Reliability versus performance for critical applications

(draft)

Alain Girault
INRIA Grenoble Rhône-Alpes, POP ART team

Érik Saule Denis Trystram
INPG - LIG, Grenoble

April 14, 2009

Abstract

Applications implemented on critical systems are subject to both safety critical and real-time constraints. Classically, applications are specified as precedence task graphs that must be scheduled onto a given target multiprocessor heterogeneous architecture. We propose a new method for optimizing simultaneously two objectives: the execution time and the reliability of the schedule. The problem is decomposed in two successive steps: a spatial allocation during which the reliability is maximized (randomized algorithm), and a scheduling during which the makespan is minimized (list scheduling algorithm). 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. Reliability is increased by replicating adequate tasks onto well chosen processors. Our fault model assumes that processors are fail-silent, that they are subject to transient failures, and that the occurrences of failures follow a constant parameter Poisson law. We assess and validate our method by running extensive simulations on both random graphs and actual application graphs. They show that it is competitive, in terms of makespan, compared to existing reference scheduling methods for heterogeneous processors (HEFT), while providing a better reliability.

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-objective 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 whose edges are dependencies between them. The maximal duration of each task is known, as well as the topology of the

multiprocessor 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 the reliability and the makespan (that is, the execution time of the scheduled application). A scheduling algorithm consists of a spatial allocation and a temporal allocation. Active replication of some well-chosen 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. A few solutions have been proposed for optimizing both objectives, but they provide only one solution, that is a single point in the (makespan, reliability) space [4, 9, 34].

We propose a new replication scheme that allows the reliability to be easily computed. Our main contribution is to propose a trade-off approach for maximizing the reliability and minimizing the makespan in static multiprocessor scheduling on heterogeneous multiprocessor architectures. We consider 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. By taking the reliability as a constraint and by minimizing the makespan for several values of the reliability, we are therefore able to provide a set of several non-dominated 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 work in Section 4. The new two-phase scheduling method is presented in Section 5. Finally, some experiments are run for assessing the method. The results are analyzed in Section 6 and compared to HEFT [41], 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 equal to $BW_{qq'}$. The inter-processor communication time between t_i on q_i and t_j on q_j is therefore

$comm_{t_i q_i t_j q_j} = data_{t_i t_j}/BW_{q_i q_j}$. The intra-processor communication time is very small and can be neglected, as is usually the case. Moreover, communications between processors can be overlapped by local computations (this is realistic since most processors now have a communication dedicated co-processor).

A **static multiprocessor schedule** of an application graph $G = (T, E)$ 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 scheduled on q).
- 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 [25].

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 formally defined as:

- π gives the set of processors where each task's replica is scheduled: $\pi : T \to 2^Q$. It can also be seen as a two-variable function: $\pi : T \times Q \to \{0, 1\}$. For the ease of notation, we write $\pi(q_i)$ the set of tasks scheduled on processor q_i. Finally, we write $\pi \subseteq \pi'$ iff $\forall t \in T, \pi(t) \subseteq \pi(t)$, or equivalently, $\pi \subseteq \pi'$ iff $\forall (t, q) \in T \times Q, \pi(t, q) \leq \pi'(t, q)$.
- 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, called its **replication factor**. The subset of processors on which t is executed is denoted $\pi(t)$. This notation is somehow abusive but it carries the right semantics.

2.2 Fault Model

The processors of Q are assumed to be **fail-silent** [11]. All the failures are transient 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**. This assumption can be met by replicating all the communication links a sufficient number of times to make them significantly more reliable than the processors, and by implementing an appropriate communication protocol able to tolerate transparently their failures (as in [15]).

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 $e^{-\lambda_q.\ell}$. Conversely, the probability that q fails during a time slot of length ℓ is $1 - e^{-\lambda_q.\ell}$. Modern fail-silent processors can have a failure rate in the order of 10^{-6} per hour.

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:

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

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 [42, 40] 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 $(5, 4)$, whose first objective is equal to 5 and whose second objective is equal to 4, is neither better nor worse than solution $(4, 5)$. 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 [25]).

