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

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

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

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

Abstract

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

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

1 Introduction

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

Our goal is to produce automatically a reliable distributed static schedule of a given algorithm onto a given distributed architecture, which satisfies two criteria: maximize the system’s reliability and minimize the system’s run-time. Concretely, we are given as input a specification of the algorithm to be distributed ($A_l$), a specification of the target distributed architecture ($A_r$), some distribution constraints ($D_i$), some information about the execution times of the algorithm blocks on the architecture processors and the communication times of the algorithm data-dependencies on the architecture communication links ($E_x$), some information about the reliability characteristics of each component of the architecture ($R_e$), a reliability objective ($R_{rel}$), and a run-time objective ($R_{rt}$). The goal is to build a static schedule of $A_l$ on $A_r$, satisfying both objectives $R_{rel}$ and $R_{rt}$, with respect to $E_x$, $D_i$, and $R_e$ (see Figure 1).

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

The majority of hard real-time distributed systems in the literature do not attempt to introduce reliability; rather, they concentrate on the problems that arise from tasks deadline assuming a reliable hardware. For example, the heuristics proposed in [2, 15, 8, 17] are based on static or dynamic allocation and scheduling of tasks to minimize the schedule length. But none of these scheduling heuristics attempt to improve the system’s reliability.
To maximize the system’s reliability in the task allocation problem, the authors of [9, 10, 11] give an explicit reliability expression in terms of system parameters. This expression is used to drive theirs algorithms in search for an allocation that maximizes the reliability. In [9], Shatz et al. present a task allocation model for reliability; failures from processors and communication links are considered to measure the system’s reliability of the proposed algorithm. In [10], Kartik et al. present an improved version of Shatz et al. algorithm, which improves the system’s reliability. In [11], Srinivasan et al. present a cluster-based allocation technique to maximize the reliability in heterogeneous systems. However, none of these heuristics attempts to minimize the length of the generated schedule.

In the literature on bi-criteria scheduling problems, only a few articles consider the reliability property [12, 13, 14, 15]. Taking both reliability and tasks deadline into account, Xiao et al. [16] propose a scheduling algorithm, called eFRCD (efficient Fault-tolerant Reliability Cost Driven Algorithm), based on the reliability model of Shatz et al. [9]. Their algorithm uses a primary-backup copy scheme that enables the system to tolerate the permanent failure of any single processor. However, the tasks deadline criterion has advantage over the reliability criterion. Dogan et al. have proposed a bi-criteria list scheduling heuristics with two objectives, minimizing the schedule length and maximizing the reliability of the obtained schedule [17]. Their cost function considers the reliability of different system-components when making decisions to schedule tasks.

The algorithm that we propose to generate a reliable distributed static schedule, called Reliable Bi-Criteria Scheduling Algorithm (RBSA), is different than the ones proposed in [9] and [16] in the sense that we use the active replication of operations [9] to improve both the system’s reliability and the schedule length (and hence the system’s run-time). Indeed, even though these two objectives are antagonistic, there are situations where replicating some operations actually reduces the schedule length, by improving the locality of computations [9].

The paper is organized as follows. Section 2 gives the system models and assumptions. The bi-criteria scheduling problem is presented in Section 3. Section 4 presents the proposed bi-criteria algorithm RBSA. Section 5 details the performances of RBSA. Finally, Section 6 concludes the paper and proposes future research directions.

2 System models and assumptions

2.1 Architecture model

The architecture is modeled by a graph, where each vertex is a processor, and each edge is a communication link. Classically, a processor is made of one computation unit, one local memory, and one or more communication units, each connected to one communication link. Communication units execute data transfers, called comms. The chosen communication mechanism is the send/receive [8], where the send operation is non-blocking and the receive operation blocks in the absence of data. Figure 2(b) is an example of architecture graph, with four processors P1, P2, P3, and P4, and four point-to-point communications links L12, L23, L24 and L34.

2.2 Algorithm model

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

Figure (a) is an example of algorithm graph, with eight operations: (L1') are sensor operations, (O,O') are actuator operations, while (A,B,C,D) are computation operations. The data-dependencies between operations are depicted by arrows. For instance, the data-dependency A→D corresponds to the sending of some arithmetic result computed by A and needed by D.

2.3 Execution characteristics and distribution constraints

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

For instance, \( \mathcal{E} \) for \( Alg \) and \( Arc \) of Figure (a) is given in Table 1. The point-to-point links L12, L23, L24 and L34 are heterogeneous. The table only gives the WCTT for inter-processor communications. For an intra-processor communication, the WCTT is always 0 time unit.

