Reliability and Scheduling on Systems Subject to Failures

Mourad Hakem and Franck Butelle
LIPN – CNRS UMR 7030
Université Paris Nord
Av. J.B. Clément
93430 Villetaneuse
France
{Mourad.Hakem,Franck.Butelle}@lipn.univ-paris13.fr

Abstract

This paper presents a new bi-objective greedy heuristic for scheduling parallel applications on heterogeneous distributed computing systems. The proposed algorithm which is called BSA (Bi-objective Scheduling Algorithm) takes into account not only the time makespan but also the failure probability of the application. Since it is not usually possible to achieve the two conflicting objectives (performance and reliability) simultaneously, a bi-objective compromise function is introduced. BSA has a low time complexity of $O(e|P| + v \log \omega)$, where e and v are respectively the number of edges and tasks in the task graph of the application. $|P|$ is the number of machines (processors) in the system and ω is the width of the task graph. Experimental results show the performance of the proposed algorithm.

Keywords: reliability, scheduling, clustering, distributed computing, precedence task graphs, directed acyclic graphs DAGs, multicriteria scheduling, heterogeneous systems.

1. Introduction

Heterogeneous distributed systems are widely deployed for executing computationally intensive parallel applications with diverse computing needs. The efficient execution of applications in such environments requires effective scheduling strategies that take into account both algorithmic and architectural characteristics to achieve a good mapping of tasks to processors, i.e., to minimize the schedule length (Makespan). In addition, failures of resources (processor/link) in such systems may occur and can have an adverse effect on applications. Consequently, there is an increasing need for developing techniques to achieve maximum reliability of a system, i.e., to minimize the failure probability of a system during execution of an application. Both heterogeneous scheduling and reliability are difficult problems and solve them together makes the problem harder because the two objectives (Makespan/Reliability) conflict. However, it may not be possible to minimize both objectives at the same time: indeed, minimize the failure probability of the application may increase the schedule length of the application. To make scheduling more realistic, we develop in this paper a Bi-objective Scheduling Algorithm (BSA) which satisfies both objectives (maximize the system's reliability and minimize the makespan) simultaneously. The salient feature of our scheme is that:

- it allows the algorithm to be performed on heterogeneous computing systems with minimum time complexity;
- it is based on a bi-objective compromise function that selects best mappings for critical free tasks at each step of the scheduling process;
- it has a very good behavior in practice.

The remainder of the paper is organized as follows: section 2 presents the basic definitions and assumptions adopted in this paper. We recall in section 3 principles of the best existing scheduling algorithms. Section 4 describes the bi-objective scheduling problem. After presentation of our compromise function in section 5, we detail our algorithm in section 6. To experimentally compare our algorithm to [2], we remind its main aspects in section 7. Before concluding, we report in section 8 some experimental results that assess the good behavior of our algorithm.

2. Basic Definitions and Notations

The execution model for task graphs is called macro-dataflow. In the macro-dataflow model, a parallel program is represented as a weighted Directed Acyclic Graph (DAG) which is defined by $G = (V, E)$ where V is the set of task nodes, $v = |V|$ is the number of nodes, E is the set of edges corresponding to the precedence relations between the tasks and $e = |E|$ is the number of edges. Let $\mathcal{V}$ be a cost function on the edges ($\mathcal{V}(t_i, t_j)$ represents the volume of data that task t_i needs to send to task t_j). The length of a path in a DAG is defined as the sum of its edges weights plus the sum of its nodes weights. In the following we will use terms node or task indifferently. In a task graph, a node which does not have any predecessors is called an *entry* node while a node which does not have any successors is called an *exit(sink)* node. Each task first receives all the needed data from its predecessors, computes without interruption and then sends the results to its successors.

A Heterogeneous System is represented by a bounded set $P = \{\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_{|P|}\}$ of processors. These processors are assumed to be fully connected. The link between processors $\mathcal{P}_k$ and $\mathcal{P}_h$ is denoted by ℓ_{kh}. Communication time between two tasks mapped on the same processor is assumed to be zero time units. Computational heterogeneity of a task is modeled by a function $\mathcal{E} : V \times P \to R^+$, which represents the execution time of each task on each processor in the system: $\mathcal{E}(t, \mathcal{P}_k)$ denotes the execution time of t on $\mathcal{P}_k$, $1 \leq k \leq |P|$. On homogeneous systems will simply denote by $\mathcal{E}(t)$ the execution time of a task t. The communicational heterogeneity is expressed by $W(t_i, t_j) = \mathcal{V}(t_i, t_j).d(\mathcal{P}_k, \mathcal{P}_h)$, where $d(\mathcal{P}_k, \mathcal{P}_h)$ is the delay to send a unit length data from $\mathcal{P}_k$ to $\mathcal{P}_h$.

Failures of resources (processors/links) in the system are assumed to be statistically independent and follow a Poisson process with a constant failure rate. We denote by $\lambda(\mathcal{P}_k)$ and $\lambda(\ell_{kh})$ the failure rate of processor $\mathcal{P}_k$ and the failure rate of the link ℓ_{kh} respectively. The mapping matrix $\mathcal{X}$ is an $v \times |P|$ binary matrix representing the mapping of the v tasks of the DAG to the $|P|$ processors. Element $\mathcal{X}_{ik}$ is 1 if task t_i has been mapped to processor $\mathcal{P}_k$ and 0, otherwise.

$\Gamma^-(t)$ and $\Gamma^+(t)$ denote the sets of immediate predecessors and successors of t in G respectively. We call $g(G)$ the *granularity* (the ratio of the computation time to communication time) of the task graph.

If $g(G) \geq 1$ a task graph G is said *coarse grain*, otherwise *fine grain*. For *coarse grain* DAGs each task receives or sends a small amount of communication compared to the computation of its adjacent tasks. During the scheduling process, the graph consists of two parts, the examined (scheduled) tasks S and the unscheduled tasks U. Initially $U = V$.