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** [33]. 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 C_{max}. 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 DAG to be scheduled. Figures 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(c)), the precedence constraints can be written as: $\forall (t_i, t_j) \in E, \forall q_j \in \pi(t_j), \sigma(t_j, q_j) \geq \max_{q_i \in \pi(t_i)}(\sigma(t_i, q_i) + p_{t_i q_i} + comm_{t_i q_i t_j q_j})$. 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 [27, 35]). Formally, an RBD is a directed graph (N, 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 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 consists of a single path from S to D, 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. Therefore, computing the reliability of the RBD is *linear* in the number of tasks.

When the spatial 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 (where dI stands for the replica of d placed on processor I). 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 rely on an efficient symbolic encoding of the RBD with BDDs (see, *e.g.*, the AltaRica workbench [13] or the Sharpe tool [17]).

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 depend 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) shows the RBD of the schedule of Figure 1(c), which uses the replication for reliability scheme. The probability that such a RBD is operational can be computed in linear time of the size of the RDB [27, 35].

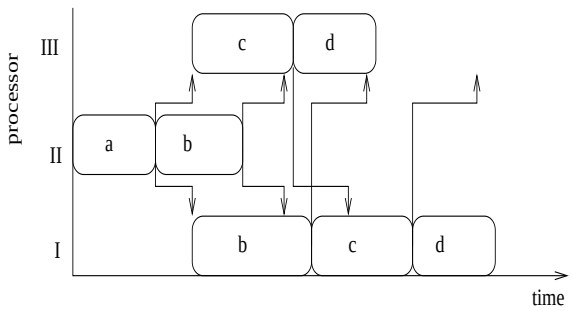
Remark that the reliability of a schedule does not depend on the temporal allocation but only on the spatial allocation. We denote by $\mathcal{P}(\pi)$ the reliability of the spatial allocation π, computed by the following expression (2):

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

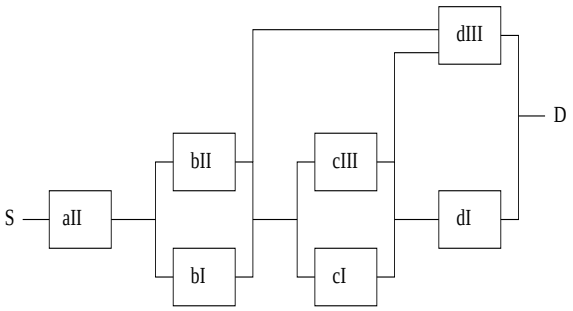
For convenience, let us define $\mathcal{P}(\pi_i) = 1-\prod_{q_j\in\pi(t_i)}\left(1-e^{-\lambda_{q_j}\cdot p_{t_iq_j}}\right)$. Using this notation, we have, $\mathcal{P}(\pi) = \prod_{i=1}^{n}\mathcal{P}(\pi_i)$. This expression stems from the following three hypotheses: fail-silent transient failures, statistically independent failure occurrences, and replication for reliability. Finally, we denote the unreliability of a schedule as $UR(\pi,\sigma) = 1-\mathcal{P}(\pi)$.

3.3 An Impossibility Result on Constant Approximation

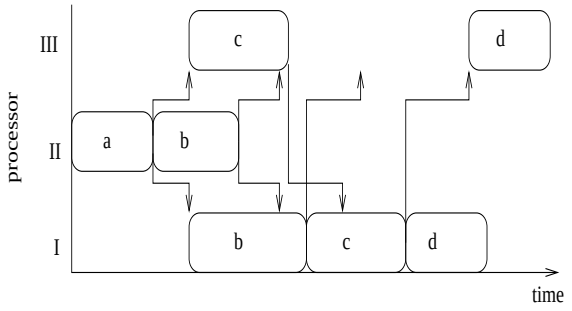
This section proves that there does not exist a "good" compromise solution. [10] contains a similar proof, but for schedules *without* replication. Here, we prove that replication does not help in finding a unique good compromise



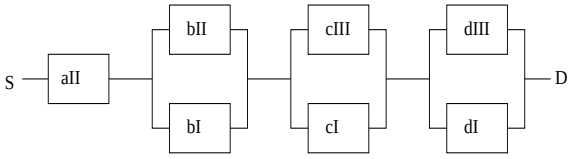
(a) Classical replication scheme.