<table>
<thead>
<tr>
<th>proc.</th>
<th>operation</th>
<th>time</th>
<th>I</th>
<th>I'</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>O</th>
<th>O'</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>2.5</td>
<td></td>
<td>2.5</td>
<td>3.0</td>
<td>2.0</td>
<td>1.5</td>
<td>3.0</td>
<td>3.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>1.5</td>
<td></td>
<td>1.5</td>
<td>1.5</td>
<td>2.0</td>
<td>1.0</td>
<td>0.5</td>
<td>2.0</td>
<td>∞</td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>2.5</td>
<td></td>
<td>2.5</td>
<td>3.0</td>
<td>2.0</td>
<td>1.5</td>
<td>3.0</td>
<td>3.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>1.5</td>
<td></td>
<td>1.5</td>
<td>1.5</td>
<td>2.0</td>
<td>1.0</td>
<td>0.5</td>
<td>2.0</td>
<td>∞</td>
<td></td>
</tr>
</tbody>
</table>

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

2.4 Reliability model

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

<table>
<thead>
<tr>
<th>processors</th>
<th>communication links</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1, P4</td>
<td>L12, L34</td>
</tr>
<tr>
<td>P2</td>
<td>L23, L24</td>
</tr>
<tr>
<td>P3</td>
<td></td>
</tr>
</tbody>
</table>

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

3 The bi-criteria problem

As said in the introduction, our goal is to find a static schedule of \( Alg \) on \( Arc \), satisfying two criteria: the run-time objective \( R_{\text{obj}} \) and the reliability objective \( R_{\text{Rel}} \). In this section, we present in details these two criteria.

3.1 Real-time criterion

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

\[
R_{\text{sched}} = \max_{p_j} \left\{ \max_{o_i} E(o_i, p_j) \right\}
\]

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

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

Our second objective is to generate a reliable schedule, that is, the system reliability $\text{Rel}_{sched}$ must be greater than $\text{Rel}_{obj}$. In order to evaluate the overall reliability of a such systems, we propose to use the Reliability Block Diagrams (RBDs) [5], which are well suited for representing and analyzing the reliability of systems with redundancy. An RBD depicts the components in a system and their connections in terms of functioning requirements.

In systems without replication, the RBD of the schedule does not have a serial/parallel structure (see Figure 3(a)); its reliability can therefore be obtained in linear time by multiplying the reliability of each component of the RBD.

In systems with replication, the RBD of the schedule has a serial structure (see Figure 3(b)); its exact reliability can only be obtained in exponential time. However, we can compute an upper bound of the reliability $\text{Rel}_{sched}$ in polynomial time, thanks to the Minimal Cuts (MCS) method [4]. The MCS is the minimum combination of failures that might cause a system to fail. When processors/links failures are assumed to be independent, the reliability of an MCS $M_i$ is computed as follows:

$$\text{Rel}_{sched}(M_i) = 1 - \prod_{(o,c) \in M_i} (1 - \text{Rel}(o,c))$$

Since cut structures operate in series and components in a cut set operate in parallel, the MCS allows us to compute the upper bound of the system’s reliability in a linear time, as follows:

$$\text{Rel}_{sched}^* \leq \sum_{i=1}^{k} \left(1 - \prod_{(o,c) \in M_i} (1 - \text{Rel}(o,c))\right)$$

4 The reliable bi-criteria scheduling algorithm $RBSA$

We now present our scheduling algorithm $RBSA$ for maximizing the system’s reliability ($\text{Rel}_{sched}$) and minimizing the system’s run-time ($R_{sched}$). We present the algorithm in macro-steps; the superscript number in parentheses refers to the step of the algorithm, e.g., $O_{sched}^{(4)}$. First, we introduce the following notations:
• \( O_{can}^{(n)} \) is the list of candidate operations, built from the algorithm graph vertices. An operation is candidate if all its predecessors are already scheduled.

• \( O_{sched}^{(n)} \) is the list of already scheduled operations.

• \( \text{pred}(o_i) \) is the set of predecessors of operation \( o_i \).

• \( \text{succ}(o_i) \) is the set of successors of operation \( o_i \).

• \( \mathcal{P} \) is the set of all processors of \( \mathcal{A} \).

• \( 2^\mathcal{P} \) is the set of combinations of processors of \( \mathcal{P} \).

• \( Rl_{sched}^{(n)} \) is the length of the temporary schedule at step \( n - 1 \).