The bi-objective function is to minimize both the Makespan (schedule length) denoted by $\mathcal{M}(G, \mathcal{X})$ and the failure probability of the application denoted by $\mathcal{F}(G, \mathcal{X})$ simultaneously with minimum time complexity and without violating the precedence constraints among the tasks.

3. Related Work

A large number of algorithms for scheduling and partitioning DAGs have been proposed in the literature. There exists mainly two categories:- Unbounded Number of processors [5, 3, 13, 10, 17] and - Bounded Number of Processors [4, 14, 9, 7, 16]. But all of these assume that the processors in the systems are completely safe.

Reliability has been considered in [11, 8, 15]. They present a task allocation model to maximize the system's reliability in heterogeneous systems. However, none of these heuristics attempts to minimize the time makespan of the application.

Only a few papers in the literature [2, 12, 6, 1], deals with both objectives (performance and reliability). In [12] the two objectives are not considered simultaneously, this algorithm, first tries to guarantee the timing constraints (deadlines) of the tasks. Then, among the processors on which task's deadline is guaranteed, the task is mapped to a processor which minimizes the failure probability of the application. However, the tasks deadline has advantage over the reliability objective. The proposed algorithm in [1] uses the active replication of tasks to improve reliability. It is based on the scheduling algorithm presented in [16]. [6] addresses the problem of allocating periodic real time tasks. The probability of meeting task's deadline is used as the objective function. To our knowledge, the RDLS algorithm [2] is the only one to address the bi-objective scheduling problem presented in this study. It is based on the Dynamic Scheduling Algorithm (DLS) developed by Sih and Lee in [14]. The objective function used in [2] has an important disadvantage: time makespan requirements may dominates the second objective (the failure probability of the application). A brief description of this algorithm is given in section 7, before comparing it to our algorithm in section 8.

4. The Bi-objective Scheduling Problem

In this section, we present in details the two objectives.

Our goal in this study is to find a schedule of the DAG $G = (V, E)$, satisfying two objectives: minimizing makespan $\mathcal{M}(G, \mathcal{X})$ and minimizing failure probability $\mathcal{F}(G, \mathcal{X})$ under task mapping $\mathcal{X}$. Thus, a bi-objective scheduling problem is formalized as follow:

$$\begin{array}{rl}
\text{Minimize} & \begin{pmatrix} \mathcal{M}(G, \mathcal{X}) \\ \mathcal{F}(G, \mathcal{X}) \end{pmatrix} \\
\text{Such That:} & G = (V, E), v = |V|, \\
& \mathcal{X}_{ik} \in \{0, 1\}, 1 \leq i \leq v, 1 \leq k \leq |P|, \\
& \sum_{k=1}^{|P|} \mathcal{X}_{ik} = 1, \forall t_i \in V, \\
& (t_i, t_j) \in E \Rightarrow t_i \prec t_j
\end{array}$$

Where $t_i \prec t_j$ means that task t_i has to be executed before task t_j.

4.1. The First Objective: Minimize Makespan

The HSA (Heterogeneous Scheduling Algorithm) algorithm that we propose in this work is a greedy scheduling heuristic based on attribute priority called *Task Criticalness* which is essentially defined to be the length of the longest path passing through free tasks (A task is called free if it is unscheduled and all of its predecessors are scheduled) in the current partially mapped DAG. Recall that the length of the path is the sum of the weights of both nodes and edges from a source (entry) node to a sink (exit) node. Once a task t is scheduled, it is mainly associated with the following values: a processor $\mathcal{P}(t)$, a start time $\mathcal{S}(t,\mathcal{P})$ and a finish time $f(t,\mathcal{P})$.

A *Critical Task* is defined as the one with the highest priority. Priority of a task t is computed by the sum $L^-(t) + L^+(t)$, where $L^-(t)$ and $L^+(t)$ are respectively the *dynamic top level* and the *static bottom level* of a free task t. They are computed as follows:

- if $\Gamma^-(t) = \phi$ then $L^-(t) \leftarrow 0$.
  Otherwise,
  $\forall\, t \in V, L^-(t) \leftarrow \max\limits_{\substack{t^* \in \Gamma^-(t)\\ \mathcal{P}=\mathcal{P}(t^*)}} \left\{f(t^*,\mathcal{P}) + W(t^*,t)\right\}$
- if $\Gamma^+(t) = \emptyset$ then $L^+(t) \leftarrow \mathcal{E}(t)$.
  Otherwise,
  $\forall\, t \in V, L^+(t) \leftarrow \max\limits_{t^* \in \Gamma^+(t)} \{\mathcal{E}(t)+W(t,t^*)+L^+(t^*)\}$

Note that the finish time of t on $\mathcal{P}$ is:

$$f(t,\mathcal{P}) = \mathcal{S}(t,\mathcal{P}) + \mathcal{E}(t,\mathcal{P})$$

$\mathcal{S}(t,\mathcal{P})$ satisfies the following conditions: - it is later than the time when all messages from t's predecessors arrive on processor $\mathcal{P}$ and it is later than $r(\mathcal{P})$ (the ready time of $\mathcal{P}$). Thus, the start time of t on $\mathcal{P}$ is:

$$\mathcal{S}(t,\mathcal{P}) = \max\Big(L^-(t,\mathcal{P}), r(\mathcal{P})\Big)$$

The *Task Criticalness* definition adopted in this work provides a good measure of the task importance, because the greater the *criticalness* is, the more work is to be performed along the path containing a task. At each step of the mapping process, HSA selects a critical free task and schedules it to a processor that allows the *minimum finish time*. HSA takes into account the computational heterogeneity of the system and is designed with the following objectives:

- To compute task priorities accurately in order that critical tasks will finish earlier;
- To have a lower time complexity compared to other algorithms in the literature.

