MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE EL HADJ LAKHDHAR BATNA
FACULTE D'INGENIEUR
DEPARTEMENT D'INFORMATIQUE

Mémoire

En vue de l'obtention du diplôme de

MAGISTER

Spécialité : Informatique

Option : Ingénierie des Systèmes Informatiques

Dans le cadre de l'**Ecole Doctorale** < **Sciences et Technologies de l'Information et Communication, STIC** >

Présenté et soutenu Par

Yasmina BERROU

Evaluation de la fiabilité des systèmes temps réel distribués embarqués

Composition du JURY:

Prof. Benmohamed mohamed	Prof Université de Constantine	Président
Dr. Bilami Azeddine	MCA Université de Batna	Examinateur
Dr. Belattar Brahim	MCA Université de Batna	Examinateur
Dr. Zidani Abdelmajid	MCA Université de Batna	Rapporteur
Dr. Kalla Hamoudi	MCB Université de Batna	Co-Rapporteur

Remerciements

J'aimerais d'abord remercier mon encadreur et mon co-encadreur qui m'a soutenu tout au long de la rédaction de ce mémoire, pour son écoute et ses conseils pertinents.

Je remercie également l'ensemble des membres du jury pour avoir consacrer leur temps à examiner ce travail malgré leurs nombreuses responsabilités, je leur suis reconnaissant pour l'attention qu'ils ont portée à mon travail.

Je tiens aussi à remercier ma famille et mes amis pour leur soutient inconditionnelet leur présence continue.

Enfin je remercie tous ceux qui m'ont aidé de près ou de loin pour la réalisation de ce travail.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction générale

Les logiciels embarqués prennent une part croissante dans la gestion de nombreux systèmes en contrôlant de plus en plus leur fonctionnement de façon automatique. Présents depuis longtemps dans des applications coûteuses (spatiales, aéronautiques, militaires...) où ils prennent une importance considérable, ces systèmes apparaissent dans des domaines grand public, tels que les appareils électroniques (téléphones portables, PDA, appareils photos...), mais aussi l'assistance à la conduite automobile (direction assistée, freinage assisté, en attendant les systèmes dits « drive-by-wire »).

Certains systèmes sont à sûreté critique, comme les avions, d'autres moins, comme les montres ou des machines à laver. Avec la miniaturisation des composants, les nouvelles technologies de batteries, et le développement des moyens de communication, il faut s'attendre à en avoir de plus en plus autour de nous, à tel point que certains prédisent l'avènement prochain de l'informatique ubiquitaire. Mais au fait, qu'est-ce qu'un système informatique embarqué ? La définition qui semble la plus communément admise stipule que ceux sont des systèmes informatiques dont les ressources sont *limitées*. Par ressource, on entend ressources de taille, de poids, de consommation électrique et de puissance de calcul. Certes, tout ordinateur de bureau a une consommation électrique limitée par la puissance de l'installation électrique de la prise sur laquelle il est branché. Il en va de même pour toutes les limites mentionnées ci-dessus puisque le monde est défini. Mais, dans le cas des ordinateurs de bureau, cette limite n'est en pratique pas prise en compte par les fabricants. En revanche, pour un PDA, un téléphone portable ou un satellite, elle est critique.

L’importance et la complexité de ces systèmes rend ces systèmes sensibles aux fautes, la présence d’une telle faute accidentelle ou intentionnelle dans un système critique amène à des résultats catastrophiques. Pour aborder ses problèmes, plusieurs recherches sont développées dans la littérature pour concevoir un système sûr de fonctionnement.

Parmi les moyens utilisés dans la littérature pour concevoir des systèmes sûrs de fonctionnement qui sont ***la fiabilité,*** que nous appelons par la suite un système fiable, et ***la tolérance de faute***, que nous appelons par la suite un système tolérant aux fautes.

Dans ce travail, on s'intéresse aux méthodes qui permettent de concevoir un système fiable, par définition, la fiabilité est la mesure qui permet d'évaluer quantitativement la sûreté de fonctionnement d'un système, et, on peut la définir aussi comme la probabilité de bon fonctionnement de système durant son exécution. Le calcul de la fiabilité d'un système est un problème NP difficile ce qui explique les nombreuses méthodes existantes pour la calculer. Cette complexité est augmentée surtout si on a une architecture distribuée hétérogène. Il y'a plusieurs modèles de calcul de fiabilité tels que les modèles combinatoires et les algorithmes d'ordonnancement….Notre travail est basé sur les algorithmes d'ordonnancement pour évaluer la fiabilité des systèmes.

Ce travail fait partie d'une collaboration avec le projet popART de l'INRIA Rhone-Alpes, l'objectif est d'intégrer les mécanismes de la sûreté de fonctionnement dans l'outil Syndex.

1.1 Problématique

Notre but dans ce travail est de concevoir un système fiable c'est-à-dire de trouver un algorithme de distribution et d'ordonnancement des composants logiciels de l'algorithme sur les composants matériels de l'architecture de telle sorte qu'on doit optimiser deux objectifs d'une part, la minimisation de la durée d'exécution du système et d'autre part, la maximisation de la fiabilité du système tout en respectant les contraintes temporelles et matérielles. Les travaux existants pour résoudre le problème d'optimisation multi-objectif sont devisés en trois grandes approches qui sont l'agrégation de deux critères dans un seul critère, la transformation d'un critère à une contrainte et l'ordonnancement des critères. Notre solution appartient à l'approche d'agrégation de deux critères à un seul.

1.2 Structure du mémoire

Le mémoire est composé de cinq chapitres :

- Le premier chapitre est une introduction générale.
- Nous présentons dans le deuxième chapitre, des notions de base et les terminologies liées aux systèmes réactifs embarqués, et les algorithmes d'ordonnancement et leur rôle dans notre travail.
- Le troisième chapitre présente la notion de fiabilité et les différentes techniques existantes dans la littérature pour estimer et calculer la fiabilité des systèmes.
- Le quatrième chapitre présente un état de l'art sur les derniers algorithmes existant dans la littérature pour aborder le problème d'optimisation bi-critères qui

sont dans notre cas: la minimisation de la durée d'exécution du système et la maximisation de la fiabilité.

- Le cinquième chapitre, présente l'algorithme de distribution et d'ordonnancement que nous proposons. Il appartient à l'approche d'agrégation de deux critères dans un seul, l'algorithme donne comme résultat, en plus de respect des contraintes temporelles et matérielles, une évaluation de la fiabilité des systèmes qui est l'objectif principale de ce travail.

Ce mémoire termine par une conclusion qui résume notre travail et suivi par quelques perspectives.

Chapitre 2

Introduction aux systèmes temps réel et les algorithmes d'ordonnancement

2.1 Introduction

Un point commun entre une usine, un avion, une centrale nucléaire, une machine à laver, une auto, et un appareil multimédia de salon, c'est qu'ils possèdent tous, de nos jours, un ou plusieurs processeur(s) ou microcontrôleur(s) pour les gérer. Un autre point commun entre ces systèmes, c'est que le logiciel qui y est déroulé, qui réagis avec l'environnement et qui a conscience de l'écoulement du temps, prend des décisions en fonction de celui-ci. On parle alors de ***système temps-réel*** sachant que ce terme revêt un sens extrêmement large.

Ces systèmes réalisent des tâches très complexes et souvent critiques, donc ce système doit respecter toutes les contraintes temporelles spécifiées avant toute exécution réelle du système d'une part, d'autre part, il doit faire une compétition entre les entités pour accéder aux ressources (processeur, lien de communication ...). On parle alors, d'un problème d'ordonnancement.

L'objectif principal de ce chapitre est de donner une introduction générale aux systèmes temps réel embarqués distribués et les algorithmes de distribution et d'ordonnancement. Dans un premier lieu nous donnons quelques définitions. Ensuite une modélisation de système sur lequel se base le travail présenté dans ce mémoire, et finalement quelques algorithmes de distribution et d'ordonnancement.

2.2 Définitions

2.2.1 Système réactif

Définition 1 (Système réactif) *"dans un sens large un système réactif peut être utilisé pour décrire tout système de traitement de l'information ou système informatisé répondant continuellement à des stimuli externes."*[16].

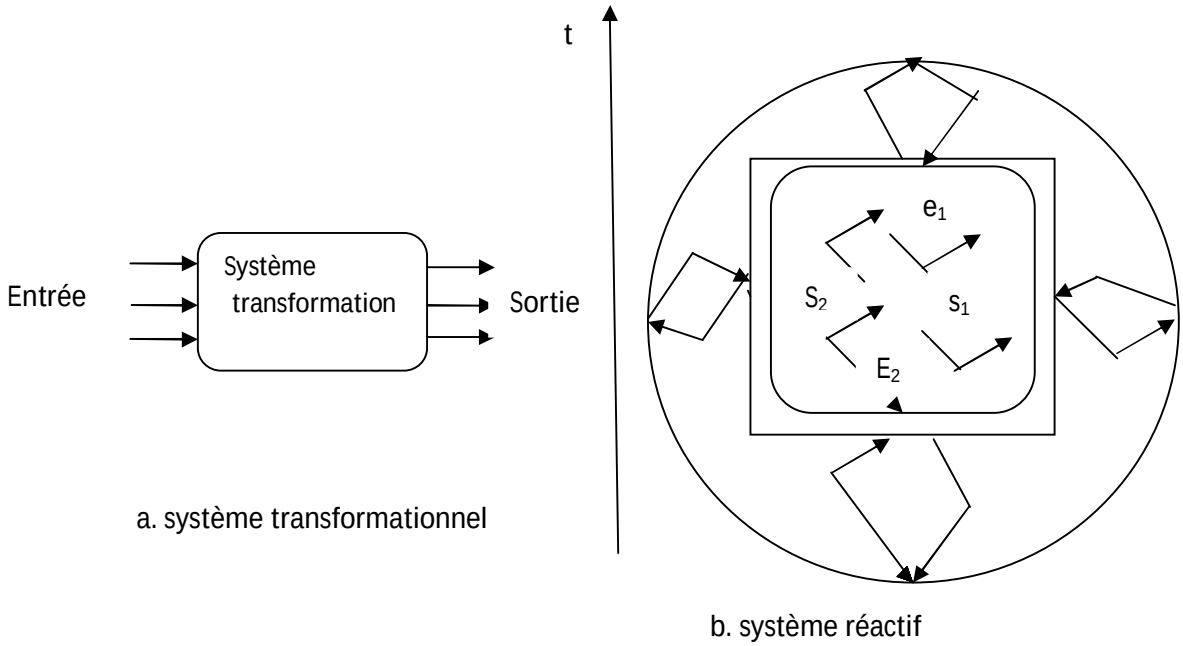

FIG. 2.1 – Système transformationnel vs système réactif

Au contraire d'un système transformationnel qui accepte des données en entrée et produit des résultats en sortie, un système réactif réagit continuellement et instantanément à des évènements ou stimuli, qu'ils soient internes ou externes, en d'autres façons, un système réactif est conduit par les stimuli, ses comportements dépendent de ces derniers et de l'ordre avec lequel ces événements surviennent.

L'introduction de la contrainte du temps inhérent à l'application, c'est à dire aux processus à contrôler en spécifiant des délais avec lesquels seront produites les réponses aux stimuli permet d'introduire la notion de ***système temps réel***.

2.2.2 Exemples des systèmes réactifs

2.2.2.1 Exemple 1

Prenons l'exemple d'un distributeur de billets. Il est constitué d'un système informatique qui interagit avec un environnement. L'environnement est ici constitué d'un clavier, d'un lecteur de carte, d'un mécanisme de distribution de billets, d'un écran, d'un stock de billets de banque et des utilisateurs. Ainsi, le clavier et le lecteur de carte permettent de connaître les désirs de l'utilisateur, l'écran et le tiroir à billets sont les actionneurs permettant de satisfaire les désirs de l'utilisateur suivant la validité de sa carte, de son code et de la disponibilité des

billets. Les événements sont produits par les utilisateurs (introduction de carte magnétique, composition d'un code, etc.). Les actions à mettre en œuvre sont l'affichage des informations sur l'écran, la lecture de la carte et la distribution du nombre de billet désiré par l'utilisateur. De ce point de vue, le système informatique qui est chargé de contrôler cet environnement est un système réactif. Il réagit aux événements d'entrée en produisant les actions de sortie adéquates.

2.2.2.2 Exemple 2

Un autre exemple très simple d'un système réactif est celui de la régulation de niveau d'eau dans un réservoir (figure 2.2). Dans cet exemple, l'environnement est constitué d'un réservoir d'eau, d'une vanne et de deux capteurs sensibles à la présence d'eau. Supposons qu'à l'instant t=0 le niveau de l'eau dans le réservoir soit au niveau du capteur 1 et que la vanne soit ouverte. Le rôle de ce système est de maintenir le niveau de l'eau entre les deux capteurs 1 et 2 : si le capteur 2 est mouillé, le système doit envoyer une commande de fermeture de la vanne avant que le réservoir déborde, et si le capteur 1 est sec, le système doit envoyer une commande d'ouverture de la vanne.

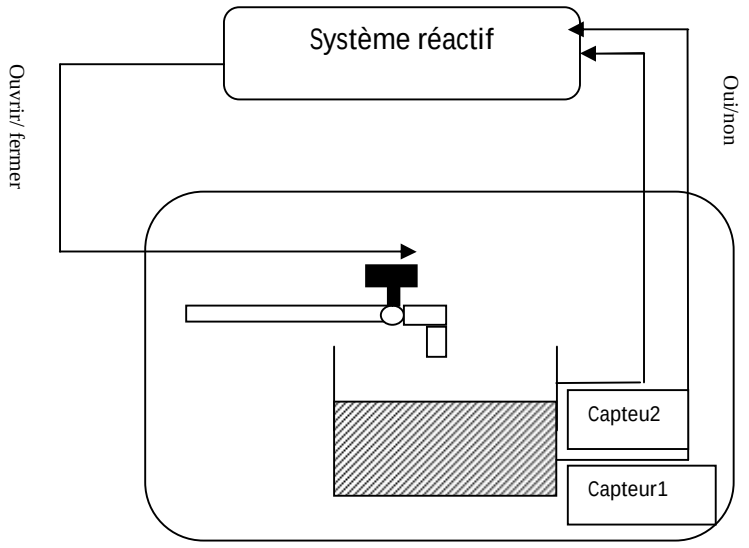

FIG. 2.2 – Exemple d'un systèmes réactif

2.2.3 Système temps réel

Il existe dans la littérature plusieurs définitions de système temps réel, on prend par exemple les trois définitions suivantes.

Définition 2 " *Un système fonctionne en temps réel s'il est capable d'absorber toutes les informations d'entrée qu'elles soient trop vieilles pour l'intérêt qu'elles présentent, et par ailleurs, de réagir à celles-ci suffisamment vite pour que cette réaction ait un sens."*[19].

Définition 3 "*Les systèmes informatiques temps réel seraient donc ceux pour lesquels le comportement temporel, non seulement en terme de séquence d'opérations (propriété logique), mais également en terme de quantification de l'écoulement de ces opérations (propriété physique), a une importance.»*[19].

Un système temps réel est un système dans lequel l'exactitude des applications ne dépend pas seulement de l'exactitude du résultat mais aussi du temps auquel ce résultat est produit. Si les contraintes temporelles de l'application ne sont pas respectées, on parle de défaillance du système. Il est donc essentiel de pouvoir garantir le respect des contraintes temporelles du système. Ceci nécessite que le système permette un taux d'utilisation élevé, tout en respectant les contraintes temporelles identifiées.

2.2.4 Temps réel souple et temps réel strict

La majorité des systèmes temps réel est exclusivement constitue des traitements qui ont des contraintes temporelles strictes, cette classe impose deux choses.

- Qu'on est capable de définir des conditions de fonctionnement nominales en termes d'hypothèses sur l'environnement avec lequel le système interagit,
- Qu'on est capable de garantir avant exécution que tous les scénarios d'exécutions possibles dans ces conditions respecteront leurs contraintes temporelles.

Une autre classe de système est moins exigeante quant au respect absolu de toutes les contraintes temporelles. Les systèmes de cette classe, dits temps-réel souple, peuvent donner un taux acceptable de fautes temporelles (non respect transitoire des contraintes) de la part d'une partie des traitements (eux-mêmes dits temps-réel souple), et sans que cela ait des conséquences catastrophiques. [7]

2.2.5 Système distribué et système embarqué

Définition 4 (Système embarqué) "*Un système embarqué peut être définis comme un système informatique autonome, qui est dédié à une tâche précise, ses ressources disponibles sont limitées, cette limitation est généralement d'ordre spatial (taille limite) et énergétique (consommation restreinte)* "

Définition 5 (Système distribué) "*Est une collection de processeurs qui sont interconnectés à l'aide d'un réseau de communication*" [17]

On dit qu'un système distribué est hétérogène si les composants de l'architecture (processeur ou moyen de communication) ont des caractéristiques différentes.

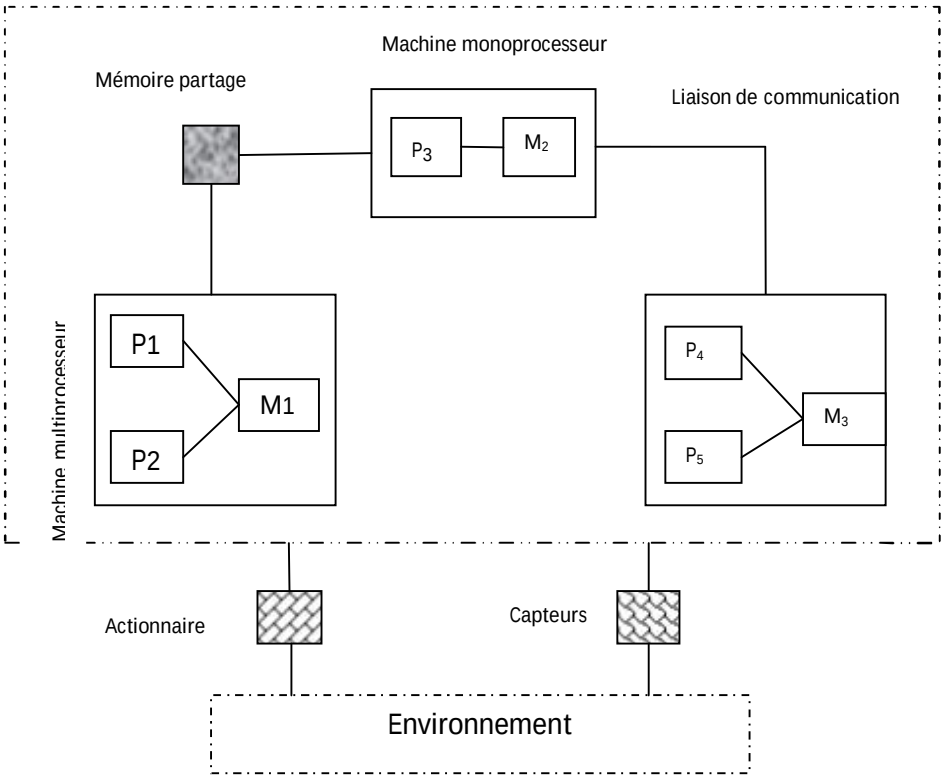

FIG. 2.3 – Exemple d'une architecture d'un systéme distribué embarqué

2.3 Modélisation des systèmes temps réel

Il existe plusieurs formalismes pour modéliser un système temps réel, telles que les méthodes formelles (logique temporelle, algèbre de processus) et les méthodes structurées (système réactif synchrone et asynchrone, réseau de Pétri),

Nous nous intéressons dans ce travail aux systèmes distribués réactifs embarqués à contraintes temps réel strictes.

