HAL
open science

Tolérance aux pannes dans des environnements de calcul parallèle et distribué : optimisation des stratégies de sauvegarde/reprise et ordonnancement

Mohamed Slim Bouguerra

To cite this version:

Mohamed Slim Bouguerra. Tolérance aux pannes dans des environnements de calcul parallèle et distribué : optimisation des stratégies de sauvegarde/reprise et ordonnancement. Autre [cs.OH]. Université de Grenoble, 2012. Français. NNT : 2012GRENM023 . tel-00910358

HAL Id: tel-00910358

https://theses.hal.science/tel-00910358v1

Submitted on 27 Nov 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministérial : 7 août 2006

Présentée par

Mohamed Slim Bouguerra

Thèse dirigée par **Denis Trystram**
et codirigée par **Thierry Gautier**

préparée au sein **Laboratoire d'Informatique de Grenoble**
et de **l'Ecole Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Tolérance aux pannes dans des environnements de calcul parallèle et distribué : optimisation des stratégies de sauvegarde/reprise et ordonnancement.

Thèse soutenue publiquement le **2 Avril 2012**,
devant le jury composé de :

Alain Girault
Directeur de recherche INRIA Rhone-aples, Président
Emmanuel Jeannot
Directeur de recherche INRIA Bordeaux Sud-Ouest, Rapporteur
Yves Robert
Professeur ENS Lyon, Rapporteur
Mohamed Jemni
Professeur Ecole supérieure de sciences et techniques de Tunis, Examinateur
Pierre Sense
Professeur Paris 6, Examinateur
Mr Denis Trystram
Professeur Grenoble INP, Directeur de thèse
Mr Thierry Gautier
Chargé de recherche INRIA Rhône-Alpes Grenoble, Co-Directeur de thèse

Remerciements

Je tiens à remercier tout d'abord mes directeurs de thèse, Denis Trystram et Thierry Gautier, pour l'aide et le soutien qu'il m'ont apporté durant ma thèse. Leur grande disponibilité et leur idées toujours pertinentes m'ont permis d'avancer sereinement tout au long de ma thèse. Un grand merci à Denis Trystram qui a fait le pont entre la Tunisie et la France en me recrutant pour le stage de M2 et ensuite pour la thèse. Merci à toi Denis pour ces années magnifiques qu'on a passé ensemble.

Je tiens à remercier également les membres de mon jury, Alain Girault, Emmanuel Jeannot, Yves Robert, Mohamed Jemni et Pierre Sense. Je remercie particulièrement Emmanuel Jeannot et Yves Robert pour avoir bien voulu consacrer une part de leur temps à rapporter sur ces travaux.

Je remercie les membres des équipes MOAIS et MESCAL avec qui j'ai collaboré notamment Derrick kondo, Jean-marc Vincent et Frederic Wagner.

Un grand merci à tous mes collègues pour les moments passés ensemble au bâtiment ENSIMAG Montbonnot lors de pauses cafés inoubliables. Mais aussi en dehors, autour de la table LRP, lors de soirées, et surtout excursions ski. Je leur suis par ailleurs reconnaissant pour l'aide qu'ils ont pu m'apportée tout au long de ma thèse. Je pense tout d'abord à Jean-louis Roch mon moniteur de ski.

Puis à ceux qui ont soutenu leur Thèse en même temps que moi et à ceux dont le tour viendra bientôt et à qui je souhaite bon courage.

Un grand merci à ma famille, particulièrement à mes parents qu'ils m'ont supporté tout au long de mon parcours scolaire et professionnel.

Sommaire

Sommaire iii

1 Introduction 1
- 1.1 Contexte et motivations . . . 1
- 1.2 Positionnement et problématiques . . . 2
- 1.3 Guide de lecture et contributions . . . 3

2 Préliminaires 5
- 2.1 Introduction . . . 5
- 2.2 Caractéristiques et conséquences des pannes . . . 6
  - 2.2.1 Caractéristiques des pannes . . . 6
  - 2.2.2 Conséquences des pannes . . . 8
- 2.3 Modélisation formelle des pannes . . . 9
  - 2.3.1 Modélisation stochastique des pannes . . . 9
    - a Collecte et analyse des traces . . . 10
    - b Inférence statistique . . . 10
  - 2.3.2 Modèle de panne typique . . . 10
  - 2.3.3 Distributions de panne typiques . . . 12
- 2.4 Approches de tolérance aux pannes . . . 13
  - 2.4.1 Prévention des pannes . . . 13
    - a Migration . . . 13
  - 2.4.2 Masquage des pannes . . . 13
    - a Duplication . . . 13
    - b Approches algorithmiques (ABFT) . . . 14
  - 2.4.3 Recouvrement après panne . . . 14
    - a Sauvegarde et reprise coordonnée . . . 16
    - b Sauvegarde et reprise non coordonnée . . . 17
- 2.5 Bilan et conclusions . . . 18

I Modélisation Stochastique du Mécanisme de Sauvegarde et Reprise 19

3 Modélisation continue 21
- 3.1 Introduction . . . 21
- 3.2 Modélisation sans tolérance aux pannes . . . 22
  - 3.2.1 Modèle de l'environnement d'exécution . . . 22
  - 3.2.2 Distribution du nombre de re-exécutions . . . 24
  - 3.2.3 Espérance du temps de complétion . . . 25
  - 3.2.4 Bilan . . . 27

3.3 Modélisation avec tolérance aux pannes . . . 28
3.3.1 Modèle du mécanisme de sauvegarde et reprise . . . 28
3.3.2 Formulation du problème d'optimisation . . . 31
a Espérance du temps de complétion avec sauvegarde et reprise 31
b Problème d'optimisation sous contraintes . . . 32
3.3.3 Résolution du problème d'optimisation . . . 32
a Solution analytique . . . 32
b Solution numérique . . . 34
3.4 État de l'art . . . 35
3.5 Validation et comparaison expérimentale . . . 41
3.5.1 Paramètres et scénarios des simulations . . . 41
3.5.2 Résultats des simulations . . . 41
3.5.3 Bilan des simulations . . . 43
3.6 Conclusion . . . 44

4 Modélisation discrète 47
4.1 Introduction . . . 47
4.2 Modèle de l'environnement d'exécution . . . 48
4.2.1 Modèle de la plateforme d'exécution . . . 48
4.2.2 Modèle d'application . . . 49
4.2.3 Modèle du mécanisme de sauvegarde . . . 50
4.3 Formulation du problème d'optimisation . . . 50
4.3.1 Introduction au processus de décision markovien . . . 50
4.3.2 Problème d'ordonnancement des points de sauvegarde . . . 51
4.3.3 Ordonnancement optimal des points de sauvegarde . . . 55
4.4 Analyse de complexité . . . 56
4.4.1 Analyse de complexité de l'algorithme proposé . . . 56
a Analyse de complexité dans le cas général . . . 56
b Cas surcoût des points de sauvegarde constant . . . 58
4.4.2 Analyse de complexité du problème . . . 58
a Définition du problème décisionnel relaxé . . . 59
b Définition de la famille d'instances . . . 59
c La NP-complétude du problème . . . 61
4.5 État de l'art . . . 62
4.6 Étude expérimentale . . . 64
4.6.1 Modèle de l'environnement d'exécution . . . 65
a Modèle d'application . . . 65
b Modèle de communication . . . 65
c Modèle de panne . . . 66
d La stratégie d'ordonnancement des points de sauvegarde . . . 66
4.6.2 Méthodologie d'expérimentation . . . 67
a Critères de performance et stratégies d'ordonnancement : . . . 67
b Modèle SimGrid . . . 68
c Les scénarios : . . . 69
4.6.3 Analyse des résultats . . . 71
4.7 Conclusions . . . 73

II Ordonnancement **77**

5 Ordonnancement tolérant aux pannes **79**
5.1 Introduction 79
5.2 Formulation du problème d'optimisation 80
5.3 État de l'art 82
5.3.1 Quelques rappels en théorie d'ordonnancement 82
5.3.2 Optimisation multi-objectifs 84
5.3.3 Résultats dans le cadre de la tolérance aux pannes 87
5.4 Optimisation du *makespan* et de la fiabilité de l'application 93
5.4.1 Optimisation de $Q||(C_{max}, \mathcal{R}_l)$ pour une loi *DFR* 93
a Tâches indépendantes et unitaires 95
b Tâches indépendantes et arbitraires 99
5.4.2 Optimisation de $P||(C_{max}, \overline{\mathcal{R}}_l)$ avec $\lambda(t)$ *IFR* 102
a Taches unitaires et indépendantes 102
b Tâches indépendantes et arbitraires 104
5.5 Conclusions 105

6 Conclusions et travaux futurs **107**
6.1 Bilan 107
6.2 Perspectives 108

Annexe **111**
6.3 Schur-convexe 112
6.4 Distribution de probabilité *IFR*/*DFR* 113