We maintain a priority list α (that contains free tasks) which is implemented by using a balanced search tree data structure (*AVL*). At the beginning, α is empty. The Head function $\mathcal{H}(\alpha)$ returns the first task in the sorted list α, which is the task with the highest priority (ties are broken randomly).

The number of tasks that can be simultaneously free at each step in the scheduling process is bounded by the *width* (ω) of the task graph (the maximum number of tasks that are independent in G). This, implies that $|\alpha| \leq \omega$.

Algorithm 4.1 The HSA algorithm

```
1:  Compute $L^+(t)$ for each task $t$ and set $L^-(t) = 0$ for each entry task $t$;
2:  $P = \{\mathcal{P}_1, \mathcal{P}_2, \ldots \mathcal{P}_{|P|}\}$; (*the set of processors in the given system*)
3:  $S = \emptyset$ ; $U = V$ ; (*Mark all tasks as unscheduled*)
4:  Put entry tasks in $\alpha$;
5:  while $U \neq \emptyset$ do
6:      $t \leftarrow \mathcal{H}(\alpha)$ ; (*Select a free task with the highest priority from $\alpha$ *)
7:      Compute $f(t,\mathcal{P})$ on each processor
            $f(t,\mathcal{P}) \leftarrow \mathcal{S}(t,\mathcal{P}) + \mathcal{E}(t,\mathcal{P})$;
8:      $\mathcal{P}(t) \leftarrow \mathcal{P} \mid f(t,\mathcal{P}) = \min_{\mathcal{P}_k \in P} \{f(t,\mathcal{P}_k)\}$;
9:      Schedule $t$ to the corresponding processor;
10:     Put $t$ in $S$ and update the priority values of $t$'s successors;
11:     Put $t$'s free successors in $\alpha$;
12:     $U \leftarrow U \backslash \{t\}$;
13: end while
```

4.2. The Second Objective: Minimize Failure Probability

4.2.1. Reliability Model: Successful execution of the task graph (DAG) requires that each resource (processor/link) be operational during the time that its mapped tasks are executing or are in communication. We do not consider failures of processors during an idle time because they only affect the task's finish time, not the systems' reliability. If a processor fails during an idle time, it will be replaced by a spare unit, and such a failure is not critical.

i) *Processor reliability:* Under a task mapping $\mathcal{X}$, the reliability of a processor $\mathcal{P}_k$ for the executions of the tasks mapped to it during the mission is:

$$\mathcal{R}_{\mathcal{P}_k}(G,\mathcal{X}) = e^{-\lambda(\mathcal{P}_k) \sum\limits_{i=1}^{v} \mathcal{X}_{ik}\mathcal{E}(t_i,p_k)}$$

The summation gives the time spent in executing tasks on $\mathcal{P}_k$.

ii) *Link reliability:* Similarly, for a link ℓ_{kh} between $\mathcal{P}_k$ and $\mathcal{P}_h$, we have

$$\mathcal{R}_{\ell_{kh}}(G,\mathcal{X}) = e^{-\lambda(\ell_{kh})\sum\limits_{i=1}^{v}\sum\limits_{j=1}^{v}\mathcal{X}_{ik}\mathcal{X}_{jh}W(t_i,t_j)}$$

The summation denotes the time required to perform the communication between $\mathcal{P}_k$ and $\mathcal{P}_h$.
iii) *System reliability:* The system reliability denoted by $\mathcal{R}(G,\mathcal{X})$ under mapping $\mathcal{X}$ is defined to be the probability that the system will not fail during the time that it is executing the task graph G. Thus, the probability of the system not to fail is:

$$\mathcal{R}(G,\mathcal{X}) = \prod_{k=1}^{|P|}\mathcal{R}_{\mathcal{P}_k}(G,\mathcal{X}).\prod_{k=1}^{|P|}\prod_{h=1}^{|P|}\mathcal{R}_{\ell_{kh}}(G,\mathcal{X}) = e^{-\mathcal{C}(G,\mathcal{X})} \quad (1)$$

Where the reliability cost $\mathcal{C}(G,\mathcal{X})$ is given by the following formula:

$$\mathcal{C}(G,\mathcal{X}) = \sum_{k=1}^{|P|}\sum_{i=1}^{v}\lambda(\mathcal{P}_k)\mathcal{X}_{ik}\mathcal{E}(t_i,\mathcal{P}_k) + \sum_{k=1}^{|P|}\sum_{h=1}^{|P|}\sum_{i=1}^{v}\sum_{j=1}^{v}\lambda(\ell_{kh})\mathcal{X}_{ik}\mathcal{X}_{jh}W(t_i,t_j) \quad (2)$$

As a result, the failure probability of the system under task mapping $\mathcal{X}$ is:

$$\mathcal{F}(G,\mathcal{X}) = 1 - e^{-\mathcal{C}(G,\mathcal{X})}$$

The first term in (2) reflects the unreliability caused by the execution of the tasks on their corresponding processors. This tells us that mapping tasks with greater execution times to more reliable processors might be a good heuristic to increase the reliability. The second term reflects the unreliability caused by the inter-processor communication. Thus, mapping larger volumes of data to more reliable links is a good approach to decrease the system's overall failure probability.

The impact of scheduling t, on $\mathcal{P}$ on the failure probability of the application, is given as follow:

$$\xi(t,\mathcal{P}) = \lambda(\mathcal{P})\mathcal{E}(t,\mathcal{P}) + \sum_{\substack{t_i\in\Gamma^-(t)\\ k\in\mathcal{P}(t_i)}}\lambda(\ell_{k\mathcal{P}(t)})\mathcal{X}_{ik}W(t_i,t)$$

From equation (2), it is clear that in order to increase the reliability $\mathcal{R}(G,\mathcal{X})$, we have to reduce $\mathcal{C}(G,\mathcal{X})$ as much as possible. We can achieve this goal by employing a greedy strategy of task selection and processor selection. We propose the Reliable Scheduling Algorithm (algorithm 4.2) which takes only reliability into account.