La spécification de ces systèmes est réalisée en trois phases complémentaires et dépendantes.

- la spécification fonctionnelle.
- la spécification architecturale.
- la spécification des contraintes.

La spécification fonctionnelle consiste à définir l'algorithme avec ses exigences fonctionnelles.

La spécification architecturale consiste à définir l'architecture matérielle qui doit implanter l'algorithme.

La spécification des contraintes consiste à attribuer des propriétés temporelles et matérielles à l'exécution de l'algorithme sur l'architecture.

Nous nous basons dans ce mémoire sur le modèle développé dans [17].

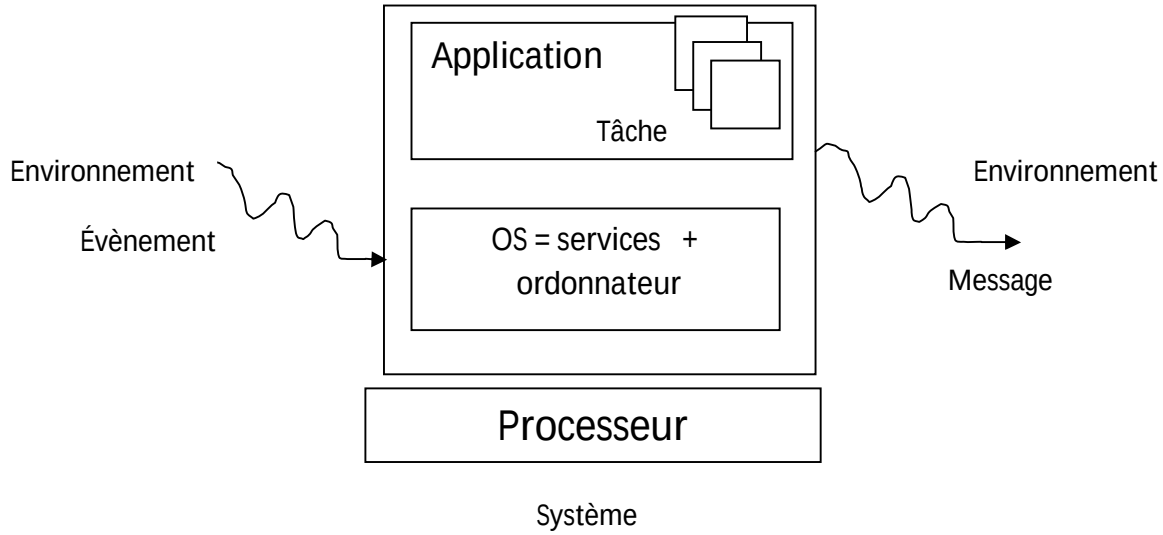

FIG. 2.4 – Modélisation des systèmes temps reél

2.3.1 Tâche

Une tâche est considérée comme l'unité de représentation des activités, elle est chargée de fournir un service de l'application, elle correspond à l'exécution d'une séquence d'opérations données sur un processeur.

La vie d'une tâche peut être représentée suivant un chronogramme ou diagramme de Guantt qui permet d'introduire les terminologies suivantes:

- **Date de création ou d'activation -** La tâche est crée et prête à être exécutée sur le processeur.
- **Date de démarrage** - Le date de début d'exécution de la tâche par le processeur
- **Date de préemption –** Le tâche est momentanément interrompue au profit d'une autre, plus prioritaire.
- **Dates de reprise -** Le processeur reprend l'exécution de la tâche là où il avait été précédemment préempté (plusieurs possibilités durant la vie de la tâche).
- **Date de terminaison** - le tâche termine de s'exécuter sur le processeur**.**
- **Temps de réponse -** l'écart entre la date de terminaison et la date d'activation.

2.3.2 Niveau logiciel

Le niveau logiciel contient l'application et le support d'exécution temps réel, il représente les fonctions nécessaires au contrôle et à la commande de l'environnement. Il est structuré en tâche et il est modélisé par un graphe flot de données (noté ***ALg***), ce niveau est appelé aussi "algorithme". Le graphe *ALg* est un **graphe orienté sans cycle** « directed acyclic graph ».

Ses sommets sont des blocs logiciels appelés **tâches**, la plupart sont sans effet de bord, à l'exception des tâches d'entrée/sortie : une tâche d'entrée est un appel à un ou plusieurs pilotes de capteurs physiques, tandis qu'une tâche de sortie est un appel à un ou plusieurs pilotes d'actionneurs physiques, les arcs représentent les dépendances de données entre ses tâches, ces arcs induisent un ordre partiel d'exécution sur les tâches appelées parallélisme potentiel, une dépendance de données entre deux tâches t_1 et t_2 note $t_1 \rhd t_2$ signifie que t_2 commence son exécution après avoir reçu les données de sortie de t_1, l'ensemble des prédécesseurs de t est noté pred(t), l'ensemble des successeurs de t est noté succ(t).

Un logiciel d'un système temps réel peut être décomposé en deux grandes parties:

- **l'algorithme applicatif -** il décrit le comportement du système. Il est composé de plusieurs sous-algorithmes applicatifs qui réalisent des fonctions spécifiques.
- **l'exécutif -** son rôle consiste à gérer et à allouer les ressources physiques à l'algorithme applicatif, et aussi à respecter les contraintes temporelles.

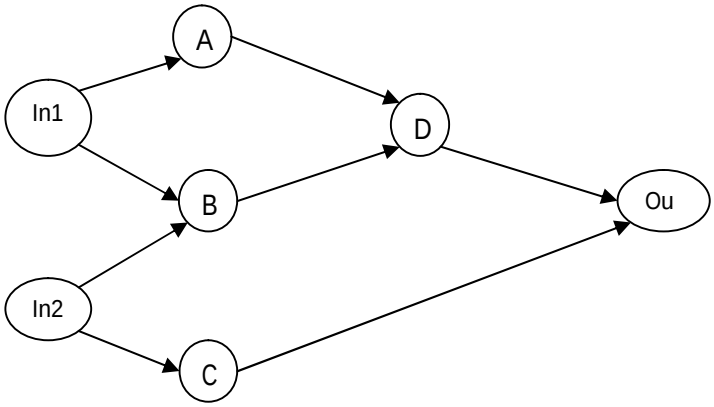

FIG. 2.5 – Exemple d'un graphe d'algorithme

Dans la figure FIG. 2.5 est un exemple d'un graphe d'algorithme constitue de sept taches (A, B, C, D, In_1, In_2, Out_1) et huit dépendances de données ({$In_1 \rhd A$}...).

2.3.3 Niveau matériel

Le système repose sur un ou plusieurs processeurs pour effectuer le traitement, il est représenté par un graphe d'architecture (noté ***ARC***), il est découpé en nœuds qui

communiquent par un moyen de communication (telle que la communication par messages ou par mémoire partagée), chaque nœud correspond à un processeur, actionneur ou capteur et les arêtes désignent les liaisons physiques entre processeurs.

D'autre part, le système utilise des actionneurs et des capteurs pour interagir avec l'environnement extérieur qu'il contrôle. Etant donné que l'architecture matérielle est distribuée, le choix de l'emplacement physique de ces capteurs/actionneurs sur l'architecture est important dans la conception de ces systèmes.

Définition 6 (Processeur) "*Un processeur est une machine à états finis, composé de quatre unités : une unité arithmétique et logique (UAL), une unité d'entrée/sortie (ES), une unité de contrôle (UC), et une mémoire locale (RAM).* "

Définition 7 (Communicateur) "*Un communicateur est une machine à états finis. Il permet le transfert de données entre deux ou plusieurs processeurs. Il est connecté à une mémoire locale RAM et une mémoire partagée SAM.* "

Définition *8* (Architecture à liaisons point-à-point) "*C'est une architecture multiprocesseur distribuée constituée d'un réseau de communication composé uniquement de mémoires SAM point à point*".

Définition 9 (Architecture à liaisons bus) "*C'est une architecture multiprocesseur distribuée constituée d'un réseau de communication composée uniquement de mémoires SAM multipoint, ou chaque mémoire SAM est connectée à tous les processeurs*"

Le graphe d'architecture ***ARC*** est un graphe biparti non-orienté *(P;M;A)* dont l'ensemble des sommets est $P \cup M$ et l'ensemble des arêtes est A; *P* est l'ensemble des processeurs et *M* est l'ensemble des medias de communication. Le degré d'un sommet est le nombre d'arêtes auquel il est relié. Un processeur est constitué d'une unité de calcul, lui permettant de séquencer les tâches, et d'une ou plusieurs unités de communication, lui permettant de séquencer les communications. Chacune des arêtes d'***ARC*** relient forcément un processeur et un médium de communication. Dans le cas d'un lien de communication point-à-point, le sommet représentant ce médium est connecté à exactement deux processeurs (il est donc de degré 2), alors que dans le cas d'un bus de communication, le sommet représentant ce medium est connecté à plusieurs processeurs (il est donc de degré $n \geq 2$). Un chemin entre les processeurs P_1 et P_n est une succession P_1-A_1-M_1-A_2-P_2-A_3-M_2-A_4-…-P_n, Le graphe ***ARC*** est connexe s'il existe un chemin entre toute paire de sommets ; cette notion est définie normalement pour les graphes non-bipartis et elle s'applique aux cas particuliers des graphes bipartis ; ce qui m'intéresse ici est que, dans un graphe Arc connexe, il existe un chemin entre toute paire de processeurs. Par analogie avec les graphes non-bipartis, le graphe *Arc* est complet s'il existe un lien de communication entre toute paire de processeurs (et non pas une arête entre toute paire de sommets). L'architecture est complètement connectée si son graphe ***ARC*** est complet. Enfin, l'architecture est dite homogène si tous ses processeurs et tous ses media de communication ont les mêmes caractéristiques (vitesse, mémoire, fiabilité, bande passante...), et hétérogène sinon.

La figure 2.6 est un exemple d'un graphe architecture, composé de trois processeurs et de trois liens de communication.

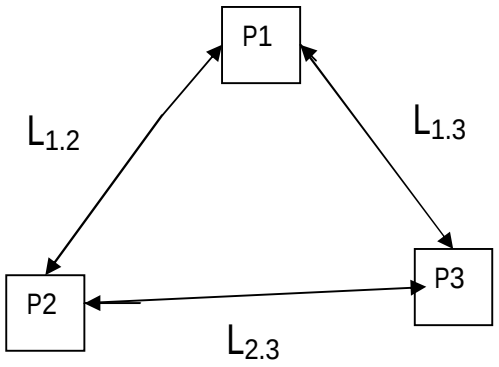

FIG. 2.6 – Exemple d'un graphe d'architecture (représentation classique)

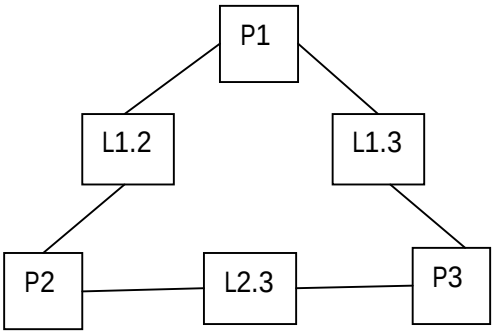

FIG. 2.7 – Exemple d'un graphe d'architecture (représentation bipartie)

La figure 2.6 représente un exemple de graphe d'architecture avec trois processeurs, P_1, P_2, et P_3, et quatre liens de communication point-à-point, L_{1-2}, L_{2-3} et L_{3-4}. La représentation bipartie est donnée dans FIG 2.7.

2.3.4 Caractéristiques et contraintes temporelles

Le système doit spécifier les caractéristiques temporelles des travaux de chaque tâche. Dans le modèle de tâche, on trouve les caractéristiques temporelles suivantes:

- **Temps d'exécution** - c'est le temps d'exécution de la tâche sur le processeur.
- **La loi d'arrive-** il s'agit de la répartition dans le temps des dates de création des tâches. On distingue couramment trois lois :

- **Périodique :** Les tâches sont crées de façon rigoureusement périodique, et la période est précisée.

- **Sporadique** les tâches sont crées de sorte qu'une durée minimale sépare deux travaux successifs : le délai d'inter-arrivée, qui est précise. Une tâche périodique est un cas particulier de tâche sporadique.

- **Apériodique** Un travail de la tâche peut être crée a tout instant. Une tâche sporadique est un cas particulier de tâche apériodique.

Le modèle de tâche doit indiquer également les contraintes temporelles telles que :

- **la date d'échéance :** il s'agit de la date à laquelle les travaux doivent être terminés, relativement à leur date de création, ou sous forme d'un vecteur de dates absolues. Relative ou (absolute deadline) en anglais

- **la gigue de démarrage :** il s'agit de la mesure de dispersion (écart-type, variance) des délais entre la date de création et la date de démarrage des travaux d'une tâche. (Release jetter) en anglais

- **la précédente** avant de pouvoir démarrer, une tâche doit attendre le résultat d'un travail d'une autre tâche

Les contraintes temporelles attribuées aux tâches peuvent être des mesures exactes, moyennes, ou majoritaires; ceci dépend des outils utilisés pour obtenir ces mesures.

2.3.5 Contraintes matérielles

Les tâches peuvent être vues comme des éléments du système qui consomment la ressource processeur. D'autres ressources existent au sein du système, qui peuvent être matérielles (processeur, mémoire, réseau, capteurs, actionneurs par exemple), ou logiques (sémaphores, ou messages par exemple). Le système d'exploitation a pour rôle de gérer toutes les ressources, c'est-a-dire d'arbitrer entre les demandes que les tâches de l'application font, et les ressources effectivement disponibles dans le système.

2.4 Problématique de distribution et d'ordonnancement temps réel embarqué

2.4.1 Description

Le problème de l'ordonnancement, sur lequel ‘ la validation de l'application, consiste à définir une politique d'attribution du processeur (et des ressources) qui assure que les contraintes temporelles et des ressources sont respectées, c'est à dire qu'aucune tâche ne terminera l'une quelconque de ses instances après l'échéance de celle-ci.

Définition 10 (Ordonnancement temps réel) " *L'ordonnancement est le terme informatique désignant le mécanisme permettant de réaliser la sélection, parmi plusieurs composants de l'algorithme (tâches), de celui qui va obtenir l'utilisation d'un composant de l'architecture pour s'exécuter, de manière à optimiser un ou plusieurs critères. L'ordonnancement temps réel a une particularité qui nécessite de respecter des contraintes de temps réel. Le processus qui réalise une telle sélection, est donc qui définit comme ordre d'exécution entre les composants de l'algorithme, est appelé algorithme d'ordonnancement.* **"**

La manière de déterminer l'ordre d'exécution et l'attribution des tâches aux processeurs est de calculer le poids pour chaque tâche en utilisant une fonction de coût.

Définition 11 (Fonction de coût) "*le rôle d'une fonction de coût est d'associer un poids à chaque composant logiciel (tâches), et ceci afin de calculer un ordre d'exécution entre ces composants***"**. [17]

Définition 12 (Algorithme de distribution et d'ordonnancement temps réel) "*C'est un algorithme d'ordonnancement temps réel qui doit réaliser, en plus d'une opération de sélection d'un composant logiciel, une opération de sélection du composant matériel qui peut exécuter le composant logiciel sélectionné. Ce type d'algorithme est nécessaire dés que l'architecture est distribuée***"**.

2.4.2 Présentation du problème

Le problème de distribution et d'ordonnancement temps réel est de chercher une allocation spatiale et temporelle des tâches sur l'architecture matérielle tout en respectant les contraintes temporelles, pour résoudre ce problème, on doit prouver l'existence d'une allocation valide.

Si on doit chercher une allocation qui satisfait plus d'un critère, ce problème de recherche devient un problème d'optimisation multi critères, les critères qui sont pris en charge dans ce mémoire est la minimisation de la longueur de l'ordonnancement et la maximisation de fiabilité du système. Ce problème d'optimisation est soit NP difficile soit polynomial [1]

Énoncé du problème

On a:

- Une architecture matérielle hétérogène ***ARC*** composée de n composant matériel

$$\boldsymbol{ARC} = \{p_1, \ldots, p_n\}$$

- Un algorithme ***Alg*** composé de m composants logiciels (tâche)

$$\boldsymbol{Alg} = \{t_1, \ldots, t_m\}$$

- Des coûts d'exécutions Exe des composants de ***Alg*** sur les composants des ***ARC***
- Des contraintes temporelles ***L*** et matérielles ***Dis***
- Un ensemble de critères à optimiser : la longueur et la fiabilité.

But

*Trouver un ordonnancement des composants de **Alg** sur les composants de **ARC**, de telle sorte qu'il respecte Dis et L, et qu'il optimise l'ensemble des critères.*

2.4.3 Classification des algorithmes d'ordonnancement temps réel

On peut classifier les algorithmes d'ordonnancement selon plusieurs critères. Nous présentons dans cette section quelques classifications existantes dans la littérature.

2.4.3.1 Monoprocesseur/multiprocesseur

L'ordonnancement est de type monoprocesseur si toutes les tâches ne peuvent s'exécuter que sur un seul et même processeur. Si plusieurs processeurs sont disponibles dans le système, l'ordonnancement est multiprocesseur**.**

2.4.3.2 En-ligne/hors-ligne

Un ordonnancement hors ligne établit avant le lancement de l'application, une séquence fixe d'exécution des tâches à partir de tous les paramètres de celles-ci. Par contre dans un ordonnancement en ligne la séquence d'exécution des tâches est établie dynamiquement par l'ordonnanceur au cours de la vie de l'application en fonction des événements qui surviennent. L'ordonnanceur choisit la prochaine tâche à élire en fonction d'un critère de priorité.

2.4.3.3 Exact/approché

Si l'algorithme donne un résultat optimale, on dit que cet algorithme appartient à la classe des algorithmes exactes, dans le cas contraire ou les résultats sont proches de la solution optimale, on dit que cet algorithme appartient à la classe des algorithmes approchés

2.4.3.4 Préemptif/non-préemptif

Si la préemption est autorisée donc l'ordonnanceur est préemptif, dans le cas contraire l'ordonnanceur est non préemptif.

2.4.3.5 Centralisé/distribué

Lorsqu'un système est distribué, l'ordonnancement (en-ligne) est distribué si les décisions d'ordonnancement sont prises par un algorithme localement en chaque nœud.

Il est centralisé lorsque l'algorithme d'ordonnancement pour tout le système, distribué ou non, est déroulé sur un nœud privilégie.

2.4.3.6 Optimal/non optimal

Par définition, un algorithme d'ordonnancement optimal pour une classe de problème d'ordonnancement donné est tel que : si un système est ordonnançable par au moins un algorithme de la même classe, alors le système est ordonnançable par l'algorithme

d'ordonnancement optimal. En conséquence, si un système n'est pas ordonnançable par un ordonnanceur optimal d'une classe donnée, alors il ne l'est par aucun ordonnanceur de la même classe.

2.5 Algorithme de distribution et d'ordonnancement de SYNDEX

SYNDEX (**Syn**chronized **D**istributed **Ex**écutive) est un outil de conception de systèmes embarqués répartis qui met en œuvre la méthode AAA (Adéquation Architecture Algorithme). La version actuelle est programmée en CAML et offre une interface graphique pour spécifier aussi bien le graphe d'algorithme ***Alg*** que le graphe d'architecture ***ARC***. En outre, SYNDEX est interfacé avec SIGNAL/POLYCHRONY**,** SCILAB/SCICOS et SCADE/ESTEREL pour construire le graphe d'algorithme à partir d'un langage de haut niveau. [14]

Afin d'effectuer des simulations sur des grands nombres de graphes, SYNDEX possède deux générateurs aléatoires de graphes, un pour les graphes d'architectures et un pour les graphes d'algorithmes. Le générateur de graphes d'architectures utilise le modèle de Waxman Ce modèle génère une grille de dimensions spécifiées par l'utilisateur, et place dans cette grille le nombre requis de sommets.