Bibliographie **117**

Table des figures **125**

Table des matières **127**

Introduction 1

Sommaire

1.1 Contexte et motivations 1
1.2 Positionnement et problématiques 2
1.3 Guide de lecture et contributions 3

1.1 Contexte et motivations

La croissance constante des besoins de puissance de calcul dans les divers domaines scientifiques, a déclenché une vraie révolution dans les environnements de calcul parallèle et distribué. En 1997 le super-calculateur le plus performant dans le classement TOP500 été composé de 100 processeurs [40]. Or aujourd'hui les applications de calcul de type HPC (*High Performance Computing*) sont exécutées en parallèle sur des plateformes de calcul qui contiennent une dizaine de milliers de processeurs. Par exemple 80% des applications parallèles exécutées sur la plateforme RedStrom à *Sandia National Laboratories*, utilisent en parallèle plus de 40% des 30,000 nœuds de calcul disponibles [79]. À terme, il est tout à fait envisageable de voir apparaître des applications de calcul scientifique composées d'un milliard de processus exécutés sur une plateforme composée d'un million de cœurs durant des jours et des jours [37].
Cette augmentation fulgurante du nombre de processeurs impliqués dans l'exécution pose de nombreux défis scientifiques en particulier sur la fiabilité et la résilience de ces nouvelles plateformes. En effet, ces plateformes ont atteint une taille si importante et un empilement de couches logicielles si compliqué que ce défi devient une problématique de premier plan. Toutes les analyses et études statistiques menées récemment dans les environnements de calcul actuels, indiquent que le temps moyen inter-panne MTTF [1] est inversement proportionnel au nombre de processeurs impliqués dans l'exécution [89, 23, 80, 75]. Ainsi, si actuellement le MTTF est de l'ordre de jour (par exemple le MTTF de la plateforme de calcul Mercury est de l'ordre de 20h [56]), à l'échelle PetaFLOP, le taux de panne sera de l'ordre d'une panne par composant chaque 30 minutes [51]. Or actuellement, l'exécution des applications de calcul parallèle existantes dure plusieurs jours. Ce défi ne concerne pas que les plateformes de type HPC. Même les applications de calcul scientifique exécutées sur les plateformes de calcul volontaires, appelées couramment *desktop grid* doivent prendre en compte la volatilité des clients volontaires. Citons par exemple les clients volontaires qui participent au projet SETI@Home où la médiane des durées de disponibilité est d'environ 1 heure avec une durée moyenne d'indisponibilité d'environ 4.5 heures [64]. Or le temps de complétion des instances

1. *Mean Time To Failure*

d'exécution du projet SETI@Home peut atteindre 5 jours [13].
Bilan, **les plateformes de calcul parallèle et distribué actuelles et futures sont de plus en plus puissantes par contre elles sont de plus en plus indisponibles et inutilisables à cause des pannes**.

Plusieurs axes de recherche sont envisageables pour assurer la complétion des applications de calcul parallèle et distribué dans ce contexte d'exécution hautement perturbé, par les interruptions des pannes. La première idée qui vient à l'esprit est de mettre plus d'efforts de recherche dans le processus de conception et de fabrication des composants électroniques de la machine. Cependant, les analyses menées pour déterminer les origines principales des pannes, indiquent que les pannes ne sont pas toujours d'origine matérielle mais dans certaines plateformes plus de 50% des pannes surgissent à la couche logicielle [80, 89, 80]. Ces analyses impliquent qu'il faut sans doute fabriquer des composants électroniques plus fiables mais qu'il faut aussi concevoir des approches algorithmiques et logicielles tolérantes aux pannes. Donc le deuxième axe de recherche où s'ancre cette thèse est la conception et la mise en place de techniques logicielles et approches algorithmiques de tolérance aux pannes pour améliorer la fiabilité et la résilience de ces environnements de calcul.

1.2 Positionnement et problématiques

Nombreux sont les défis scientifiques liés à la tolérance aux pannes dans les environnements de calcul parallèle et distribué. Principalement, la mise en place d'une approche de tolérance aux pannes soulève deux grandes problématiques.

- La première est de nature conceptuelle, où il faut concevoir et développer techniquement le mécanisme de tolérance aux pannes en fonction du type de pannes à tolérer, la nature de l'application, l'architecture de la plateforme *etc*.
- La deuxième problématique est soulevée lors de l'intégration et la mise en place du mécanisme de tolérance aux pannes dans le couple application/plateforme d'exécution, où il faut optimiser le compromis entre le surcoût de fonctionnement du mécanisme de tolérance aux pannes d'un coté et l'impact des pannes sur l'exécution d'un autre coté.

Cette thèse traite deux principales questions issues de la deuxième branche.

La première est soulevée lors de la mise en place du mécanisme de tolérance aux pannes le plus populaire dans le contexte considéré, à savoir le mécanisme de sauvegarde et reprise (*Checkpoint/Restart*) [21, 52]. Le principe de ce mécanisme est de sauvegarder périodiquement l'état de l'application sur un support de stockage fiable [38]. En cas de panne, l'application redémarre à partir du dernier point de sauvegarde. Toutefois, une sauvegarde a un surcoût, dans les plateformes actuelles le temps nécessaire pour réaliser une sauvegarde est de l'ordre de 25 minutes [22]. Ainsi, lors de la mise en place du mécanisme de sauvegarde et reprise, la première question à laquelle il faut trouver une réponse est "**à quelle fréquence doit-on sauvegarder l'application pour minimiser le temps de complétion ?**". La première partie de cette thèse est dédiée à la modélisation et la résolution de la problématique d'ordonnancement des points de sauvegarde.

La deuxième question se pose naturellement lors de la phase d'ordonnancement de l'application sur l'ensemble des processeurs disponibles. En effet, lors de la phase

d'ordonnancement de l'application, l'utilisateur est en face d'un compromis où **"il faut choisir entre utiliser plusieurs processeurs en parallèle pour accélérer la complétion de l'application ou utiliser un nombre limité de processeurs pour minimiser la probabilité de panne"**. Dans la deuxième partie de cette thèse, nous nous focalisons sur la conception des stratégies d'ordonnancement tolérant aux pannes pour optimiser à la fois le temps de complétion et la probabilité de succès de l'application.

1.3 Guide de lecture et contributions

Le Chapitre 2 fournit une vue d'ensemble des différents concepts et problèmes associés à la discipline de tolérance aux pannes dans les environnements de calcul parallèle et distribué. Nous présentons principalement trois points essentiels pour aborder les différentes problématiques étudiées dans cette thèse. Le premier point concerne les attributs et les caractéristiques des pannes dans le contexte des environnements de calcul parallèle et distribué. Le deuxième point introduit les concepts et les outils mathématiques utilisés pour modéliser formellement les arrivées des pannes et quantifier leurs impacts sur l'environnement de calcul. Dans le troisième point nous exposons les diverses approches de tolérance aux pannes et les problématiques d'optimisation qui en découlent. Ce chapitre est inspiré d'un chapitre de livre, que nous avons écrit, qui synthétise les différentes approches et techniques de tolérance aux pannes dans les environnements de calcul parallèle et distribué [10] et d'une publication qui présente les connaissances actuelles, les défis et les opportunités de recherche dans le domaine de la tolérance aux pannes dans les environnements de calcul de type HPC [21].

Nous entamons dans le Chapitre 3 la modélisation et la résolution du problème d'ordonnancement des points de sauvegarde dans les environnements de calcul parallèle et distribué. Le critère de performance retenu est la minimisation de l'espérance du temps de complétion. Trois contributions principales sont présentées dans ce chapitre. Dans la première contribution nous exhibons une forme analytique de la politique d'ordonnancement optimale dans le cas où le taux de panne et les surcoûts des points de sauvegarde sont constants. Dans la deuxième contribution nous présentons une approche de résolution numérique dans le cas où la distribution de panne et les surcoûts des points de sauvegarde sont arbitraires. Finalement, dans la dernière contribution nous mettons en évidence à travers des simulations les apports et la qualité de la politique d'ordonnancement des points de sauvegarde proposée. Les contributions présentées ici ont fait l'objet de deux publications. La première est publiée conjointement avec Thierry Gautier, Denis Trystram et Jean-marc Vincent à PPAM 2009 [17]. La deuxième est publiée à Renpar 2009 [15] avec Thierry Gautier.

Nous continuons dans le Chapitre 4 l'étude du problème de compromis soulevé par la mise en place du mécanisme de sauvegarde et reprise. Contrairement au chapitre précédent, ici l'objectif est la minimisation de l'espérance de la quantité totale de temps perdu avant la première panne en considérant une modélisation discrète de l'application. Plusieurs contributions sont exposées dans ce chapitre. Nous démontrons tout d'abord que si les surcoûts des points sauvegarde sont arbitraires alors le problème d'ordon-

nancement des points de sauvegarde est dans la classe des problèmes $\mathcal{NP}$-complet. Ensuite, nous exhibons deux schémas de programmation dynamique pour calculer la politique d'ordonnancement optimale dans le cadre général et dans le cas où les surcoûts de points de sauvegarde sont constants. Finalement, nous conduisons une campagne expérimentale pour comparer l'approche d'ordonnancement proposée avec les approches existantes et mettre en évidence l'apport du mécanisme de sauvegarde et reprise dans les environnements de calcul de type *desktop grid*. Les résultats présentés dans ce chapitre ont fait l'objet de deux publications. La première est publiée en commun avec Derrick Kondo et Denis Trystram à CCGRID 2011 [18]. La deuxième est acceptée en dans la revue IEEE-TC [19] et elle est réalisée conjointement avec Denis Trystram et Frédéric Wagner.