Algorithm 4.2 The *RSA* algorithm

```
1:  Compute $L^+(t)$ for each task $t$ and set $L^-(t) = 0$ for each entry task ;
2:  $P = \mathcal{P}_1, \mathcal{P}_2, \ldots \mathcal{P}_{|P|}$; (*the set of processors in the given system *)
3:  $S = \emptyset$ ; $U = V$ ; (*Mark all tasks as unscheduled*)
4:  Put entry tasks in $\alpha$;
5:  while $U \neq \emptyset$ do
6:      $t \leftarrow \mathcal{H}(\alpha)$ ; (*Select a free task with the highest priority from $\alpha$ *)
7:      Calculate $\xi(t,\mathcal{P})$ on each processor
            $\xi(t,\mathcal{P}) \leftarrow \lambda(\mathcal{P})\mathcal{E}(t,\mathcal{P}) + \sum_{t_i\in\Gamma^-(t),\, k\in\mathcal{P}(t_i)} \lambda(\ell_{k\mathcal{P}(t)})\mathcal{X}_{ik}W(t_i,t)$;
8:      $\mathcal{P}(t) \leftarrow \mathcal{P} \mid \xi(t,\mathcal{P}) = \min_{\mathcal{P}_k\in P}\{\xi(t,\mathcal{P}_k)\}$;
9:      Schedule $t$ to the corresponding processor;
10:     Put $t$ in $S$ and update the priority values of $t$'s successors;
11:     Put $t$'s free successors in $\alpha$;
12:     $U \leftarrow U\backslash\{t\}$;
13: end while
```

4.2.2. Complexity Analysis of HSA and RSA

Theorem 4.1 *The time complexity of HSA and RSA is* $O(e|P| + v\log\omega)$.

Proof: Computing $L^+(t)$ (line 1) takes $O(e+v)$. Insertion or deletion from α costs $O(\log|\alpha|)$ where $|\alpha| \leq \omega$. Since each task in a DAG is inserted into α once and only once and is removed once and only once during the entire execution of HSA and RSA, the time complexity for α management is in $O(v\log\omega)$. The main computational cost of HSA and RSA is spent in the while loop (Lines 5 to 13). This loop is executed v times. Line 6 costs $O(\log\omega)$ for finding the head of α. Line 7 costs $O(|\Gamma^-(t)||P|)$ in examining the immediate predecessors of task t on each processor $\mathcal{P}_k \mid k = 1\ldots|P|$. For the whole v loops the cost for this line is at most $\sum_{i=1}^{v} O(|\Gamma^-(t)||P|) = O(e|P|)$. Line 10 costs $O(|\Gamma^+(t)|)$ to update the priority values of the immediate successors of t, and similarly the cost for the v loops of this line is $O(e)$. Thus the total cost of HSA and RSA is $O(e|P| + v\log\omega)$. $\square$

5. The Compromise function

In this section, we present our compromise function for minimizing the two objectives simultaneously. The HSA algorithm is modified to take into account reliability of the resources in the system. The new modified algorithm that we propose is a greedy heuristic and will be referred to as the Bi-objective Scheduling Algorithm (BSA). In a BSA, a bi-objective compromise function $\mathcal{D}(t,\mathcal{P})$ between $f(t,\mathcal{P})$ and $\xi(t,\mathcal{P})$ (reliability cost of t if scheduled on $\mathcal{P}$) is used as a cost function to select the best processor $\mathcal{P}^{best}$ to which a free critical task t is scheduled at each step in the mapping process. The compromise function is defined to be:

$$\mathcal{D}(t,\mathcal{P}) = \sqrt{\theta\left(\frac{f(t,\mathcal{P})}{\max\limits_{\mathcal{P}_k\in P} f(t,\mathcal{P}_k)}\right)^2 + (1-\theta)\left(\frac{\xi(t,\mathcal{P})}{\max\limits_{\mathcal{P}_k\in P} \xi(t,\mathcal{P}_k)}\right)^2}$$

Where $\theta \in [0,1]$ is a weighting parameter that is introduced with the aim of being able to privilege one of the objectives compared to the other. Consequently:

- If $\theta = 0$, BSA $\xrightarrow{reduces\ to}$ HSA;
- If $\theta = 1$, BSA $\xrightarrow{reduces\ to}$ RSA;
- If $\theta = 0.5$, the two objectives have the same importance.

Thus, the selected best processor $\mathcal{P}^{best}(t)$ is obtained as follow:

$$\mathcal{P}^{best}(t) \leftarrow \mathcal{P} \text{ such that } \mathcal{D}(t,\mathcal{P}) = \min_{\mathcal{P}_k\in P}\left\{\mathcal{D}(t,\mathcal{P}_k)\right\} \quad (3)$$

Since the ranges of values that $f(t,\mathcal{P})$ and $\xi(t,\mathcal{P})$ can take are different, they are normalized by dividing them by their respective $\max(\text{values})$, before combining them inside the compromise function $\mathcal{D}(t,\mathcal{P})$. The advantage of such a normalization is that $f(t,\mathcal{P})$ cannot dominate $\xi(t,\mathcal{P})$ or vice-versa during the scheduling process.