L'heuristique de distribution et d'ordonnancement implanté sous SYNDEX appartient à la classe des algorithmes approche hors ligne [12].

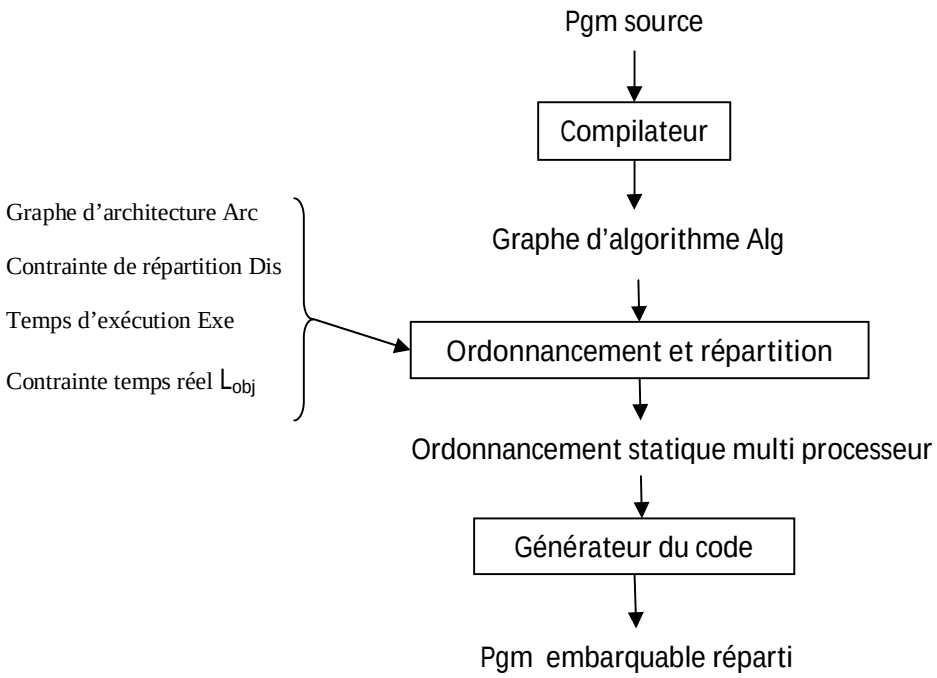

FIG 2.8 - Schéma générale de la méthode AAA

Le niveau logiciel est spécifié par un flot de données ou les nœuds représentent les tâches (opération) et les arcs représentent les dépendances de données entre ces tâches. La tâche ne peut être exécutée sauf si toutes ses entrées sont disponibles, elle consomme les données et produit des données en sortie, une tâche sans prédécesseur est un capteur (sans successeur est un actionneur).

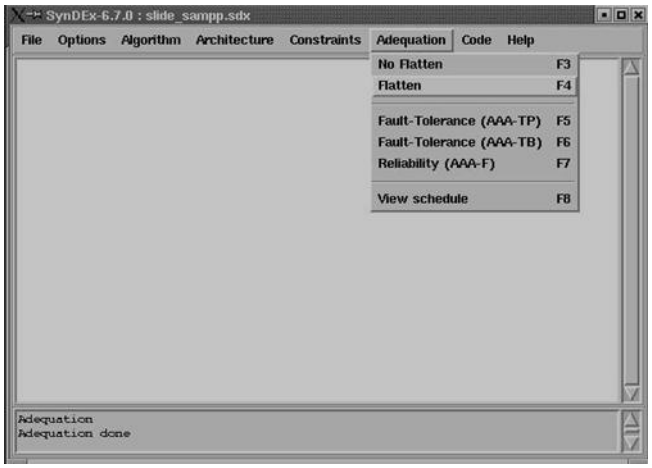

FIG 2.9- Le logiciel Syndex

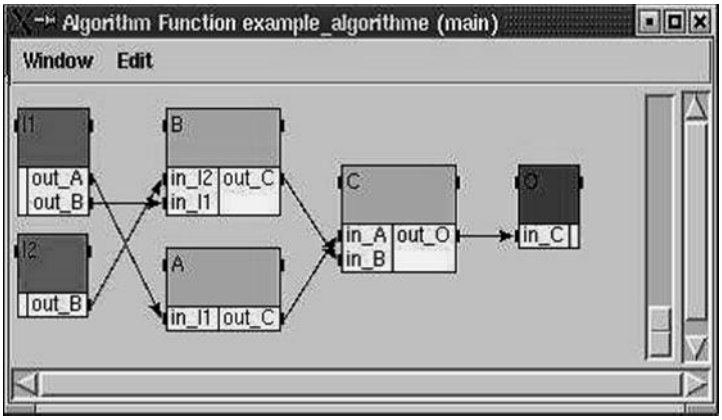

FIG 2.10 - Exemple d'un graphe d'algorithme<

Le niveau matériel est représenté par graphe non orienté appelé graphe d'architecture les nœuds désignent les processeurs ou les mémoires, les liens entre les nœuds représentent les liens physiques entre les processeurs.

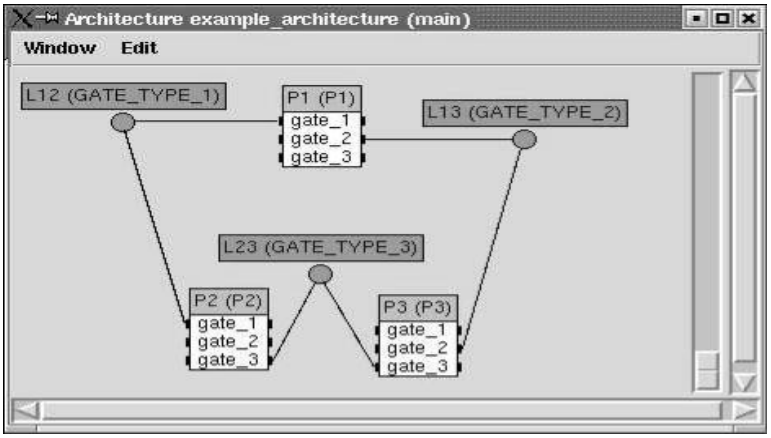

FIG 2.11 - Exemple d'un graphe d'architecture

Les contraintes temporelles pour chaque tâche, on associe un temps d'exécution différent selon le processeur et pour chaque dépendance de données, on associe un coût de transfert.

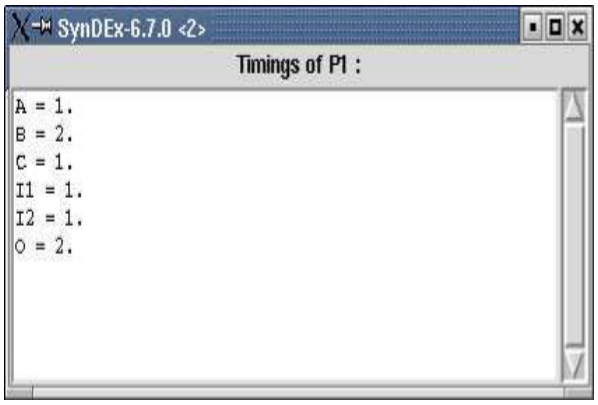

FIG 2. 12- Exemple des contraintes temporelles

Les contraintes matérielles sont spécifiées par l'association de la valeur ∞ à ***Exe*** (t_i, pj) qui signifie que l'opération t_i ne peut être exécutée sur le processeur P_j.

Le but principal de cet algorithme est de chercher une meilleure allocation spatiale et temporelle du graphe d'algorithme sur le graphe d'architecture tout en respectant les contraintes temporelles et matérielles

Afin de choisir une tâche à ordonnancer parmi toutes celles de l'ensemble des tâches candidates, l'algorithme AAA utilise une fonction de coût, appelée « pression d'ordonnancement » et notée FC_{AAA} [12]. La pression d'ordonnancement de tâche t_i sur le processeur p_j est donnée par :

$$FC_{AAA}(t_i, p_j) = P(t_i, p_j) - F(t_i, p_j) \qquad (2.1)$$

Où :

$P(t_i, p_j)$: La pénalité d'ordonnancement, c'est l'allongement de la longueur maximale quand on décide de placer t_i sur p_j. Elle est calculée par la formule suivant

$$P_{t_i,p_j} = R_{t_i,p_j} - R \tag{2.2}$$

Où R est la longueur calculée.

$F(t_i, p_j)$: La flexibilité d'ordonnancement, c'est-à-dire la différence entre la date de début d'exécution au plus tard de t_i sur p_j, calculée depuis le début de l'ordonnancement et la date de début d'exécution au plus tôt de t_i sur p_j, elle est calculée aussi depuis le début de l'ordonnancement.

$$F_{t_i,p_j} = st_{t_i,p_j} - St_{t_i,p_j} \tag{2.3}$$

L'heuristique AAA de SYNDEX est présentée dans la figure suivante :

Algorithme AAA	
Entrée	P, T, Exe, Dis et L
Sortie	Ordonnancement multiprocesseur et statique de L sur P en fonction de *Exe* et *Dis* qui satisfait L
Initialisation	$T_{ord} = Ø$ T_{cond} = {ensemble des composants de T sans prédécesseur}
Sélection de meilleur couple	**Tant que** $T_{cond} \neq Ø$ **faire** - Calculer la pression d'ordonnancement pour chaque tâche candidate $FC_{AAA}(t_i, p_j) = P(t_i, p_j) - F(t_i, p_j)$ - Pour chaque tâche choisir le couple (t_i, p_j)qui minimise la pression d'ordonnancement $FC_{AAA}(t_i, p_{meil}) = \min_{p_j \in P} FC_{AAA}(t_i, p_j)$ - Choisir le meilleur couple (t_i, p_j) qui maximise la pression d'ordonnancement $FC_{AAA}(t_{meil}, p_{mail}) = \max_{t_i \in T_{cond}} FC_{AAA}(t_i, p_{meil})$ - Allouer temporellement et spatialement la tâche choisie sur le processeur p choisi.
Mise à jour	- Mettre à jour la liste des tâches candidates T_{cond} - Mettre à jour la liste des tâches ordonnances T_{ord} **Fin Tant que**

2.7 Conclusion

Dans ce chapitre, nous avons présenté les concepts de base des systèmes temps réel, le problème d'ordonnancement et quelques algorithmes d'ordonnancement temps réel.

Le chapitre suivant décrit la notion de fiabilité et les différentes techniques utilisées dans la littérature pour la calculer.

Chapitre 3

Fiabilité des systèmes

3.1 Introduction

L'évaluation de la fiabilité d'un système est un problème *NP* difficile. Plusieurs recherches existent dans la littérature pour aborder ce problème. On s'intéresse dans ce chapitre sur les travaux basés sur les systèmes représentés par des graphes (réseau), où les nœuds représentent les tâches et les arcs représentent les dépendances de données entre ces tâches.

La fiabilité d'un ordonnancement est la probabilité qu'il existe pour chaque dépendance de donnée $x \triangleright y$ un lien opérationnel entre les deux nœuds (un nœud de source et un nœud de destination*)* par définition *un chemin est opérationnel pendant un intervalle de temps si et seulement si il est constitué de processeurs et de liens de communications opérationnels pendant cet intervalle* [14]. Ce problème, connu sous le nom de problème des paires terminales (« terminal-pair problem ») [11].

En fonction de la complexité du système, on peut diviser les méthodes de calcul de fiabilité en méthodes qui donnent des résultats exacts et des méthodes qui donnent des résultats approchés.

Dans ce chapitre, nous présentons tout d'abord la notion de sûreté de fonctionnement ensuite une notion probabiliste qui calcule le niveau de sûreté de fonctionnement qui est la fiabilité et les différentes méthodes existantes dans la littérature pour la mesurer ou estimer.

3.2 Terminologie

3.2.1 Sûreté de fonctionnement

La sûreté de fonctionnement (« dependability ») d'un système informatique est son aptitude à délivrer un service de confiance justifiée.

La sûreté de fonctionnement se décline en :

- **la disponibilité** (« availability ») qui est le fait d'être prêt à l'utilisation ;
- **la fiabilité** (« reliability ») qui mesure la continuité de service ;
- **la sécurité-innocuité** (« safety ») qui est l'absence de conséquences catastrophiques pour l'environnement
- **la confidentialité** (« confidentiality ») qui est l'absence de divulgations non autorisées d'informations ;
- **l'intégrité** (« integrity ») qui est l'absence d'altérations inappropriées de l'information ;
- **la maintenabilité** (« maintainability ») qui est l'aptitude aux réparations et aux évolutions.

La sûreté de fonctionnement est-elle vraiment nécessaire pour les systèmes informatiques ? La réponse est clairement « Oui ». Pour s'en convaincre, il suffit de consulter les deux pages web suivantes, qui recensent des cas célèbres de fautes informatiques ayant entraîné des catastrophes, depuis l'explosion de la fusée Ariane 5 jusqu'aux erreurs de division en virgule flottante du processeur Pentium: http://www5.in.tum.de/~huckle/bugse.html et http://www.qucis.queensu.ca/Software Engineering/archive/horror. On peut aussi considérer les cas, encore plus nombreux, de systèmes informatiques qui fonctionnent correctement, en dépit des inévitables défaillances matérielles et des « bugs » : des bases de données bancaires aux systèmes de contrôle/commande des avions, en passant par le réseau de télécommunication planétaire (à la fois flaire, hertzien et satellitaire)

3.2.2 Moyen de la sûreté de fonctionnement

Le développement d'un système sûr de fonctionnement repose sur l'utilisation combinée de plusieurs moyens :

- **la prévention des fautes** qui sert à empêcher l'occurrence ou l'introduction de fautes ;
- **la tolérance aux fautes** qui sert à fournir un service même en présence de fautes ;
- **l'élimination des fautes** qui sert à réduire le nombre et la sévérité des fautes ;
- **la prévision des fautes** qui sert à estimer la présence, le taux futur et les conséquences possibles des fautes

3.2.3 Fiabilité

Il existe plusieurs définitions de la fiabilité, la définition suivante est la plus adapté à notre modèle, elle est donné par [15]

Définition 3.1 (fiabilité) *La notion de fiabilité comme mesurant la continuité de service, elle permet de mesurer le niveau de sûreté de fonctionnement d'un système, elle peut donc être définie comme étant la probabilité qu'il fonctionne correctement pendant un intervalle de temps donné.*

La fiabilité généralement désigné par ***R***, et comme toute probabilité son intervalle des valeurs est [0.1], mathématiquement la fiabilité peut être calculée par la formule suivante:

$$R(x) = \int_0^t r(x)\,dx \qquad (3.1)$$

$R(t) = P(T > t)$ Probabilité que le temps *T* soit supérieur à temps *t* (temps opérationnel ou temps de fonctionnement) on peut aussi le définir comme la probabilité que le système ne soit pas défaillant pendant l'intervalle de temps [0, t].

3.2.4 Faute, Erreur, Défaillance :

On distingue trois concepts qui sont liés entre eux et qui sont les entraves de la sureté de fonctionnement qui sont: faute, erreur, défaillance.

La faute dans un système informatique représente un défaut d'un composant physique ou logique, l'activation d'une faute produit un état interne erroné qui conduit à une erreur. L'erreur dans un système peut changer l'état d'un système qui produit un non-respect de spécification, c'est une défaillance du système.

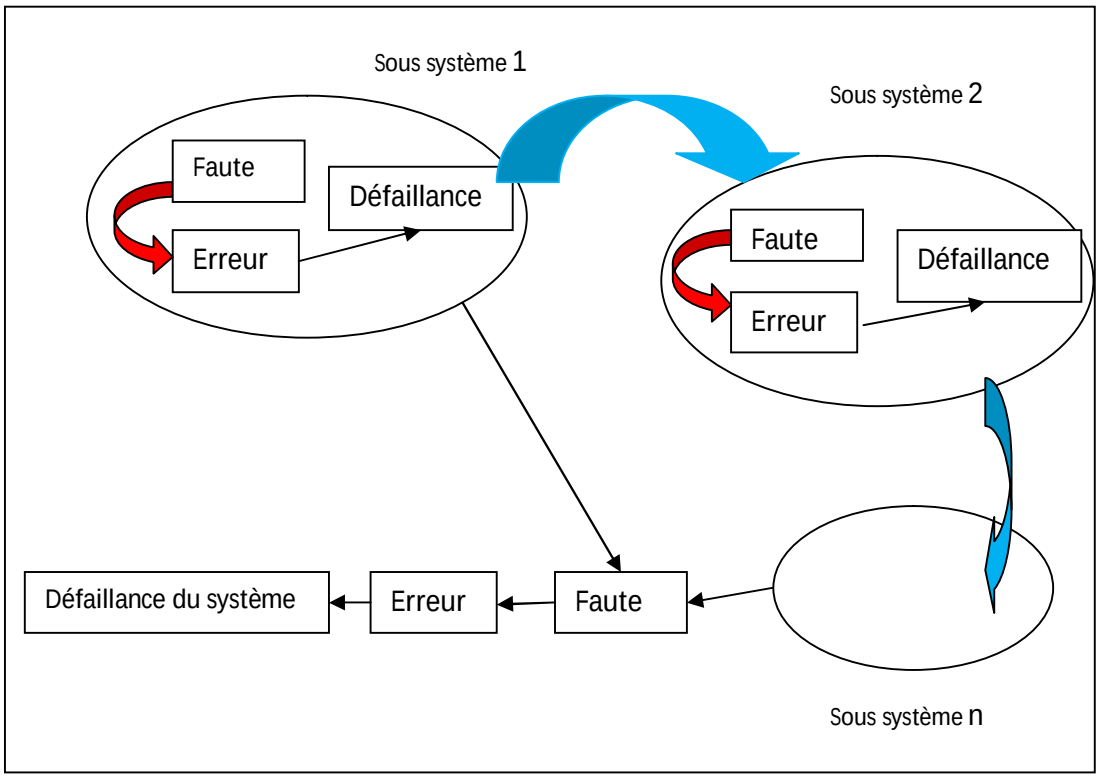

FIG 3. 1 Relation entre faute, erreur et défaillances

Dans la figure FIG 3.1, la défaillance d'un système est la conséquence d'une erreur, et l'erreur est la conséquence d'une faute activée. En plus, étant donné qu'un système informatique est souvent composé de plusieurs sous-systèmes, la défaillance d'un sous-système peut créer et/ou activer une faute dans un autre sous systèmes, ou dans le système lui-même.

Le terme ***panne*** existe également et est très populaire, notamment dans l'expression ***tolérance aux pannes***. Cependant, les spécialistes de la sûreté de fonctionnement le trouvent mal défini et préfèrent utiliser plutôt celui de défaillance (quand ou veut dire que le système ne délivre plus le service attendu) ou celui de faute (quand on cherche la raison pour laquelle le système est défaillant).

Dans un système informatique, trois principales classes de fautes sont à envisager :

Les fautes matérielles.

Les fautes logicielles.

Les fautes d'interaction.

Les fautes ***matérielles*** affectent les composants matériels du système : processeurs, mémoires, capteurs, actionneurs, media de communication...

Les fautes ***logicielles*** affectent les composants logiciels du système et sont communément appelées « bugs ».

Les fautes ***d'interaction*** concernent les interactions entre le système et son environnement.

On peut classifier les fautes d'un système selon plusieurs critères tels que:

- **leur nature:** accidentelle ou intentionnelle.
- **leur origine**: physique, humaine, interne, externe.
- **leur persistance**: permanente ou temporaire.