Le Chapitre 5 traite la deuxième problématique d'optimisation de compromis entre la fiabilité et le temps de complétion de l'application. Ici, nous nous focalisons sur la conception des stratégies d'ordonnancement tolérantes aux pannes pour optimiser à la fois le temps de complétion de la dernière tâche et la probabilité de succès de l'application. Dans ce chapitre nous présentons une analyse plus fine du problème en exploitant les diverses propriétés mathématiques des distributions de panne. Tout d'abord nous démontrons qu'en fonction de la nature de croissance du taux de panne les deux objectifs à optimiser sont tantôt antagonistes, tantôt congruents. Ensuite, nous donnons en fonction de la nature du taux de panne différentes politiques d'ordonnancement avec des ratios de performance constants par rapport aux deux objectifs considérés. Nous tenons à signaler que la majorité des contributions présentées dans ce chapitre reposent sur la notion mathématique de Schur-convexité issue de la théorie de majoration [77] et les différentes propriétés mathématiques des distributions de panne usuelles [8]. Nous incitons le lecteur à lire la synthèse de ces deux éléments présentée dans les Sections 6.3 et 6.4 à l'annexe.

Le Chapitre 6 conclut cette thèse en présentant une synthèse des divers problèmes de tolérance aux pannes abordés ainsi que les divers résultats obtenus. Nous présentons aussi dans ce chapitre les perspectives et opportunités de recherche qui en découlent des diverses problématiques abordées dans cette thèse.

Préliminaires 2

Sommaire

2.1 Introduction **5**
2.2 Caractéristiques et conséquences des pannes **6**
2.2.1 Caractéristiques des pannes 6
2.2.2 Conséquences des pannes 8
2.3 Modélisation formelle des pannes **9**
2.3.1 Modélisation stochastique des pannes 9
a Collecte et analyse des traces 10
b Inférence statistique 10
2.3.2 Modèle de panne typique 10
2.3.3 Distributions de panne typiques 12
2.4 Approches de tolérance aux pannes **13**
2.4.1 Prévention des pannes 13
a Migration 13
2.4.2 Masquage des pannes 13
a Duplication 13
b Approches algorithmiques (ABFT) 14
2.4.3 Recouvrement après panne 14
a Sauvegarde et reprise coordonnée 16
b Sauvegarde et reprise non coordonnée 17
2.5 Bilan et conclusions **18**

2.1 Introduction

Nous présentons dans ce chapitre les concepts et les acquis essentiels pour aborder les défis de résilience dans les environnements de calcul parallèle et distribué. Cette présentation commence par une analyse des différents attributs qui caractérisent les pannes dans ces environnements. Ensuite nous exhibons, dans la deuxième section, les divers approches et notions mathématiques utilisées pour modéliser formellement les pannes. Finalement nous donnons dans la dernière section une synthèse des différentes approches de tolérance aux pannes utilisées pour assurer et améliorer la résilience des applications exécutées dans ces environnements.

2.2 Caractéristiques et conséquences des pannes

Soulever et relever les défis posés par les problématiques de résilience et de fiabilité dans le contexte des plateformes actuelles de calcul parallèle et distribué, nécessite en premier lieu l'analyse et la compréhension profonde des caractéristiques et des conséquences des pannes dans ces environnements d'exécution. Dans la suite de cette section nous exposons les différents attributs des pannes, en exhibant tout d'abord les caractéristiques fondamentales de l'événement "panne" et en présentant par la suite les diverses conséquences des pannes dans le contexte considéré.

2.2.1 Caractéristiques des pannes

Nous détaillons dans cette partie les différents attributs qui caractérisent les pannes et qui sont essentiels dans le cadre de ce travail. Nous notons que les diverses notions et propriétés des pannes présentées dans la suite de ce chapitre sont relatives aux environnements de calcul parallèle et distribué.

Environnement de calcul parallèle et distribué. Nous nous restreignons dans cette thèse aux environnements de calcul parallèle et distribué de type HPC (*i.e. High Performance Computing*) et aux environnements de calcul volontaire appelés *desktop grid*.

- Le premier environnement que nous considérons contient les plateformes de calcul et les applications de type HPC. Usuellement, une application de type HPC est exécutée en parallèle sur plusieurs processeurs. Techniquement, cette application est composée d'un ensemble de processus qui sont repartis sur l'ensemble des coeurs disponibles. Pour exécuter l'application, ces processus coopèrent entre eux en échangeant des messages via les canaux de communication ou par une mémoire partagée.
- Le deuxième environnement contient les plateformes de calcul qui se basent sur le principe de calcul volontaire, appelées couramment *desktop grid*. Typiquement les applications exécutées dans les plateformes de type *desktop grid* reposent sur une architecture Client/Serveur (ou appelée aussi Maitre/Esclave). L'application est composée de plusieurs processus qui sont exécutés sur un ensemble de processeurs. Contrairement aux applications HPC, dans ce cas de figure les processus (les clients) sont complètement indépendants et ils ne communiquent qu'avec le serveur.

Définition de la panne. La première question de très grand intérêt qui se pose dans notre étude concerne la définition de l'événement "panne". La réponse informelle adéquate à cette question est la suivante : une panne est un événement qui cause le dysfonctionnement dans le couple composé par l'application et la plateforme d'exécution. Plus précisément nous définissons une panne, comme étant un événement qui cause, premièrement **l'indisponibilité** d'un ou plusieurs composants de la plateforme de calcul durant une durée qui peut être transitoire ou permanente et deuxièmement la **perte** ou la **non-conformité** des résultats calculés depuis le début de l'exécution. Ainsi selon le type de la plateforme de calcul et la nature de l'application l'événement panne peut avoir une nature, une origine et une étendue spatio-temporelle différente.

Nature des pannes. Nous distinguons principalement dans les environnements de calcul parallèle et distribué deux natures de panne. La première classe contient les pannes dont l'origine de déclenchement est un événement purement accidentel. En effet ces pannes arrivent pour diverses raisons par exemple :

- dans l'environnement HPC, les pannes sont dues à l'usure physique du matériel, une erreur de manipulation humaine, à cause des interférences de l'environnement externe telle que les rayons cosmiques *etc.*
- dans l'environnement *desktop grid*, les pannes sont dues à la mise hors tension ou le redémarrage de la machine par le volontaire lui même, la suppression du logiciel client par le volontaire *etc.*

La deuxième classe contient les pannes d'origine malicieuse. Ces pannes malveillantes sont le pur produit du coté obscur de l'être humain. Elles se manifestent de la même façon dans les deux environnements de calcul considérés et elles se distinguent principalement des pannes accidentelles par leur aspects malicieux qui rend la tâche de leur détection et correction plus compliquée. La détection de ces pannes est délicate car assez souvent l'attaquant exploite les failles du système pour renvoyer des résultats qui sont similaires aux résultats attendus mais qui sont erronés. Par exemple un client volontaire qui renvoie toujours des faux résultats. Dans cette étude nous nous focalisons plutôt sur la classe des pannes qui sont de nature accidentelle pour deux raisons. Nous tenons à signaler tout d'abord que tous les travaux existant à notre connaissance qui se sont intéressés à l'analyse de la nature des pannes dans les centres de calcul HPC [56, 89, 23, 21, 80, 75], ne signalent pas l'existence des pannes d'origine malicieuse. Ce constat est tout à fait raisonnable, puisque l'accès physique à ces centres de calcul institutionnels est bien protégé. Nous notons aussi que même dans les plateformes qui reposent sur le principe de volontariat les pannes de nature malicieuse sont quasi inexistantes.

Racine principale de la panne (*root cause breakdown*). La racine principale de la panne est l'attribut qui joue le rôle le plus important dans la conception des approches de tolérance aux pannes.
Les travaux existants distinguent principalement deux racines de panne dans le cadre des environnements HPC. La première classe contient les pannes dont la racine principale est un problème matériel (*hardware failures*) tandis que la deuxième classe contient les pannes dont la racine principale est un problème logiciel (*software failures*). Ainsi la question qui se pose immédiatement lors de la conception des approches de tolérance aux pannes dans ce contexte est : "faut il concevoir une approche de tolérance aux pannes plus orientée vers les pannes d'origine matérielle ou bien le contraire ?". Toutefois en déduire quelle est la racine principale des pannes la plus répandue dans ces environnements est une tâche très difficile et les quelques travaux existants ont des conclusions différentes. Par exemple Gibson *et al.* [89] concluent que 53% des pannes qui ont touché les 4750 nœuds du centre de calcul de Los Alamos National Laboratory (LANL) durant la période 1995-2005 sont d'origine matérielle. Par contre Oliner *et al.* [80] constatent que 64% des pannes ont une racine qui est en lien avec la couche logicielle en analysant des traces provenant des centres de calcul LLNL et SNL où nous trouvons par exemple la machine Blue Gene/L avec 131072 processeurs. Cette divergence est due probablement aux raisons suivantes. Tout d'abord, nous notons que toutes les analyses existantes sont menées sur des traces d'exécution de provenances

différentes. Deuxièmement, seuls quelques échantillons de trace d'exécution sont disponibles puisqu'obtenir des informations et des données telles que les traces d'exécution ou les fichiers journaux des super-calculateurs est difficile pour diverses raisons, comme par exemple la sécurité et la sensibilité de telles informations, l'image de marque des fabricants *etc*. De plus, les quelques traces disponibles sont généralement des fichiers très volumineux par exemple l'analyse menée par Oliner *et al.* [80] est effectuée en utilisant une trace de la machine Blue Gene/L de taille 111.67 Go et qui contient 178,081,459 événements. Rajoutons à tout cela, le fait que les informations contenues dans ces traces sont ambiguës et elles nécessitent dans la majorité des cas l'expertise des administrateurs du système informatique en question pour en déduire les racines des pannes [56, 80].