6. Our Scheduling Algorithm BSA

Algorithm 6.1 The *BSA* algorithm

```
1:  Compute $L^+(t)$ for each task $t$ and set $L^-(t) = 0$ for each entry task ;
2:  $P = \mathcal{P}_1, \mathcal{P}_2, \dots \mathcal{P}_{|P|}$; (*the set of processors in the given system*)
3:  $S = \emptyset$ ; $U = V$ ; (*Mark all tasks as unscheduled*)
4:  Put entry tasks in $\alpha$;
5:  while $U \neq \emptyset$ do
6:      $t \leftarrow \mathcal{H}(\alpha)$ ; (*Select a free task with the highest priority from $\alpha$ *)
7:      Calculate $\left(\max_{\mathcal{P}_k\in P} f(t,\mathcal{P}_k)\right)$ and $\left(\max_{\mathcal{P}_k\in P} \xi(t,\mathcal{P}_k)\right)$ values;
8:      Calculate $\mathcal{D}(t,\mathcal{P})$ on each processor $\mathcal{P}_k \in P$ ;
9:      $\mathcal{P}^{best}(t) \leftarrow \mathcal{P} \mid \mathcal{D}(t,\mathcal{P}) = \min_{\mathcal{P}_k\in P}\{\mathcal{D}(t,\mathcal{P}_k)\}$;
10:     Schedule $t$ to the corresponding processor;
11:     Put $t$ in $S$ and update the priority values of $t$'s successors;
12:     Put $t$'s free successors in $\alpha$;
13:     $U \leftarrow U \backslash \{t\}$;
14: end while
```

Theorem 6.1 *The time complexity of BSA is* $O(e|P| + v\log\omega)$.

Proof: Computing $L^+(t)$ (line 1) takes $O(e+v)$. Insertion or deletion from α costs $O(\log|\alpha|)$ where $|\alpha| \le \omega$. Since each task in a DAG is inserted into α once and only once and is removed once and only once during the entire execution of *BSA*, the time complexity for α management is in $O(v\log\omega)$. The main computational cost of *BSA* is spent in the while loop (Lines 5 to 14). This loop is executed v times. Line 6 costs $O(\log\omega)$ for finding the head of α. Line 7 and 8 costs $O(|\Gamma^-(t)||P|)$ in examining the immediate predecessors of task t on each processor $\mathcal{P}_k \mid k = 1\dots|P|$, while computing $\left(\max_{\mathcal{P}_k\in P} f(t,\mathcal{P}_k)\right)$, $\left(\max_{\mathcal{P}_k\in P} \xi(t,\mathcal{P}_k)\right)$ and $\mathcal{D}(t,\mathcal{P}_k)$ values on each processor. For the whole v loops the cost for these lines is at most $\sum_{i=1}^{v} O(|\Gamma^-(t)||P|) = O(e|P|)$. Line 11 costs $O(|\Gamma^+(t)|)$ to update the priority values of the immediate successors of t, and similarly the cost for the v loops of this line is $O(e)$. Thus the total cost of *BSA* is $O(e|P| + v\log\omega)$. $\square$

7. A Brief Description of RDLS Algorithm

In order to compare our algorithm to RDLS algorithm proposed in [2], which is the only algorithm to our knowledge that addresses the bi-objective scheduling problem discussed in this study, we give in the following section a brief description of this algorithm. The RDLS (Reliable Dynamic Level Scheduling) algorithm is based on an existing scheduling algorithm DLS (Dynamic Level Scheduling [14]). The DLS algorithm computes the *Dynamic Level* (DL) value for all free tasks on all processors. The task-processor pair which gives the largest value of DL is selected for scheduling. $DL(t_i,\mathcal{P}_k)$, is defined to be:

$$\begin{aligned} DL(t_i,\mathcal{P}_k) &= SL(t_i) \\ &\quad - \max\{L^-(t_i,\mathcal{P}_k), r(\mathcal{P}_k)\} + \Delta(t_i,\mathcal{P}_k) \end{aligned} \quad (4)$$

The first term in (4) is called the static level of the task. It is defined to be the longest sum of the median execution time of the tasks along any directed path from t_i to an exit task. The max term defines the time when task t_i can begin execution on processor $\mathcal{P}_k$. The third term accounts for the processor speed differences and is defined to be:

$$\Delta(t_i,\mathcal{P}_k) = \widehat{\mathcal{E}}(t_i) - \mathcal{E}(t_i,\mathcal{P}_k)$$

where $\widehat{\mathcal{E}}(t_i)$ denotes the median execution time of task t_i across all processors in the system and $\mathcal{E}(t_i,\mathcal{P}_k)$ denotes the execution time of t_i on $\mathcal{P}_k$.
To take reliability measures into account the authors incorporate a new cost function term called $C(t_i,\mathcal{P}_k)$ as follows:

$$DL'(t_i,\mathcal{P}_k) = DL(t_i,\mathcal{P}_k) - C(t_i,\mathcal{P}_k) \quad (5)$$

The new term promotes resources with high reliability to maximize the reliability of the application. The idea of the RDLS algorithm is that it seeks a task-processor pair with highest dynamic level given by 5.

8. Experimental results

To evaluate the performance of BSA algorithm, series of simulations have been carried out. Due to the NP-completeness of this scheduling problem, the proposed algorithm cannot always lead to an optimal solution. Thus it is necessary to test its performance using randomly generated graphs.

