 1997.

[2] I. Ahmad, Y. Kwok, and M. Wu. Performance comparison of algorithms for static scheduling of dags to multiprocessors. In *Proceedings of the 2nd Australian Conference on Parallel and Real-Time Systems*, pages 185–192, Sep 1995.

[3] I. Ahmad and Y.-K. Kwok. On exploiting task duplication in parallel program scheduling. In *IEEE Transactions on Parallel and Distributed Systems*, volume 9, pages 872–892, September 1998.

[4] Y. Chen and M. Yuang. A cut-based method for terminal-pair reliability. In *IEEE Trans. Reliability*, pages 413–416, september 1996.

[5] A. Dogan and F. Özgüner. Optimal and suboptimal reliable scheduling of precedence-constrained tasks in heterogeneous distributed computing. In *Proceedings of the 2000 International Conference on Parallel Processeing (ICPP00-Workshops)*, Toronto, Canada, August 2000.

[6] A. Dogan and F. Özgüner. Reliable matching and scheduling of precedence-constrained tasks in heterogeneous distributed computing. In *Proceedings of the 2000 International Workshops on Parallel Processeing (ICPP00)*, Toronto, Canada, August 2000.

[7] K. D. Figiel and D. R. Sule. A generalized reliability block diagram (rbd) simulation. In *Proceedings of the 22nd conference on Winter simulation*, pages 551 – 556, New Orleans, Louisiana, United States, 1990.

[8] A. Girault, H. Kalla, M. Sighireanu, and Y. Sorel. An algorithm for automatically obtaining distributed and fault-tolerant static schedule. In *The International Conference on Dependable Systems and Networks*, San Francisco, California, USA, June 2003.

[9] T. Grandpierre and Y. Sorel. From algorithm and architecture specifications to automatic generation of distributed real-time executives: a seamless flow of graphs transformations. In *MEMOCODE'2003, Formal Methods and Models for Codesign Conference*, Mont Saint-Michel, France, June 2003.

[10] R. Guerraoui and A. Schiper. Fault-tolerance by replication in distributed systems. In *Proceeding Conference on Reliable Software Technologies*, pages 38–57. Springer-Verlag, 1996.

[11] M. Gupta and E. Schonberg. Static analysis to reduce synchronization cost in data-parallel programs. In *23rd Symposium on Principles of Programming Languages*, pages 322–332, january 1996.

[12] K. Hashimoto, T. Tsuchiya, and T. Kikuno. Effective scheduling of duplicated tasks for fault-tolerance in multiprocessor systems. *IEICE Transactions on Information and Systems*, E85-D(3):525–534, march 2002.

[13] S. Kartik and C. S. R. Murthy. Improved task allocation algorithms to maximize reliability of redundant distributed computing systems. *IEEE Transactions On Reliability*, VOL. 44(NO. 4 DECEMBER), 1995.

[14] S. Kartik and C. S. R. Murthy. Task allocation algorithms for maximizing reliability of distributed computing systems. *IEEE Transactions On Computers*, VOL. 41(NO. 9 September), 1997.

[15] Y.-K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys*, 31(4):406–471, 199.

[16] C. Lavarenne, O. Seghrouchni, Y. Sorel, and M. Sorine. The SYNDEX software environment for real-time distributed systems design and implementation. In *European Control Conference*, volume 2, pages 1684–1689. Hermès, July 1991.

[17] G. Manimaran and C. S. R. Murthy. A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis. *IEEE Transactions on Parallel and Distributed Systems*, 9(11):1137–1151, november 1998.

[18] X. Qin and H. Jiang. Dynamic, reliability-driven scheduling of parallel real-time jobs in heterogeneous systems. In *Proceedings of the 30th International Conference on Parallel Processing (ICPP 2001*, pages 113–122, Valencia, Spain, September 2001.

[19] X. Qin, H. Jiang, and D. R. Swanson. An efficient fault-tolerant scheduling algorithm for real-time tasks with precedence constraints in heterogeneous systems. In *Proceedings of the 31th International Conference on Parallel Processing (ICPP 2002)*, pages 360–386, Vancouver, British Columbia, Canada, August 2002.

[20] R. A. Sahner and K. S. Trivedi. A hierarchical, combinatorial-markov method of solving complex reliability models. In *Proceedings of the Fall Joint Computer Conference*, pages 817–825, 1986.

[21] S. Shatz, J. Wang, and M. Goto. Task allocation for maximizing reliability of distributed computer systems. In *IEEE Trans. Computers*, volume 41, pages 156–168, September 1992.

[22] S. Srinivasan and N. Jha. Safety and reliability driven task allocation in distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 10(3):238–251, march 1999.

[23] T. Yang and A. Gerasoulis. List scheduling with and without communication delays. *Parallel Computing*, 19(12):1321–1344, 1993.