Selon les analyses menées dans les environnements de type *desktop grid* [64, 55], la racine principale des pannes est liée directement au comportement du volontaire c'est-à-dire quand il décide d'éteindre ou redémarrer sa machine ou quand il décide d'arrêter le processus client qui tourne sur sa machine *etc*. De plus dans cet environnement la couche matérielle et la couche logicielle du client formée par l'application, l'intergiciel et le système d'exploitation sont suffisamment simples pour supposer qu'elles sont fiables.

Pour finir, nous concluons à l'issue de cette partie, qu'il faut concevoir des approches de tolérance aux pannes qui tolèrent à la fois les pannes d'origine matérielle et les pannes d'origine logicielle.

Detection des pannes. Il est sans doute évident que la tolérance aux pannes passe avant toute chose par la détection de ces pannes. Nous trouvons typiquement dans les deux environnements de calcul parallèle et distribué considérés deux classes de pannes. Nous trouvons dans la première classe les pannes franches (*i.e. crash fault* ou *fail stop*)[6, 23]. Dans ce cas de figure, dés que l'événement panne survient, l'application s'arrête en renvoyant par exemple un code d'erreur. Donc les pannes franches sont assez souvent détectées immédiatement.
La deuxième classe contient les pannes de nature silencieuse (*i.e. silent failures*). Les pannes qui ont une nature pareille sont simplement les pannes indétectables, ou qui sont détectées après une très grande durée depuis leurs déclenchements ce qui implique nécessairement que le résultat fourni est incorrect [23]. Nous citons l'exemple le plus répandu dans les systèmes informatiques en général qui est l'inversement des bits (*i.e. flipping of bits*) dans la mémoire centrale ou les caches des processeurs qui peut être provoqué par les rayons cosmiques ou les interférences électromagnétique *etc*.
Dans ce travail nous considérons qu'il existe forcément un détecteur de panne intégré dans le couple application/ plateforme d'exécution qui permet la détection des pannes. Nous tenons aussi à signaler que même en absence d'un tel mécanisme nous proposons dans le chapitre 4 une technique de détection intégrée dans les points de sauvegarde durant l'exécution.

2.2.2 Conséquences des pannes

Les pannes se distinguent aussi par leurs conséquences sur le couple formé par l'application et la plateforme d'exécution. Dans la suite de cette partie nous présentons

le régime des pannes qui représente l'étendue temporelle des pannes sur la plateforme d'exécution. Finalement nous exposons les impacts des pannes sur l'application.

Régime de panne. Dans notre contexte d'étude deux régimes de panne sont possibles et qui se distinguent selon le temps durant lequel les pannes durent. Le régime de panne est permanent si l'élément touché par la panne ne fonctionnera plus jamais. Par contre le régime de panne est transitoire si l'élément touché par la panne est hors d'usage seulement durant une durée de temps transitoire qui peut être de nature déterministe ou aléatoire.

Impacts des pannes. Deux cas de figures sont envisageables suite à l'arrivée d'une panne durant l'exécution. Dans le premier cas la panne provoque la perte de la totalité du travail effectué depuis le début de l'exécution. Ce modèle est sans doute le cas de figure le plus répandu dans le contexte des applications parallèles et distribuées. Dans le deuxième cas de figure l'occurrence de la panne provoque simplement la préemption de l'exécution pendant une durée de temps qui peut être déterministe ou aléatoire. Ensuite une fois que la plateforme d'exécution est disponible de nouveau l'application redémarre là ou elle a été interrompu.
Dans ce travail nous considérons le premier modèle d'impact qui reflète la quasi majorité des intergiciels des applications parallèles tels que KAAPI [11], CHARM ++ [26], LAM-MPI [85], Open-MPI [60] et MPICH-V [20].

2.3 Modélisation formelle des pannes

Pour concevoir les approches de tolérance aux pannes et comparer par la suite les améliorations apportées à la résilience du couple application plateforme d'exécution. Il est primordial que nous puissions modéliser formellement les arrivées des pannes durant l'exécution ou la durée qu'il faut pour la réparation. Dans la suite de cette section, nous exposons tout d'abord les différentes approches existantes pour construire les modèles statistiques qui modélisent formellement les instants d'arrivée des pannes et la durée nécessaire pour la réparation des impacts de ces pannes. Ensuite, nous présentons les différents concepts et modèles mathématiques utilisés comme étant des paramètres d'entrée dans cette thèse pour concevoir et optimiser les stratégies de tolérance aux pannes.

2.3.1 Modélisation stochastique des pannes

Plusieurs efforts de recherche ont été conduits pour développer des modèles statistiques qui décrivent l'inter-arrivées des pannes ou la durée de réparation des composants dans les deux contextes d'exécution parallèle retenus. Sans trop rentrer dans les détails des différents travaux de modélisation puisque ceci n'est pas l'objet de cette étude, les travaux de modélisation des durées aléatoires de l'arrivée ou de la réparation des pannes se divisent principalement en deux étapes.

a Collecte et analyse des traces

La première étape primordiale est la phase de collecte et de filtrage des traces d'exécution de la plateforme en question. Pour réaliser cette étape, les travaux existants se distinguent essentiellement en trois points.

1. Le premier point concerne la provenance et la méthode de collecte des traces d'exécution. Par exemple les traces étudiées par Gibson *et al.* [89], sont créées manuellement par les administrateurs système suite à chaque dysfonctionnement. Ces traces concernent 4750 nœuds de calcul, 24101 processeurs situés dans le centre de calcul Los Alamos National Laboratory (LLAN) entre la période 1996-2005. Par contre les traces étudiées par Heien *et al.* [56], sont les fichiers journaux produits automatiquement par le système d'exploitation du cluster Mercury localisé au centre de calcul National Center for Supercomputing Applications(NCSA) durant 2004-2010.
2. Le deuxième point qui différencie les travaux de modélisation est la méthode utilisée pour analyser les traces et extraire l'instant et la racine principale de la panne. Par exemple dans l'analyse proposée par Gibson *et al.* [89] cette tâche est confiée aux administrateurs système qui déterminent ces attributs après chaque panne. Tandis que dans l'analyse menée par Heien *et al.* [56] un ensemble d'expression régulière est construit en s'appuyant toujours sur l'expertise des administrateurs système pour extraire les événements qui correspondent aux pannes dans les traces.
3. Le dernier point qui change d'un travail à un autre est la granularité considérée. Par exemple un modèle de panne par nœud de calcul est construit dans [89]. Tandis que dans [56] une granularité plus fine est adoptée en construisant un modèle de panne pour chaque composant c'est-à-dire un modèle de panne pour, la mémoire, le processeur, le support de stockage partagé (NFS) *etc*

Ainsi à l'issue de cette première étape nous avons un échantillon de données empiriques qui décrivent les instants d'arrivée des pannes et les instants de réparation de chaque élément touché par la panne.

b Inférence statistique

Une fois que l'étape de collecte et analyse est faite, la deuxième phase est simplement l'étape d'inférence statistique pour trouver des correspondances entre les données empiriques et les lois de probabilité usuelles telles que la loi exponentielle, la loi Weibull, la loi Gamma *etc*. Cette étape d'inférence est conduite en supposant que les événements de panne ou de réparation sont indépendants et identiquement distribués ($i.i.d$).

2.3.2 Modèle de panne typique

Ainsi à l'issue des deux étapes de modélisation décrites auparavant, nous obtenons un modèle stochastique qui décrit les instants d'arrivée et les durées de réparation des pannes dans la plateforme. Nous présentons dans cette partie les divers concepts, modèles et notations mathématiques typiques utilisés dans la totalité de la thèse. Sans perte de généralité supposons que la granularité utilisée pour construire le modèle stochastique est l'élément de calcul et que le régime de panne de ce dernier est un

régime transitoire. L'état de cet élément de calcul alterne entre deux états selon un processus semi-Markovien [83] comme l'indique la Figure 2.1.

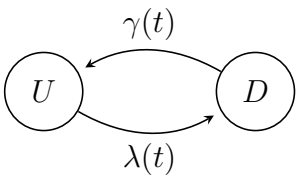


FIGURE 2.1 – Chaîne de Markov associée à l'élément de calcul

Le premier est l'état disponible noté par U (Up), dans lequel l'élément de calcul est disponible pour exécuter les instructions de l'application pendant U unités de temps. À la fin de cet intervalle une panne surgit et l'élément de calcul passe alors dans le second état indisponible noté par D ($Down$). Une fois dans cet état, l'élément de calcul est indisponible durant une durée de temps notée par D. Par exemple cette durée représente le temps de réparation d'un composant défectueux dans l'élément de calcul ou elle peut être considérée comme la durée pendant laquelle le client volontaire est absent du réseau de la plateforme de calcul volontaire. Les deux intervalles de temps écoulés dans les états disponible et indisponible ont une durée aléatoire et ils sont modélisées par des variables aléatoires réelles ($v.a.r$) supposées indépendantes et identiquement distribuées ($i.i.d$). Nous associons à ces deux variables les fonctions distribution et densité notées respectivement $F(t), f(t)$ pour l'inter-arrivées des pannes et $D(t), d(t)$ pour la durée d'indisponibilité. Considérons par exemple la distribution des pannes comme cas d'illustration. Par définition la fonction distribution F est une fonction croissante monotone en t, à valeur dans l'intervalle $[0, 1]$. Formellement $F(t)$ représente la probabilité que l'élément de calcul subit une panne dans l'intervalle $[0, t]$. Nous définissons respectivement $\overline{F}(t) = 1 - F(t)$ la probabilité qu'une panne surgisse durant l'intervalle $[t, +\infty]$. À la fonction distribution nous associons la fonction densité notée par $f(t)$ tel que $F(t) = \int_0^t f(x)dx$. Ainsi le temps moyen inter-panne ($MTTF$) est donné par $\int_0^\infty xf(x)dx$. La transition de l'état disponible vers l'état indisponible et le retour de l'état indisponible vers l'état disponible se produit en fonction du taux de panne dans un sens et du taux d'indisponibilité dans l'autre sens. Les fonctions taux de transition sont notées respectivement par $\lambda(t)$ et $\gamma(t)$. La fonction taux de panne ou taux d'indisponibilité est reliée à la fonction distribution et à la fonction densité par l'identité suivante [8] :