Et on peut aussi classifier les défaillances des systèmes selon les critères suivants:

- **leur domaine**: en valeur ou temporelle
- **détectabilité**: par l'utilisateur;
- **leurs conséquences**

En particulier, les systèmes dont toutes les défaillances sont temporelles *et* permanentes sont appelés **systèmes à silence sur défaillance** (« fail-silent » en anglais). Ces systèmes sont relativement faciles à étudier en raison de la nature même de leurs défaillances (qu'on appelle aussi « crash »). À l'inverse, les systèmes ayant des défaillances en valeur et incohérentes sont beaucoup plus difficile à étudier. Suivant les travaux de Lamport et al, les défaillances en valeurs incohérentes sont communément appelées **défaillances byzantines**. Dans les applications critiques, l'hypothèse de silence sur défaillance a longtemps été considérée et à juste titre, comme étant trop restrictive pour des composants non dotés de mécanismes d'autotest, ce qui est le cas pour la très grande majorité des composants sur étagère. Ainsi, Brasileiro et al, ont proposé une mise en œuvre logicielle de nœuds à silence sur défaillance avec deux processeurs ; selon le type de protocole de communication utilisé et la perte de performances par rapport à un nœud monoprocesseur qui varie de 80% à 60%. Toutefois, une étude récente de Baleani et al. proposent des architectures de système-sur-puce (« system-on-chip ») à silence sur défaillance ; ces circuits ont un surcoût acceptable par rapport à des circuits usuels, ce qui, en fait de bons candidats pour être vendus sur étagère.

3.2.5 Taux de défaillance

Définition 3.2 (taux de défaillance) *Le taux de défaillance d'un composant est une fonction du temps, note* $\lambda(t)$ *qui peut varier dans le temps et donne une fréquence d'occurrence instantanée pour un intervalle de temps très court. Cette fréquence d'occurrence instantanée augmente généralement avec le temps.* [19]

La probabilité qu'un système défaille à un instant donné ***t*** peut être représenté par la loi exponentielle, le taux de défaillance de chaque composant est représenté par un constant strictement positive $\lambda(t) = \lambda$ donc pour tout $t > 0$:

$$R(t) = 1 - e^{-\lambda t} \qquad (3.2)$$

Les processeurs modernes à silence sur défaillance peuvent avoir un taux de défaillance de l'ordre de 10^{-6} / *hg*. Par exemple, la probabilité qu'un processeur, dont le taux de défaillance est égal à 10^{-6} / *hr*, soit opérationnel pendant 3 heures est $exp^{10^{-6}*3} \approx 0.999998$. Quant à sa probabilité de défaillance pendant cette même durée, est $1 - exp^{10^{-6}*2} \approx 0.0002$.

De plus, il faut distinguer les fautes **« à chaud »** des fautes **« à froid »**. Dans le premier cas, un composant ne peut subir une défaillance que quand il est actif, alors que dans le second cas un composant peut subir une défaillance n'importe quand. L'hypothèse de faute étant qu'une défaillance n'affecte qu'une seule tâche, c'est le modèle « à chaud » qui est pertinent.

Il existe plusieurs représentation du taux de défaillance telles que:

- **MTTF (Mean Time To Failure)** mesure le temps moyen jusqu'a l'occurrence de la première défaillance.
- **MTBF (Mean time between two failure)** représente le temps moyen entre deux défaillances.

3.3 Modèles pour le calcul de la fiabilité

Le calcul de la fiabilité d'un système nécessite la définition d'un modèle de fiabilité. Le choix d'un tel modèle de fiabilité a un effet direct sur le niveau de confiance de ce calcul. Ce modèle doit définir les paramètres de performance des composants logiciels et/ou matériels du système (tels que les taux de défaillance), le niveau et le type de la redondance si elle existe, ainsi que les hypothèses de défaillance. Il existe dans la littérature plusieurs modèles de fiabilité. Les modèles les plus utilisés sont : les modèles combinatoires [1], les modèles basés sur les chaînes de Markov [14], les modèles basés sur les réseaux de Petri [13] et les modèles basés sur la théorie de l'ordonnancement [28]. Nous les présentons dans les paragraphes suivants.

3.3.1 Modèles combinatoires

Généralement, ces modèles utilisent la théorie des graphes pour représenter graphiquement (par un graphe orienté) toutes les combinaisons d'événements élémentaires qui peuvent causer la défaillance d'un système. À chaque nœud son prédécesseur du graphe, qui représente un événement élémentaire, est associé un ensemble de paramètres de performance, telle que la probabilité de son apparition. La fiabilité de ce système est calculée à partir d'une analyse quantitative de chaque graphe. Les deux modèles les plus fréquemment

utilisés sont le diagramme de blocs et l'arbre de fautes. Par exemple, l'arbre de fautes est constitué de plusieurs niveaux, où chaque nœud d'un niveau supérieur représente une combinaison de deux ou plusieurs événements liés aux nœuds de niveau inférieur. Les feuilles de l'arbre représentent les événements élémentaires qui peuvent causer la défaillance d'un système et la racine de l'arbre représente l'événement de défaillance du système. Donc, la probabilité que le système défaille est la probabilité d'atteindre la racine de l'arbre à partir de ses feuilles. Les modèles combinatoires sont faciles à comprendre, mais il n'est pas facile de représenter le comportement non indépendant des événements, au sens probabiliste

3.3.2 Modèles basés sur la chaine de Markov

Les chaînes de Markov permettent de modéliser le comportement dynamique d'un système par un graphe d'états, qui représente tous les états du système et les transitions possibles entre ces états. Les transitions sont pondérées par des probabilités suivant des lois exponentielles. Le calcul de la fiabilité d'un système peut être effectué grâce à des méthodes de résolution numérique ou par simulation. À la différence des modèles combinatoires les chaînes de Markov permettent la modélisation des événements non indépendants et aussi des événements de réparation des composants du système. Cependant, l'espace d'état peut grossir exponentiellement avec le nombre de composants d'un système, d'où des problèmes algorithmiques pour calculer la fiabilité.

3.3.3 Modèles basés sur les réseaux de pétri

Le comportement dynamique d'un système est ici représenté par un ensemble d'états, de jetons et de transitions. À la différence des modèles combinatoires, les transitions peuvent être associées à n'importe quel type de loi probabiliste. Les réseaux de Petri peuvent être utilisés pour générer des chaînes de Markov. En plus, ils peuvent être utilisés facilement pour représenter les caractéristiques des systèmes concurrents, telles que la synchronisation et le partage des ressources, et aussi pour valider des propriétés d'un système, telle que l'absence de blocage. Le calcul de la fiabilité est basé sur la simulation [24]. Le but de la simulation est d'appliquer à un système un ensemble de tests aléatoires, et d'utiliser ensuite les résultats de ces tests pour calculer la fiabilité de ce système. Cependant, la précision de ce calcul dépend du choix de l'ensemble de tests et augmente avec la durée de la simulation. Or la procédure de production des tests introduit toujours un biais, qui est difficile à mesurer.

3.3.4 Modèles basés sur les algorithmes d'ordonnancement

Il existe plusieurs travaux qui montrent que l'utilisation de la théorie de l'ordonnancement pour la conception des systèmes peut améliorer la fiabilité de ces systèmes, c'est pourquoi, nous nous sommes attachés à ce modèle. Dans ce modèle, un taux de défaillance est associé à chaque composant matériel et le calcul de la fiabilité est basé sur une fonction d'évaluation (Fiab) qui permet d'évaluer la probabilité du bon fonctionnement du système (ou de l'allocation résultante d'un algorithme de distribution/ordonnancement). La fiabilité d'un système dépend donc de l'allocation des composants logiciels de l'algorithme sur les composants matériels de l'architecture.

La conséquence de l'utilisation d'une telle fonction d'évaluation est que l'algorithme de distribution/ ordonnancement doit prendre en compte les taux de défaillance des composants matériels durant sa phase d'allocation spatiale des composants logiciels sur les composants matériels.

3.4 Technique d'évaluation de la fiabilité

Dans la littérature, il existe plusieurs techniques pour évaluer la fiabilité des systèmes, les techniques les plus utilisées sont:

3.4.1 Arbre de défaillance

Les arbres de défaillances modélisent l'ensemble des combinaisons d'événements qui conduisent à un événement redouté.

Definition3.4 (arbre de défaillance) *L'arbre de défaillance est une représentation graphique de type arbre généalogique. Il représente une démarche d'analyse d'événement. L'arbre de défaillance est construit en recherchant l'ensemble des événements élémentaires ou les combinaisons d'événements, qui conduisent à un Evénement Redouté (ER)* ».

L'objectif est de suivre une logique déductive en partant d'un Evénement Redouté pour déterminer de manière exhaustive l'ensemble de ses causes jusqu'aux plus élémentaires.

Exemple

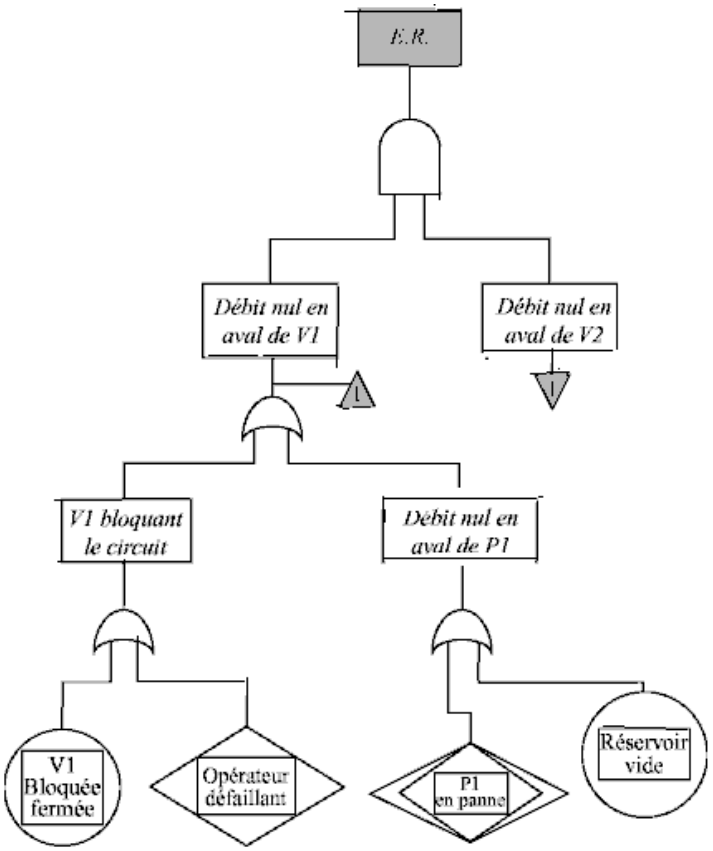

FIG 3.2 Exemple d'un arbre de défaillance

Les objectifs des arbres de défaillance sont résumés en quatre points :

- La recherche des événements élémentaires, ou leurs combinaisons qui conduisent à un ER.
- La représentation graphique des liaisons entre les événements. Il existe une représentation de la logique de défaillance du système pour chaque ER ; ce qui implique qu'il y aura autant d'arbres de défaillance à construire que d'ER retenus.
- La représentation graphique des liaisons entre les événements. Il existe une représentation de la logique de défaillance du système pour chaque ER ; ce qui implique qu'il y aura autant d'arbres de défaillance à construire que d'ER retenus.
- Enfin, il est possible d'évaluer la probabilité d'apparition de l'ER connaissant la probabilité des événements élémentaires : c'est l'analyse quantitative qui permet de déterminer les caractéristiques de fiabilité du système étudié. L'objectif est en particulier de définir la probabilité d'occurrence des divers événements analysés. Les calculs reposent sur les équations logiques tirées de la structure de l'arbre de défaillance et des probabilités d'occurrence des événements élémentaires.

On distingue trois types d'événements

Definition3.5 (**événement redoute**) *"l'événement redouté est l'événement indésirable pour lequel on fait l'étude de toutes les causes qui y conduisent. Cet événement est unique pour un arbre de défaillance et se trouve au "sommet" de l'arbre."*

Avant de commencer la décomposition qui permet d'explorer toutes les combinaisons d'événements conduisant à l'événement redouté, il faut définir avec précision cet événement ainsi que le contexte de son apparition.

L'événement redouté est représenté par un rectangle au sommet de l'arbre

Définition 3.6 (**événement intermédiaire**) "les événements intermédiaires sont des événements à définir comme l'événement redouté. La différence avec l'événement redouté est qu'ils sont des causes pour d'autres événements. "

Par exemple c'est la combinaison d'événements intermédiaires qui conduit à l'événement redouté.

Un événement intermédiaire est représenté par un rectangle comme l'événement redouté

Définition 3.7 (**événement élémentaire**) "*les événements élémentaires sont des événements correspondants au niveau le plus détaillé de l'analyse du système.* "

Dans un arbre de défaillance, ils représentent les défaillances des composants qui constituent le système étudié. Pour fixer le niveau de détail de l'étude, on considère en général que les événements élémentaires coïncidents avec la défaillance des composants qui sont réparables ou interchangeables.

Les événements élémentaires sont représentés par des cercles.

On utilise les portes logiques pour représenter la combinaison logique des événements intermédiaires qui sont à l'origine de l'événement décomposé.

Porte AND : dans ce porte un événement redoute est opérationnel si tous ses événements élémentaires sont opérationnels.

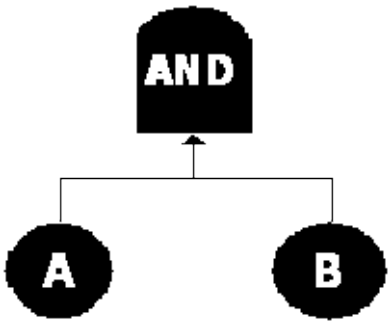

FIG 3.3 Exemple de porte AND

Dans ce cas l'équation de la fiabilité est $R_{systeme} = R_A + R_B - R_A * R_B$.

Porte OR : dans ce porte un événement redoute est opérationnel si et seulement si un parmi ses événements élémentaires est opérationnel.

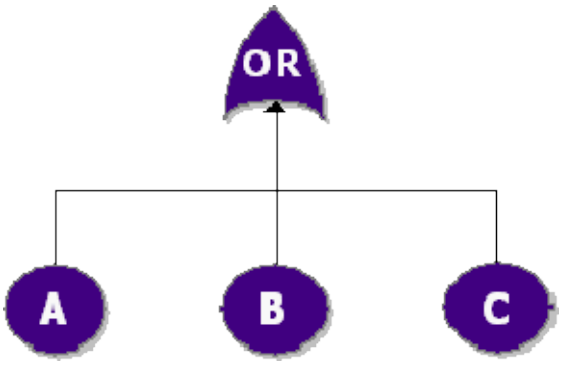

FIG 3.4 Exemple de porte OR

Dans ce cas l'équation de la fiabilité est $R_{systeme} = R_A * R_B * R_C$.

3.4.2 Bloc-diagramme de Fiabilité

Un BDF est un graphe orienté (N, E), dont chaque sommet de N est un bloc représentant un composant du système et chaque arc de E est un lien de causalité (dépendance) entre deux blocs. Dans tout BDF, deux blocs particuliers sont identifiés: ceux sont sa source S et sa destination D. Un BDF représente un système et il est utilisé pour calculer la fiabilité : un BDF est opérationnel si et seulement si, il existe au moins un chemin opérationnel de S à D. Un chemin est opérationnel si et seulement si, tous les blocs de ce chemin sont opérationnels. La probabilité qu'un bloc soit opérationnel est sa fiabilité. Par construction, la probabilité qu'un BDF soit opérationnel est donc égale à la fiabilité du système qu'il représente.

Quand le BDF est construit, on distingue trois types de système : série, parallèle ou série parallèle.

Définition 3.3 (Système en série) *Deux blocs sont en série si le fonctionnement des deux est nécessaire pour assurer le fonctionnement de l'ensemble.* [19].

La fiabilité de se système est calculée par la formule suivante

$$R = \prod_{i=1,n} r_b \qquad (3.3)$$

Où r_b représente la fiabilité d'un bloc et n est le nombre de bloc en série.

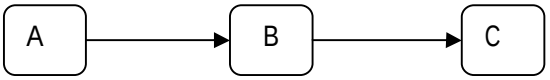

FIG 3. 5 Configuration série

Définition 3.4 (Système en parallèle) *Deux blocs sont en parallèle si le fonctionnement d'au moins un des deux est suffisant pour assurer le fonctionnent de l'ensemble.*

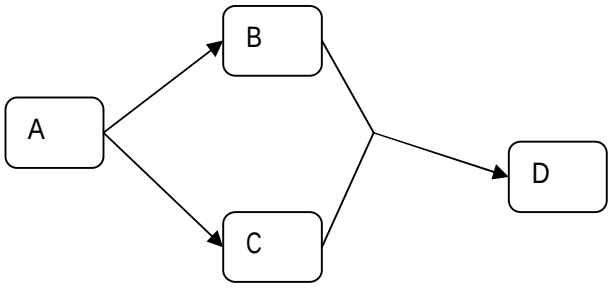

FIG 3. 6 Configuration parallèle

La fiabilité de se système est calculée par la formule suivante :

$$R = 1 - \prod_{i=1,n} r_b \qquad (3.4)$$

Définition 3.5 (Système Série/Parallèle) *Deux éléments sont en série/parallèle si le fonctionnement d'au moins un sous élément de chacun est suffisant pour assurer le fonctionnement de l'ensemble*

La fiabilité de se système est calculée par la formule suivante :

$$R = 1 - \prod_{i=1,n} r_b \qquad (3.5)$$

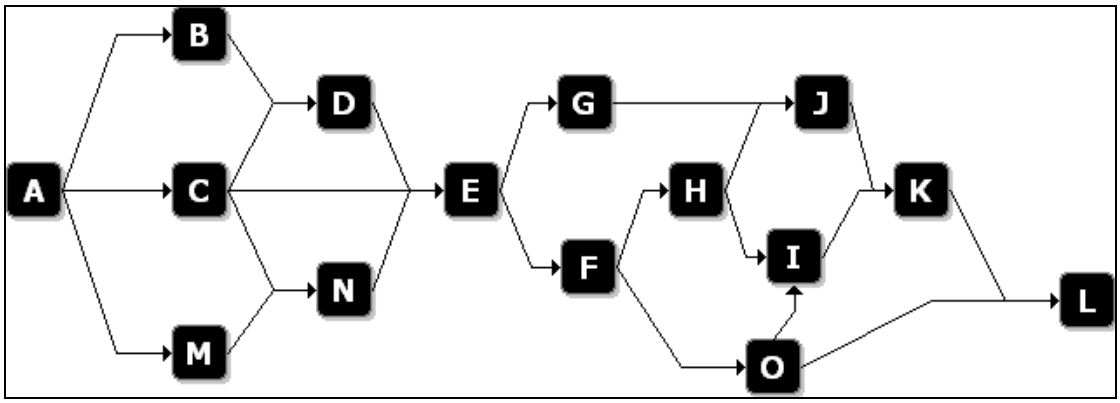

FIG 3. 7 Exemple d'une configuration série-parallèle

Dans le cas où le système de structure quelconque, la méthode proposé dans [19] une solution basée sur la transformation des graphes pour transformer un système de structure quelconque à un système série-parallèle.

