Reliability versus performance for critical applications

authors:

Alain Girault (INRIA, POP ART team)

Érik Saule (INPG, LIG, MOAIS team)

Denis Trystram (INPG, LIG, MOAIS team)

contact:

Denis Trystram (denis.trystram@imag.fr)

Laboratoire d'Informatique de Grenoble

ENSIMAG - antenne de Montbonnot

ZIRST 51, avenue Jean Kuntzmann

38330 MONTBONNOT SAINT MARTIN

France

tel:+33 (0)4 76 61 20 79

fax:+33 (0)4 76 61 20 99

Reliability versus performance for critical applications

Abstract: Applications implemented on critical systems are subject to safety critical constraint and are represented by task graphs. This paper considers transient faults on processors with constant failure rates. Reliability is increased by replicating adequate tasks onto well chosen processors. We propose a method for optimizing simultaneously two objectives: the execution time and the reliability. The problem is decomposed in two successive steps: a spatial allocation during which the reliability (randomized algorithm) is maximized, and a scheduling during which the execution time (list scheduling algorithm) is minimized. It allows us to produce several trade-off solutions among which the user can choose the solution that fits the application's requirements the best. This method is assessed by running extensive simulations on both random graphs and actual application graphs. They show that it is competitive compared to existing reference scheduling methods for heterogeneous processors (HEFT) while providing a better reliability.

Keywords: Scheduling, Multi-objective optimization, Reliability

Reliability versus performance for critical applications

Alain Girault Érik Saule Denis Trystram

INRIA, POP ART team INPG - LIG, Grenoble

March 19, 2008

1 Introduction

This work concerns reliability for distributed safety critical reactive systems. Such systems should be reliable and efficient for reacting to their environment. We study the bi-objectives optimization problem for maximizing the reliability and minimizing the execution time.

Informally, an application is represented by a precedence task graph whose vertices are the instructions and the edges are dependencies between them. The maximal duration of the tasks is known, like the structure of the distributed architecture. The failure rate per time unit of each processor is also an input of the problem. We are looking for a scheduling algorithm that optimizes simultaneously both reliability and makespan (execution time). A scheduling algorithm consists of a spatial allocation and a temporal allocation. Active replication of some tasks is used for increasing the reliability.

This problem has been partially studied. Several algorithms have been proposed for minimizing only the makespan, some of them use the replication of tasks for limiting the influence of the communications. This is usually in contradiction with the replication used for increasing the reliability. In presence of replication, the reliability is usually difficult to compute. Thus, some assumptions about duplications should be made. Few solutions were proposed for optimizing both objectives, but they only provide one solution, that is a single point in the (makespan, reliability) space.

Within this work, we propose a new kind of duplication that allows to compute easily the reliability. Our main contribution is to propose a trade-off approach for maximizing the reliability and minimizing the makespan in static multiprocessor scheduling on heterogeneous distributed architectures. It considers successively the problem of determining a spatial allocation and then a temporal allocation. Several random choices are repeated for the first phase while the second is solved by an algorithm derived from the well-known list scheduling algorithm HEFT. We are therefore able to provide a set of several good compromise solutions.

This paper is organized as follows: Firstly, we present the general scheduling problem with reliability with a new model of replication in Sections 2 and 3. Then, we discuss related works in Section 4. The new two phases scheduling method is presented in Section 5. Finally, some experiments are run for assessing the method. The results are analyzed in Section 6 in regard to HEFT [38] which is a reference scheduling algorithm.

2 Problem Statement

2.1 System Model

Let us consider an **application graph** $G = (T = \{t_1, \ldots, t_n\}, E)$, where T is a set of tasks and E is a set of precedence constraints between tasks. For each directed edge $e = (t_i, t_j) \in E$, the amount of data to transfer from t_i to t_j is $data_{t_i t_j}$.

We are also given a **network of heterogeneous (unrelated) processors** $Q = \{q_1, \ldots, q_m\}$. The processing time of task t on processor q is denoted by p_{tq}. The communication network is fully connected and the data link between q and q' has a bandwidth of $BW_{qq'}$. The inter-processor communication time between t_i on q_i and t_j on q_j is $comm_{t_i q_i t_j q_j} = data_{t_i t_j}/BW_{q_i q_j}$. The intra-processor communication time is very small and can be neglected, as it is usually the case.

A **static distributed schedule** of an application graph $G = (\{t_1, \ldots, t_n\}, E)$ onto a network of processors $Q = \{q_1, \ldots, q_m\}$ consists of an assignment of each task of T to one processor of Q, along with a starting time. It is therefore formally represented by two functions:

- A **spatial allocation function** π that gives the processor where each task is to be executed: $\pi : T \to Q$. It can be seen either as a set of pairs $(t, q) \in T \times Q$, or as a matrix of size $n \times m$ whose elements belong to $\{0, 1\}$, where a '1' (resp. a '0') in position (t, q) means that t is scheduled on q (resp. is not executed).
- A **temporal allocation function** σ that gives the starting time of each task: $\sigma : T \to \mathbb{R}^+$.

Finally, the **length** or **makespan** of a schedule (denoted C_{max}) is the completion time of its last operation. Formally, $C_{max} = \max_{t \in T}(\sigma(t) + p_{t\pi(t)})$. This problem is classic and basic results are available in the chapter 3 of [21].

The above two definitions actually concern the particular case of a **schedule without replication**. For the general case of a **schedule with active replication**, a schedule is defined as:

- π gives the set of processors where each task's replica is scheduled: $\pi : T \to \{0, 1\}^m$.
- σ gives the starting time of each task's replica onto each processor: $\sigma : T \times Q \to \mathbb{R}^+$.

For a schedule without replication, π is such that $\forall t \in T, \Sigma_{j=1}^{m} \pi(t, q_j) = 1$. For a schedule with replication, it is such that $\forall t \in T, \Sigma_{j=1}^{m} \pi(t, q_j) = r_t$, where r_t is the number of replicas of task t (also called its replication factor). The subset of processors on which t is executed is denoted $\pi(t)$. This notation is a bit abusive but carries the right semantic.

The makespan corresponds to the maximum completion time over all the processors.

2.2 Fault Model

The processors of Q are assumed to be **fail-silent** [26]. All the failures are temporal and we assume that their maximal duration is such that a failure affects only the task currently executed on the faulty processor, and not the following tasks. We also assume that communication links are reliable.

Failure occurrences are supposed to be **statistically independent events**, and the occurrence of a failure on a processor q follows a **Poisson's law** with constant parameter λ_q, called the **failure rate by time unit** of q. It follows that the probability that q is operational in a time slot of length ℓ is $e^{-\lambda_q \cdot \ell}$. Conversely, the probability that q fails during a time slot of length ℓ is $1 - e^{-\lambda_q \cdot \ell}$.

A schedule is **operational** iff all its tasks are operational. A task t scheduled on processor q is **operational** iff q does not fail during the whole duration of t. Therefore, the probability that t is operational on q is:

$$\mathcal{P}(t, q) = e^{-\lambda_q \cdot p_{tq}} \tag{1}$$

Finally, the **reliability** of a schedule is the probability that it is operational. We will denote by UR the unreliability of a schedule, equal to 1 minus its reliability.