$$F(t) = 1 - \exp^{-\int_0^t \lambda(u)du} = \int_0^t f(x)dx. \tag{2.1}$$

La fonction $\lambda(t)$ joue un rôle très important dans la théorie de fiabilité. Par exemple la quantité $\lambda(t)dt$ représente la probabilité qu'un élément de calcul qui fonctionne depuis t unité de temps subisse une panne durant l'intervalle $[t, t + dt]$. Nous donnons également dans la Section 6.4 d'autres interprétations et propriétés mathématiques qui sont utilisées dans cette thèse.

Les distributions de pannes typiques utilisées pour modéliser formellement les pannes ont une fonction de taux dont la croissance est monotone dans le temps [8].

Plus précisément les fonctions de distributions de panne typiques sont classées en deux classes en fonction du sens de croissance de la fonction taux. La première classe contient les distributions dont le taux de panne est croissant *IFR* (*Increasing failure rate*) dans ce cas la fonction $\lambda(t)$ est croissante monotone en t. La deuxième classe contient les distributions dont le taux de panne est décroissant *DFR* (*Decreasing failure rate*) et dans ce cas la fonction $\lambda(t)$ est décroissante monotone en t. Ainsi les distributions de cette classe modélisent par exemple l'usure physique d'un composant qui voit sa probabilité de panne augmentée au fur et à mesure dans le temps. Gibson *et al.* [89] indiquent que la classe des distributions *DFR* modélise par exemple la première période dans la durée de vie d'un cluster. En effet très souvent quand un cluster est installé il faut du temps aux administrateurs du centre de calcul pour fixer les bons paramètres de fonctionnement et pour éliminer les composants vulnérables ce qui implique que le taux de panne décroit en fonction du temps.

2.3.3 Distributions de panne typiques

Dans le cadre de la théorie de fiabilité, nous trouvons essentiellement les deux lois de probabilité suivantes.

- Loi exponentielle ou processus de Poisson : C'est la loi la plus populaire pour les deux raisons suivantes. Premièrement pour sa simplicité mathématique puisque c'est la seule loi sans mémoire [1]. Deuxièmement la distribution résultante de la superposition de n distributions arbitraires et identiques converge vers une distribution exponentielle quand n est très grand [1]. Ceci implique qu'il est possible que la distribution de panne d'un système formé par une composition en série [2] de plusieurs composants qui ont une distribution de panne arbitraire et identique converge vers une distribution exponentielle. Formellement la loi exponentielle a un seul paramètre noté par λ dans le cas des inter-arrivées des pannes. Les fonctions distribution, densité et taux associées à la loi exponentielle sont données par les expressions suivantes :

$$F(t) = 1 - \exp^{-\lambda t},\ f(t) = \lambda \exp^{-\lambda t},\ \lambda(t) = \lambda$$

  Ainsi par définition la loi exponentielle est à la fois *IFR* et *DFR*.
- Loi Weibull : Cette loi est aussi très populaire pour la simple raison de sa grande flexibilité. En effet cette loi à queue lourde passe avec succès la majorité des tests statistiques. Formellement, cette loi a deux paramètres un paramètre de forme noté par β_w et un paramètre d'échelle noté α_w.

$$F(t) = 1 - \exp^{-(\alpha_w t)^{\beta_w}},\ f(t) = \beta_w \alpha_w (t\alpha_w)^{\beta_w - 1} \exp^{-(\alpha_w t)^{\beta_w}}, \lambda(t) = \beta_w \alpha_w (\beta_w t)^{\beta_w - 1}$$

  Ainsi si $\beta_w = 1$ la loi Weibull est une loi exponentielle. Nous notons aussi que le paramètre de forme détermine si la fonction taux est croissante ($\beta_w \geq 1$) ou décroissante ($\beta_w \leq 1$).

1. L'absence de mémoire se traduit mathématiquement par $\mathbb{P}\{U \geq t\} = \mathbb{P}\{U \geq s + t | U \geq s\}$
2. La composition série implique que le système fonctionne si et seulement si tous ses composants fonctionnent

2.4 Approches de tolérance aux pannes

Ayant introduit les différentes caractéristiques et conséquences des pannes dans la première section, nous présentons dans cette section une synthèse des différentes approches conçues pour tolérer les pannes dans les environnements de calcul parallèle et distribué. Ces approches se distinguent principalement par la procédure adoptée pour traiter les arrivées des pannes durant l'exécution.

2.4.1 Prévention des pannes

La première classe contient les approches qui sont conçues pour éviter les pannes moyennant des techniques de prévention ou de maintenance. Dans le contexte des applications parallèles et distribuées, nous trouvons essentiellement l'approche de migration comme technique de prévention.

a Migration

L'approche de migration est l'approche de prévention la plus utilisée dans les environnements de calcul parallèle et distribué pour éviter les occurrences des pannes durant l'exécution. Le principe de cette technique est de migrer les processus exécutés sur un processeur qui est susceptible de tomber en panne vers un autre plus robuste. Différentes techniques sont proposées dans le contexte des applications parallèles et distribuées pour implémenter techniquement la migration comme par exemple la migration des processus [27, 100] ou encore la migration des machines virtuelles [90]. Les approches de migration s'intègrent techniquement entre la couche système d'exploitation de la plateforme et l'intergiciel utilisé pour exécuter l'application sans changer quoi que se soit dans l'application.

Toutefois nous notons que les approches de cette classe reposent principalement sur la prédiction des occurrences des pannes. Il est évident que leur efficacité dépend fortement de la qualité de la prédiction spatio-temporelle des occurrences des pannes avant qu'elles arrivent. Selon l'analyse menée dans [21], les quelques approches de prédiction des pannes existantes ne sont pas très efficaces. En effet ces approches de prédiction de panne prédisent les occurrences des pannes en analysant les traces des pannes or ces traces ne sont pas toujours représentatives des plateformes actuelles.

2.4.2 Masquage des pannes

Nous trouvons dans la deuxième classe les approches de tolérance aux pannes qui masquent les impacts des pannes et permettent à l'application de se terminer même si des pannes surgissent. Les deux approches différentes conçues dans le contexte des applications parallèles et distribuées sont la duplication et les approches algorithmiques.

a Duplication

L'approche de tolérance aux pannes par duplication consiste en la création de plusieurs copies des processus sur différents nœuds de calcul. Ainsi en cas de dysfonctionnement d'une copie l'exécution se poursuit normalement puisque la panne

est masquée via la duplication. Cette approche est intégrée techniquement au niveau système c'est-à-dire entre le système d'exploitation de la plateforme et l'intergiciel utilisé pour exécuter l'application. Deux techniques différentes sont proposées pour implémenter cette approche. Dans la première technique la duplication est de nature active [88] c'est-à-dire les copies secondaires jouent le même rôle que la copie primaire en exécutant l'application au même temps. Dans la deuxième technique la duplication est de nature passive (*standby redundancy*) [9] où seulement la copie primaire exécute l'application tandis que les autres copies sont en état passif et elles ne font que se synchroniser avec la copie primaire. Ainsi en cas de panne une copie secondaire prend la place de la copie primaire

Toutefois la tolérance aux pannes via duplication est une approche relativement couteuse en terme de ressource de calcul. En effet dans les deux cas de figure (passif ou actif) pour tolérer une panne il faut dupliquer chaque processus au moins une fois, c'est-à-dire la moitié des ressources de calcul est dédiée à la tolérance aux pannes. Par exemple les auteurs de P2PMPI [44] reporte que la mis en place d'une stratégie de duplication active de degré deux c'est-à-dire chaque processus est dupliqué une fois, augmente de 60% le temps de complétion de l'application. Ainsi l'approche de tolérance aux pannes via la duplication est principalement utilisée dans les environnements de calcul parallèle et distribué de type *Grid* ou P2P, car les processeurs de calcul ne sont pas fortement synchronisés et de plus ils ont un coût relativement faible.

b Approches algorithmiques (ABFT)

Le principe des approches de type ABFT (*Algorithmic Based Fault Tolerance*) est la détection et la correction des erreurs causées par les pannes durant l'exécution de l'application. Généralement la détection et la correction sont effectuées à la fin de l'exécution. Typiquement les approches de type ABFT rajoutent quelques informations de redondance dans les données d'entrées comme par exemple le calcul d'un *Checksum* en fonction d'une matrice d'entrée [59]. Ensuite un algorithme est conçu en fonction du schéma d'exécution de l'application pour détecter et corriger les erreurs causés par les pannes dans les résultats à la fin de l'exécution. Cet algorithme de détection et de correction prend en entrée les informations de redondance calculées préalablement et le résultat final. Contrairement aux approches de tolérance aux pannes de type duplication, les approches de type ABFT s'intègrent au niveau applicatif puisque les algorithmes de détection et de correction sont conçus en fonction du schéma d'exécution de l'application. Ainsi cette approche reste applicable que pour certaines classes d'applications de calcul numérique telle que les applications de produit matriciel. De plus les approches de ce type telle que celle proposée par Huang *et al.* [59] ne tolèrent pas les pannes franches puisque en cas de panne franche l'application est immédiatement abandonnée d'où le résultat final est inexistant.