The parameters used in the experimental study are chosen in such a way that they are representative and cover a wide spectrum of real-life parallel applications. They are based on those used in the literature (see for example [2, 11, 8, 15, 12]). In addition a random DAG generator (see [5, 4]) used in this study generates DAGs that are quite close to those occuring in practical application. The failures rates of processors and links are assumed to be uniformly distributed from $5 \times 10^{-6}/h$ to $15 \times 10^{-6}/h$ and from $15 \times 10^{-6}/h$ to $30 \times 10^{-6}/h$ respectively. The worst-case execution time of each task in the DAG is assumed to be uniformly distributed between 80 and 120 units of time, where the executions times of a given task are different on different processors. The granularity of the task graph is 1.0, the number of processors is set to 40 and $\theta = 0.5$ (the two objectives have the same importance). To model the communicational heterogeneity of the system, the unit message delay of the links and the message volume between two tasks are chosen uniformly from the ranges $[5, 15]$ and $[10, 100]$ respectively. Each point in all figures represent the mean of executions on 30 random graphs. The time makespan and the reliability cost are the two main metrics to test the performance of BSA algorithm. The simulation studies are mainly grouped into six sets:

***a*) Performance comparison between BSA, RSA and HSA:** First, we compare the performance of BSA with our algorithms HSA and RSA presented previously in section 4. From Fig. 1, it is clear that we deal with two conflicting objectives. RSA algorithm minimizes well the reliability cost of the application but it produces task mappings that increase the time makespan of the application. Conversely, HSA algorithm produces better schedule lengths, but it suffers in terms of reliability. This experiment demonstrates that BSA algorithm alleviates this problem by finding better compromise solutions between the two objectives. This is due to the greedy processor selection adopted by our algorithm. It is guided by the compromise function which does not allow to make one objective dominates the other. The processors are selected in such a way that $\mathcal{D}(t, \mathcal{P})$ value is minimized.

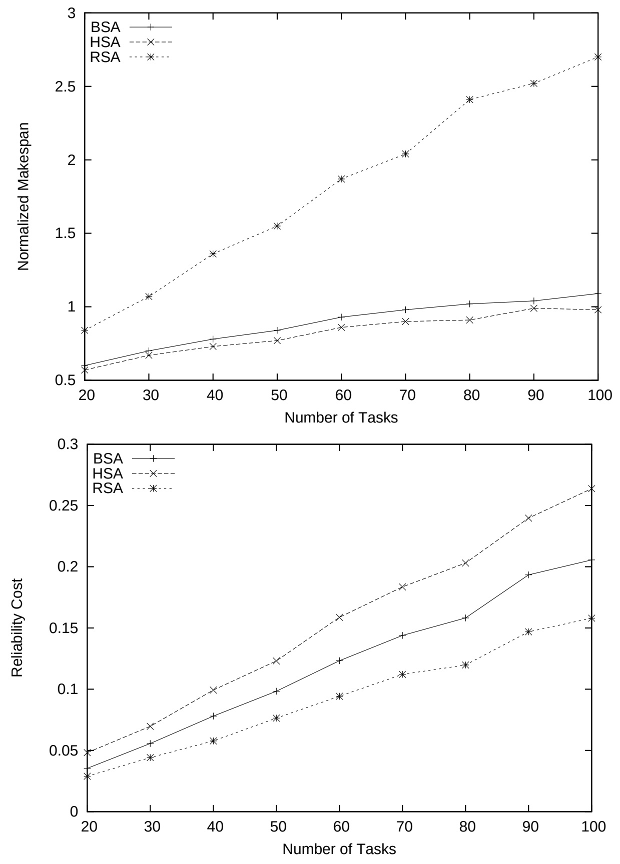


Figure 1. Average normalized makespan and reliability cost comparison between BSA, HSA and RSA

***b*) Effect of granularity:** In this simulation, the number of tasks is chosen uniformly from the range $[80, 120]$. The granularity of the task graph is varied from 0.2 to 2.0, with increments of 0.2. Fig. 2 shows that both time makespan an reliability increase as the granularity increases. This is due to the fact that the increase of the tasks' execution times leads to an increase of both time makespan and tasks' reliability costs.

***c*) Effect of computational heterogeneity:** Computational heterogeneity is modeled by varying the execution times of tasks. Five sets of execution times with the same average value, are used in the simulation study: $[90, 110], [70, 130], [50, 150], [30, 170]$ and $[10, 190]$. These ranges reflect 5 different levels of heterogeneity. In this experiment , the number of tasks is cho-