2.3 Bi-objective Problem Definition

Given an application graph G and a set of processors Q, the problem is to determine a static distributed schedule with active replication of G onto Q, with a minimal C_{max} and a minimal UR. This is a **bi-objective** static scheduling problem.

In multi-objective optimization, optimality is not defined as an absolute best solution. We will consider the notion of Pareto dominance [39, 37] in order to get a partial order between solutions. The quality of a solution is represented by the values of all the objective functions. For instance, solution $(2, 1)$ is neither better nor worse than solution $(1, 2)$. Those two solutions are not comparable and said to be **Pareto independent**.

DEFINITION 1 *Consider two solutions S_0 and S_1 of a multi-objective problem. S_0 **Pareto dominates** S_1 if S_0 is as*

good as S_1 for all objective functions and strictly better on at least one. A solution is said to be ***Pareto optimal*** *if no Pareto solution dominates it. The set of all Pareto solutions is the* ***Pareto set*** *of the problem.*

Determining if a point is Pareto-optimal or not is NP-hard, as minimizing the makespan is already a NP-hard problem (see chapter 2 of [21]).

3 Principle

3.1 Task Replication

The idea is to improve the reliability of the schedule thanks to the **active replication** of tasks. This technique is also known as the **state machine approach** [30]. It involves scheduling several copies of a task onto as many distinct processors, so that they can be executed in parallel by those processors.

Adding more replicas decreases the UR of the schedule, but in general increases its makespan. In this sense, we say that both objectives C_{max} and UR are antagonistic.

In order to get a worst-case scheduling, we force a replica (t, q) to await for the completion of *all* the replicas of all its predecessor tasks before starting its own execution. We call this replication scheme **replication for reliability**. It differs from the usual replication considered in scheduling, in which a replica only awaits for the completion of the *first* replica of all its predecessor tasks. This difference is illustrated in Figure 1. Figure 1(e) is the task to be scheduled. Figure 1(a) and (c) are two possible schedules onto a fully connected three processors architecture: the former uses the replication for efficiency scheme, while the latter uses the replication for reliability scheme. In these schedules, each task is represented by a box whose width is proportional to its execution time on its processor; each inter-processor communication is represented by an arrow whose projection on the time axis is proportional to the communication delay. Formally, in the replication for reliability case (Figure 1(a)), the precedence constraints can be written as: $\forall (t_i, t_j) \in E, \forall q_j \in \pi(t_j), \sigma(t_j, q_j) \geq \max_{q_i \in \pi(t_i)}(\sigma(t_i, q_j) + p_{t_i q_j} + comm_{t_i q_i t_j q_j})$. In the replication for efficiency case, the $\max$ would be replaced by a $\min$.

3.2 Computing the Reliability of a Spatial Allocation

A spatial allocation can be represented as a **Reliability Block Diagram** (RBD) (see for instance [23, 32]). Formally, an RBD is a directed graph $(N, \mathcal{E})$, such that each vertex of N is a **block** representing an element of the allocation (*i.e.* a replica of a task placed on a processor), and each edge of $\mathcal{E}$ is a **causality link** between two blocks. N has two particular vertices namely, the **source** S and the **destination** D (S has no incoming edges and D has no outgoing

edges). A RBD is **operational** iff there exists at least one operational path from S to D. A path is operational iff all its blocks are operational. The probability that a block is operational is equal to its reliability (computed by Equation (1)). By construction, the probability that a RBD is operational is equal to the reliability of the spatial allocation it represents.

When the spatial allocation contains no replication (*i.e.* a schedule without replication), its RBD is **serial**. Indeed, it is composed of n blocks where the i-th block represents the execution of the i-th task. In order to deliver an operational schedule, all the tasks must be executed without any fault. The RBD consists of a single path from S to D which goes through all blocks. Therefore, computing the reliability of the RBD is *linear* in the number of task.

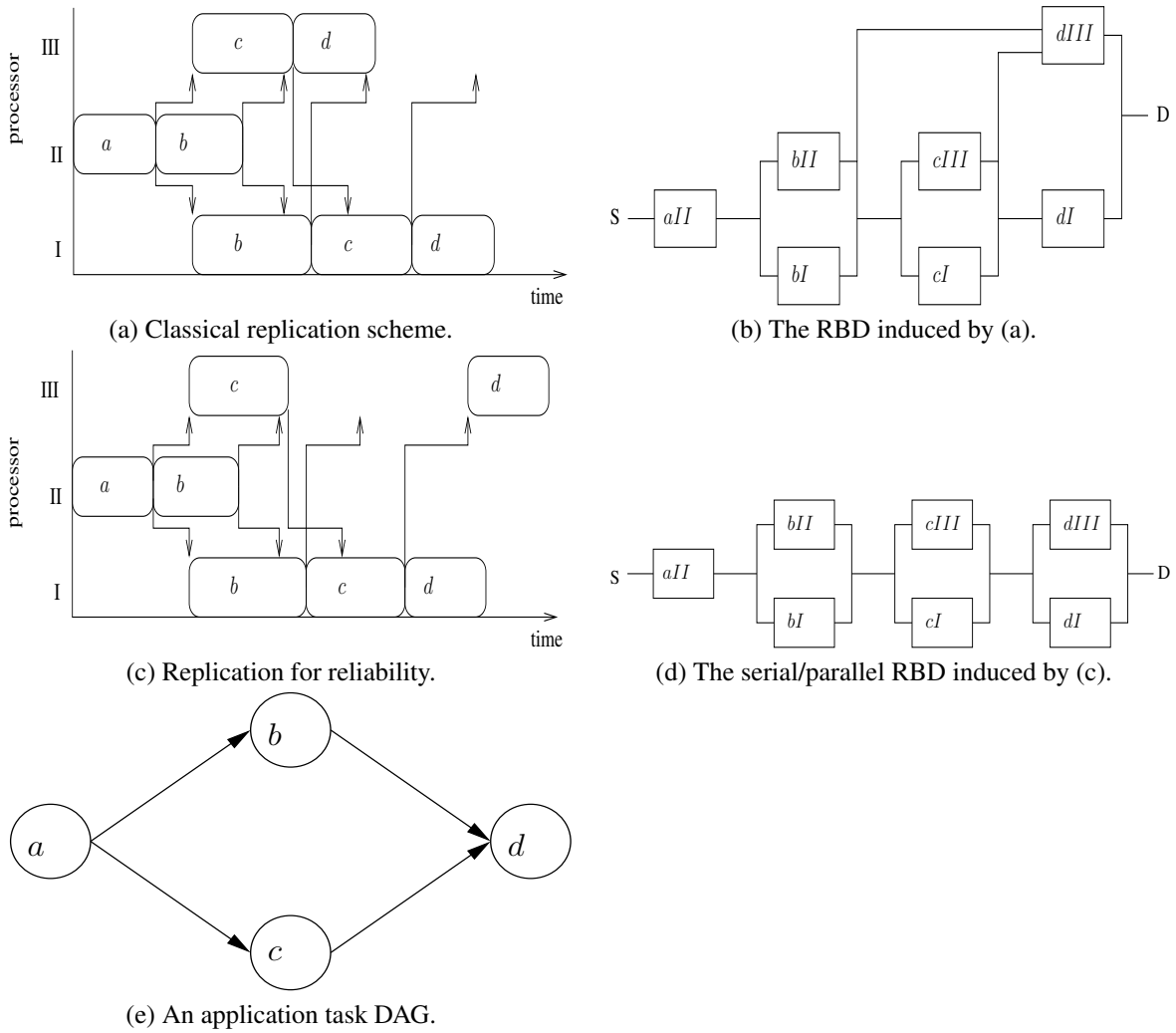
When the allocation contains replications, its RBD has no particular predefined form. For instance, Figure 1(b) shows the RBD corresponding to the schedule presented in Figure 1(a). Either $dIII$ or dI must be operational. On one hand, $dIII$ can not be operational if its predecessors are not operational. At the time where $dIII$ is scheduled, bI and cI are not completed. $dIII$ can only receive its data from bII and $cIII$. On the other hand, dI can be operational if either cI or $cIII$ is operational and either bI or bII is operational. Computing the reliability of such a RBD can only be done in *exponential* time in the size of the RBD (unless P=NP). Classical methods require to enumerate all the minimal cuts of the RBD (see *e.g.* the AltaRica workbench[12]).