3.4.3 Representation BDD (binary decision diagrams)

BDD est un graphe acyclique direct basé sur la décomposition de Shannon [25] :

- Si ***F*** est une fonction booléenne sur les variable$x_1, \dots, x_n$, alors pour chaque variable la décomposition de Shannon est défini par :

$$F = x_1 \cap F(x_1 = 1) \cup not(x) \cap F(x_1 = 0) \qquad (3.6)$$

- L'application itérative de la formule de décomposition de Shannon avec le respect de l'ordre de séquence des variables, On obtenant la décomposition globale de Shannon.
- Chaque nœud représente une variable. (composant de système)
- Pour chaque nœud, la branche gauche représente le sous arbre correspondant au cas où la variable est vrai c'est-à-dire le composant est opérationnel, et la branche droite représente le sous-arbre correspondant au cas où le composant n'est pas opérationnel
- Si la probabilité de x_i est p_i alors, on peut calculer la probabilité de $P\{F\}$ par l'application récursive de la formule suivante :

$$P\{F\} = p_1 \, P\{F_{x_1=1} = 1\} + (1 - p_1)\{F_{x_1=0} = 0\} \qquad (3.7)$$

Les différentes implémentations de BDD sont basées sur les deux algorithmes :

- La recherche de plus court chemin (minpath).
- La visite récursive du graphe.

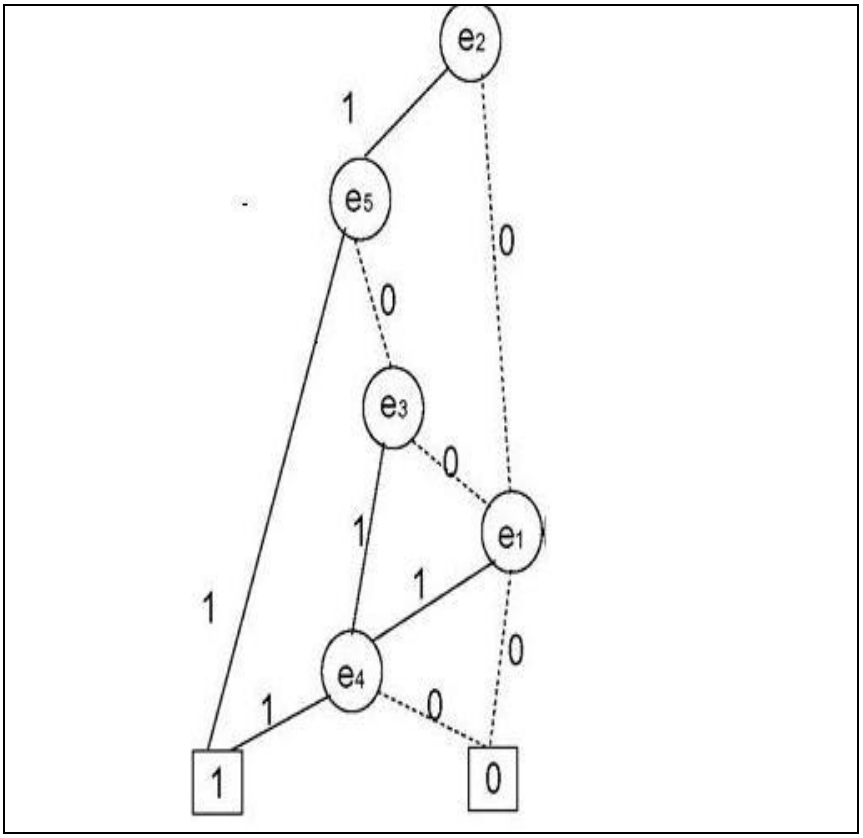

FIG 3.8 Exemple de représentation BDD

3.4.4 Ensemble Coupes minimale (mincut set) et Chemin Plus Court (minpath)

Dans certain système, le calcul exacte de la fiabilité est impossible, parce que trop couteux, la méthode la plus répondu est l'ensemble des coupes minimales, ou le chemin le plus court. [7]

Définition 3.6 (chemin) *Pour un graphe donne* ***G = (V, E)****, un chemin est un sous-ensemble des composants, arcs et/ou nœud qui garantit que la source O et la destination Z sont reliés si tous les composants de ce sous-ensemble fonctionnent. Un chemin H est un chemin court (minpath) s'il n'existe pas un sous-ensemble d'éléments dans H qui est également un chemin.*

Définition 3.7 (coupe) *Pour un graphe donné* ***G = (V, E),*** *une coupe est un sous-ensemble de composants, d'arcs et/ou de nœuds, si dont l'échec déconnecte la source O et la destination Z. Une coupe K est un mincut s'il n'existe pas un sous-ensemble d'éléments en K qui est également une coupe.*

Si on Considère un système général où les ensembles de coupe minimale sont dénotés par A_i où $1<i<n$ (n est le nombre de min cut set) alors, la probabilité de ce système est calculé par la formule suivante [5] :

$$Pr(S) = pr(A_1 + A_2 \ldots\ldots + A_n) \tag{3.8}$$

Les limites supérieures et inférieures de la fiabilité de système sont données respectivement par [5]:

$$pr(A_1 + A_2 \ldots\ldots + A_n) \leq \sum_{1}^{n} P(A_n) \tag{3.9}$$

$$pr(A_1 + A_2 \ldots\ldots + A_n) \geq \sum_{i=1}^{n-m} \sum_{j=n-m}^{n} P(A_n A_m) \tag{3.10}$$

Pour résoudre l'équation (3.8) plusieurs techniques sont développées :

- Algorithme d'expansion d'inclusion -exclusion. [27].
- Les minpath ou minset est la somme de produit disjoint SPD [4]
- La représentation booléenne de diagramme de décision [25]

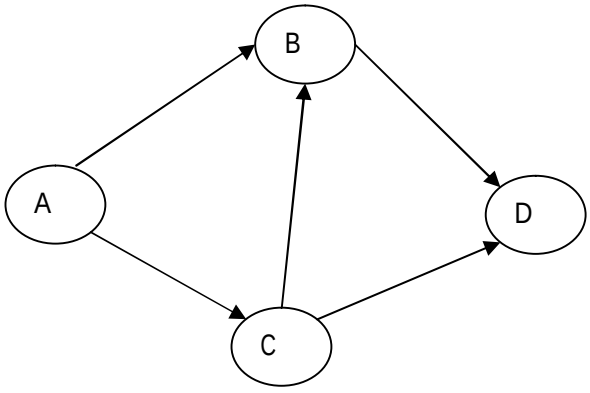

FIG 3. 1 Exemple d'un graphe oriente

Le graphe dans FIG 3. 5 possède deux minpaths et trois mincuts. Qui sont respectivement :

$$H_1 = \{ A, B , D\} ; H_2 = \{ A, C, D\}$$

$$K1 = \{A\} ; K2 = \{B, C\} ; K3 = \{D\}$$

La fiabilité de ce système est calculée par la formule suivante :

$$F_S = F(A)\big(F(B) + F(C)\big)F(D)$$

3.4.5 algorithme direct

Les algorithmes directes sont dédiés à trouver la fonction logique de connectivite du graphe et l'expression de fiabilité de système sans le passage par les algorithmes de recherche de mincut set et minpath. Plusieurs techniques sont développées dans ce but. L'algorithme proposé dans [15] est basé sur la construction et la sauvegarde de différent pathset. Donc, l'expression logique de connectivité du graphe est la somme des termes logiques et l'expression de fiabilité est la somme des produits (SDP). Un autre algorithme basé sur la construction de BDD sans le passage de la construction des expressions logiques, l'algorithme propose dans [25] utilise les algorithmes de factorisation pour calculer la fiabilité.

L'algorithme présenté dans [27] génère la représentation BDD sans le passage par les expressions logiques :

```
gen_bdd(noeud_dedut)

{   T_bdd = 0

   Regrouper noeud_debut dans ce chemin

      pour (arc_i dans l'ensemble des arc debut a partir noeud_debut) faire

{   noeud_suiv = noeud distination de arc_i

        Si (noeud_suiv == noeud_sink)      alors

              souschemin_bdd = arc_i_bdd

       Si non

          Si (noeud_suiv est tjrs dans ce chemin)

            Continue;

          Si non

             souschemin_bdd  = gen_bdd(noeud_suiv) AND arc_i_bdd

            T_bdd=T_bdd OR souschemin_bdd }

effacer noeud_debut dans ce_chemin

retourne T_bdd}
```

3.4 Conclusion

Nous avons présenté dans ce chapitre les différentes techniques existantes dans la littérature pour évaluer la fiabilité d'un système, dans ce qui suit, nous donnerons un état de l'art des algorithmes de distribution et d'ordonnancement temps réel fiable.

Chapitre 4

État de l’art : Algorithme d’ordonnancement et de distribution bi-critères

4.1 Introduction

Durant le processus de conception d'un système temps réel, le but d'un algorithme de distribution et d'ordonnancement est d'allouer spatialement et temporellement les composants logiciels sur les composants matériels, tout en respectant les contraintes temporelles et matérielles, cependant, cette allocation ne prend pas en compte deux critères en même temps.

Le problème qui consiste à définir un algorithme d'ordonnancement et distribution qui prend en charge plus d’un seul critère peut être ramené à un problème d'ordonnancement et d'optimisation multicritère.

Notre but dans ce travail est de trouver un algorithme de distribution et d'ordonnancement qui minimise la longueur de l'ordonnancement et maximise la fiabilité du système.

Nous nous intéressons dans ce chapitre sur les différents algorithmes d'ordonnancement existant dans la littérature pour la résolution du problème d'optimisation bi-critère.

4.2 Problème d'optimisation bi-critères

Avant de décrire les différents algorithmes d'ordonnancement et d'optimisation bi-critères, nous devons tout d'abord donner une définition du problème d'optimisation. La définition suivante est donnée par [3].

Définition (4.1) (Problème d'optimisation) *"un problème d'optimisation est défini par un espace d'état, une ou plusieurs fonctions d'objectifs, et un ensemble de contraintes."*

Où :

- **Espace d'état** : est défini comme l'ensemble des domaines de définition des variables du problème.
- **Fonction objectif** - représente le but à atteindre par le décideur.
- **L'ensemble des contraintes** - définis les conditions sur l'espace d'état que les variables doivent satisfaire.
- **Une méthode d'optimisation** - recherche le point ou un ensemble de point de l'espace des états possibles qui satisfait au moins un ou plusieurs critère(s), le résultat est appelé valeur optimale ou optimum.

Notre objectif dans ce mémoire est de trouver un algorithme de distribution et d'ordonnancement bi-critères qui minimise la longueur de l'ordonnancement et qui maximise la fiabilité du système, malheureusement, ces deux objectifs sont contradictoire [12].

4.3 Classification des algorithmes bi-critères

Il existe dans la littérature plusieurs recherches pour résoudre ce problème bi-critères. Nous les classons en trois grandes classes [14] :

4.3.1 Agrégation des deux critères en un seul

4.3.1.1 Principe

Cette méthode de résolution de problème d'optimisation bi-critère est la plus utilisée dans la littérature, elle permet de transformer un problème bi-objectif à un problème mono-objectif.

Par exemple, nous prenons chacune des deux fonctions d'objectifs et leur appliquer un coefficient de pondération et ensuite on fait la somme ou le produit des fonctions objectifs, on obtient alors une seule fonction objectif.

Nous pouvons formaliser cette approche comme suit [8]:

$$\begin{cases} \max f(x) = \sum_{i=1}^{n} \delta_i \, f_i(x) \\ \sum_{i=1}^{n} \delta_i = 1 \end{cases} \quad (4.1)$$

Et sa représentation graphique comme suit :

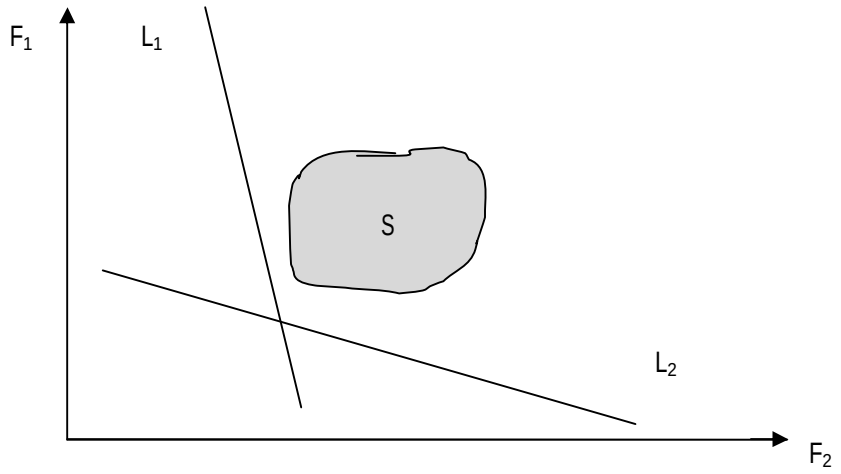

FIG 4. 1 La méthode d'agrégation de deux critères dans un seul

Dans la figure 4.1, *S* correspond a l'ensemble des valeurs des pairs $(f_{1,} f_2)$, les droites L_1 et L_2 représente les coefficients de pondération, cette approche consiste a tangente la droite L_1 et la droite L_2 avec l'ensemble *S*, le point de tangence représente la solution cherchée, l'itération de ce processus donne l'ensemble des solutions optimales.

Pour que cette approche fonctionne bien, nous devons être capables de bien choisir les coefficients de pondération. Sinon, les solutions que nous trouverons ne seront pas forcément optimales.

4.3.1.2 Présentation de quelques algorithmes

Dogan et Ozgüner ont proposé dans [10] et [11] deux extensions différentes de la méthode DLS (dynamic level scheduling) de Sih et Lee. L'algorithme DLS calcule à chaque étape le niveau dynamique pour chaque tâche libre sur tous les processeurs, la paire tâche-processeur qui possède le niveau dynamique le plus maximale est sélectionné pour ordonnancer cette fonction de coût qui est calculée de la manière suivante:

$$DL(t_i, p_j) = SL(t_i) - \max(t^A_{(i,j)}, t^M_j) + \delta(t_i, p_j) \quad (4.2)$$

Où :

$SL(t_i)$: représente le niveau statique.

$\max\left(t^A_{(i,j)}, t^M_j\right)$: La date de début de tâche t_i sur la machine (processeur) p_j

$\delta(t_i, p_j)$: La vitesse du processeur p_j

L'apport de Dogan et Ozgüner dans [10] est d'ajouter un nouveau terme à la fonction de coût DLS pour prendre en compte la fiabilité (RDLS: Reliability Level Dynamic Scheduling). Ils considèrent que l'occurrence des défaillances des ressources matérielles (processeurs et liens de communication) suit une loi de Poisson de la forme $\exp^{-\lambda t}$, où λ est le taux de défaillance de la ressource et t est son temps d'utilisation. Ils supposent de plus, que l'architecture est hétérogène pour les fiabilités, que les défaillances des diverses ressources matérielles sont permanentes et que ce sont des événements statistiquement indépendants.

Dans RDLS, les trois premiers termes servent pour trouver le temps d'exécution le plus minimum. Le quatrième terme permet d'introduire la fiabilité dans le calcul de fonction de coût désigné par $C(t_{i,}\ _{mj})$ donc le nouveau calcul de niveau dynamique se fait de la façon suivante:

$$DL(t_i, p_j) = SL(t_i) - \max\left(t^A_{(i,j)}, t^M_j\right) + \delta(t_i, p_j) \quad - C(t_i, p_j) \qquad (4.3)$$

L'inconvénient principal de cette méthode est que le temps d'exécution peut diminuer la fiabilité du système.

Pour résoudre ce problème Dogan et Ozgüner proposent un autre algorithme dans [11] qui calcule à chaque étape deux tableaux :

- le premier tableau est trié selon l'ordre croissant de la fonction *DLS*.
- le deuxième tableau est trié selon l'ordre croissant de $\delta COST$ qui rend compte de la baisse de fiabilité due à cette même décision.

Ainsi, chaque paire (t_i, p_j) a un rang dans chacune de des deux listes triées, respectivement notée $Rank^1_{(i,j)}$ et $Rank^2_{(i,j)}$. Ces deux rangs sont alors combinés en:

$$Rank_{(i,j)} = \delta_1\ Rank^1_{(i,j)} + \delta_2\ Rank^2_{(i,j)} \qquad (4.4)$$

Où les deux nombres δ_1 et δ_2 sont les coefficients de pondération choisis de manière à équilibrer les deux critères. Enfin, la paire (t_i, p_j) ayant le plus petit $Rank_{(i,j)}$ est choisie et $t_i$est alors placée surp_j.

Dans ces deux travaux ([10] et [11]), la fiabilité est améliorée uniquement en choisissant d'ordonnancer certaines tâches d'*Alg* sur un processeur plutôt qu'un autre, en étant pour cela guidé par le calcul de la fiabilité. On voit donc que la fiabilité, qui est a priori une mesure de la tolérance aux fautes, peut être utilisée comme un moyen. Les résultats des simulations effectuées par Dogan et Özgüner montrent que leur algorithme RDSL est systématiquement.

L'algorithme présente dans [2] [16] [21] désigné par AAA-F est une heuristique gloutonne de type ordonnancement de liste, il utilise la redondance active des composants logiciels pour améliorer la fiabilité, AAA-F est une variante de la méthode AAA (Adéquation Algorithme Architecture) de deux choses. D'une part, le calcul de la fonction de coût et d'autre part, l'utilisation de plusieurs processeurs au lieu d'un seul. AAA-F fait la compromise entre deux fonctions le gain en longueur et la perte en fiabilité, AAA-F calcule à chaque étape les grandeurs suivantes pour toutes les tâches libres (candidates) sur tous les processeurs:

Gain en longueur = $$L\left(Ord^{(n)}(o_i, p_j)\right) - L(Ord^{(n-1)})$$ (4.5)

Marge en longueur= $$L(Ord_{obj}) - L(Ord^{(n-1)})$$ (4.6)

Perte en fiabilité = $$F\left(Ord^{(n)}(o_i, p_j)\right) - F(Ord^{(n-1)})$$ (4.7)

Marge en fiabilité = $$F(Ord_{obj}) - F(Ord^{(n-1)})$$ (4.8)

Où :

- $L\left(Ord^{(n)}(o_i, p_j)\right)$ **:** désigne la longueur d'ordonnancement quand placer o_i sur p_j à l'étape n.
- $L(Ord^{(n-1)})$ **:** désigne la longueur d'ordonnancement à l'étape n-1.
- $L(Ord_{obj})$: désigne la longueur d'ordonnancement objectif qu'on veut atteindre.
- $F\left(Ord^{(n)}(o_i, p_j)\right)$: désigne la fiabilité d'ordonnancement quand placer o_i sur p_j à l'étape n.
- $F(Ord^{(n-1)})$ **:** désigne la fiabilité d'ordonnancement à l'étape n-1
- $F(Ord_{obj})$: désigne la fiabilité d'ordonnancement objectif qu'on veut atteindre.

Puisque les deux critères ne sont pas de même ordre de grandeur, il doit normaliser les deux objectifs:

$$L^n = \frac{L\left(Ord^{(n)}(o_i, p_j)\right) - L(Ord^{(n-1)})}{L(Ord_{obj}) - L(Ord^{(n-1)})} \quad (4.9)$$

$$G^n = \frac{F\left(Ord^{(n)}(o_i, p_j)\right) - F(Ord^{(n-1)})}{F(Ord_{obj}) - F(Ord^{(n-1)})} \quad (4.10)$$

Donc la fonction de coût est calculée de la manière suivante:

$$f(o_i, p_j) = \cos\theta\, L^n + \sin\theta\, G^n \quad (4.11)$$

Pour chaque tâche AAA-F choisit le couple (o_i, p_j) qui minimise f et parmi les tâches candidates AAA-F choisit le couple (o_i, p_j) qui maximise f pour placer o_i sur p_j.