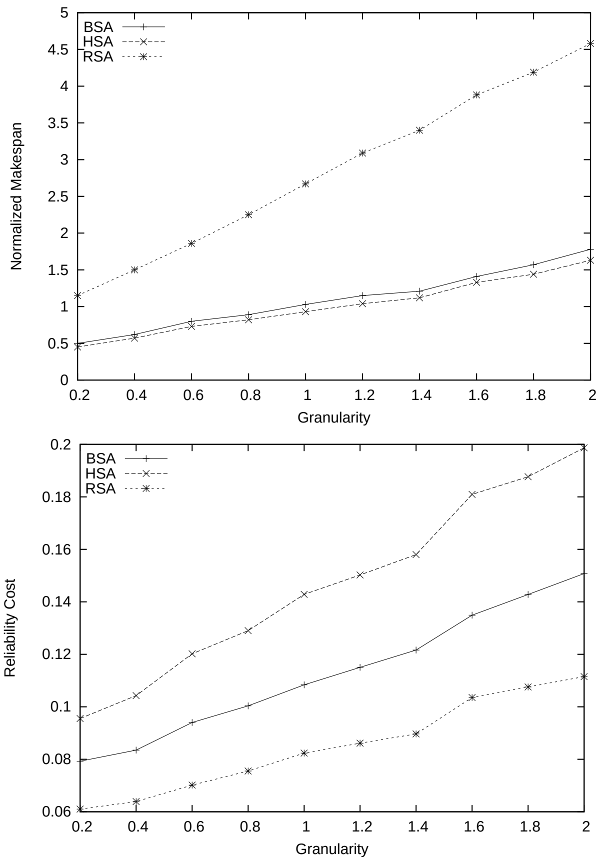


Figure 2. Impact of the granularity on reliability cost and makespan

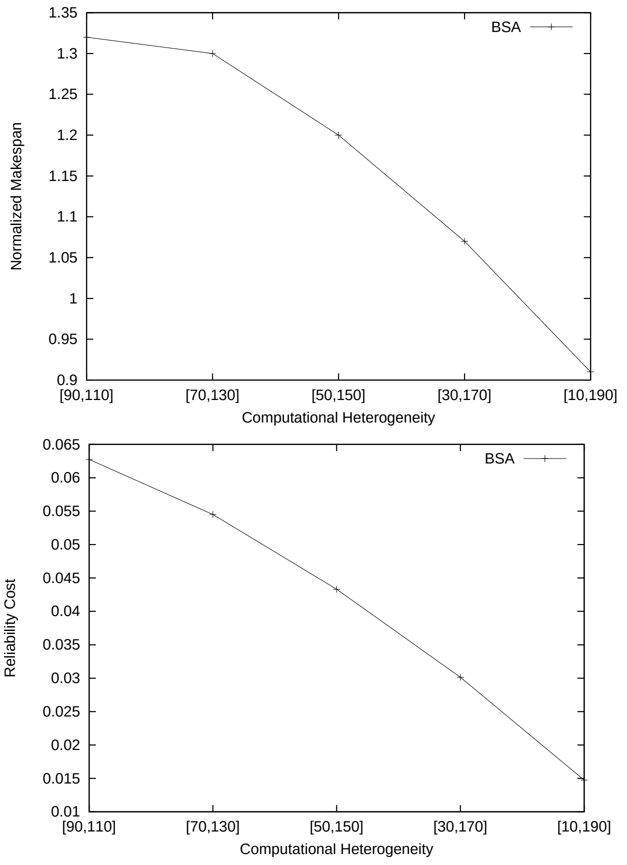


Figure 3. Impact of computational heterogeneity on reliability cost and makespan

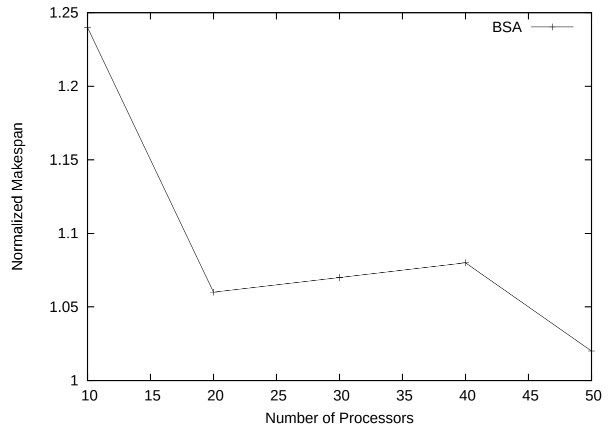
sen uniformly from the range $[80, 120]$. As we can see from Fig. 3, for both time makespan and reliability measure the BSA algorithm has better performance on systems with higher computational heterogeneity. This can be explained by the fact that the tasks' reliability costs and finish times decrease accordingly when the heterogeneity of the systems increases.

***d*) Effect of system sizes:** To test the impact of the number of processors on the performance of BSA algorithm, we chose the number of tasks uniformly from the range $[80, 120]$ and increased $|P|$ from 10 to 50. In Fig. 4, we observe that the performance of BSA algorithm in terms of reliability improves as the system size increases. This is because for large number of processors, the BSA algorithm has more choices for scheduling tasks. However, system sizes does not make a significant effect on the time makespan, because processors on which tasks are mapped are not usually fastest.

***e*) Performance comparison between BSA and RDLS:** For the experimental performance comparison between BSA and RDLS we do not take link failures into account, because the *link reliability computation* of both algorithms is different. However, this do not change performances of these algorithms. Thus the overall reliability cost of the application becomes:

$$\mathcal{C}(G, \mathcal{X}) = \sum_{k=1}^{|P|} \sum_{i=1}^{v} \lambda(\mathcal{P}_k)\mathcal{X}_{ik}\mathcal{E}(t_i, \mathcal{P}_k)$$

Comparing the results of our algorithm to the results of RDLS [2], we can find that the RDLS is very easy to fail (see Fig. 5). The time makespan produced by RDLS is close to that produced by HSA algorithm (this is the reason for which we take HSA in this set of study). This is explained by the fact that the ranges of values of $DL(t, \mathcal{P})$ and $C(t, \mathcal{P})$ can take are different. As a result, the value of $DL(t, \mathcal{P})$ dominates the value of $C(t, \mathcal{P})$. To cure this problem, both objective values must be normal-



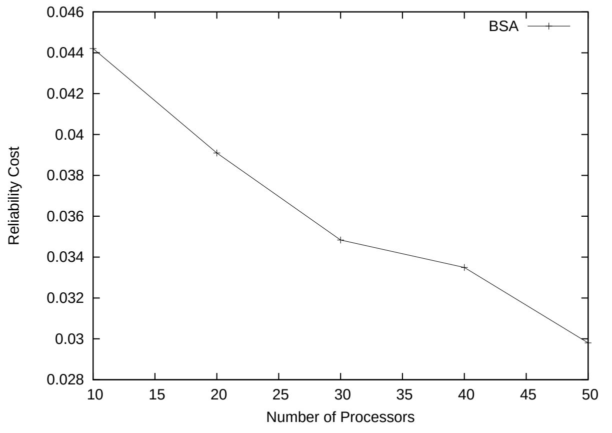
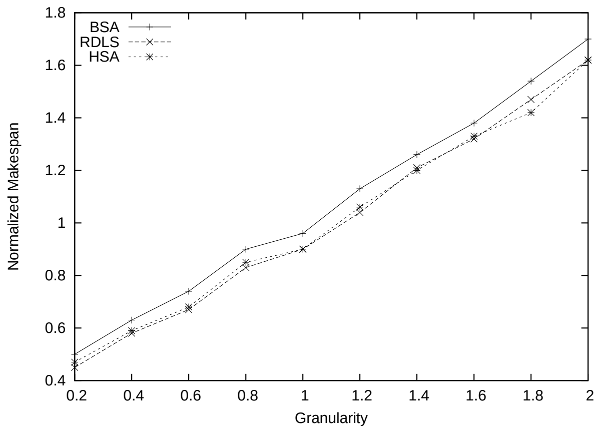