Using replication for reliability leads to different properties. Indeed, if all but one replicas of task i failed, the scheduled is still valid. All tasks that depends on the result of i could still be executed without any problem. Therefore, the schedule is operational if a single replica of each task is operational. As a consequence, the RBD is always a serial/parallel graph, *i.e.* it is a chain of parallel macro blocks. Figure 1(d) presents the RBD induced by the schedule that uses replication for reliability presented in Figure 1(c). The probability that such a RBD is operational can be computed in linear time (of the size of the RDB) [23, 32].

Remark that the reliability of a schedule does not depend on the temporal allocation but only on the spatial allocation. We denote by $\mathcal{P}(\pi)$ the reliability of the spatial allocation π, computed by the following expression (2):

$$\begin{aligned} \mathcal{P}(\pi) &= \prod_{i=1}^{n} \left(1 - \prod_{q_j \in \pi(t_i)} \left(1 - \mathcal{P}(t_i, q_j)\right)\right) \\ &= \prod_{i=1}^{n} \left(1 - \prod_{q_j \in \pi(t_i)} \left(1 - e^{-\lambda_{q_j} \cdot p_{t_i q_j}}\right)\right) \\ &= \prod_{i=1}^{n} \mathcal{P}(\pi_i) \end{aligned} \tag{2}$$

We denote the unreliability of a schedule as $UR(\pi, \sigma) = 1 - \mathcal{P}(\pi)$. Remark that the expression is derived from the following two hypotheses: fail-silent transient fault, and using replication for reliability.



(a) Classical replication scheme.

(b) The RBD induced by (a).

(c) Replication for reliability.

(d) The serial/parallel RBD induced by (c).

(e) An application task DAG.

Figure 1: Difference between classical and reliable replication applied on the example of a diamond task graph.

3.3 A Preliminary Result

This section proves that there does not exist a "good" compromise solution. [8] also contains a similar proof but for schedules without replication. A schedule with replication could have been a good compromise solution. This section proves that replication does not help for finding a single good compromise solution.

Let us first recall the definition of approximation in mono-objective optimization. S is a ρ-approximation ($\rho \geq 1$) of an instance I according to the function to minimize f iff $f(S) \leq \rho f^*(I)$ where $f^*(I)$ is the minimum value of f among the solutions of I. Constant approximations is the result works are aiming at.

In multi-objective optimization, the definition is extended in order to take into account several functions. A solution S $\rho = (\rho_1, \rho_1, \ldots, \rho_k)$**-approximates** I according to $f = (f_1, f_2, \ldots, f_k)$ iff S is a ρ_i-approximation of I

according to f_i.

THEOREM 1 *The problem of minimizing C_{max} and UR can not be approximated within constant factors $\rho = (\rho_{C_{max}}, \rho_{UR})$ by a single solution.*

Proof: Consider the instance composed of 2 processors and 1 task. The processing time of the task on processor 1 is 1 and it is k on processor 2. The failure rate of processor 1 is left unspecified to λ_1. Let $\lambda_2 = \lambda_1/k^2$.

Consider all the three solutions of this instance: S_1 in which the task is scheduled on processor 1; S_2 in which the task is scheduled on processor 2; and S_3 were the task is scheduled on the two processors. Note that S_1 is optimal for C_{max} leading to $C_{max}(S_1) = 1$ and $UR(S_1) = 1 - e^{-\lambda_1 p_1}$, that S_2 is close to the optimal for UR where $C_{max}(S_2) = k$ and $UR(S_2) = 1 - e^{-\lambda_1 p_1/k}$, and that S_3 is optimal for the reliability with $C_{max}(S_3) = k$ and $UR(S_3) = 1 - e^{-\lambda_1 p_1 - \lambda_1 p_1/k}$.

Both ratios $C_{max}(S_2)/C_{max}(S_1)$ and $UR(S_1)/UR(S_2)$ go to infinity with k. Ratios involving S_3 and S_1 go to infinity with k too. As a conclusion, none of these three solutions can approximate all the solutions within a constant factor. ■

Because of this theorem, we propose to compute a *set* of compromise solutions, among which the user will choose the one that matches the best his/her applicative requirements.

3.4 On the Model Assumptions

Before going in more details, we would like to discuss some choices made on the model.

On the WCET analysis. The WCET analysis has been extensively studied (see [27, 22] for surveys). Knowing the execution characteristics is not a critical assumption since WCET analysis has been applied with success to real-life processors with branch prediction [4] or with caches and pipelines [36]. In particular, it has been applied to the most critical embedded system, namely the Airbus A380 avionics software running on the Motorola MPC755 processor [11, 34].

On transient faults and their negligible duration. Computing the reliability with RBDs requires the assumption that failures are statistically independent events. This in turn requires that the failure of a processor can only impact the task currently executing onto this processor, and not the future tasks scheduled onto it (otherwise the failure of the current task and the failure of the future task would not be independent). This in turn requires that the processor failures be transient. Note also that, from the point of view of safety critical embedded systems, transient failures are far more common than permanent failures. Moreover, processors in critical systems are usually **fail-silent** [26], so the statistical independence of faults on different processors is a pertinent assumption. This reliability model is directly

borrowed from [31] and is widely used in the literature [2, 18, 25].

On fully reliable communication links. If communication links are subject to failures, then the RBD will not be serial-parallel. Thus, computing the reliability will become a complex problem. Having failures on communication links is much harder since it requires to address extra problems, such as routing. Moreover, having a failure on a link will result in a failure on several tasks' execution, which makes failure non statistically independent. [17] deals with this problem by adding for each task t, a special synchronization task which depends on all replicas of t and on which depend all successors of t. By doing that, performances and reliability are degraded in order to gain more tractability.

4 Discussion on Related Approaches

In this section, we first review the main results for minimizing the objectives independently (the makespan in heterogeneous computing and the unreliability). Then, we introduce the multi-objective optimization, and we present the main approaches proposed for dealing with this problem.

4.1 Optimizing the Makespan in Heterogeneous Computing

Off-line scheduling for optimizing the makespan on homogeneous resources (machines, computers ...) has lead to a huge number of studies [21]. Even if most variants of this problem are NP-hard, it is often possible to derive a theoretical analysis for designing approximation algorithms. In the context of parallel processing, scheduling on heterogeneous resources is a more recent problem, which sounds harder. The existing results consist mainly in designing smart heuristics that have no theoretical guarantees. Considering the relaxed hypothesis of tasks independence, there is a known 2-approximate algorithm [13]. Let us now survey briefly the main existing results.