• \( Rl_{sched}(o_i, \{p_1, \ldots, p_j\}) \) is the length of the temporary schedule at step \( n \) where the \( j \) replicas \( o_i^1, \ldots, o_i^j \) of \( o_i \) are scheduled respectively on the \( j \) processors \( p_1, \ldots, p_j \).

• \( Rl_{sched}^{(n-1)} \) is the reliability of the temporary schedule at step \( n - 1 \).

• \( Rl_{sched}^{(n)}(o_i, \{p_1, \ldots, p_j\}) \) is the reliability of the temporary schedule at step \( n \) where the \( j \) replicas \( o_i^1, \ldots, o_i^j \) of \( o_i \) are scheduled respectively on the \( j \) processors \( p_1, \ldots, p_j \).

4.1 Algorithm principles

The algorithm that we propose is a greedy list scheduling heuristic [7], called \( RBSA \) (Reliable Bi-Criteria Scheduling Algorithm), which uses a bi-criteria compromise function \( \mathcal{Bf} \) as a cost function to introduce priority between operations to be scheduled. It is based on two functions: the reliability loss \( \mathcal{L} \) and the schedule length gain \( \mathcal{G} \). The first function \( \mathcal{L}^{(n)}(o_i, \{p_1, \ldots, p_j\}) \) computes, at each step \( n \) of the algorithm, the loss on reliability resulting from the scheduling of the \( j \) replicas \( o_i^1, \ldots, o_i^j \) of \( o_i \) respectively on the \( j \) processors \( p_1, \ldots, p_j \) (Figure 5):

\[
\mathcal{L}^{(n)} = \frac{Rl_{sched}^{(n)}(o_i, \{p_1, \ldots, p_j\}) - Rl_{sched}^{(n-1)}}{Rl_{obj} - Rl_{sched}^{(n-1)}}
\]  

(2)

The second function \( \mathcal{G}^{(n)}(o_i, \{p_1, \ldots, p_j\}) \) computes, at each step \( n \) of the algorithm, the gain on the schedule length resulting from the scheduling of the \( j \) replicas \( o_i^1, \ldots, o_i^j \) of \( o_i \) respectively on the \( j \) processors \( p_1, \ldots, p_j \) (Figure 6):

\[
\mathcal{G}^{(n)} = \frac{Rl_{sched}^{(n)}(o_i, \{p_1, \ldots, p_j\}) - Rl_{sched}^{(n-1)}}{Rl_{obj} - Rl_{sched}^{(n-1)}}
\]  

(3)

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

\[
\mathcal{Bf}^{(n)} = \cos(\theta)\mathcal{L}^{(n)} + \sin(\theta)\mathcal{G}^{(n)}
\]  

(4)

Figure 6. Selection of the best choice with \( \mathcal{Bf} \).

Figure 5 illustrates the \( \mathcal{Bf} \) computation for operation \( o_1 \).

We first compute the set \( 2^\mathcal{P} \) of all combinations of all processors of \( \mathcal{P} \). We then compute, for each set \( P \subseteq 2^\mathcal{P} \),
the compromise value $Bf(\theta)(o_1, P_1)$. Graphically, it is the length of the $Bf(\theta)(o_1, P_1)$ vector, whose end is the orthogonal projection of the point $(G(o_1, P_1), L(o_1, P_1))$ onto the line $L = \tan(\theta)G$ (with $\theta = 45^\circ$ here). In the present case, the best compromise value is reached for $P_1 = \{p_1, p_3\}$. As a consequence, $o_1$ is replicated onto two processors, $p_1$ and $p_3$. In general, replicating an operation maximizes the system reliability [24] and minimizes the schedule length [24].

4.2 Our scheduling algorithm \textit{RBSA}

The \textit{RBSA} scheduling algorithm is shown in Figure 4.