Figure 4. Impact of system sizes on reliability cost and makespan



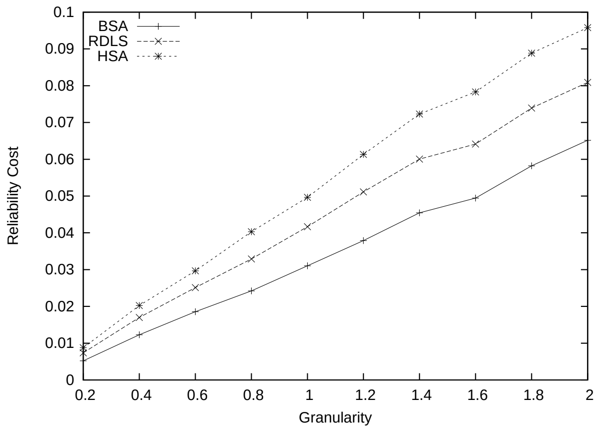


Figure 5. Average normalized makespan and reliability cost comparison between BSA and RDLS

ized and then combined. Doing so, the algorithm becomes heavier, due to the scheduling criterion of RDLS. In addition, normalizing both objective values is a necessary condition to have the same range of values, but not sufficient to achieve a better compromise solutions between the two objectives. This is due to the disadvantage of the objective function given by 5. Unlike RDLS, the BSA algorithm does not have this problem, because the compromise function $\mathcal{D}(t, \mathcal{P})$ provides better decisions during the mapping process.

***f*) Running times:** the running times of BSA and RDLS are given in Table 1. The implementation is in C programming language, the computer used is a Pentium 4 (CPU 2.0 GHz). From this table, we can observe that our algorithm is considerably faster than RDLS. We conclude that for large task graphs, RDLS algorithm is impractical. The time complexity of BSA makes it practical for very large task graph which is the case in many practical applications.

Number of tasks	BSA	RDLS
1000	0.011	2.079
2000	0.023	8.680
3000	0.039	16.675
4000	0.045	34.604
5000	0.067	47.917
6000	0.072	123.844
7000	0.093	159.256
8000	0.108	205.778

Table 1. Running Times of BSA vs RDLS in seconds

9. Conclusion

The algorithm discussed in this paper for scheduling parallel applications on heterogeneous computing systems provides achieve two conflicting objectives: minimum time Makespan and maximum reliability at the same time. The algorithm, which is called BSA, is based on a compromise function that selects processors on which critical free tasks should be mapped to during the scheduling process. The main features of our algorithms are:

- It has a low time complexity;
- It can attain both objectives to some degree simultaneously;
- It has a very good behavior in practice.

Indeed, simulations studies show that the solution quality and the time complexity of our algorithm makes it a viable choice for compile-time scheduling large tasks graphs on systems subject to failures.

References

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

[2] A. Dogan and F. Ozguner. Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(03):308–323, 2002.

[3] A. Gerasoulis and T. Yang. Dsc: Scheduling parallel tasks on an unbounded number of processors. *IEEE Transactions on Parallel and Distributed Systems*, 5:951–967, 1994.

[4] M. Hakem and F. Butelle. Efficient critical task scheduling parallel programs on a bounded number of processors. In Actapress, editor, *Proc. of the 17th International Conference on Parallel and Distributed Computing and Systems. (PDCS'05 - IASTED)*, pages 139–144, 2005.

[5] M. Hakem and F. Butelle. Critical path scheduling parallel programs on unbounded number of processors. *International Journal of Foundations of Computer Science*, 17(2):287–301, 2006.

[6] C.-J. Hou and K. G. Shin. Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems. *IEEE Transactions on Parallel and Distributed Systems*, 46(12):1338–1356, 1997.

[7] J.-J. Hwang, Y.-C. Chow, F. D. Anger, and C.-Y. Lee. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM Journal on Computing*, 18(2):244–257, 1989.

[8] S. S. N. K. Jha. Safety and reliability driven task allocation in distributed systems. *IEE Trans. on Parallel and Dist. Syst.*, 10(03):238–251, 1999.

[9] Y.-K. Kwok and I. Ahmad. Bubble scheduling: A quasi dynamic algorithm for static allocation of tasks to parallel architectures. In *Proc. 7th IEEE Symp. on Parallel and Distr. Processing*, pages 36–43, 1995.

[10] Y.-K. Kwok and I. Ahmad. Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 7(5):506–521, 1996.

[11] S. K. C. S. R. Murthy. Task allocation algorithms for maximizing reliability of distributed computing systems. *IEEE Trans. on Computers*, 41(06):719–724, 1997.

[12] X. Qin and H. Jiang. A dynamic and reliability driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters. *Journal of Parallel and Distributed Computing*, 65(08):885–900, 2005.

[13] V. Sarkar. *Partitionning and Scheduling Parallel Programs for Execution on Multiprocessors*. MIT Press, 1989.

[14] G. Sih and E. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans. on Parallel and Dist. Systems*, 4(2):75–87, 1993.

[15] J. P. W. Sol M. Shatz and M. Goto. Task allocation for maximizing reliability of distributed computer systems. *IEEE Trans. on Computers*, 41(09):1156–1168, 1992.

[16] Y. Sorel. Massively parallel computing systems with real-time constraints: the ”algorithm architecture adequation”. In *Proc. of Massively Parallel Comput. Syst., MPCS*, 1994.

[17] M. Y. Wu and D. Gajski. Hypertool: Aprogramming aid for message-passing systems. *IEEE Trans. on Parallel and Dist. Syst.*, 1(3):330–343, 1990.