The most popular proposed methods are extensions of list scheduling algorithms for heterogeneous resources. HEFT (Heterogeneous Earliest First Task scheduling) [38] sorts the tasks by decreasing order of average remaining critical path. It is a greedy algorithm: each task is considered one after the other, and is mapped to the processor that can complete it the soonest. HEFT is often considered as a reference heuristic for running experiments. It has been improved in [40] by using a different ordering of tasks.

The min-min heuristic and its variants [15] consider, for each task, the processor that can complete it the soonest. Min-min assigns the task with the smallest minimum completion time in order to optimize processor's usage (in this sense, it is a dual view of HEFT). The max-min variant assigns the task with the greatest minimum completion time. The idea here is to take into account early tasks that will potentially penalize the global makespan. The number of proposed variants reflects the heuristic character of the method. These algorithms are designed for independent tasks,

but can be easily adapted for tasks with data dependencies.

Other approaches are based on clustering, which involves grouping tasks that communicate heavily. Tasks grouped together will be executed on the same resource, and groups are then assigned to a resource. Finally, the starting time of each task is determined. [3] is an example of such an approach which deals with heterogeneity by considering the resources of similar capabilities.

4.2 Optimizing the Reliability Only

Let us remind first that the reliability is increased by replicating some tasks. When the communication links are reliable, the computation of the reliability of the induced RBD can be done in linear time. With link failures, computing the reliability of the RBD is NP-complete. Thus, there exists no polynomial time algorithm (unless P=NP). However it is possible to design an efficient exact exponential algorithm based on the minimal cuts of the RBD [12].

Another way to compute the reliability of the RBD is to add extra synchronization tasks in order to transform it into a serial/parallel graph. Despite the fact that the problem becomes polynomial, there is an additional overhead due to these synchronization tasks (see details in [17]).

[35] optimizes the reliability by using clustering techniques in order to minimize the communication times. Then, heavier communications between clusters are mapped to the more reliable links, while clusters with higher computation cost are mapped to the more reliable processors.

With our assumptions, the optimal reliability can be easily computed. The reliability of a schedule is increased by adding replicas. The maximum reliability is therefore obtained by executing a replica of each task on all processors.

4.3 General Purpose Multi-objective Optimization

In this section, we discuss the main way used in the literature for optimizing several objectives simultaneously. This topic has recently received more and more attention [9, 10, 1].

The most commonly used approaches (and the simplest ones) transform the multi-objective problem into a mono-objective one, namely:

1. We can fix a threshold value for one objective. The solution is then constrained to a minimum (or maximum) value on all but one objective functions. This technique is sometime called ϵ-constraint [37].

2. We can aggregate all the objectives into a single function. A new objective function is derived usually by a linear or convex combination of the objectives. If a solution is optimal for a linear or convex aggregation function, then it is Pareto optimal. However, finding such a solution is usually NP-hard.

3. We can sort the objectives by a hierarchy that reflects the relative importance of the objectives. The problem is solved iteratively for each objective, like in [14].

Most of these approaches lead to a single solution. In contrast, we are proposing a new method that generates many compromise solutions. By iteratively using a threshold method applied at different levels (*i.e.* solution 1), we will derive a set of trade-off solutions.

Only very few works proposed integrated approaches. [24] formalized in a nice way a generic method for obtaining an ϵ-approximation of a Pareto set by partitioning the solution space in increasing size area of a factor ϵ. Then, looking for a solution in each area gives an approximation of the Pareto set. Such a set has a polynomial cardinality if the problem belongs to NP. Finding a point in each area is a NP-hard problem in the unreliability and makespan minimization problem. However, the main idea helps to construct an interesting set of independent solutions.

4.4 Reliability and Makespan Bi-objective Optimization

In [2], the authors proposed a heuristic for a similar problem in which communication links are not reliable. The authors compute an upper-bound of the reliability thanks to the minimal cut sets method applied to the RBD. The proposed heuristic optimizes a linear combination of the two objectives, normalized with respect to threshold provided by the user.

[6] proposed a bi-objective scheduling problem for non fully connected networks of processors. The exact reliability is NP-hard to compute, which makes the problem harder. The optimization problem is treaten by adapting a list-based heuristic called DLS [33] into RDLS. DLS is a greedy algorithm for heterogeneous scheduling for makespan that iterates by scheduling a pair (task, processor) that has the greatest Dynamic Level. RDLS considers a Reliable Dynamic Level that is obtained by adding a term to the Dynamic Level to take into account the reliability of the processors. However, tasks are not replicated, so the impact on reliability is very limited.

[8] tackled the problem of maximizing the reliability and minimizing the makespan on related machines where processors are subject to crash fault. Approximation of the Pareto set is given for the case of unitary execution time. This work also proposes a general way to transform heuristics for minimizing the makespan into bi-objective heuristics. Here again, the tasks are not replicated.

5 A New Bi-objective Scheduling Heuristic

In this section, we present our two steps heuristic for optimizing the makespan and reliability. First, it computes the spatial allocation function in order to set UR lower than a threshold value; then it computes the temporal alloca-

tion function aiming at minimizing C_{max}. It is inspired by [24] for approximating Pareto sets but does not ensure performance ratios.

5.1 Principle

As shown in Section 3.3, it is in general impossible to achieve simultaneously a solution approximating both objectives within a constant factor. Thus, we will provide a set of several Pareto independent solutions in order to help a decision maker to determine a trade-off solution. Then, we propose to consider one objective as a threshold, set to several successive levels, and to solve the resulting mono-objective problem for each level.

As the unreliability of a solution $S = (\pi, \sigma)$ depends only on π, it is easier to threshold on the unreliability. We start by computing an allocation π that satisfies a fixed unreliability. As there is a huge number of π that satisfy the threshold, we will generate them randomly. This will be explained in more detail in 5.2. Spatial allocations are post-optimized using a greedy algorithm by removing longest jobs (details are given in 5.2.2).

Since we have fixed the allocation π, we only have to care about the makespan while generating σ. The problem of choosing σ to minimize the makespan is a pre-allocated scheduling problem, which is discussed in 5.3.

In summary, our method involves two successive phases: first we compute a spatial allocation π that satisfies the unreliability threshold, and then we compute a temporal allocation σ that minimizes C_{max}. Decoupling those two phases is possible because the reliability of a static multiprocessor schedule depends only on its spatial allocation π.

5.2 Phase 1: Allocation Scheme

5.2.1 Properties and Algorithm

The following property helps to obtain a partial order in the allocation space.

PROPERTY 1 *Let π_0 and π_1 be two spatial allocations.* $\pi_0 \subset \pi_1 \Rightarrow UR_{\pi_0} > UR_{\pi_1}$

This property can be used by a greedy algorithm to improve the reliability of a spatial allocation by adding active replicas. The two following properties help us to choose which replica to add. Property 2 formally states that the reliability of a spatial allocation can not be better than the worst reliability of a task. Property 3 states that there exists a task having a reliability greater than the n-th root of the allocation's reliability.