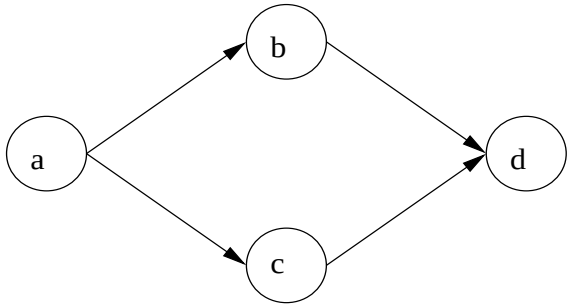
(b) The RBD induced by (a).



(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.

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 all the solutions of I. Finding such an approximation constant ρ is what most works are aiming at.

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

Proof: Consider the instance composed of two processors p_1 and p_2 and one task t, such that the processing time of t on p_1 is 1, while it is k on p_2. The failure rate of p_1 is left unspecified to λ_1, while that of p_2 is equal to $\lambda_2 = \lambda_1/k^2$.

Consider all the three solutions of this instance: S_1 in which t is scheduled on p_1; S_2 in which t is scheduled on p_2; and S_3 were t is scheduled on both p_1 and p_2. Remark that S_1 is optimal for C_{max}, with $C_{max}(S_1) = 1$ and $UR(S_1) = 1 - e^{-\lambda_1 p_1}$. S_2 is close to the optimal for UR, with $C_{max}(S_2) = k$ and $UR(S_2) = 1 - e^{-\lambda_1 p_1/k}$. S_3 is optimal for the UR, 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 application 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 [30, 26] 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 [6] or with caches and pipelines [39]. In particular, it has been applied to the most critical embedded system, namely the Airbus A380 avionics software running on the Motorola MPC755 processor [12, 37].

On transient faults and their negligible duration. Computing the reliability with RBDs requires the assumption that failure occurrences 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. Remark 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** [11], so the statistical independence of faults on different processors is a pertinent assumption. This reliability model is directly borrowed from [34] and is widely used in the literature [4, 22, 29].

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. [21] deals with this problem by adding for each task t, a special synchronization task that depends on all the replicas of all the predecessor tasks of t, and on which depend all the replicas of all the successor tasks of t. By doing that, performances and reliability are degraded in order to gain more tractability.

On the efficiency of the replication for reliability scheme. In the approaches targetting fault-tolerance, a task does not need to wait for the completion of all the replicas of its predecessors. It just needs to wait for the completion of the first replica. But here, we target reliability instead of fault-tolerance; thus, we need to ensure that the reliability block-diagram representing the schedule is serial-parallel in order to make the computation of the reliability tractable. For this reason, we adopted the replication for reliability scheme. One can wonder if this particular scheme induces a significant overhead. In another article, Girault and Kalla [14] have shown that a close scheme only incurs a small overhead on the makespan of the final schedule (less than 4% on average). For this reason, we believe that our replication scheme is realistic in practice.

4 Discussion on Related Approaches

In this section, we first review the main results for independently minimizing the makespan and the unreliability objectives. 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 [25]. 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, and more difficult, problem. The existing results consist mainly of smart heuristics that have no theoretical guarantees. Considering the relaxed hypothesis of tasks independence, there is a known 2-approximate algorithm [16]. Let us now survey briefly the main existing results.

The most popular methods are extensions of list scheduling algorithms for heterogeneous resources. HEFT (Heterogeneous Earliest First Task scheduling) [41] 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 onto the processor that can complete it the soonest. HEFT is often considered as a reference heuristic for running experiments. It has been improved in [43] by using a different ordering of tasks.

The min-min heuristic and its variants [19] 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 approach 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 could easily be adapted for tasks with data dependencies.

Other approaches are based on clustering, which involves grouping tasks that communicate heavily. Tasks grouped together are executed on the same resource. Finally, the starting time of each task is determined, thanks to classical scheduling methods. [5] 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. The maximum reliability is therefore obtained by executing a replica of each task on all processors. 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 thanks to a symbolic encoding of the RBD with BDDs [17, 13].

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 [21]).

[38] 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.

4.3 General Purpose Multi-objective Optimization

In this section, we discuss the main way apporaches in the literature for optimizing several objectives simultaneously. This topic has recently received more and more attention [2, 3, 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 [40].

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 [18].

Most of these approaches lead to a single solution. In contrast, we are proposing a new method that generates several non-dominated solutions, by iteratively using a threshold method applied at different levels (i.e., solution 1).

Only very few works proposed integrated approaches. [28] 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 ε-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. Still, the main idea helps to construct an interesting set of independent solutions.