Hakem et Butelle présentent dans [18] une heuristique d'ordonnancement bi-critère désignée par BSA (bi-objectif Scheduling Algorithem), Hakem utilise une fonction de calcul de la fiabilité différente de celle définie dans les algorithmes précédents, il calcule la probabilité de défaillance de tâche sélectionnée sur le processeur et ne prend pas en compte les tâches déjà ordonnancées. À chaque étape, BSA attribue la priorité supérieure à la tâche libre qui appartient au chemin critique ou le chemin le plus long et il est noté tâche critique, ensuite il sélectionne la tâche critique et ordonnance sur le processeur qui minimise la fonction de coût suivante:

$$f(t_i, p_j) = \sqrt{\theta\left(\frac{f(t_i, p_j)}{\max_{p\in P} f(t_i, p)}\right)^2 + (1-\theta)\left(\frac{\varepsilon(t_i, p_j)}{\max_{p\in P} \varepsilon(t_i, p)}\right)^2} \quad (4.12)$$

Où :

$f(t_i, p_j)$: Temps de fin d'exécution de t_i sur p_j

$\max_{p\in P} f(t_i, p)$: Temps maximum de fin d'exécution de t_i sur tous les processeurs.

$\varepsilon(t_i, p_j)$: La probabilité de défaillance de composant logiciel $t_i$sur composant matériel p_j

$\max_{p \in P} \varepsilon(t_i, p)$: La probabilité de défaillance de composant logiciel $t_i$sur tous les processeurs.

4.3.2 Transformation d'un critère en contrainte

4.3.2.1 Principe

Cette approche aussi permet de transformer un problème bi-objectif en un problème mono-objectif de telle sorte, on choisit une fonction prioritaire et ensuite en transformer le problème en conservant la fonction prioritaire et l'autre fonction objectif comme une contrainte. On peut la formaliser comme suit**:**

$$\begin{cases} \max f_1(x) \\ sc\ f_2(x) < M \end{cases} \tag{4.13}$$

On peut représenter cette méthode par le graphe suivant:

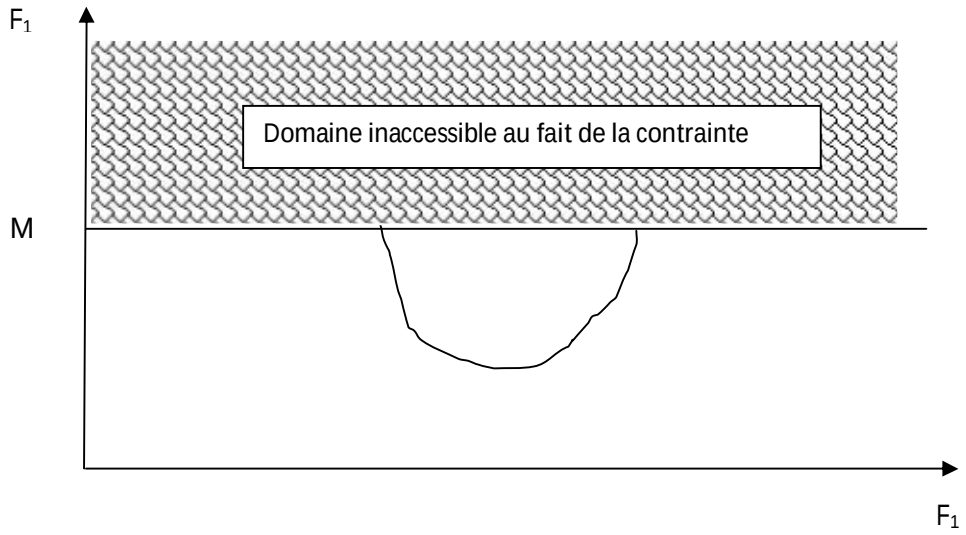

FIG 4. 2 Transformation d'un critère en contrainte

Cependant, la relative simplicité de l'énoncé de la méthode l'a rendue populaire.

4.3.2.2 Présentation de quelques algorithmes

Xiao et al. présentent dans [25][26] une heuristique de distribution et d'ordonnancement bi-critère en ligne appelée RCD, cette dernière est basée sur deux algorithme ***SORT-EDF*** (Storting Algorithm) et ***AEAP*** (As Early As Possible), ***SORT-EDF*** génère le séquencement des tâches, et ***AEAP*** ordonnance les tâches selon le temps de début le plus court. Donc, ***RCD*** permet à chaque étape de placer t_i sur p_j le plus fiable et ordonnancer selon le temps de début le plus court. Le modèle de fiabilité est le même que celui de Dogan et Ozgüner. Qin et al. définissent le coût de fiabilité de la tâche t_i ordonnancée sur le processeur p_j comme étant le produit de λ_j par le temps d'exécution de t_i sur p_j

4.3.3 Hiérarchisation des critères

4.3.3.1 Principe

Est appelée aussi "l'ordonnançabilité des critères", elle permet d'ordonner totalement les optimum de Pareto, puis résolution du problème dans l'ordre des critères: on optimise le premier critère: puis parmi les solutions optimales pour le premier critère on optimise le deuxième critère (ainsi de suite dans les cas multi critère).

Optimum de Pareto : Une solution du problème d'optimisation est un **optimum de Pareto** si et seulement si elle est optimale par rapport aux deux critères, c'est-à-dire qu'il n'existe aucune autre solution qui soit meilleure pour les deux critères à la fois ; mais il peut tout à fait exister d'autres solutions qui soient meilleures pour un des deux critères.

Formellement, la structure d'ordre utilisée dans R^2est définie par $x \leq y \Leftrightarrow \forall i, 1 \leq i \leq 2, x_i \leq y_i \ et \ x = y \Leftrightarrow \mathbf{1} \leq i \leq 2, x_i = y_i$, où x_i est la i-ième composante du vecteur x. Comme avec toute structure d'ordre, on écrit x ≠ y, ¬(x = y), et x < y ⇔ x ≤y ^ x ≠y. Ainsi donc, x est un optimum de Pareto ssi ∃y tel que Z(y) ≤ Z(x), où Z est la fonction de l'espace des solutions vers l'espace des critères (iciR^2). Si ∄y tel que $\forall i, 1 \leq i \leq 2$; $Z_i(y) \leq Z_i(x)$ avec au moins une inégalité stricte, alors x est un **optimum de Pareto strict**. Si ∃y tel que $\forall i, 1 \leq i \leq 2$; $Z_i(y) \leq Z_i(x)$, alors x est un **optimum de Pareto faible** (notons au passage que strict implique faible).

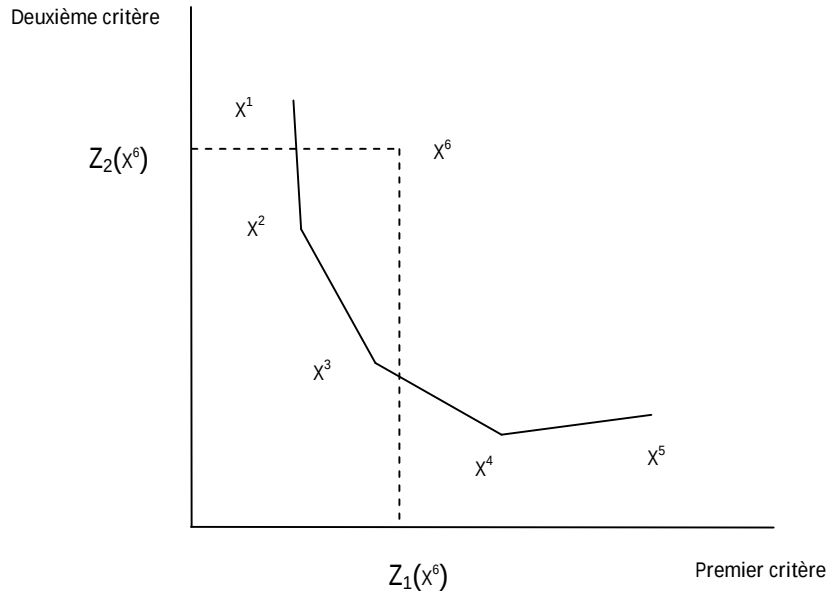

FIG 4. 3 Optima et courbe de Pareto pour un problème de minimisation bi-critère.

Graphiquement, on représente chaque solution du problème d'optimisation par un point dans le plan R^2, dont l'abscisse est la mesure du premier critère (c'est-à-dire $Z_1(x)$) et dont l'ordonnée est la mesure du second critère (c'est-à-dire $Z_2(x)$). Ceci est valable dans le cas de deux critères ; dans le cas de n critères, nous somme dans un espace à n dimensions. Sans perte de généralité, supposons que le problème consiste à minimiser les deux critères. La figure 4.3 illustre la notion d'optimum de Pareto : chaque point numéroté de x_1 à x_6 est une solution du problème d'optimisation bi-critère ; l'enveloppe convexe des points est appelée **courbe de Pareto** (même si en réalité c'est une ligne brisée plutôt qu'une courbe) ; tous les points qui sont sur la courbe de Pareto (ici x_1, x_2, x_3, x_4 et x_5) sont des optima de Pareto ; les points x_1 et x_5 sont des optima faibles, alors que les point x_2, x_3 et x_4 sont des optima stricts.

En général, il n'existe pas de solution qui optimise *les deux critères à la fois*. C'est le cas dans la figure 4.3 : les points x_1 et x_2 optimisent le premier critère mais pas le second ; pour les points x_4 et x_5 c'est le contraire ; quant au point x_3 il n'en optimise aucun. Le choix d'une solution est donc un compromis entre les deux critères. Les optima de Pareto jouent un rôle important dans la résolution des problèmes d'optimisation bi-critères puisqu'ils permettent à l'utilisateur de choisir son compromis parmi les meilleures solutions. Malheureusement, la recherche des optima de Pareto est un problème NP-difficile [13].

Nous pouvons formaliser cette approche comme suit:

$$\min f_1(x) \tag{4.14}$$

Nous notons f_1^* la solution de l'équation (4.14)

Ensuite, nous transformons le premier objectif en contrainte d'égalité puis nous prenons le deuxième objectif et nous résolvons le problème suivant:

$$\begin{cases} \max f_2(x) \\ sc\ f_1(x) = f_1^* \end{cases} \tag{4.15}$$

Si nous supposons que la solution de (4.15) est f_2^* alors la paire (f_1^*, f_2^*) est la solution qui minimise les deux objectifs.

L'inconvénient présenté par cette méthode est le mauvais choix de séquence des fonctions objectives à minimiser. Ce choix est largement arbitraire, un peu comme celui des coefficients de pondération dans la première approche.

Deux ordonnancements différents de fonction objective n'aboutissent pas généralement à la même solution.

4.3.3.2 Présentation de quelques algorithmes

L’algorithme présenté dans [12] [26] est une heuristique d'ordonnancement bi-critère qui cherche l’optimum de Pareto où la longueur d'ordonnancement est minimisée, il donne la priorité à la longueur d'ordonnancement et après, chercher de maximiser la fiabilité comme deuxième critère.

Le modèle de fiabilité est d’associé à chaque processeur un taux de défaillance, la fiabilité d'un ordonnancement est la probabilité qu'il finisse correctement et il est donné par la probabilité que tous les processeurs soient fonctionnels pendant l'exécution de toutes les tâches assignées

4.4 Discussion

L’algorithme de distribution et d'ordonnancement que nous proposons dans le chapitre suivant est parmi les algorithmes d'agrégation de deux critères en un seul, la solution est une extension de l'algorithme de distribution et d'ordonnancement de SYNDEX de telle sorte qu’on ajoute un deuxième objectif pour prendre la fiabilité du système en compte.

Chapitre 5

Algorithme d'Ordonnancement Bi-critères : temps réel et fiable

5.1 Problème d'ordonnancement et d'optimisation bi-critères

Dans ce chapitre, nous nous intéressons à une mesure qui permet d'évaluer la sûreté de fonctionnement d'un système qui est la fiabilité. Nous avons proposé une solution qui appartient aux méthodes d'agrégation de deux critères dans un seul, qui maximise la fiabilité du système et en même temps qui minimise la durée d'exécution de l'algorithme sur l'architecture tout en respectant les contraintes temporelles et matérielles.

5.1.1 Modèle de faute

Les composants matériels (processeurs et liens de communication) sont supposés être à silence sur défaillance. Les défaillances sont temporaires. La durée maximale d'une défaillance est telle qu'une défaillance affecte une seule tâche. Cela veut dire que la défaillance d'un processeur n'affecte que la tâche qui est en cours d'exécution sur le processeur au moment de la défaillance. De plus, l'occurrence des défaillances de chaque processeur p suit une loi de Poisson de paramètre constant **λ,** appelé « taux de défaillance de p par unité de temps ». Enfin, les défaillances des composants matériels sont des événements statistiquement indépendants.

5.1.2 Formalisation du problème

Énoncé du problème :

- *Une architecture matérielle distribuée hétérogène* ***ARC*** *composée de n processeur* ***:***

$$ARC = \{p1, \ldots, p_n\}$$

- *Un algorithme* ***Alg*** *composé de m composant logiciel (tâche) :*

$$Alg = \{t_1, \ldots, t_m\}$$

- *Des caractéristiques d'exécution Exe des composants logiciels* ***Alg*** *sur les composants matériels* ***ARC.***
- *Un ensemble des contraintes matérielles Dis*
- *Les taux de défaillance* Λ *des processeurs*

$$\Lambda = \{\lambda_1, \ldots, \lambda_n\}$$

- *Les temps d'exécution d'une instruction unitaire T des processeurs*

$$T = \{\tau_1, \ldots, \tau_n\}$$

- *Un coût statique des processeurs au produit de taux de défaillance et le temps d'exécution d'une instruction unitaire C*

$$C = \{C_1^{\lambda}, \ldots, C_n^{\lambda}\}$$

But

Trouver un ordonnancement statique et multiprocesseur des composants logiciels ***Alg*** *sur les composants matériels ARC qui respecte Dis et L, qui minimise la durée d'exécution de l'algorithme et maximise la fiabilité du système.*

Nous pouvons définir ce problème formellement comme suit :

$$\min_{\substack{t_i \in T \\ p_i \in P}} \begin{cases} L_{ord} \\ C_p \end{cases}$$

5.1.3 Premier critère : Minimisation de la longueur

Pour minimiser la longueur d'un ordonnancement L_{ord}, nous devons tout d'abord définir la formule qui calcule celle-ci, la formule suivante est la plus utilisée :

$$L_{ord} = \max_{p_j \in P}\{t_i^F\} \quad (5.1)$$

Où t_i^F dénote le temps de terminaison de la dernière tâche à exécuter, et il est calculé par la formule suivante :

$$t_i^F = t_{i,j}^s + exe(t_i, p_j) \quad (5.2)$$

Où $t_{i,j}^s$ représente le temps de début d'exécution de la tâche t_i sur processeur p_j et $exe(t_i, p_j)$ est le temps d'exécution de tâche t_i sur processeur p_j .

La tâche t_i ne peut pas être exécutée sur le processeur p_j sauf si toutes ses entrées sont disponibles et le processeur est prêt pour exécuter t_i.

Donc $t_{i,j}^s$ est calculé par la formule suivante :

$$t_i^s = \max\{t_{i,j}, t_{i,j}^d\} \quad (5.3)$$

Où :

$t_{i,j}$: représente le temps où le processeur p_j est prêt pour exécuter la tâche t_i

$t_{i,j}^d$: représente le temps où toutes les entrées pour exécuter t_i sont disponibles.

5.1.4 Deuxième critère : Maximisation de fiabilité

Modèle de fiabilité : la fiabilité est définie comme la probabilité qu'un système fonctionne correctement pendant le temps où la tâche en exécution, donc la fiabilité est calculée par la formule suivante :

$$F_{ord} = e^{-\sum exe(t_i, p_j)\lambda_j} \quad (5.4)$$

5.2 l'heuristique d'ordonnancement et d'optimisation bi-critère

L'algorithme AAA (Adéquation Architecture Algorithme) de SYNDEX est un algorithme statique et multiprocesseur, il est développé pour allouer les composants de l'algorithme ***Alg*** sur les composants d'architecture ***ARC*** tout en respectant les caractéristiques matérielles et les

contraintes temporelles. Pour minimiser la longueur d'exécution de système, AAA calcule à chaque étape la pression d'ordonnancement pour toutes les tâches candidates sur tous les processeurs ensuite pour chaque tâche, il choisit le processeur qui minimise la pression et parmi tous les couples, il choisit le couple qui maximise la pression. (L'algorithme est présenté dans le chapitre2)

L'algorithme AAA ne prend pas en compte la fiabilité du système comme critère à optimiser. Donc notre, solution permet d'introduire la fiabilité on se basant sur l'agrégation de fonction de pression de SYNDEX avec un nouveau terme pour le deuxième critère de fiabilité.

Ce terme appelé ***coût statique*** $C^{\lambda}_{p_j}$ qui représente le produit de taux de défaillance et le temps d'exécution d'une instruction unitaire.

5.2.1 Importance de coût statique

D'après la proposition **1** la fiabilité optimale est obtenue lorsque nous allouons les tâches au processeur qui possède un coût statique (le produit de taux de défaillance et le temps d'exécution d'une instruction unitaire) minimal.

Proposition1 *Si on a un ordonnancement Ord_1, toutes ses tâches sont allouées dans un ordre topologique au processeur qui possède un coût statique minimal, et si on a F_{ord1} est la fiabilité obtenue par cet ordonnancement, alors aucun autre ordonnancement $ord_2 = ord_1$ avec une fiabilité F_{ord2} telle que : $F_{ord1} < F_{ord2}$*

Preuve

Si nous supposons que, i = 0 (c.à.d. $\forall j: \lambda_0\tau_0 < \lambda_j\tau_j$) alors $F_{ord_1} = e^{-f(t,p_o)\lambda_o}$

Où, $f(t,p_0)$ désigne la date de terminaison de la dernière tâche à exécuter sur p_o avec l'ordonnancement Ord_1.

Si nous avons $f'(t,p_j)$ la date de terminaison de la dernière tâche à exécuter sur p_j avec l'ordonnancement Ord_2, alors $F_{ord_{j_1}} = e^{-f'(t,p_j)\lambda_j}$

Si nous avons T l'ensemble des tâches qui ne sont pas exécutées sur le processeur p_o avec Ord_2 alors, $f'(t,p_o) \geq f(t,p_0) - \tau_0 \sum_{t_i \in T} o_i$ (reste les tâches de T qui sont exécutées sur processeur p_o)

Si nous avons $T = T_1 \cup T_2 \cup \ldots \cup T_m$ ou T_j est un sous ensemble des tâches exécutées sur processeur j par l'ordonnancement Ord_2, ces sous-ensembles sont disjoints alors $\forall 1 \leq j \leq m, f'(t,p_j) \geq \tau_j \sum_{t_i \in T} o_i$

Donc, si on compare les deux fiabilités on obtient :

$$\sum_{j=0}^{m} f'(t,p_j)\,\lambda_j \; - f(t,p_0)\lambda_0 \tag{5.5}$$

$$\geq f(t,p_0)\lambda_0 - \lambda_0\tau_0 \sum_{t_i \in T} o_i \; + \sum_{j=1}^{m}\left(\lambda_j\tau_j \sum_{t_i \in T} o_i\right) - f(t,p_0)\lambda_0 \tag{5.6}$$

$$= \sum_{j=1}^{m}\left(\lambda_j\tau_j \sum_{t_i \in T} o_i\right) - \lambda_0\tau_0 \sum_{t_i \in T} o_i \tag{5.7}$$

$$= \sum_{j=1}^{m}\left(\lambda_j\tau_j \sum_{t_i \in T} o_i\right) - \lambda_0\tau_0 \sum_{j=1}^{m}\left(\sum_{t_i \in T} o_i\right) \tag{5.8}$$

On a T_j disjoint alors

$$= \sum_{j=1}^{m}\left((\lambda_j\tau_j - \lambda_0\tau_0) \sum_{t_i \in T} o_i\right) \tag{5.9}$$

$$\geq 0$$

Parce que $\forall\, j : \lambda_0\tau_0 < \lambda_j\tau_j$

5.2.2 Fonction de Syndex

L'heuristique de distribution et d'ordonnancement AAA de SYNDEX est un algorithme glouton de type d'ordonnancement de liste, basée sur une fonction de coût appelée la pression d'ordonnancement, dont l'objectif est de minimiser la longueur de la distribution/ordonnancement.