2.4.3 Recouvrement après panne

Nous trouvons dans la dernière classe les approches de tolérance aux pannes qui sont conçues pour amortir les effets des pannes en reconstruisant l'état de l'application en cas de panne. Le principe de base de ces approches est de sauvegarder périodiquement une image de l'état global de l'application (*i.e. Checkpoint*). Cette sauvegarde est placée sur

un support de stockage stable. Ainsi en cas de panne l'application redémarre à partir de ce point de sauvegarde au lieu de redémarrer du début. À la différence des approches décrites auparavant, ces approches reposent sur un support externe qui est le support stable pour sauvegarder les données nécessaires au redémarrage de l'application. Le support stable est un élément clé et il est défini comme étant un dispositif de stockage persistant et fiable. Plus précisément le support stable doit vérifier nécessairement les points suivants :

1. Accessibilité : Les données de sauvegarde sont accessibles à tout moment.
2. Fiabilité : Les données sauvegardées ne doivent pas être perdues ou changées même en cas de panne.

L'architecture et la nature du support stable sont conçues en fonction des types de panne qu'on veut tolérer sur ce dernier. Usuellement dans les centres de calcul HPC le support stable est une baie de stockage dédiée à cet effet [52].

Dans le contexte des applications parallèles et distribuées, nous trouvons essentiellement trois variantes d'implémentation de cette approche. Ces variantes se distinguent entre elles en la couche d'implémentation choisie pour s'intégrer dans le couple application plateforme d'exécution.

Niveau applicatif (*Application level checkpointing*). Cette variante est aussi appelée *user-defined checkpointing*, car c'est à l'utilisateur de définir l'état de l'application à sauvegarder par exemple quelle variable dans l'application doit être sauvegardée sur le support stable et il doit aussi prédéfinir à quel instant ces variables doivent être sauvegardées sur le support stable. Cette approche est efficace puisqu'elle bénéficie des informations et des nuances appropriées de l'évolution de l'état de l'application apportées par l'utilisateur. Cependant cette approche est définie au moment du codage en fonction de l'application. Elle nécessite plus d'effort dans la phase de codage de l'application

Niveau compilateur (*Compiler-level checkpointing*). Nous trouvons aussi des compilateurs qui rajoutent des points de sauvegarde dans l'exécutable au moment de la compilation de l'application [74]. Cependant il n'est pas aussi évident pour le compilateur de trouver les emplacements optimaux des points de sauvegarde puisqu'il n'a aucune information sur l'évolution de l'exécution au cours du temps.

Niveau système (*System level checkpointing*). L'approche de sauvegarde au niveau système est la variante qui a reçu le plus d'attention dans les environnements de calcul HPC. Dans cette approche, la sauvegarde de l'état global de l'application est implémentée d'une façon transparente par rapport à l'application. En effet cette approche s'intègre directement au niveau de l'intergiciel utilisé pour exécuter l'application. Nous rappelons que l'état global d'une application parallèle est composé des états locaux des processus et des états des canaux de communication entre les processus. Donc pour sauvegarder l'état de l'application il faut sauvegarder les états des processus et les messages circulant dans les canaux de communication. Sauvegarder l'état d'un processus est une tâche relativement facile en utilisant des bibliothèques spécialisées telle que BLCR [54]. Cependant la sauvegarde des états des canaux de communication est une

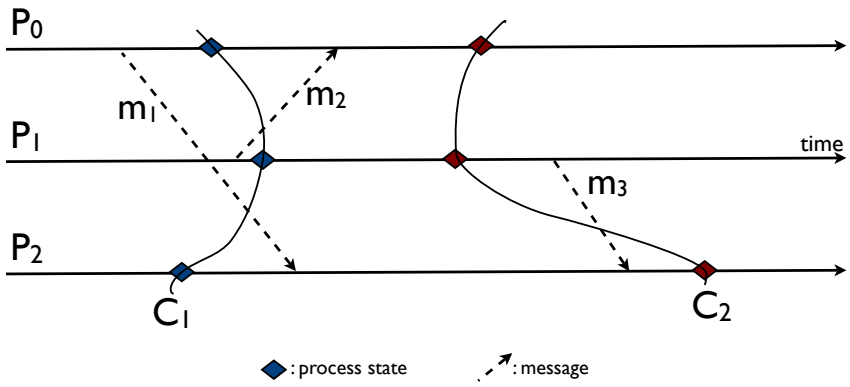


FIGURE 2.2 – Exemple typique d'un point de sauvegarde cohérent C_1 et un autre point incohérent C_2 [10]

tâche plus délicate qui nécessite un protocole d'orchestration pour synchroniser les processus impliqués dans l'exécution. Plus précisément, pour qu'un point de sauvegarde soit cohérent il faut s'assurer que si l'état sauvegardé d'un processus acquitte la réception d'un message alors il faut aussi que l'état sauvegardé du processus émetteur acquitte l'envoi de ce message. Considérons l'exemple des deux points de sauvegarde C_1 et C_2 dans la Figure 2.2. Le point de sauvegarde C_1 est cohérent car toutes les réceptions acquittées leurs envois respectifs sont aussi acquittés. Par contre dans le point de sauvegarde C_2 la réception du message m_3 est acquittée dans la sauvegarde du processus P_2 tandis que l'envoi n'est pas acquitté par P_1.

Pour garantir que le point de sauvegarde représente un état cohérent de l'application, nous trouvons principalement dans le contexte des applications parallèles et distribuées deux variantes de protocole de synchronisation de processus.

a Sauvegarde et reprise coordonnée

Dans le protocole de sauvegarde coordonnée, tous les processus se synchronisent pour créer un point de sauvegarde qui représente un état cohérent de l'exécution. En supposant que les messages sont délivrés selon la stratégie FIFO (First In First Out) dans les canaux de communication. Chandy et Lamport [29] proposent une procédure distribuée d'échange de message pour vider les canaux de communication lors de la prise du point de sauvegarde. Cette procédure est enclenchée par le processus qui commande la prise des points de sauvegarde en arrêtant tout d'abord l'exécution de l'application localement et en envoyant ensuite un message spécial d'initiation de sauvegarde à tous ses voisins. Ensuite chaque processus qui reçoit ce message d'initiation de sauvegarde fait exactement la même chose. Une fois qu'un processus reçoit autant de message d'initiation de sauvegarde que le nombre de ses voisins il entame à ce moment la sauvegarde de son état local et il envoie ensuite cette sauvegarde sur le support

stable. Finalement chaque processus reprend l'exécution de l'application une fois qu'il a envoyé la sauvegarde local sur le support stable. En cas de panne un redémarrage global est effectué, en redémarrant tous les processus à partir du dernier point de sauvegarde. Finalement nous tenons à signaler que l'approche de tolérance aux pannes par sauvegarde coordonnée est implémentée dans la majorité des intergiciels de calcul parallèle KAAPI [11], LAM-MPI [85], Open-MPI [60] et c'est la seule solution intégrée par défaut dans la machine HPC IBM Blue Gene/P [94].

b Sauvegarde et reprise non coordonnée

Le protocole de sauvegarde non coordonnée est conçu principalement pour éviter la congestion réseau sur le support stable et la contrainte de redémarrage global en cas de panne. Typiquement dans ces approches, chaque processus est autonome c'est-à-dire qu'il décide quand et où la sauvegarde locale est enclenchée. De même en cas de panne seul le processus touché par la panne est redémarré. Plus précisément, ces approches reposent principalement sur l'hypothèse PWD (*PieceWise Deterministic*) [95]. Formellement l'hypothèse PWD signifie qu'un processus, peut être modélisé par une séquence d'intervalles d'états déterministes et chaque intervalle d'état commence par un événement non déterministe, dont la sauvegarde permet la reconstruction de l'état du processus en rejouant les différents événements. Un événement non déterministe peut être la réception d'un message ou un événement interne au processus. Ainsi contrairement à l'approche coordonnée qui repose sur le vidage des canaux de communication, les approches non coordonnées sauvegardent l'état des canaux de communication et l'état local de chaque processus d'une manière non coordonnée. Nous trouvons principalement trois protocoles pour sauvegarder les messages circulant dans les canaux de communication.