4.4 Reliability and Makespan Bi-objective Optimization

In [4], 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 thresholds provided by the user.

[8] 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 treated by adapting a list-based heuristic called DLS [36] 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.

[10] 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 times. 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, so the impact on reliability is very limited.

5 A New Bi-objective Scheduling Heuristic

In this section, we present our two-steps heuristic for optimizing both the makespan and the reliability. First, it computes the spatial allocation function in order to set UR lower than a threshold value; then, it computes the temporal allocation function aiming at minimizing C_{max}. It is inspired by [28] 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 impose a threshold on UR. Hence, we start by computing a spatial allocation π that satisfies a fixed UR. As there is a huge number of π that satisfy a given threshold, we will generate them randomly. This will be explained in more detail in 5.2. Spatial allocations are post-optimized thanks to a greedy algorithm that removes the tasks having the longest execution times (see Section 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 Section 5.3.

In summary, our method involves two successive phases: first we compute a spatial allocation π that satisfies the UR 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: Spatial Allocation Scheme

5.2.1 Properties and Algorithm

The following property defines a partial order relationship in the allocation space.

PROPERTY 1 *Let π and π' be two spatial allocations. $\pi \subset \pi' \Rightarrow UR_\pi > UR_{\pi'}$.*

Proof: For any $(t'_i, q'_j) \in \pi' - \pi$, we have $0 \leq \mathcal{P}(t'_i, q'_j) \leq 1$, or equivalently:

$$0 \leq 1 - \mathcal{P}(t'_i, q'_j) \leq 1 \tag{3}$$

Now, according to equation (2), $\mathcal{P}(\pi) = \prod_{i=1}^{n}(1 - \prod_{q_j \in \pi(t_i)}(1 - \mathcal{P}(t_i, q_j)))$. Thanks to equation (3), $0 \leq \prod_{(t'_i,q'_j)\in\pi'-\pi}(1 - \mathcal{P}(t'_i, q'_j)) \leq 1$. Hence, multiplying this term with the term $\prod_{q_j \in \pi(t_i)}(1 - \mathcal{P}(t_i, q_j))$ results in a smaller term:

$$\prod_{(t'_i,q'_j)\in\pi'-\pi}(1 - \mathcal{P}(t'_i, q'_j)) \times \prod_{q_j \in \pi(t_i)}(1 - \mathcal{P}(t_i, q_j)) \leq \prod_{q_j \in \pi(t_i)}(1 - \mathcal{P}(t_i, q_j))$$

As a consequence, $\forall 1 \leq i \leq n$, the term $1 - \prod_{q'_j \in \pi'(t'_i)}(1 - \mathcal{P}(t'_i, q'_j))$ is greater than the term $1 - \prod_{q_j \in \pi(t_i)}(1 - \mathcal{P}(t_i, q_j))$. Therefore, $\mathcal{P}(\pi') \geq \mathcal{P}(\pi)$. ■

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 $\exists t \in T, \mathcal{P}(\pi_t) > \sqrt[n]{1 - UR_0}$.*

Proof: By contradiction. Suppose that, for all task t, we have $\mathcal{P}(\pi_t) \leq \sqrt[n]{1 - UR_0}$. The reliability of the spatial allocation is $\mathcal{P}(\pi) = \prod_{i=1}^{n} \mathcal{P}(\pi_i)$. Since all $\mathcal{P}(\pi_i)$ are probabilities, we have $0 \leq \mathcal{P}(\pi_i) \leq 1$. Thus, $\mathcal{P}(\pi) \leq (\max_{i=1}^{n} \mathcal{P}(\pi_i))^n$. The hypothesis leads to the following bound on the unreliability of the schedule: $UR(\pi) = 1 - \mathcal{P}(\pi) \geq 1 - \sqrt[n]{1 - UR_0}^n \geq UR_0$, which is a contradiction. ■

We now present the `IRSAG` algorithm (Iterative Randomized Spatial Allocation Generator) that constructs a spatial allocation with an unreliability smaller than a threshold UR_0 (see Figure 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 satisfies 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 `IRSAG` algorithm works with any initial allocation π_0, and not only with the empty set.

The `IRSAG` 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)).