La pression d'ordonnancement est une fonction de coût qui mesure à la fois la marge d'ordonnancement et l'allongement de la longueur de distribution et d'ordonnancement (voir chapitre 2)

D'après [16] la pression d'ordonnancement est calculée par la fonction suivante :

$$FC_{AAA}(t_i, p_j) = P(t_i, p_j) - F(t_i, p_j)$$

Où :

$P(t_i, p_j)$: La pénalité d'ordonnancement, c'est l'allongement de la longueur maximal quand on décide de placer t_i sur p_j. Elle est calculée par la formule suivant

$$P_{t_i,p_j} = R_{t_i,p_j} - R \tag{5.10}$$

Où R est la langueur calculée.

$F(t_i, p_j)$: La flexibilité d'ordonnancement, c'est-à-dire la différence entre la date de début d'exécution au plus tard de t_i sur p_j, calculée depuis le début de l'ordonnancement, et la date de début d'exécution au plus tôt de t_i sur p_j, calculée aussi depuis le début de l'ordonnancement.

$$F_{t_i,p_j} = st_{t_i,p_j} - St_{t_i,p_j} \tag{5.11}$$

Donc :

$$FC^N_{AAA} = St^{(n)}_{t_i,p_j} + \overline{st^n}(t_i) - L^{(n-1)} \tag{5.12}$$

Où :

$St^{(n)}_{t_i,p_j}$: La date de début au-plus-tôt de t_i sur p_j ,

$\overline{st^n}(t_i)$: La date de début au-plus-tard de t_i depuis la fin,

$L^{(n-1)}$: La longueur d'ordonnancement à l'étape n-1.

Donc, pour chaque tâche candidate t_i, l'algorithme calcule sa pression d'ordonnancement sur chaque processeur P_j de Arc et conserve le couple (t_i; p_j) pour lequel cette pression d'ordonnancement est *la plus petite* : chaque couple indique quel est le meilleur processeur pour exécuter t_i. Puis, parmi tous les couples conservés, l'algorithme choisit celui dont la pression d'ordonnancement est *la plus grande* : ce couple indique quelle est la tâche la plus pénalisante, c'est-à-dire celle qui accroît le plus la longueur de l'ordonnancement. Le fait de choisir la tâche la plus pénalisante permet, au final, de minimiser la longueur de l'ordonnancement. C'est dans la pression d'ordonnancement, utilisée comme fonction de coût, que réside l'aspect heuristique de l'algorithme AAA.

5.2.3 Fonction est compromise entre fonction de SYNDEX et le coût statique

La solution que nous proposons pour résoudre le problème défini précédemment appartient aux méthodes d'agrégations de deux critères dans un seul, donc de chercher une fonction de coût qui permet à chaque étape de choisir la meilleure tâche qui satisfait la fonction de SYNDEX FC_{AAA} et le coût statique C_p^{λ} .

Puisque les deux critères ne sont pas de même grandeur, nous normalisons les deux critères par rapport au maximum, donc nous obtenons les deux grandeurs suivantes :

:

$$FC_{AAA}^{N} = \frac{FC_{AAA}(t_i, p_j)}{\max\limits_{p_i \in Arc} FC_{AAA}(t_i, p_i)} \tag{5.13}$$

$$C_p^{N} = \frac{C_{p_j}^{\lambda}}{\max\limits_{p_i \in Arc} C_{p_i}^{\lambda}} \tag{5.14}$$

Donc notre fonction est définie comme suit :

$$f(t_i, p_j) = \theta\, FC_{AAA}^{N}(t_i, p_j) + (1-\theta) C_p^{N} \tag{5.15}$$

θ et $(1-\theta)$ représentent les coefficients de pondération

5.3 Présentation de l'algorithme de l'heuristique

Algorithme d'Ordonnancement Bi –critères AOB	
Entrées et sorties	Entrées : ARC, Alg, Exe, Dis et Rtc Sortie : ordonnancement statique et multiprocesseur de Alg sur ARC qui respecte Dis et Rtc et qui optimise la durée d'exécution et la fiabilité.
Initialisation	T_{con} = {l'ensemble des tâches d'entrées} T_{ord} = Ø Calculer pour chaque processeur le coût statique $C_{p_i}^{\lambda}$ $C_{p_i}^{\lambda} = \lambda_i * \tau_i$
sélection	Tant que $T_{cond} \neq$ Ø alors • Pour chaque tâche, calculer la pression d'ordonnancement sur tous les processeurs $FC_{AAA}(o_i, p_j) = P(o_i, p_j) - F(o_i, p_j)$ • Pour chaque tâche, le couple calcule la fonction suivante $f(t_i, p_j) = \theta\, FC_{AAA}^{N}(t_i, p_j) + (1-\theta) C_p^N$ • Pour chaque tâche, choisir le processeur qui minimise $f(t_i, p_j)$ $f(t, p_{mail}) = \min_{p_j \in Arc} f(t, p_j)$ • Pour chaque couple (t_i, p_j) , choisir le couple qui maximise $f(t_i, p_j)$ $f(t_{mail}, p_{mail}) = \max_{t_i p_j} f(t_i, p_j)$
Allocation spatiale te temporelle	• Placer la tâche t_{mail} sur processeur p_{mail} allocation spatiale • Ordonnancer la tâche t_{mail} sur processeur p_{mail} allocation temporelle
Mise à jour	T_{con} = T_{con} - {t_{mail}} + {tous ses successeurs qui sont ses prédécesseurs appartiennent a T_{ord}} T_{ord} = T_{ord} + {t_{mail}} **Fin Tant que**

Nous pouvons deviser notre algorithme en quatre étapes :

- **Etape d'initialisation** : Dans cette étape, nous avons initialisé la liste des tâches candidates et la liste des tâches déjà ordonnancées, la première liste initialement contient l'ensemble des tâches sans prédécesseurs qui sont les tâches d'entrées (capteurs), la deuxième liste initialement est vide. Ensuite nous avons calculé le coût statique pour chaque processeur.

- **Etape de sélection** : Dans cette étape, nous avons calculé la pression d'ordonnancement d'AAA et la fonction compromise pour chaque tâche sur tous les processeurs. Ensuite, pour chaque tâche nous choisissons le processeur qui minimise notre fonction de coût et le couple (tâche, processeur) qui maximise la fonction objectif

- **Etape d'allocation spatiale et temporelle** : Selon le couple sélectionné nous ordonnançons la tâche sur le meilleur processeur.

- **Etape de mise à jour :** Dans cette étape, nous supprimons la tâche sélectionnée de la liste des tâches candidates et nous la mettons dans la liste des tâches ordonnancées, et nous mettons tous ses successeurs dans la liste des tâches candidates.

- L'algorithme est terminé lorsque la liste des tâches candidates se vide ou les contraintes temporelles ne sont pas vérifiées.

5.4 BSA (Bi-objective Scheduling Algorithm)

BSA appartient aussi aux méthodes d'agrégations de deux critères dans un seule, il utilise une fonction de calcul de la fiabilité différents de celle définie dans les algorithmes précédents, il calcul la probabilité de défaillance de tâche sélectionné sur le processeur et il ne prend pas en compte les tâche déjà ordonnancer. À chaque étape, BSA attribue la priorité supérieure à la tâche libre qui appartient au chemin critique ou le chemin le plus long et il est noté tâche critique, ensuite il sélectionne la tâche critique et ordonnance sur le processeur qui minimise la fonction de coût suivante:

$$f(t_i, p_j) = \sqrt{\theta \left(\frac{f(t_i, p_j)}{\max\limits_{p \in P} f(t_i, p)} \right)^2 + (1-\theta) \left(\frac{\varepsilon(t_i, p_j)}{\max\limits_{p \in P} \varepsilon(t_i, p)} \right)^2} \qquad (5.16)$$

Où :

$f(t_i, p_j)$: Temps de fin d'exécution de t_i sur p_j

$\max\limits_{p \in P} f(t_i, p)$: Temps maximum de fin d'exécution de t_i sur tout les processeurs.

$\varepsilon(t_i, p_j)$: La probabilité de défaillance de composant logiciel t_i sur composant matériel p_j

5.5 Génération aléatoire de graphe d'architecture et d'algorithme

Afin de mener une étude comparative des heuristiques de distribution/ordonnancement que nous avons proposées dans ce travail, il à été nécessaire de disposer d'un ensemble de graphes d'algorithmes et de graphes d'architectures. Pour cela, nous présentons dans cette partie, deux générateurs aléatoires de graphes qui nous ont permis d'obtenir ces graphes. Le premier générateur génère des graphes d'algorithmes, et le deuxième génère des graphes d'architectures.

a. Génération aléatoire de graphe d'algorithme

Nous avons choisi d'utiliser la méthode proposé dans [21] pour la génération aléatoire des algorithmes, Le principe de la méthode est de :

1. Tirer aléatoirement *v* nœuds de type capteur, calcul et actionneur dans une matrice de dimension *height * width*.
2. Construit les *n* niveaux du graphe, où chaque niveau *i* est composé des nœuds situés sur une même ligne horizontale dans la matrice.
3. Choisir pour chaque nœuds *x* de niveau *i* un nombre *input* aléatoire de prédécesseurs parmi les nœuds de niveau $j < i$, avec au moins un prédécesseur *y* de niveau *i-1*. Nous avons choisi de générer deux types d'arcs : avec diffusion ou sans diffusion.
4. Choisir aléatoirement pour chaque nœud *x* de niveau *i* sans successeurs un nœud *y* comme successeur de niveau $j > i$.
5. Enfin, ajouter aléatoirement au graphe un nombre aléatoire *k* de nœuds de type mémoire et constante.

b. Génération aléatoire de graphe d'architecture

Pour générer aléatoirement de graphes d'architectures nous avons choisi l'algorithme de Waxman [29], le principe de la méthode est de :

1. Tirer aléatoirement *v* nœuds de type processeurs dans une matrice
2. Calculer la probabilité *P(x,y)* d'ajouter un lien entre chaque paire de processeurs *x* et *y* :

$$P(x,y) = \beta\, e^{\frac{-d(x,y)}{\alpha * L}}$$

Où *L* est la distance maximale entre tous les processeurs,
$\beta=2.2$ et $\alpha=0.15$ sont des constantes, et
d(x, y) est la distance euclidienne entre *x* et *y*.

3. Pour chaque *P(x, y)*, tirer un nombre *T* aléatoire entre 0 et 1. Si $T < P$, alors ajouter un nœud *m* de type media point-a-point et relier *m* avec *x* e

5.6 Etude Comparative

Dans cette section, nous comparons les heuristiques présentées dans BSA et AOB sur les différents types d'algorithmes ***ALg***, pour voir dans quel cas chaque heuristique est meilleure par rapport aux autres,

- **Algorithme séquentiel**

On dit qu'un algorithme est séquentiel si l'exécution de ses tâches est séquentiel un après l'autre. Nous représentons ce modèle d'algorithme par le schéma suivant :

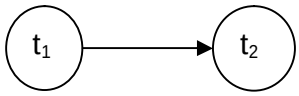

FIG 5. 1 Exemple de représentation séquentielle

Considérant par exemple le système constitué :

- d'une architecture ***ARC*** composée de deux processeurs p_1 et p_2, avec les taux de défaillance respectifs $\lambda_1 = 7*10^{-6}$ et $\lambda_2 = 2*10^{-6}$
- d'un algorithme ***Alg*** (FIG **5.1**) composé de deux tâches en série t_1 et t_2 et leurs temps d'exécution sont :

	p_1	**p_2**
t_1	1	2
t_2	1	2

TAB 5. 1 Temps d'exécution des tâches de FIG 5.1

Les résultats obtenus par chaque algorithme sont donnés dans le tableau suivant :

	AOB	**BSA**
Allocation spatiale	t_1 et t_2 sur p_2	t_1 sur p_1 t_2 sur p_1
La longueur	4	2
La fiabilité	$e^{-8*10^{-6}}$	$e^{-14*10^{-6}}$

TAB 5. 2 Résultats obtenus par les algorithmes sur FIG 5.1

- **Algorithme parallèle**

Nous avons deux cas d'un algorithme parallèle

Premier cas :

Nous ne pouvons pas exécuter les tâches en parallèle avant d'exécuter la tâche source. Nous pouvons représenter cet algorithme par le schéma suivant :

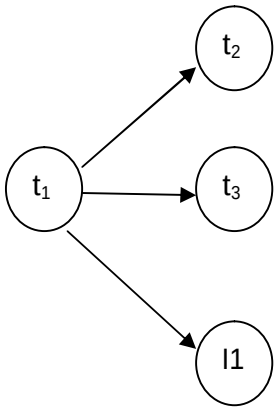

FIG 5. 2 Exemple 1 de représentation parallèle

Nous prenons le même modèle d'architecture précédent mais l'algorithme est constitué d'une tâche source t_1 et trois tâches en parallèle t_2 ,t_3 et t_4 avec leurs temps d'exécution sont :

	p_1	**p_2**
t_1	1	2
t_2	2	4
t_3	1	2
t_4	2	4

TAB 5. 3 Temps d'exécution des tâches de FIG 5.2

Les résultats obtenus par chaque algorithme sont donnés dans le tableau suivant :

	AOB	**BSA**
Allocation spatiale	t_1 et t_2 sur p_2 t_3 sur p_2 t_4 sur p_1	t_1 et t_2 sur p_2 t_3 sur p_2 t_4 sur p_1
La longueur	7	5
La fiabilité	$e^{-28*10^{-6}}$	$e^{-32*10^{-6}}$

TAB 5. 4 Résultats obtenus sur FIG 5.2

Deuxième cas :

On doit exécuter les tâches en parallèle avant d'exécuter la tâche destination, nous pouvons représenter ce modèle par le schéma suivant :

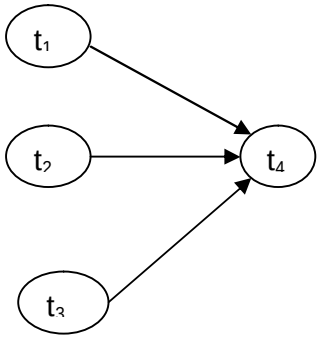

FIG 5. 3 Exemple 2 de représentation parallèle

Nous prenons le même modèle d'architecture précédent mais l'algorithme est constitué de trois tâches en parallèle t_1, t_2 et t_3 et une tâche destination t_4 ,et avec leurs temps d'exécution sont :

	p_1	**p_2**
t_1	3	6
t_2	1	2
t_3	1	2
t_4	1	2

TAB 5. 5 Temps d'exécution des tâches de FIG 5.3

Par l'application des deux heuristiques AOB et BSA sur l'algorithme de la figure FIG 5.3, nous obtenons les résultats suivants :

	AOB	**BSA**
Allocation spatiale	t_1 et t_4 sur p_2 t_2 et t_3 sur p_1	t_1 et t_4 sur p_2 t_2 et t_3 sur p_1
La longueur	10	8
La fiabilité	$e^{-34*10^{-6}}$	$e^{-37*10^{-6}}$

TAB 5. 6 Résultats obtenus sur FIG 5.3

Cas général

La figure 5.4 présente un algorithme dans le cas général, où il contient des tâches exécutées séquentiellement et des tâches exécutées en parallèle.

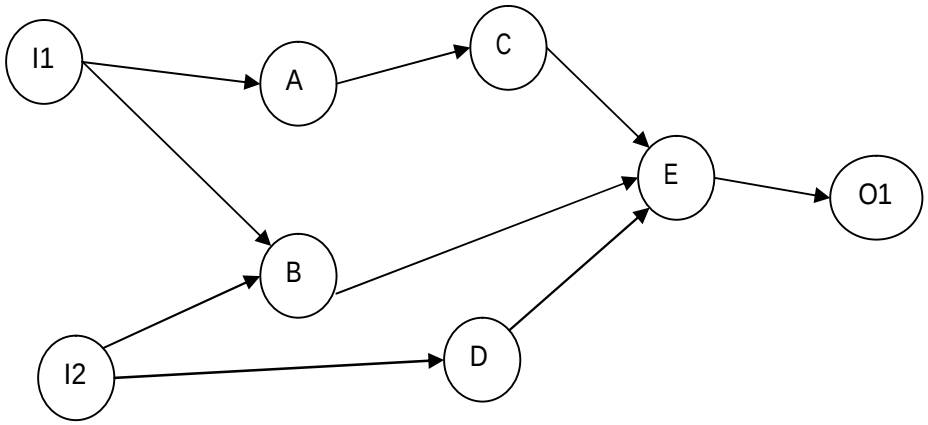

FIG 5. 4 Exemple général d'un graphe d'algorithme

	P1	**P2**
Temps d'exécution d'une unité	1	2
Taux de défaillance	$7*10^{-6}$	$2*10^{-6}$

TAB 5.7 Caractéristique matérielle de FIG 5.4

	P1	**P2**
I1	1	2
I2	1	2
A	3	6
B	5	10
C	4	8
D	3.	6
E	1	2
O1	1	2

TAB 5. 8 Temps d'exécution des tâches de FIG 5.4

Les résultats obtenus sont présentés dans le tableau suivant

	AOB	BSA
Allocation statique	I1 et A sur P2 I2 et B sur P1 C sur P2 D sur P1 E et O sur P2	I1 et A sur P1 I2 et B sur P2 C et D sur P1 E et O sur P2
La longueur	20	16
La fiabilité	$e^{-69*10^{-6}}$	$e^{-72*10^{-6}}$

TAB 5. 9 Résultats obtenus sur la figure 5.4

5.7 Simulation

Dans le but de valider notre heuristique, nous avons comparé ses résultats avec ceux de BSA, BSA est basé sur une fonction de coût qui compromet entre la longueur de l'ordonnancement et la fiabilité de système. Elle attribue à chaque tâche libre une priorité appelée tâche critique, et après, elle choisit pour chaque tâche critique le processeur qui minimise sa fonction de compromise (voir chapitre 4). L'objectif principal de cette simulation est de comparer la longueur moyenne et la fiabilité moyenne générée par BSA et AOB.

Les figures FIG 5.5 et FIG 5.6 comparent respectivement la longueur et la fiabilité moyenne des ordonnancements générés par les heuristiques d'AOB et de BSA. Chaque point est la moyenne de 50 graphes ***Alg*** générés aléatoirement avec CCR= 1 (le rapport entre le temps moyen d'exécution et le temps moyen de communication) et N variant de 20 à 100 tâches sont ordonnancées sur un graphe ***ARC*** complet à 6 processeurs avec liens point-à-point. Les taux de défaillance des processeurs ont été tirés aléatoirement entre $2*10^{-6}$ et$7*10^{-6}$ (θ =1/2), Les ordonnancements obtenus avec AOB sont systématiquement meilleurs en fiabilité mais moins bon en longueur.