PROPERTY 2 *Let π be a spatial allocation such that $UR(\pi) < UR_0$, then* $\forall t \in T, \mathcal{P}(\pi_t) > 1 - UR_0$

PROPERTY 3 *Let π be a spatial allocation such that $UR(\pi) < UR_0$, then* $\exists t \in T, \mathcal{P}(\pi_t) > \sqrt[n]{1 - UR_0}$

Algorithm 1 IRSAG: Spatial allocation generation

```
1  input : an instance, a value UR_0 and an allocation π_0
2  output : an allocation π
3  begin
4      π := π_0;
5      prop2 := false;
6      forall t ∈ T do
7          while P(π_t) < 1 − UR_0 do
8              π := π ∪ (t, alea(1, m));
9              if P(π_t) > ∛(1 − UR_0) then
10                 prop2 := true;
11             end if
12         end while
13     end forall
14     if prop2 = false then
15         t_0 := alea(1, n);
16         while P(π_{t_0}) < ∛(1 − UR_0) do
17             π := π ∪ (t_0, alea(1, m));
18         end while
19     end if
20     while (UR(π) > UR_0) do
21         if π = T × Q then
22             return NIL;
23         end if
24         π := π ∪ (alea(1, n), alea(1, m));
25     end while
26     return π;
27 end
```

Algorithm 2 CommBlevelList

```
1  input : an instance and an allocation π
2  output : an temporal allocation σ
3  begin
4      Let blevel[t] be the longest past from t to the end
       of the graph.
5      forall t ∈ T in anti-topological order
6          if t is a communicating task or a leaf
           then
7              prio[t] := blevel[t]
8          else
9              prio[t] := max_{t'|∃(t,t')} prio[t']
10         end if
11     end forall
12     forall time x from 0 to ∞
13         forall q ∈ Q
14             if j is idle at x
15                 Select t ready on q at time x minimiz-
                   ing prio[t]
16                 Schedule t in σ from x to x + p_t on q
17             end if
18         end forall
19     end forall
20     return σ
21 end
```

The proofs of both properties are obtained directly from the fact that the RBDs are serial/parallel (thanks to the reliability of the communication links) and from the multiplication of probabilities, so we skip them.

We now give the IRSAG (Iterative Randomized Spatial Allocation Generator) algorithm that constructs a spatial allocation with an unreliability smaller than a threshold UR_0. It is formally stated in Algorithm 1. The principle is to add random replicas to the current allocation. It starts by fulfilling Property 2, that is, it adds a replica for each task as long as its unreliability is greater than the threshold (lines 6 to 13). If no task respects Property 3, a task is randomly chosen (lines 14 to 19). Finally, replicas are added until the unreliability threshold is reached (lines 20 to 25).

Remark that the algorithm works with any initial allocation π_0, and not only with the empty set.

The algorithm uses a uniform distribution for choosing the replica (t, q) to add. The last part taken alone ensures that tasks are in average replicated evenly. This seems to be a good property according to the structure of the reliability function (Equation (2)).

5.2.2 Improvement

Algorithm 1 returns a spatial allocation that has an unreliability smaller that UR_0. But it does not ensure that the allocation is minimal for this property. Due to the replication model, removing replicas can only lead to a better

makespan. This is why we propose a `maopt` procedure that iteratively removes replicas as long as the unreliability remains below UR_0. Replicas will be sorted in decreasing order of execution times in order to remove first the longest replica.

5.3 Phase 2 : Pre-allocated Scheduling

Since we are using the model of replication for reliability (Figure 1(b)), we cannot reuse the classical literature on multiprocessor scheduling with duplication, because all these results assume the classical model of replication (see Figure 1(a)) and the related discussion).

Finding the best temporal allocation σ with a fixed spatial allocation π is equivalent to schedule a DAG of tasks where tasks are forced to be scheduled on a given processor. This problem can be seen as a special case of scheduling on dedicated machine. However, this particular problem is the Scheduling with Preallocation Problem which is known to be strongly NP-hard [29]. In the following, all replicas are equivalent and therefore, the term "task" refers to a replica on a processor.

The following definition leads to a weakly dominant property for schedules that helps to solve the problem. We introduce a notion of priority between tasks allocated in order to characterize the temporal allocation on a given processor. When two tasks t and t' are ready, if t has a greater priority than t', then t is scheduled before t'. t is a **communicating task** if one of its direct successor t' is scheduled on a different processor than t.

DEFINITION 2 *Let σ be a schedule. For each processor q, $\{tc_1^q, \ldots, tc_{k_q}^q\}$ denotes the set of communicating tasks ordered in the same way as in the schedule. The schedule is* ***communication friendly*** *if for all processor q and for each pair of communicating task tc_i^q, tc_j^q such that $i < j$, the predecessors of tc_i^q have a greater priority than the predecessors of tc_j^q.*

PROPERTY 4 *Let σ be a valid schedule that is not communication friendly, then there exists a communication friendly schedule having a better (or equal) makespan.*

Sketch of the proof: From σ we derive, on each processor, the total ordering of communicating tasks. By iteratively swapping consecutive tasks to comply with Definition 2, the makespan can only decrease because the communicating tasks are done sooner (or at the same time) than in the original schedule. ■

The proof gives us an algorithm that improves an existing temporal allocation. We first need an existing temporal allocation to use it. So we generate it using a list scheduling algorithm.

In the literature, list scheduling algorithms are widely used for classical scheduling problem [21]. The principle can be stated as "compute if there is something to compute". In order to break tie, we usually introduce priority between tasks. The B-level of a node is a classical lower bound of the minimum completion time of the application from the execution date of the node. It can be seen as the application's completion time assuming there is an infinite number of processors. The B-level of a leaf node (*i.e.* without successors) is its execution time, while the B-level of a non-leaf task is the sum of its execution time and of the maximum B-level of its successors. As our problem is pre-allocated, we can take into account the communication times when computing the B-level. It remains a lower bound of the optimal makespan.

The CommBlevelList algorithm is focused on communicating tasks by setting non communicating tasks' priority to the maximum priority of their successors, and communicating tasks' priority to their B-level. CommBlevelList is formally given in Algorithm 2. The reader should note that the formal statement includes a loop over the time to compute the starting time of the tasks. This loop is included for the sake of clarity: an algorithm producing the same schedules could be written using heaps instead of iterating on time slices.

6 Experimental Study

In this section, we present experimental results about the method proposed in Section 5. There is also no interest in comparing the method we proposed with a method that does not use duplication. Indeed, previous works study the problem without duplicating tasks and thus do not improve the reliability by more than one order [8, 7, 6]. So we do not compare the proposed algorithm with other bi-objective algorithms. But we give responses to some questions detailed in 6.1. Some instances for experimental tests were constructed by using a model of random networks and applications from an existing benchmark. Details about the instances generation are given in 6.2. Experiences are described in 6.3. Finally, the results are analyzed in 6.4.

6.1 Goals of the Experiments

Assessing the efficiency of the method is difficult, since extracting the Pareto set of an instance requires an exponential time. We try to validate it by studying several points:

In order to be competitive, the algorithm for the pre-allocated scheduling problem should be efficient. If it is not, the complete method would not give interesting results, even if the first phase gives the best spatial allocation possible.

It is also interesting to study the impact of the quality of the spatial allocation on the makespan of the resulting solution. In particular, changing the number of iteration of the random algorithm and using the optimization presented

in 5.2.2 should be considered.