Algorithm 1 IRSAG: Spatial allocation generation

input : an instance, a value UR_0 and an allocation π_0
output : an allocation π
begin
  $\pi := \pi_0$;
  $prop3 := false$;
  forall $t \in T$ **do**
    while $\mathcal{P}(\pi_t) < 1 - UR_0$ **do**
      $\pi := \pi \cup (t, alea(1, m))$;
      if $\mathcal{P}(\pi_t) > \sqrt[n]{1 - UR_0}$ **then**
        $prop3 := true$;
      end if
    end while
  end forall
  if $prop3 = false$ **then**
    $t_0 := alea(1, n)$;
    while $\mathcal{P}(\pi_{t_0}) < \sqrt[n]{1 - UR_0}$ **do**
      $\pi := \pi \cup (t_0, alea(1, m))$;
    end while
  end if
  while $(UR(\pi) > UR_0)$ **do**
    if $\pi = T \times Q$ **then**
      return NIL;
    end if
    $\pi := \pi \cup (alea(1, n), alea(1, m))$;
  end while
  return π;
end

Algorithm 2 CommBlevelList

input : an instance and an allocation π
output : an temporal allocation σ
begin
  Let $blevel[t]$ be the longest path from t to the end of the graph.
  forall $t \in T$ in anti-topological order
    if t is a communicating task or a leaf **then**
      $prio[t] := blevel[t]$
    else
      $prio[t] := \max_{t' | \exists (t,t')} prio[t']$
    end if
  end forall
  forall time x from 0 to ∞
    forall $q \in Q$
      if j is idle at x
        Select t ready on q at time x minimizing $prio[t]$
        Schedule t in σ from x to $x + p_t$ on q
      end if
    end forall
  end forall
  return σ
end

5.2.2 Improvement

The IRSAG algorithm returns a spatial allocation that has an unreliability smaller that UR_0. But IRSAG does not ensure that its allocation is the best among such allocations. It is not even minimal by inclusion. Due to the replication model, removing replicas can only lead to a better C_{max}. This is why we propose a procedure called maopt (for Minimal Allocation Optimization) which is basically a greedy local descent. It iteratively removes replicas as long as the unreliability remains below UR_0. Replicas are 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 scheduling a DAG of tasks where tasks are forced to be scheduled on a given processor. This particular problem is the Scheduling with

Preallocation Problem, which is known to be strongly NP-hard [32]. In the following, all replicas are equivalent, in the sense that there is no distinction between two replicas of the same task and two replicas of two different tasks. Therefore, the term "task" refers to a replica on a processor.

Definition 2 below leads to a weakly dominant property for schedules and will help us to solve the problem. We introduce a notion of priority between the tasks 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 any 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 with 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 earlier (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 one using a list scheduling algorithm.

In the literature, list scheduling algorithms are widely used for classical scheduling problem [25]. The principle can be stated as "compute if there is something to compute". In order to break tie, tasks are ordered according to some priority. 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 that 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 greatest 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 focuses on communicating tasks by setting non communicating tasks' priority to the maximum priority of their successors, and communicating tasks' priority to their B-level (see Figure 2). It 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.

Let us now study the approximation ratio of `CommBlevelList`. The following proposition states that no constant approximation of the makespan can be obtained by any List Scheduling algorithm LS.

PROPOSITION 1 *Let LS be a list scheduling algorithm. There exist instances of the pre-allocated scheduling problem such that $C_{max}^{LS} \geq (m-1)C_{max}^{*}$, where C_{max}^{LS} is the makespan obtained by LS and C_{max}^{*} is the optimal makespan.*

Proof: This proposition is proved by constructing an instance that asymptotically reaches the bound $(m-1)$. The result concerns any List Scheduling algorithm. Thus, the schedule should not be function of the priority list of the algorithm. Only one task should be ready at a time per processor.

The constructed instance on m processors is the following. Processor 1 has a single task t_1^1 of processing time $p_1^1 = \varepsilon$. Every other processors q ($2 \leq q \leq m$) have two tasks t_q^1 and t_q^2 of processing time $p_q^1 = \varepsilon$ and $p_q^2 = 1$. Each task t_q^i ($q \neq 1$) is the successor of t_{q-1}^1. The communication delay between t_{q-1}^1 and t_q^1 is ε, while the communication delay between t_{q-1}^1 and t_q^2 is $\frac{\varepsilon}{2}$.