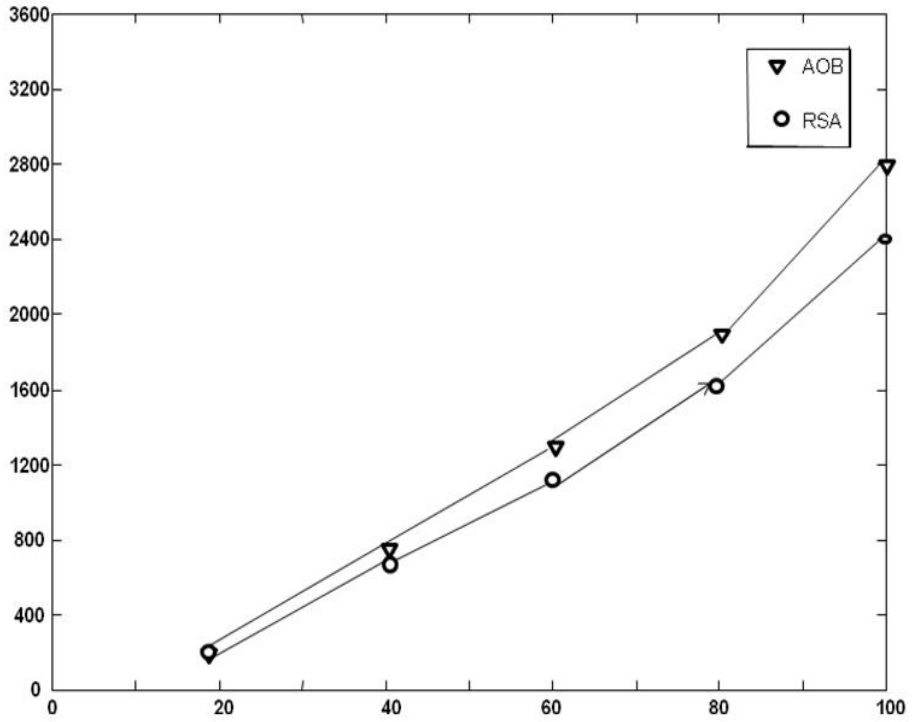

FIG 5. 5 -Comparaison entre AOB et BSA en longueur

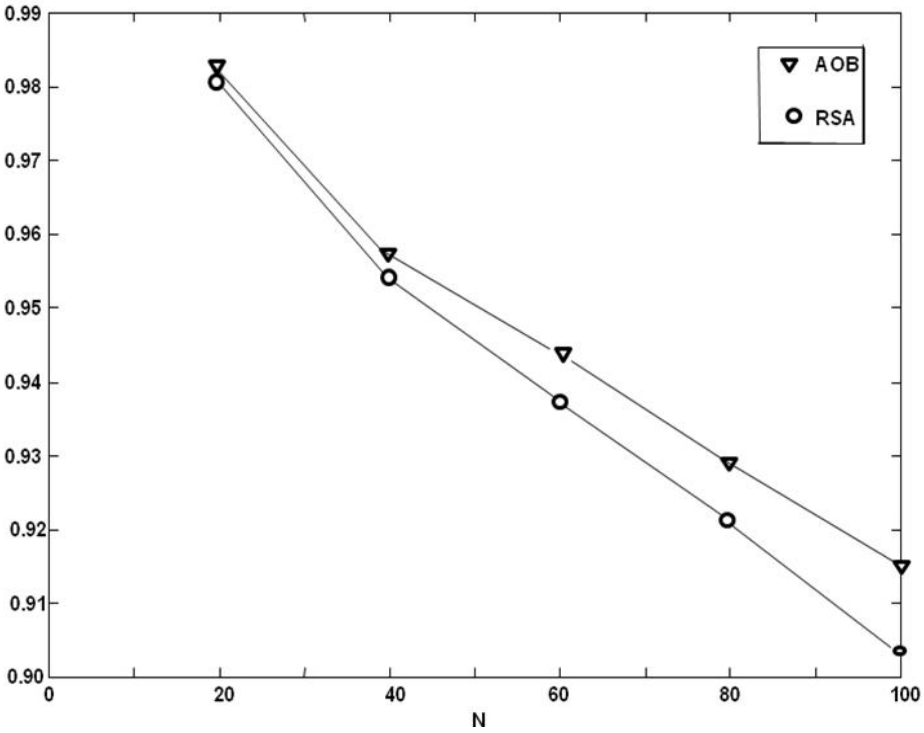

FIG 5. 6 -Comparaison entre AOB et BSA en fiabilite

Dans la figure FIG 5.5 montre que l'algorithme BSA donne de bons résultats pour la longueur mieux qu'AOB. Nous résolvons ce problème par le teste de contrainte temporelle,

par contre, la deuxième figure FIG 5.6 montre que notre algorithme donne de bons résultats dans le cas de fiabilité par rapport BSA.

Les figures FIG 5.7 et FIG 5.8 comparent respectivement la longueur moyenne des ordonnancements générés par les heuristiques d'AOB et de BSA. Nous prenons N=80 tâches, et ordonnancées sur un graphe ***ARC*** complet à 4 processeurs avec lient point à point et CCR= 0.1 1 et 10 ; Les taux de défaillance des processeurs ont été tirés aléatoirement entre $2*10^{-6}$ et$7*10^{-6}$ (θ =1/2) et la valeur de$\theta = 1$. Nous ne prenons pas en compte le critère de fiabilité.

Le ratio entre les coûts de communication et les coûts d'exécution < CCR > : le choix de la valeur de ce paramètre a des conséquences importantes sur l'efficacité d'une heuristique de distribution et d'ordonnancement. Un CCR supérieur à 1 indique que les communications coûtent < plus cher > que les calculs.

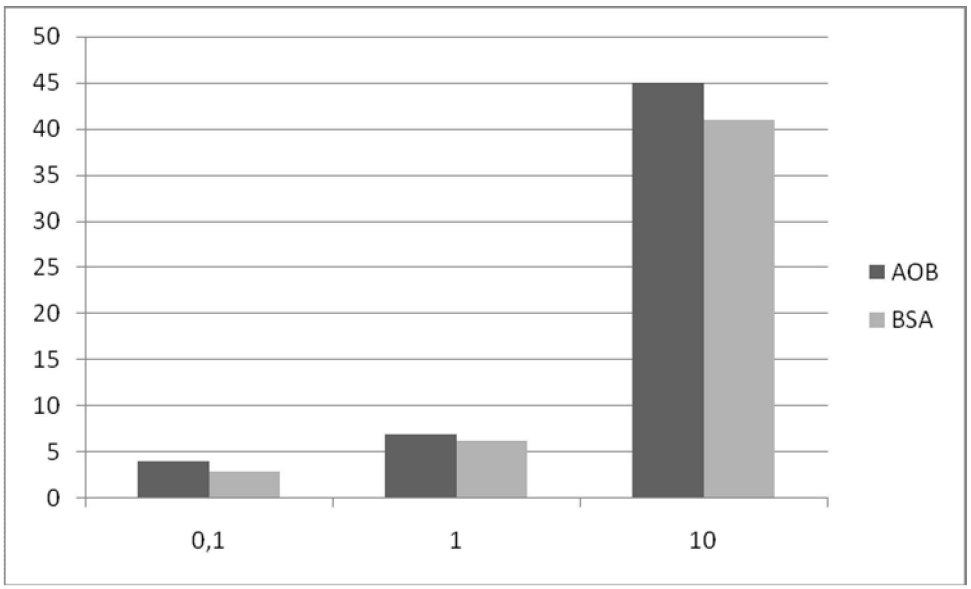

FIG 5. 7 -Comparaison entre AOB et BSA en longueur où $\theta = 1$

Dans la figure FIG 5.7, montre que lorsque CCR augmente, le temps de communication augmente donc la longueur d'ordonnancement des deux algorithmes augmente, et il montre aussi, que pour n'importe quelle valeur de CCR la longueur d'ordonnancement produit par les deux algorithmes est presque similaire.

Nous prenons le même exemple de la figure précédente mais dans ce cas la valeur de $\theta = 0.5$, nous obtenons les résultats représentés dans les deux figures suivantes :

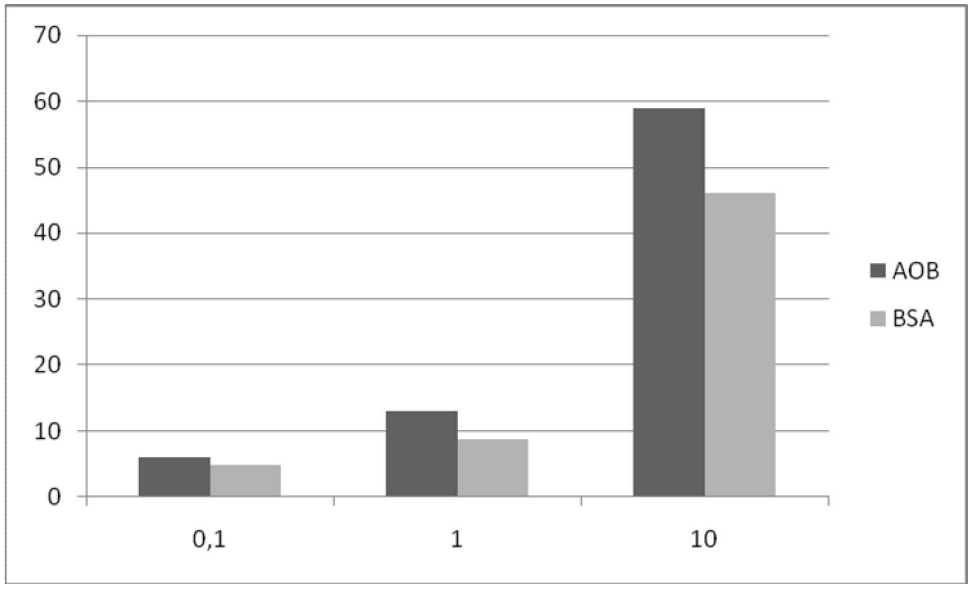

FIG 5. 8-Comparaison entre AOB et BSA en langueur où $\theta = 0.5$

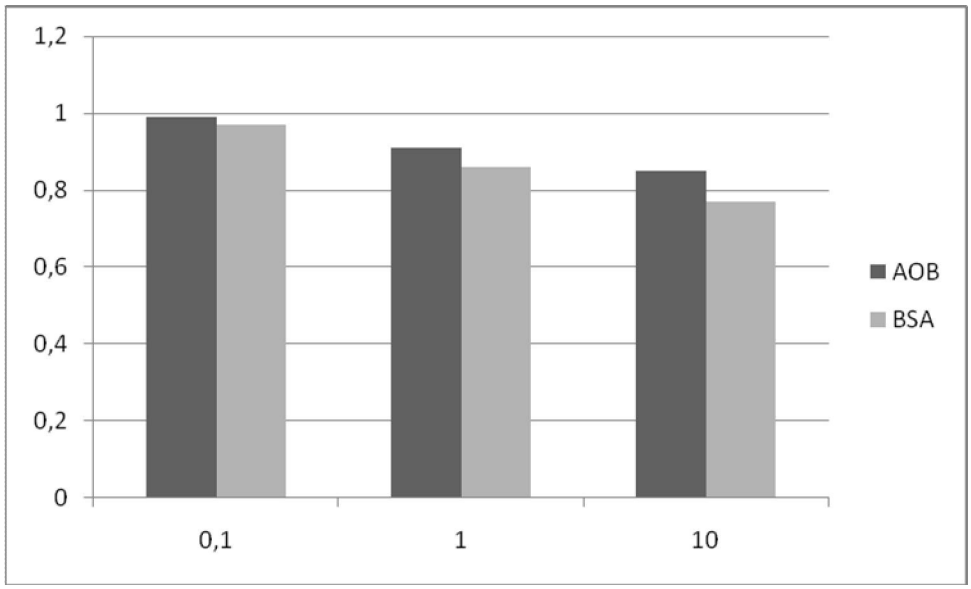

FIG 5. 9-Comparaison entre AOB et BSA en fiabilite où $\theta = 0.5$

Dans les figures FIG 5.8 et FIG 5.9, pour CCR=0.1, les deux algorithmes ayant presque les mêmes performances mais lorsque CCR =1 et 10, AOB donne des résultats meilleurs dans le cas de fiabilité par rapport BSA est moins performant dans le cas de longueur.

5.8 Conclusion

Nous avons présenté dans ce chapitre une heuristique pour concevoir un système fiable, basé sur la théorie d'ordonnancement, notre algorithme génère un ordonnancement statique et multiprocesseur des composants logiciels sur les composants matériels tout en respectant les contraintes temporelles, elle utilise une fonction de compromise de fonction de coût de Syndex et un terme statique. Nous avons démontré l'importance de coût statique pour générer une allocation fiable.

.

Conclusion et perspective

Dans le but de concevoir un système fiable, nous avons proposé dans ce mémoire un algorithme d'ordonnancement et de distribution bi-critère des composants logiciels sur les composants matériels qui optimise la duré d'exécution et la fiabilité du système, tout en respectant les contraintes temporelles et les caractéristiques matérielles.

Le problème de distribution et d'ordonnancement temps réel et fiable est un problème NP difficile, puisqu'il s'agit de trouver une solution qui minimise le temps globale d'exécution et maximise la fiabilité du système. La solution optimale à ce problème NP-difficile ne peut être trouvée que par des algorithmes exacts de complexité exponentielle ; c'est pourquoi nous avons proposé d'utiliser dans ce travail des heuristiques qui cherchent une solution si possible proche de la solution optimale, tout en étant de complexité polynomiale.

Notre solution appartient aux méthodes d'optimisation d'agréger deux critères dans un seul et qui améliore la fiabilité par l'allocation des composants logiciels au processeur qui possède un coût statique minimal tout en respectant la contrainte temporelle.

Nous avons prouvé que la fiabilité optimale obtenue lorsque nous allouons les composants logiciels sur le processeur ayant un coût statique minimal, mais nous devons toujours minimiser la durée d'exécution du système. C'est pour ça, nous avons proposé une fonction de compromise entre le coût statique et la fonction de Syndex pour garantir les deux objectifs.

La fonction de compromise utilise le coefficient de pondération fournis par l'utilisateur, qui donne plus de poids soit à l'objectif de fiabilité s'il est plus grand que 0,5, soit à l'objectif de durée d'exécution s'il est moins.

Les distributions/ordonnancements temps réel et fiables générées par notre algorithme sont prédictives, c'est à dire que les contraintes de temps réel et de fiabilité peuvent être vérifiées avant la mise en exploitation du système, à la fois en l'absence et en présence de défaillances des processeurs.

Notre algorithme est une extension d'AAA implémentée sous l'outil de Syndex.

Syndex génère automatiquement un code exécutable distribué, au premier temps, il produit un ordonnancement distribué et statique des composants logiciels sur les composants

matériels, ensuite, il génère un exécutif temps réel embarqué, qui implémente notre heuristique.

Les principaux avantages de notre algorithme sont :

1. Complexité de temps est minimale
2. Atteindre les deux objectifs simultanément en même degré.
3. Facile à implémenter dans la pratique.

Le mauvais choix de coefficient de pondération peut donner des résultats qui ne sont pas forcément optimum, donc, parmi les perspectives c'est d'essayer de fixer les coefficients de pondération ou de mettre des contraintes pour avoir de meilleurs choix.

Bibliographies

[1] A.Abd-allah. Extending reliability block diagrams to software architectures. Research report, Center for Software Engineering, Computer Science Department, University of Southern California, Los Angeles (CA), USA, 1997.

[2] I Assayad, A. Girault, and H. Kalla. A bi-criteria scheduling heuristics for distributed embedded systems under reliability and real-time constraints. In *International Conference on Dependable Systems and Networks, DSN'04*, pages 347..356, Firenze, Italy, June 2004. IEEE.

[3] Alian Berro. Optimisation multi-objectif et stratégie d'évolution en environnement dynamique. Phd Thesis. Université de Toulouse 2001.

[4] A.Bobbio. C.Ferraris. R.terruggia. New challenges in network reliability analysis. Département informatique. University de Piemonte Orientale. Italy 2006.

[5] A.Bobbio. R.terruggia. E.ciancamerla. M.Minichino. Evaluating Network reliability versus topology by BDD algorithms. PSMA HongKong-China. May 2008.

[6] A.O.Charle Elgebed.C.Chu. Reliabiliyu allocation through Cost minimisation. March 2003. IEEE.

[7] Y.G. Chen and M.C. Yuang. A cut-based method for terminal-pair reliability. *IEEE Trans. On Reliability*, 45 :413.416, September 1996.

[8] Y.Collette. P.Siarry. Optimisation multiobjectif . Louis-jeans. Septembre 2002.

[9] David Decotigny. Bibliographie d'introduction a l'ordonnancement dans les systèmes informatiques temps réel. Novembre 2002

[10] A. Dogan and F. Özgüner. Bi-objective scheduling algorithms for execution time-reliability tradeoff in heterogeneous computing systems. *The Computer Journal*, 48(3):300.314, 2005.

[11] A.Dogan and F. Özgüner. Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing. *IEEE Trans. on Parallel and Distributed Systems,* 13(3) :308.323, March 2002.

[12] J . Dongarra and E . Jean. Bi-objectif scheduling algorithms for optimizing makespan and reliability on heteregouns systems SPAA'07. June 9-11 2007 .sandiago USA

[13] J. K. Muppala, G. Ciardo, and K. Trivedi. Stochastic reward nets for reliability prediction

[14] R.A. Sahner and K.S. Trivedi. A hierarchial, combinatorial-markov model of solving complex reliability models. In *Proceedings of 1986 ACM Fall joint computer conference,* pages 817–825, 1986

[15] R.E. Bryant.Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers,* C-35:677--691, 1986.

[16] Alain Girault. *Contributions à la conception sûre des systèmes embarqués sûrs*. Thèse d'Habilitation à dirige a la recherche. INPG, Grenoble, France, Septembre 2006.

[17] N.K. Goyal, Network reliability evaluation: a new modeling approach, *Int Conference on Reliability and Safety Engineering (INCRESE2005),* 473-488, 2005.

[18] Mourad Hakem and Frank Butelle. A Bi-objectif Algorithm for scheduling Parallel application on heterogeneous system subject to failures. In renpar , Canet en Roussillon, October 2006.

[19] G. Hardy and C. Lucet and N. Limnios, Computing all-terminal reliability of stochastic networks by Binary Decision Diagrams, *Proceedings Applied Stochastic Modeling and Data Analysis, ASMDA2005*, 2005

[20] Y. He, Z. Shao, B. Xiao, Q. Zhuge, and E. Sha. Reliability driven task scheduling for heterogeneous systems. In *IASTED International Conference on Parallel and Distributed Computing and Systems*, Marina Del Ray, USA, November 2003.

[21] H. Kalla. *Génération automatique de distributions/ordonnancements temps-réel, fiables et tolérants aux fautes*. Thèse de doctorat, INPG, Grenoble, France, December 2004.

[22] S. Kartik and C.S.R. Murthy. Task allocation algorithms for maximising reliability of distributed computing systems. *IEEE Trans. on Computers,* 46(6) :719.724, June 1997.

[23] J.-C. Laprie. Sûreté de fonctionnement informatique : concepts de base et terminologie. Rapport technique, LAAS-CNRS, Toulouse, France, 2004.

[24] J. Rushby. Critical system properties : Survey and taxonomy. *Reliability Engineering and System Safet*, 43(2) :189–219, 1994

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

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

[27] K. Sekine, H. Imai, A unified approach via BDD to the network reliability and path number, Department of Information Science, University of Tokyo, TR-95-09, 1995

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

[29] M. Thomas and E. W. Zegur. Generation and analysis of random graphs to model internetworks. Technical report, College of Computing Georgia Institute of Technology, 1994.

[30] K. Trivedi. Probability & Statistics with Reliability, Queueing & Computer Science applications, Wiley, II Edition, 2001.

[31] X. Zang, H. Sun, K. Trivedi, A BDD-based algorithm for reliability graph analysis, Department of Electrical Engineering, Duke University, 2000