Those first two questions are crucial for improving the method. However, the main question remains: can we derive a set of interesting trade-off solutions from our method? And finally, is the method competitive with existing scheduling methods for makespan only?

6.2 Benchmark Construction

We construct an instance for the heterogeneous scheduling problem by putting together, on the one hand, an application graph with communication and processing times, and on the other hand, a network with processors of different speeds and communication links of different bandwidths.

6.2.1 Application Generation

We consider an instance of the problem of static scheduling on identical processors with precedence constraints $P \mid prec, p_i, c_{ij} \mid C_{max}$ [21]. A classical instance for this problem is described by m processors, an application graph $G = (V, E)$, computation times p_v of all tasks $v \in V$ and data sizes $Comm_{ij}$ between tasks, for all $(i, j) \in E$.

We uses an existing benchmark [20] that considers 602 instances of various structures. This benchmark has been used to test different scheduling algorithms in [5, 19].

6.2.2 Processor Set Generation

A set of processors is characterized by the number of processors m, their speeds $speed_i$, their failure rates per time unit λ_i, and by the bandwidth of the link between two processors BW_{ij}.

According to [28, 7], in actual systems, ratios between computational speeds of any two different processors or bandwidth of any two different links are uniformly distributed in $[1, 10]$. Real systems have failure rates uniformly distributed in $[10^{-6}/h; 10^{-5}/h]$. We randomly generated 60 sets of processors with a number of processors between 5 and 10.

6.2.3 Instance Construction

For an application and a network of processors, we constructed an instance for the heterogeneous scheduling problem. A task i scheduled on processor j takes $p_i/speed_j$ time units to compute.

We removed a class of 275 big instances from the benchmark. Despite the fact that they were solvable separately in a reasonable time, solving all the 16200 instances (275×60) would have taken a prohibitive time (about five minutes by instances). We kept the 327 other instances. Thus, our experiments were made on 19620 instances (327×60).

The instances we generated are not uniformly distributed among the instance space and are probably not representative of any application field. Thus, all results are valid only considering this particular benchmark. However, this benchmarking method shows the general behavior of tested algorithms.

6.3 Experimental Protocol

Our method is not monolithic, several options and parameters can vary. First, as our algorithm is randomized, we can change the number of draws. We have considered the following number of iterations: 1, $\log nm$, $\sqrt{nm}$ (where n is the number of tasks and m is the number of tasks). The `maopt` improvement given in 5.2.2 can be used or not. Finally, the first phase algorithm can be initialized by several allocations: the empty allocation and the HEFT allocation were considered. Remember that the whole method is parametric. It takes a minimum reliability as a parameter.

Experiments are run under the following protocol. First, HEFT [38] is run on each instance. Let ${C_{max}}^{HEFT}$ and UR^{HEFT} be the values obtained on the schedule computed by HEFT. The algorithm is run with different thresholds on unreliability. Each unreliability value is obtained by multiplying UR^{HEFT} by values of r with r is one of the following values: 1, 0.9, 0.8, 0.7, 0.1, 0.01, 0.001, 0.0001 or 0.00001. For instance, if $UR^{HEFT} = 0.1$ and $r = 0.1$, then our UR objective is 0.01, so conversely, the minimum reliability we accept is 0.99.

We also run experiments in "real condition" by setting the threshold to a geometrical increasing level from UR_{min} to UR_{max}. Only Pareto independent solutions art kept. This is the way our method should be used in an actual production environment, and this confirms its practical usability.

6.4 Results and Analysis

Results are given for each r as the geometric mean (over all instances) of the ratio between the makespan of our algorithm and the makespan of HEFT. We use the geometric mean because it is the one which makes sense when considering ratios [16].

In order to verify the efficiency of our pre-allocated B-Level scheduling, we compare the makespan obtained by HEFT and the makespan of our B-Level scheduling with the spatial allocation of HEFT. The mean ratio between our algorithm and HEFT is 1.01009 with a minimum ratio of 0.885272 and a maximum ratio of 2.29752. Those results show that, despite the non existence of a guaranteed performance, the second phase's algorithm performs well in practice.

We test the impact of a bad spatial allocation by comparing the results of the experience without optimization (`iterlog` curve) and with `maopt` (`maoptiterlog` curve), both of them with $\log nm$ iterations. The results are presented in Figure 2. It shows, for each value of factor of unreliability r, the mean ratio of the makespan to

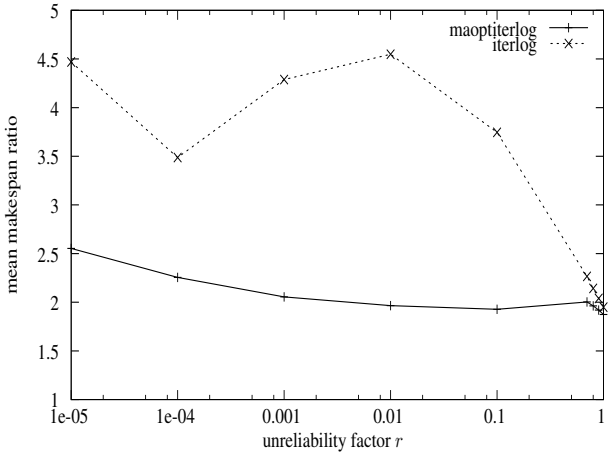


Figure 2: Comparing random allocation with optimization

${C_{max}}^{HEFT}$. The point for the `maoptiterlog` curve with $r = 00.1$ and mean makespan ratio of 2 can be interpreted as: On this benchmark, to gain two orders of unreliability, we have (in average) twice the makespan. Results show that the optimization presented in 5.2.2 really improves the makespan as the $maoptiterlog$ curve is widely below the $iterlog$ curve. Thus, getting a good allocation for makespan is really important. Some strange results occur for $r = 10^{-4}$ and $r = 10^{-3}$ without `maopt`. They are due to a threshold effect of Properties 3 and 2. That is to say that Property 3 is useless for nearly all valid schedules for $r \geq 0.01$; only one replica for each task is needed to match the property but obtaining a reliability greater than the threshold will require extra-replica on most of tasks. When $0.0001 \leq r \leq 0.01$, the mean number of replication which are required to match Property 3 is closer to the number of replica needed to match the reliability goal.

Figure 3 shows the results obtained by changing the number of iterations of our algorithm. Three numbers of iterations were considered: $\sqrt{nm}, \log(nm), 1$, which stand respectively for `maoptitersqrt`, `maoptiterlog`, and `maoptiter1`. The greatest number of iterations tested is $\sqrt{nm}$ because the computation time for an instance with nm iterations became prohibitive (several minutes for small instances against several second for $\sqrt{nm}$). Moreover, it seems hard to do better than what we obtain with those parameters as the difference between the curves `maoptiterlog` and `maoptitersqrt` is much smaller than the difference between `maoptiter1` and `maoptiterlog`. We can conjecture (but experiments should be run) that increasing the number of iterations will not improve significantly the performances. The mean value obtained with $r = 0.1$ is rather strange: the mean value of the makespan for $r = 0.1$ is better than the mean makespan for $r = 0.7$. This is more visible when there is a single iteration of the random algorithm. This shows that the `maopt` optimization becomes more efficient when r increases, *i.e.* when there are more replicas.