The optimal makespan of this instance is obtained by scheduling all the t_q^1 tasks as soon as possible and then executing all tasks t_q^2 in parallel, except on processor m where task t_m^2 is scheduled before t_m^1. Hence, the optimal makespan is $C_{max}^{*} = (2m - \frac{3}{2})\varepsilon + 1$.

However, according to the principle of list scheduling algorithms, the tasks t_q^2 are executed before the tasks t_q^1 since the corresponding communications are shorter. This postpones the release of the tasks on the next processor. In this case, all tasks are executed sequentially. The resulting makespan is therefore $C_{max}^{LS} = m\epsilon + \frac{(m-1)\epsilon}{2} + m - 1$.

As a consequence, the ratio $\frac{C_{max}^{LS}}{C_{max}^{*}}$ tends to $(m-1)$ when ε tends to 0. ■

In other words, this proposition states that, to reach a constant approximation, an algorithm should be able to wait for important tasks instead of executing unimportant ones. The main problem in the design of approximation algorithms for this problem is the lack of "good" lower bounds of the optimal makespan. Indeed, classical lower bounds, such as the critical path or the maximal load of a processor, are not within a constant factor of the optimal makespan. In our problem, we do not know any criterion to distinguish important tasks from unimportant ones.

5.4 Discussion

We proposed a two phases algorithm, where the first phase generates a spatial allocation and the second phase generates a temporal allocation. This decomposition is pertinent since the reliability of a schedule only depends on the spatial allocation and not on the local ordering of the tasks. In our method, the spatial allocation is generated randomly using suitable properties. One can wonder whether we can expect more from a more sophisticated deterministic algorithm

or not.

The first phase does not give directly the makespan of the schedule. It ensures that the reliability is greater than a given threshold. It is very hard to optimize the makespan in this phase because it remains unknown until the second phase is finished. Thus, the first phase should provide a spatial allocation which will be scheduled efficiently in phase 2. The problem is that the problem tackled in phase 2 is hard, and, as we proved in the last section, classical algorithms like List Scheduling cannot guarantee a constant approximation. Thus, useful lower bounds of the optimal makespan can not be easily determined. The lower bounds we know are not sufficient since we could reach very different makespans with two instances featuring the same values of lower bound. Since we do not know which function to optimize in phase 1, random generation seems to be the only reasonable policy.

6 Experimental Study

In this section, we present experimental results about the method proposed in Section 5. There is no interest in comparing our method with a method that does not use task replication, and, as a consequence, does not improve the reliability by more than one order [10, 9, 8]. 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 the experimental tests are constructed by using a model of random networks, while others are applications taken 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 possible spatial allocation.
- 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 iterations 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 is: 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 $P \mid prec, p_i, c_{ij} \mid C_{max}$ [25]. 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 [24] that considers 602 instances of various structures. This benchmark has been used to test different scheduling algorithms in [7, 23].

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 [31, 9], 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, $\log nm$, and $\sqrt{nm}$, where n is the number of tasks and m is the number of tasks. Also, the `maopt` improvement shown in Section 5.2.2 can be used or not. Finally, the first phase algorithm can be initialized with several allocations: the empty allocation and the HEFT allocation were considered. Remember that the whole method takes a minimum reliability as a parameter.

Experiments are run under the following protocol. First, HEFT [41] is run on each instance. Let ${C_{max}}^{HEFT}$ and UR^{HEFT} be the values obtained on the schedule computed by HEFT. Then, our algorithm is run with different thresholds of UR. Each UR value is obtained by multiplying UR^{HEFT} by values of r, with r taking 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 threshold 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 our simulations 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 [20].

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 experiments 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 the UR factor r, the mean ratio of the makespan to ${C_{max}}^{HEFT}$. In the `maoptiterlog` curve, the point $r = 0.1$ of mean makespan ratio 2 can be interpreted as: On this benchmark, to gain two orders of unreliability, we worsened (in average) the makespan by a factor of 2 (this holds also for point