- La première variante adopte une approche pessimiste en supposant qu'une panne se produit après chaque événement non déterministe dans l'exécution. Les déterminants de chaque événement non déterministe sont sauvegardés dans la mémoire stable. Ainsi en sauvegardant tous les messages non déterministes, cette approche garantit toujours la possibilité de recouvrement à partir d'un point de sauvegarde valide. Cependant la sauvegarde de tous les messages non déterministes dégrade les performances de l'application.
- La deuxième variante adopte une approche optimiste en écrivant les déterminants des événements non déterministes de façon asynchrone sur la mémoire stable. Dans cette approche, la panne est supposée survenir une fois que la procédure de sauvegarde est terminée. Il est évident que cette approche élimine le blocage de l'exécution, ce qui réduit le surcoût du mécanisme. Mais cette approche peut provoquer l'effet domino c'est-à-dire la reprise de l'exécution depuis le début.
- La dernière variante appelée journalisation causale combine les deux approches précédentes en gardant localement les sauvegardes de chaque processus et en rajoutant des information de causalité dans les messages échangées entre les processus. Ainsi cette approche évite l'effet domino tout en réduisant les surcoût de sauvegarde des messages. Cependant en cas de panne la procédure de redémarrage doit gérer la causalité entre les différents états des processus.

2.5 Bilan et conclusions

Les analyses actuelles ainsi que les projections futures des impacts des pannes sur les environnements d'exécution parallèle et distribuée soulèvent un défi scientifique de très grande importance. Ces analyses montrent que les applications exécutées sur ces plateformes doivent faire face à des pannes très occasionnelles et qui ont plusieurs caractéristiques et conséquences sur le couple application plateforme d'exécution. Dans cette thèse nous nous focalisons sur les pannes franches de nature accidentelle qui surviennent au niveau logiciel ou matériel. Nous supposons que ces pannes sont modélisées formellement par des $v.a.r$ de nature $i.i.d$. Pour tolérer ces pannes, nous trouvons dans l'état de l'art plusieurs approches de tolérance aux pannes. Nous constatons que dans le contexte des applications parallèles et distribuées l'approche de tolérance aux pannes la plus populaire est l'approche sauvegarde et reprise coordonnée. Cette approche est implémentée dans la quasi-totalité des intergiciels de calcul parallèle KAAPI [11], LAM-MPI [85], Open-MPI [60] et de plus elle est disponible dans les machines de production comme par exemple la machine IBM Blue Gene/P [94]. Cette approche est populaire car elle repose sur un principe naturel et simple. De plus elle peut être utilisée sans changer les codes source ou les exécutables des applications et finalement elle ne consomme pas les ressources de calcul.

Par contre l'utilisation de cette approche soulève une problématique d'optimisation combinatoire très importante. En effet même si cette approche ne consomme pas de ressources de calcul elle introduit un surcoût supplémentaire au moment de la prise du point de sauvegarde. Donc elle soulève un problème de gestion de compromis entre la quantité de travail perdu et le surcoût accumulé des points de sauvegarde. Dans la première partie de cette thèse nous nous intéressons à la modélisation et à la conception de stratégie d'ordonnancement de point de sauvegarde pour optimiser ce compromis.

I

Modélisation Stochastique du Mécanisme de Sauvegarde et Reprise

Modélisation continue 3

Sommaire

3.1 Introduction **21**
3.2 Modélisation sans tolérance aux pannes **22**
3.2.1 Modèle de l'environnement d'exécution 22
3.2.2 Distribution du nombre de re-exécutions 24
3.2.3 Espérance du temps de complétion 25
3.2.4 Bilan 27
3.3 Modélisation avec tolérance aux pannes **28**
3.3.1 Modèle du mécanisme de sauvegarde et reprise 28
3.3.2 Formulation du problème d'optimisation 31
a Espérance du temps de complétion avec sauvegarde et reprise 31
b Problème d'optimisation sous contraintes 32
3.3.3 Résolution du problème d'optimisation 32
a Solution analytique 32
b Solution numérique 34
3.4 État de l'art **35**
3.5 Validation et comparaison expérimentale **41**
3.5.1 Paramètres et scénarios des simulations 41
3.5.2 Résultats des simulations 41
3.5.3 Bilan des simulations 43
3.6 Conclusion **44**

3.1 Introduction

L'intégration des mécanismes de tolérance aux pannes dans les environnements de calcul parallèle et distribué actuels et futurs est une tâche incontournable. Parmi les problématiques soulevées par cette tâche est l'optimisation du compromis entre le surcoût induit par le mécanisme de tolérance aux pannes et les impacts des pannes sur l'environnement d'exécution. Dans ce chapitre nous présentons une modélisation et une méthodologie d'optimisation pour cette problématique. Tel que le nom du chapitre l'indique, nous nous focalisons sur l'intégration du mécanisme de sauvegarde et reprise coordonnée. De plus la formulation du problème repose sur une modélisation stochastique continue des différents critères de performance. Dans ce modèle, nous formalisons analytiquement le compromis entre l'impact des pannes et l'impact du surcoût accumulé des points de sauvegarde sur le temps de complétion de l'application.

Comme ligne directrice, nous commençons par l'étude des performances de l'application exécutée dans un environnement d'exécution perturbé par des pannes sans mécanisme pour tolérer les pannes. Ce modèle servira comme modèle référentiel pour analyser et comparer les améliorations apportées par l'utilisation du mécanisme de sauvegarde et reprise coordonnée. Dans un deuxième temps, nous généralisons dans la Section 3.3 le modèle de performance pour inclure le mécanisme de sauvegarde et reprise coordonnée. Ce nouveau modèle exprime analytiquement le compromis entre la quantité de travail perdu à cause des pannes et le surcoût accumulé des points de sauvegarde. Ce compromis est encapsulé dans une fonction objectif qui exprime l'espérance du temps de complétion en fonction des paramètres de l'application, des propriétés de la plateforme d'exécution et de la stratégie d'ordonnancement des points de sauvegarde. En s'appuyant sur ce modèle de performance nous formulons un problème d'optimisation sous contraintes pour calculer la stratégie d'ordonnancement des points de sauvegarde qui minimise l'espérance du temps de complétion de l'application. Finalement, pour valider le modèle et la stratégie d'ordonnancement proposée nous présentons dans la Section 3.5 une comparaison avec les travaux existants au travers de simulations.

3.2 Modélisation sans tolérance aux pannes

Nous décrivons dans cette section un modèle de performance du couple formé par l'application et la plateforme d'exécution pour mettre en évidence l'impact des pannes sur les performances de l'application en absence de tout mécanisme de tolérance aux pannes. Nous commençons par la présentation du modèle de l'environnement d'exécution considéré où nous décrivons à la fois, le modèle d'exécution d'application et le modèle de plateforme ainsi que le modèle de panne sous-jacent. Finalement, en se basant sur ces deux modèles, nous présentons le modèle de performance qui exprime analytiquement l'espérance du temps de complétion ainsi que la distribution du nombre de re-exécutions en fonction des paramètres de l'application et du taux d'inter-arrivées des pannes de la plateforme.

3.2.1 Modèle de l'environnement d'exécution

Nous ciblons dans ce travail les environnements de calcul parallèle et distribué de type HPC. Typiquement dans un tel environnement l'application est caractérisée par la quantité de travail qu'elle contient. Ainsi, l'application parallèle est supposée composée de w unités de travail. Cette application est exécutée sur un ensemble de processeurs identiques et homogènes. Par convention dans ce travail, cet ensemble de processeurs est considéré comme formant un seul élément de calcul. Formellement, l'élément de calcul est modélisé par les trois attributs suivants.

1. un taux de traitement d'instructions par unité de temps noté par ν. Cet attribut encapsule à la fois le nombre des processeurs impliqués dans le calcul et la puissance de calcul de chacun.
2. un taux de disponibilité (nommé aussi par taux de panne) noté par $\lambda(t)$ qui représente la durée de temps où l'élément de calcul est disponible pour exécuter l'application. Typiquement cet attribut représente la superposition des lois de

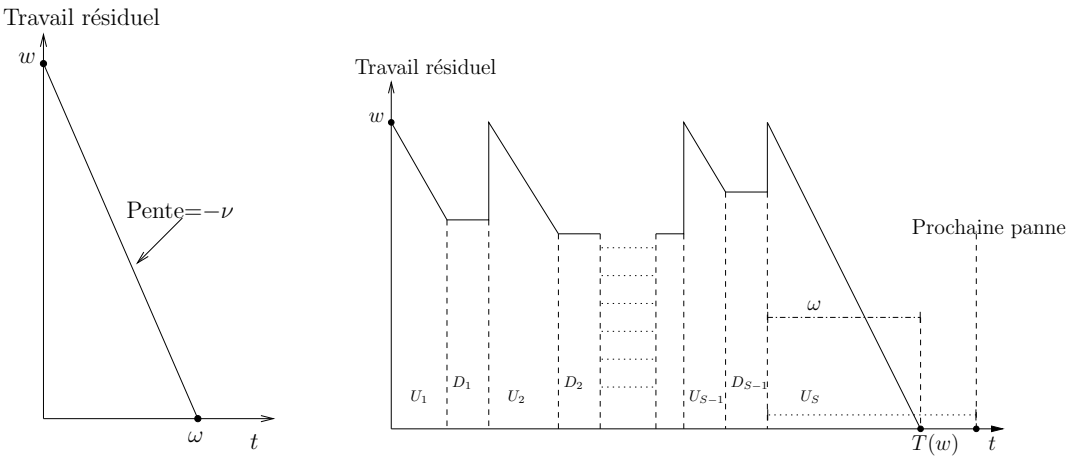
panne de tous les processeurs. Plus précisément, nous rappelons qu'une application de type HPC est touchée par une panne si au moins un processeur est touché par une panne. Ainsi dans ce cas, la loi de panne globale de l'élément de calcul est donnée par la composition en série des différentes lois de panne des processeurs.