Algorithm \textit{RBSA}:

\begin{itemize}
  \item \textbf{input}: $Alg$, $Arc$, $Ect$, $Dis$, $Rel$, $Relobj$, and $\theta$
  \item \textbf{output}: a reliable distributed static schedule of $Alg$ on $Arc$ satisfying $Relobj$ and $Relobj$, or a fails message
  \item \textbf{begin}
  \item Compute the set $2^n$ of all combinations of processors of $P$; let’s the user can limit the degree $k$ of processor combinations $\forall j$
  \item Initialize the lists of candidate and scheduled operations:
    \begin{itemize}
      \item $n := 0$
      \item $O_{\text{cand}} := \{o \in O \mid \text{pred}(o) = \emptyset\}$
      \item $O_{\text{sched}} := \emptyset$
    \end{itemize}
  \item while $O_{\text{cand}} \neq \emptyset$ do
    \begin{itemize}
      \item For each candidate operation $o_{\text{cand}}$, compute $Bf(\theta)$ on each set $P_i$ of $2^n$:
        \begin{equation}
        Bf(\theta)(o_{\text{cand}}, P_k) := \cos(\theta)ct^{(\theta)}(o_{\text{cand}}, P_k) + \sin(\theta)et^{(\theta)}(o_{\text{cand}}, P_k)
        \end{equation}
      \item For each candidate operation $o_{\text{cand}}$, select the best set $P^{o_{\text{cand}}}_{\text{best}}$ such that:
        \begin{equation}
        Bf(\theta)(o_{\text{cand}}, P^{o_{\text{cand}}}_{\text{best}}) := \min_k Bf(\theta)(o_{\text{cand}}, P_k)
        \end{equation}
      \item Select the most urgent candidate operation $o_{\text{urgent}}$ between all $o_{\text{cand}}$ of $O_{\text{cand}}$ such that:
        \begin{equation}
        Bf(\theta)(o_{\text{urgent}}, P^{o_{\text{urgent}}}_{\text{best}}) := \max_k Bf(\theta)(o_{\text{cand}}, P^{o_{\text{cand}}}_{\text{best}})
        \end{equation}
      \item Schedule actively each replica of $o_{\text{urgent}}$ on each processor of $P^{o_{\text{urgent}}}_{\text{best}}$; the implied communications are also implemented actively on the communications links;
      \item Compute the new values $Rel_{\text{sched}}$ and $Rel_{\text{obj}}$;
      \item if $(Rel_{\text{sched}} < Rel_{\text{obj}})$ or $(Rel_{\text{sched}} > Rel_{\text{obj}})$ then return “fails to satisfy objectives” $\forall j$ the user can re-execute the algorithm by changing $\theta$ or the objectives $\forall i$
    \end{itemize}
  \item Update the lists of candidate and scheduled operations:
    \begin{itemize}
      \item $O^{(n)}_{\text{cand}} := O^{(n-1)}_{\text{cand}} \cup \{o_{\text{urgent}}\}$
      \item $O^{(n-1)}_{\text{cand}} = \{o \in O_{\text{cand}} \mid \text{pred}(o) \subseteq O_{\text{sched}}\}$
    \end{itemize}
  \item $n := n + 1$
\end{itemize}
\item \textbf{end while}
\item \textbf{end}
\end{itemize}

Figure 7. The \textit{RBSA} scheduling algorithm.

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

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

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

Finally, we check at the micro-step [5] if the two objectives $Rel_{\text{obj}}$ and $Rel_{\text{obj}}$ are satisfied or not. If they are not, the user can change $\theta$ or the objectives and re-execute the algorithm.

4.3 An example

Figure 4 shows the final reliable schedule produced by \textit{RBSA} with $\theta = 45^\circ$ for the graphs $Alg$ and $Arc$ of Figure 3. The objectives taken for running \textit{RBSA} were $Rel_{\text{obj}} = 16$ and $Rel_{\text{obj}} = 0.999997$. The results obtained by \textit{RBSA} are $Rel_{\text{final}} = 13$ and $Rel_{\text{final}} = 0.9999991$.

Figure 8. The final reliable schedule produced by \textit{RBSA} with $\theta = 45^\circ$.

4.4 Run-time behavior

In order to give the same weight to $L$ and $G$ in the $Bf$ computation, we first take $\theta = 45^\circ$. If the $Rel_{\text{obj}}$ or $Rel_{\text{obj}}$
requirements are not met at the micro-step, then the user can refer to Table 3 to change, Rel_{obj}, or \( R_{obj} \), and re-execute RBSA until both requirements are met.

| RBSA output | table 3. Re-execution strategy for RBSA.
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Rel_{obj} and ( R_{obj} )</td>
<td>( \text{Rel}<em>{obj} ) and ( \text{R}</em>{obj} )</td>
</tr>
<tr>
<td>user action</td>
<td>change Rel_{obj}, ( R_{obj} ), and/or ( t );</td>
</tr>
<tr>
<td></td>
<td>re-execute RBSA;</td>
</tr>
</tbody>
</table>

\( \text{Rel}_{obj} \) (resp. \( \text{R}_{obj} \)) means that \( \text{Rel}_{obj} \) (resp. \( \text{R}_{obj} \)) is not satisfied

4. If both criteria are satisfied, then the user can generate executable code from the final schedule, for instance by using the SYNDEX tool.