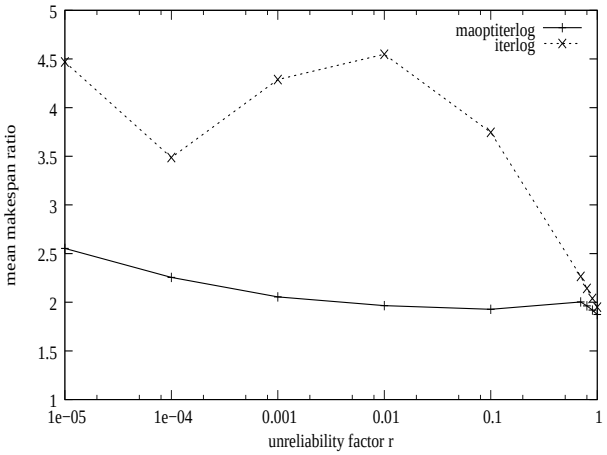


Figure 2: Comparing random allocation with optimization

$r = 0.01$). Our results show that the optimization presented in Section 5.2.2 really improves the makespan, since 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. In other words, 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, and 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 that 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: $\sqrt{nm}$, $\log(nm)$, and 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 seconds with $\sqrt{nm}$ iterations). 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 average value obtained with $r = 0.1$ is rather strange: the average makespan for $r = 0.1$ is better than the average 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.

Despite the efficiency of our algorithm's second phase, the method tested does not give solutions close to the HEFT

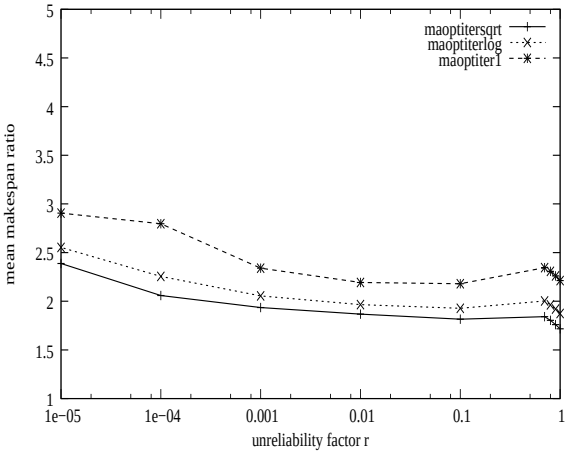


Figure 3: Impact of the number of generation on the makespan

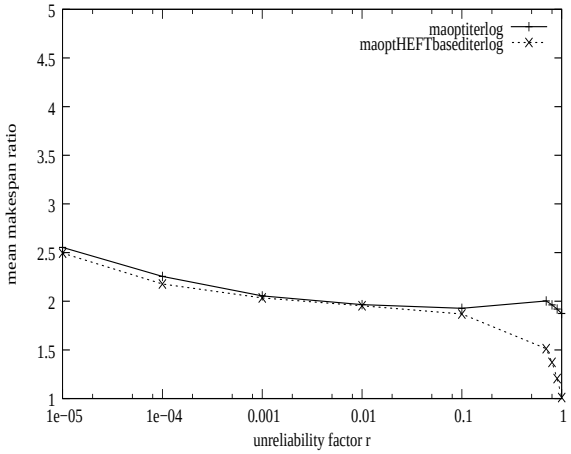


Figure 4: Comparing HEFT based allocation with pure random one

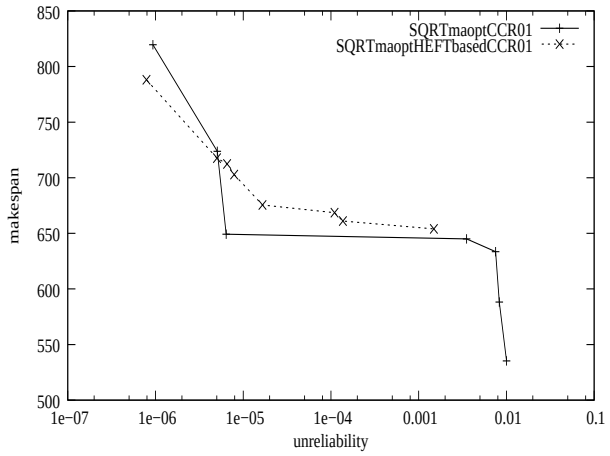
makespan. Therefore, our algorithm's first phase returns spatial allocations that are not "makespan friendly". All those results mean that completely random spatial allocation are bad for the makespan. Basing the random allocation on an existing allocation for the makespan should improve the obtained 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.