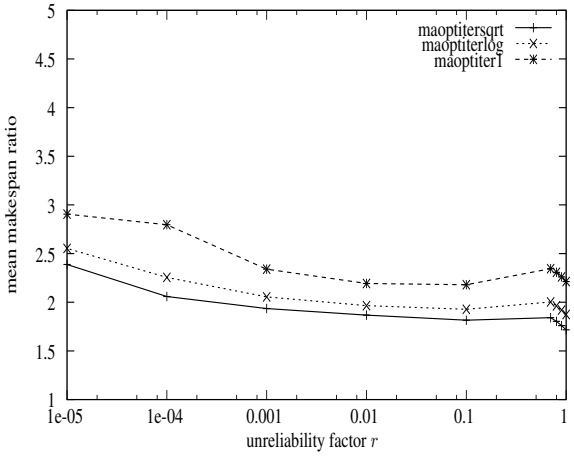


Figure 3: Impact of the number of generation on the makespan

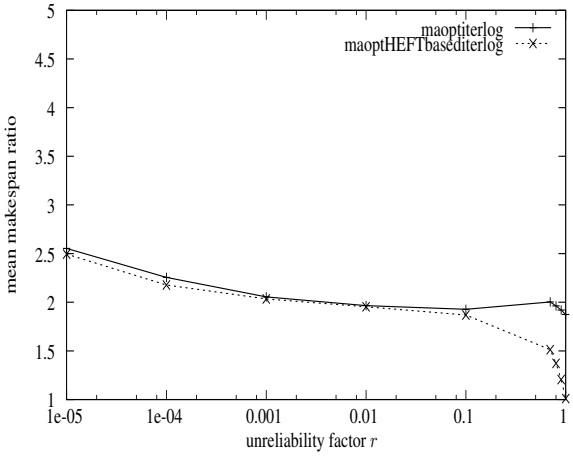
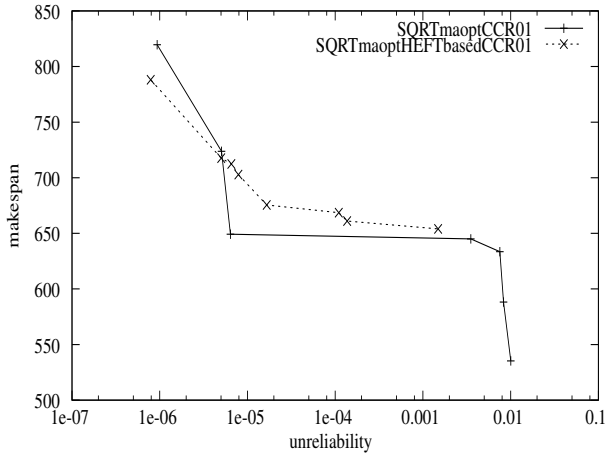


Figure 4: Comparing HEFT based allocation with pure random one

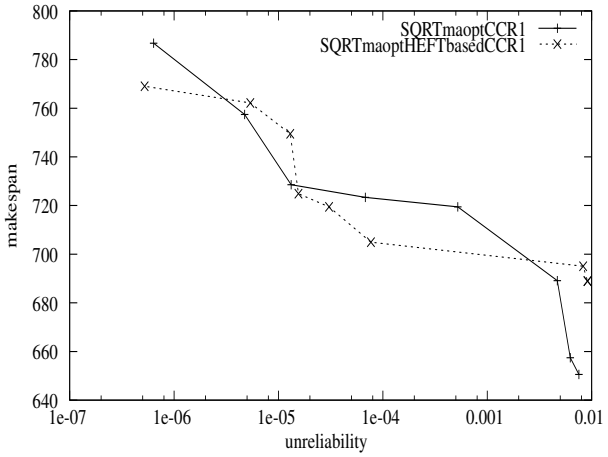
Despite the efficiency of our algorithm's second phase, the method tested does not give solutions close to the HEFT makespan. Therefore, our algorithm's first phase returns spatial allocations that are not "makespan friendly". All those results mean that completely random spatial allocation are bad for the makespan. Basing the random allocation on an existing allocation for the makespan should improve the obtained makespan for a reliability factor close to 1. Figure 4 confirms that basing random allocations on the HEFT spatial allocation really improves the makespan for small reliability. However, it seems to be of no use if we want to gain an order (or more) on the reliability. In Figure 4, the `maoptiterlog` curve is the same as in Figure 3, while the `maoptHEFTbaseditерlog` curve is based on initial allocations produced by HEFT.

Considering a particular instance, Figure 5 shows for three levels of CCR (Communication to Computation Ratio) the set of trade-off solutions returned by our method (that is, the approximated Pareto curve). The results are obtained with and without basing our allocation on HEFT. The results obtained really differ with the CCR. On one hand, using HEFT results in a worse curve when the CCR is low. On the other hand, when the CCR is high, the curve obtained when basing the method on HEFT completely dominates the random one. The (a) and (b) curves seem to warn us against using exclusively HEFT. This can result in missing interesting solutions.

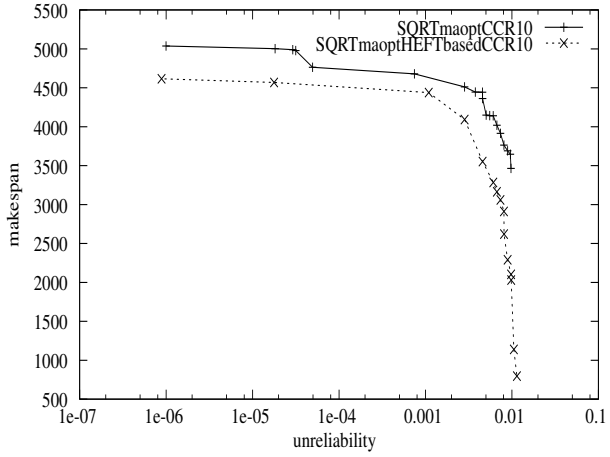
We can remark that when the CCR is high, improving the reliability has a more notable impact on the makespan than when the CCR is low. This is due to the replication model that implies a lot of communications.



(a) CCR=0.1



(b) CCR=1



(c) CCR=10

Figure 5: r150 with different CCR levels

7 Conclusion

We have presented a new bi-objective scheduling method for task graphs with data-dependencies onto heterogeneous distributed memory architectures. The two objectives considered are the schedule length (real-time constraint) and the reliability (safety critical constraint). The failure model assumes that the processors are fail-silent, communication links are reliable, and failure occurrences are statistically independent event and follow a Poisson law with a constant parameter (called the failure rate per time unit of the processor). To improve significantly the reliability, we use the active replication of tasks. A key issue is then to compute the reliability of the system. We showed that using a "replication for reliability" scheme allows us to improve the reliability and to compute it easily by slightly degrading the makespan.

Since we are solving a bi-objective problem, we face the problem that there exist several non Pareto-dominated solutions. To alleviate this problem, we transform the reliability objectives into a constraint. For a given instance, we apply iteratively our method with several reliability constraint levels, in order to obtain a set of trade-off solutions. This will allow the user to choose the solution that fits best his applicative requirements.

Our method proceeds in two phases: first a spatial allocation of the tasks onto the processors, then a temporal allocation to determine at what time each task should start its execution. According to our failure model, the reliability of a schedule depends only on its spatial allocation. During the first phase, we only produce spatial allocations whose reliability are greater than the reliability threshold; to achieve this, we replicate some well-chosen tasks. And during the second phase, we only attempt to minimize the schedule length.