Finally, remember that for complexity reasons, we compute in \( Bf \) only the upper bound of the partial schedule’s reliability. Hence, once we have obtained the final schedule, we compute its exact reliability, which we compare to \( \text{Rel}_{obj} \).

5  RBSA time complexity

We compute the time complexity of RBSA as follows. Among the micro-steps, the dominant one is

The computational complexity of the combinations of processors of \( P \) is \( O(n^{k+1}) \), where \( n \) is the number of processors in \( \text{Arc} \) and \( k \) is the degree of maximum processor combinations. Thus, the time complexity of micro-step \( J \) is \( O(Nm^{k+1}) \), where \( N \) is the number of operations in \( \text{Arc} \). Thus, for \( n \) iterations the overall time complexity is \( O(nNm^{k+1}) \). Finally, since exactly one operation is replicated and scheduled at each iteration, \( n = N \), and the total time complexity is therefore \( O(N^2m^{k+1}) \).

6 Performance evaluation

6.1 Simulation parameters

To evaluate RBSA, we have compared its performances with our previous algorithm proposed in [9], called FTBAR (Fault-Tolerance Based Active Replication), and with the algorithm proposed by Hashimoto in [10], called HBP (Height-Based Partitioning). HBP actively replicates all operations once, therefore producing schedules that tolerate one processor failure, while FTBAR actively replicates all operations a fixed number of times, say \( n \), therefore producing schedules that tolerate \( n-1 \) processor failures. We have implemented all three algorithms within the SYNDEX tool [11, 12]. SYNDEX generates automatically executable fault-tolerant distributed code, by first producing a static fault-tolerant distributed schedule of a given algorithm on a given distributed architecture (either with FTBAR, HBP, or RBSA), and then by generating sreal-time fault-tolerant distributed executive implementing this schedule.

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

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

6.2 Performance of \( RBS_A \) against HBP and FTBAR for \( \theta = 0^\circ \)

In this simulation, the architecture graph \( A_{rc} \) was a fully connected network of 4 processors, with the failure rates of processors and communications links given in Table 2. For each schedule, we have computed the normalized schedule length (NSL), obtained by dividing the output schedule length by the sum of the computation costs on the critical-path of each graph [1]. Thus, we have compared the average NSL produced by \( RBS_A \) with those produced by FTBAR and HBP, averaged over 50 random \( A_{rc} \) graphs. To make the comparison fair, FTBAR and HBP were run without any replication of operation.

In Figure 10, we have plotted the average NSL as a function of CCR, for N=100 operations. \( RBS_A \) was run with \( \theta = 0^\circ \), meaning that only \( R_{t, obj} \) was taken into account as objective (i.e., no reliability objective).

![Figure 10. Average NSLs for \( \theta = 0^\circ \) and \( N = 100 \) operations.](image1)

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

6.3 Performance of \( RBS_A \) against HBP and FTBAR for \( \theta = 45^\circ \)

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

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

![Figure 11. Average NSLs for \( \theta = 45^\circ \) and \( N = 100 \) operations.](image2)

![Figure 12. Average reliability for \( \theta = 45^\circ \) and \( N = 100 \) operations.](image3)

For CCR=0.1, all three algorithms have similar performances. For CCR=1, \( RBS_A \), FTBAR and HBP are similar for the NSL, while HBP is slightly less efficient for the reliability. However, for CCR=10, \( RBS_A \) outperforms signif-
icantly FTBAR and HBP both for the NSL and the reliability. This is due to the fact that we use the active replication of Alg’s operations, not only to improve the system’s reliability, but also to improve the locality of computations and hence the schedule length. Not surprisingly, this has more influence when CCR=10 because communications are more expensive compared to computations. Our results indicate that the bi-criteria heuristics of RBSA can meet its two requirements, and still outperform other existing single-criterion heuristics.

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

![Figure 13. Average NSLs for θ = 45° and CCR = 1.](image1)

![Figure 14. Average reliability for θ = 45° and CCR = 1.](image2)

Again, we see that RBSA outperforms both FTBAR and HBP, and that this effect becomes greater when N increases.

7 Conclusion and future work

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

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

Our algorithm can be re-executed if the system’s reliability or run-time objective is not met, by changing the $\theta$ parameter of Bcf, until both objectives are met.

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

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

References