Taking a particular instance of our benchmark, 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 depending on the CCR. On one hand, using HEFT results in a worse curve when the CCR is low (Figure 5(a)), while, on the other hand, when the CCR is high, the obtained curve when basing the method on HEFT completely dominates the random one (Figure 5(b)). Figures 5(a) and (b) warn us that using exclusively HEFT can result in missing interesting solutions.

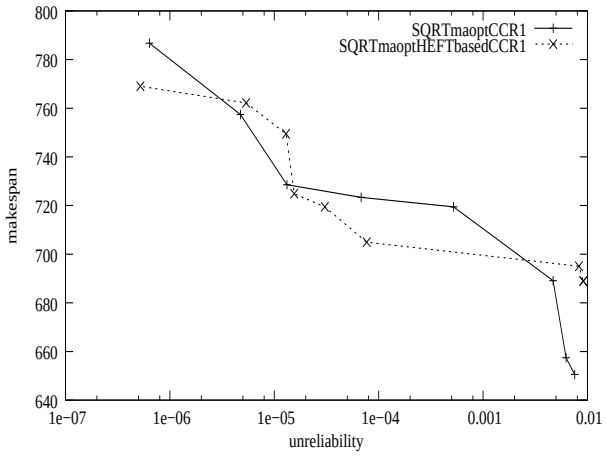
We can remark that, when the CCR is high, improving the reliability has a more significant impact on the makespan than when the CCR is low. This is due to the replication model that implies a lot of communications.

7 Conclusion

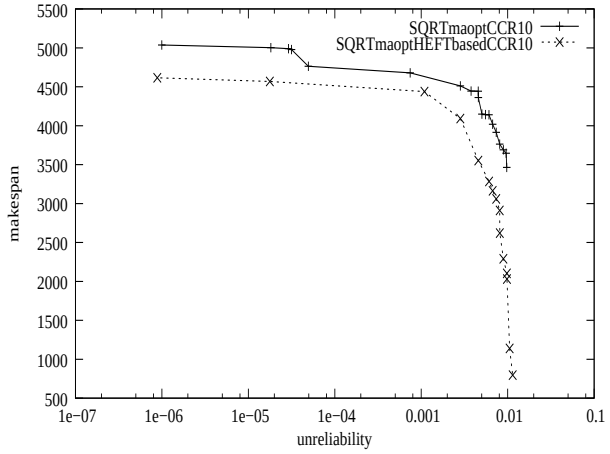
We have presented a new bi-objective scheduling method for task graphs with data-dependencies onto heterogeneous distributed memory architectures. The two objectives considered are the schedule length (real-time constraint) and



(a) CCR=0.1



(b) CCR=1



(c) CCR=10

Figure 5: r150 with different CCR levels

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 efficiently 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. The drawback is that the makespan is slightly degraded.

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 non Pareto-dominated solutions. This will allow the user to choose the solution that fits best his/her applicative requirements.

Our method proceeds in two phases: first a spatial allocation of the tasks onto the processors, and then a temporal allocation to determine at what time each task must 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. During the second phase, we only attempt at minimizing the schedule length.

We have performed extensive simulations of our method, 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 our 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] 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.

[3] 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.

[4] 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.

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

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] A. Girault and H. Kalla. A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate. *IEEE Transactions on Dependable and Secure Computing*, To appear. INRIA research report 6319. http://hal.inria.fr/inria-00177117.

[15] A. Girault, H. Kalla, and Y. Sorel. Transient processor/bus fault tolerance for embedded systems. In *IFIP Working Conference on Distributed and Parallel Embedded Systems, DIPES'06*, pages 135–144, Braga, Portugal, October 2006. Springer-Verlag.

[16] 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.

[17] C. Hirel, R. Sahner, X. Zang, and K.S. Trivedi. Reliability and performability modeling using Sharpe. In *International Conference on Computer Performance Evaluation: Modelling Techniques and Tools, TOOLS'00*, volume 1786 of *LNCS*, pages 345–349. Springer-Verlag, March 2000.

[18] K. Ho. Dual criteria optimization problems for imprecise computation tasks. In Leung [25], chapter 35.

[19] 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.

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

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

[22] 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.

[23] 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.

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

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

[26] 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.

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

[28] 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.

[29] 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.

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

[31] 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.

[32] 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.

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

[34] 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.

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

[36] 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.

[37] 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.

[38] 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.

[39] 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.

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

[41] 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.

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

[43] 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.