We have performed extensive simulations of our scheduling algorithms, with various optimizations and various initial spatial allocations (either obtained with a random placement algorithm or with the popular HEFT algorithm). These simulations prove the efficiency of the method.

References

[1] A. Agnetis, P. B. Mirchandani, D. Pacciarelli, and A. Pacifici. Scheduling problems with two competing agents. *Operations Research*, 52(2):229–242, April 2004.

[2] I. Assayad, A. Girault, and H. Kalla. A bi-criteria scheduling heuristic for distributed embedded systems under reliability and real-timeconstraints. In *2004 International Conference on Dependable Systems and Networks (DSN'04)*, pages 347–356, June 2004.

[3] B. Cirou and E. Jeannot. Triplet: a clustering scheduling algorithm for heterogeneous platforms. Technical Report RT-0248, INRIA, Febuary 2001.

[4] A. Colin and I. Puaut. Worst case execution time analysis for a processor with branch prediction. *Real-Time Systems*, 18(2/3):249–274, 2000.

[5] T. Davidovic, L. Liberti, N. Maculan, and N. Mladenovic. Mathematical programming-based approach to scheduling of communicating tasks. Technical report, LIX, December 2004.

[6] A. Dogan and F. Özguner. Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):308–324, March 2002.

[7] A. Dogan and F. Özgüner. Biobjective scheduling algorithms for execution time-reliability trade-off in heterogeneous computing systems. *The Computer Journal*, 48(3):300–314, 2005.

[8] J. Dongarra, E. Jeannot, E. Saule, and Z. Shi. Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. In *SPAA '07: Proceedings of the nineteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 280–288. ACM press, June 2007.

[9] E.Angel, E. Bampis, and L. Gourvès. Approximation results for a bicriteria job scheduling problem on a single machine without preemption. *Information Processing Letters*, 94(1):19–27, April 2005.

[10] E.Angel, E. Bampis, and A. Kononov. A FPTAS for approximating the unrelated parallel machines scheduling problem with costs. *In Proceeding of European Symposium on Algorithms (ESA)*, pages 194–205, 2001.

[11] C. Ferdinand, R. Heckmann, M. Langenbach, F. Martin, M. Schmidt, H. Theiling, S. Thesing, and R. Wilhelm. Reliable and precise WCET determination for a real-life processor. In *International Workshop on Embedded Software, EMSOFT'01*, volume 2211 of *LNCS*, Tahoe City (CA), USA, October 2001. Springer-Verlag.

[12] J. Gauthier, X. Leduc, and A. Rauzy. Assessment of large automatically generated fault trees by means of binary decision diagrams. *J. of Risk and Reliability*, 221(2):95–105, 2007.

[13] L.A. Hall. Approximation algorithms for scheduling. In D.S. Hochbaum, editor, *Approximation Algorithms for NP-hard problems*, chapter 1, pages 1–45. PWS publishing company, Boston, MA 02116, USA, 1997.

[14] K. Ho. Dual criteria optimization problems for imprecise computation tasks. In Leung [21], chapter 35.

[15] O. H. Ibarra and C. E. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the Association for Compution Machinery*, 24(2):280–289, April 1977.

[16] R. Jain. Ratio games. In *The Art Of Computer Systems Performance Analysis*, chapter 11, pages 165–174. Wiley professional computing, New York, USA, 1991.

[17] H. Kalla. *Génération automatique de distributions/ordonnancements temps réel, fiables et tolérants aux fautes*. PhD thesis, INPG, December 2004.

[18] S. Kartik and C.S.R. Murthy. Task allocation algorithms for maximising reliability of distributed computing systems. *IEEE Transaction on Computers*, 46(6):719–724, June 1997.

[19] V. Kianzad and S. Bhattacharyya. A comparison of clustering and scheduling techniques for embedded multiprocessor systems. Technical report, Institute for Advanced Computer Studies, December 2003.

[20] Y-K. Kwok and I. Ahmad. Benchmarking the task graph scheduling algorithms. In *IPPS/SPDP*, pages 531–537, 1998.

[21] J. Y-T. Leung, editor. *Handbook of Scheduling*. CRC Press, Boca Raton, FL, USA, 2004.

[22] B. Lisper. Trends in timing analysis. In *IFIP Working Conference on Distributed and Parallel Embedded Systems, DIPES'06*, pages 85–94, Braga, Portugal, October 2006. Springer.

[23] D. Lloyd and M. Lipow. *Reliability: Management, Methods, and Mathematics*, chapter 9. Prentice-Hall, 1962.

[24] C. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *Proc. of the 41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 86–92, 2000.

[25] P. Pop, K. Poulsen, and V. Izosimov. Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems. In *CODES-ISSS'07*, Salzburg, Austria, October 2007. ACM.

[26] D. Powell et al. The Delta-4 approach to dependability in open distributed systems. In *International Symposium on Fault-Tolerant Computing, FTCS-18*, pages 246–251, Tokyo, Japan, June 1988. IEEE.

[27] P. Puschner and A. Burns. A review of worst-case execution-time analysis. *Real-Time Systems*, 18(2/3):115–128, 2000.

[28] 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 *International Conference on Parallel Processing, ICPP'02*, pages 360–386, Vancouver, Canada, August 2002.

[29] V. Rayward-Smith, F.N. Burton, and G.J. Janacek. Scheduling parallel program assuming preallocation. In J.K Lenstra P. Chretienne, E.G. Coffman and Z. Liu, editors, *Scheduling Theory and its Applications*, chapter 7, pages 146–165. Wiley, 1995.

[30] F.B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.

[31] S. Shatz and J.P. Wang. Task allocation for maximizing reliability of distribued computer systems. *IEEE Transactions on Computers*, 41(9):1156–1169, September 1992.

[32] D.P. Siewiorek and R.S. Swarz. *Reliable Computer Systems, Design and Evaluation*. A.K. Peters, third edition, 1998.

[33] G.C. Sih and E.A Lee. A compile-time scheduling heuristic for interconnection-constraint heterogeneous processor architectures. *IEEE Transactions on Parallel and Distributed Systems*, 4:175–187, February 1993.

[34] J. Souyris, E.L. Pavec, G. Himbert, V. Jégu, G. Borios, and R. Heckmann. Computing the worst case execution time of an avionics program by abstract interpretation. In *International Workshop on Worst-case Execution Time, WCET'05*, pages 21–24, Mallorca, Spain, Juillet 2005.

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

[36] H. Theiling, C. Ferdinand, and R. Wilhelm. Fast and precise WCET prediction by separate cache and path analyses. *Real-Time Systems*, 18(2/3):157–179, May 2000.

[37] V. T'kindt and J-C. Billaut. *Multicriteria Scheduling*. Springer, 2006.

[38] H. Topcuoglu. Performance-effective and low-complexity task scheduling for heterogeneous scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, March 2002.

[39] M. Voorneveld. Characterization of Pareto dominance. *Operations Research Letters*, 31(1):7–11, January 2003.

[40] H. Zhao and R. Sakellariou. An experimental investigation into the rand function of the heterogeneous earliest finish time scheduling algorithm. *In Proc. of EuroPar 03. LNCS 2790*, pages 189–194, 2003.