3. un taux d'indisponibilité noté par $\gamma(t)$ qui représente par exemple la durée nécessaire à la réparation d'un composant de l'élément de calcul. Typiquement les applications HPC fonctionnent seulement si la totalité des processeurs sont opérationnels. Ainsi si un processeur est indisponible l'élément de calcul est considéré comme étant indisponible.

Par conséquent sans la présence de pannes, le temps de complétion noté par ω d'une application composée de w unités de travail sur un élément de calcul est une quantité déterministe donnée par :

$$\omega = w/\nu.$$

Ceci est illustré sur la Figure 3.1(a), où le temps de complétion de l'application sur l'axe des abscisses est proportionnel à la quantité de travail résiduel dans l'application sur l'axe des ordonnées et qui décroit selon la pente $-\nu$.



(a) modèle d'exécution sans pannes

(b) modèle d'exécution avec pannes

FIGURE 3.1 – modèle d'exécution sans panne (a) avec pannes (b)

Par contre, dans le cas d'une exécution réelle, des pannes surviennent nécessairement pendant l'exécution ce qui augmente le temps de complétion de l'application, comme le montre la Figure 3.1(b). Usuellement dans le cadre des applications HPC, la panne cause nécessairement la perte de tout le travail fait depuis le début de l'exécution. Nous soulignons que cette hypothèse est vraie pour la majorité des applications de calcul scientifique. Dans ce travail nous appelons cette durée de temps perdu par la quantité de travail perdu notée par L (*Lost work*). Ensuite, une fois interrompue par la panne, l'exécution de l'application ne pourra reprendre qu'après une durée de temps D où l'élément de calcul est indisponible. Une fois que l'élément de calcul est de nouveau disponible, l'exécution de l'application redémarre depuis le début. Notons que dans ce travail nous supposons que le temps de redémarrage nécessaire pour la reprise de l'application après la panne est équivalent au temps de déploiement initial de

l'application donc qu'il est inclus par défaut dans ω.
Ce processus composé des trois étapes : exécution, interruption et redémarrage, est répété tant qu'il n'y a pas un intervalle de disponibilité plus grand que le temps nécessaire pour finir l'application sans aucune interruption de panne. Formellement, soient U et D deux $v.a.r$, tel que U_i est la durée de l'intervalle de disponibilité avant la $i^{ème}$ panne et D_i est la durée d'indisponibilité après la $i^{ème}$. Dans ce cas suite à la $i^{ème}$ panne la quantité de travail perdu est exactement U_i et D_i unités de temps sont rajoutées au temps de complétion. Supposons que le $S^{éme}$ intervalle inter-pannes est plus grand que ω. Le temps de complétion de l'application en présence des pannes noté par $T(\omega)$ est donné par l'expression suivante :

$$T(\omega) = \sum_{i=1}^{S-1} (U_i + D_i) + \omega. \tag{3.1}$$

En se basant sur cette modélisation , nous exprimons dans la section suivante l'espérance du temps de complétion ainsi que la distribution du nombre de re-exécutions en fonction de la distribution des pannes, la distribution d'indisponibilité et la quantité de travail initial à exécuter.

3.2.2 Distribution du nombre de re-exécutions

Dans cette section nous donnons une expression analytique de la distribution du nombre de re-exécutions en fonction de la quantité de travail à exécuter ainsi que du taux de panne. Soit S_ω le nombre de re-exécutions nécessaire pour exécuter les w unités de travail. À partir de l'abstraction présentée précédemment. L'événement $\{S_\omega = s\}$ est équivalent à l'événement $\{U_1 < \omega, U_2 < \omega, \cdots, U_{s-1} < \omega, U_s > \omega\}$. En conséquence, S_ω est une $v.a.d$ (variable aléatoire discrète) distribuée selon une loi géométrique de paramètre q_ω donné par :

$$q_\omega = \mathbb{P}\{U > \omega\} = \overline{F}(\omega).$$

La densité ainsi que la distribution du nombre de re-exécutions sont exprimées de la manière suivante :

$$\mathbb{P}\{S_\omega = s\} = (1 - q_\omega)^{s-1} q_\omega,\ \mathbb{P}\{S_\omega \leq s\} = 1 - (1 - q_\omega)^s. \tag{3.2}$$

À partir de la distribution S_ω tous les moments peuvent être calculés. Par exemple, l'espérance et la variance du nombre de re-exécutions sont exprimées ci-dessous.

$$\mathbb{E}[S_\omega] = \frac{1}{q_\omega},\ \mathrm{Var}[S_\omega] = \frac{1 - q_\omega}{q_\omega^2}. \tag{3.3}$$

Supposons que la distribution d'inter-arrivées des pannes suit un processus de Poisson avec un paramètre λ. L'espérance et la variance du nombre de re-exécutions sont données par :

$$\mathbb{E}[S_\omega] = \exp^{\lambda\omega},\ \mathrm{Var}[S_\omega] = \exp^{2\lambda\omega} - \exp^{-\lambda\omega}. \tag{3.4}$$

En utilisant l'Expression 3.4, nous traçons sur la Figure 3.2 la variation de l'espérance du nombre re-exécutions en fonction de la quantité de travail initial pour différentes valeur du taux de panne. Nous tenons à signaler que le taux de traitement de l'élément

de calcul est donné par $\nu = 1$. Cette figure reporte trois configurations de variation de l'espérance du nombre de re-exécutions en fonction de la quantité de travail initial qui varie entre 1 jour et 4 jours. Nous tenons à signaler que le taux de traitement de l'élément de calcul est une unité de travail par unité de temps ($\nu = 1$). D'une configuration à une autre nous faisons varier le MTTF [1] de l'élément de calcul entre les trois valeurs suivantes : 2 jours ($\lambda(t) = 1/2$), 1.5 ($\lambda(t) = 2/3$) jours et 1 jour ($\lambda(t) = 1$). Cette figure et l'Expression 3.4 montent très bien que l'espérance du nombre de re-exécutions a une croissance exponentielle par rapport à la croissance de la quantité de travail dans l'application ou de la croissance du taux de panne. En se basant sur ce critère de performance, nous montrons par la suite dans la Section 3.5 qu'il est possible de réduire cette croissance en une croissance linéaire moyennant un mécanisme de tolérance aux pannes.

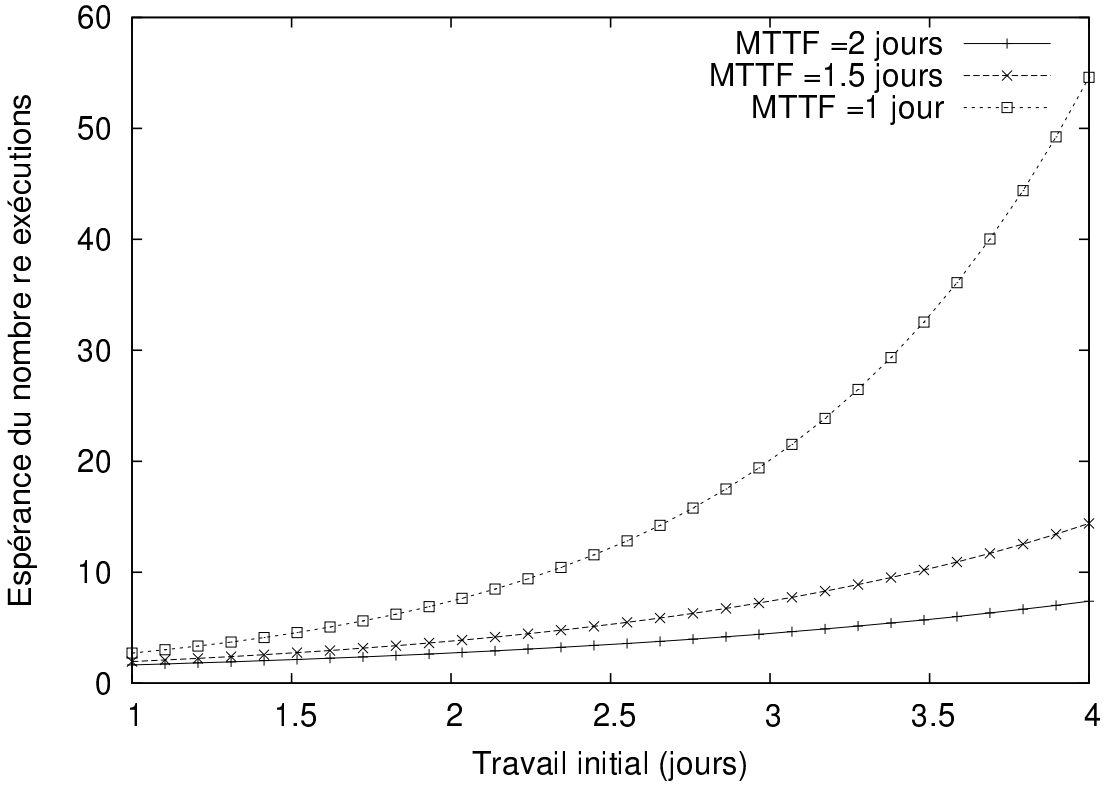


FIGURE 3.2 – L'espérance du nombre de re-exécutions

3.2.3 Espérance du temps de complétion

Nous présentons maintenant l'expression de l'espérance du temps de complétion nécessaire pour exécuter les w unités de travail. Cette expression permet d'illustrer la variation de l'espérance du temps de complétion par rapport à la quantité de travail initial, le taux de panne et le taux d'indisponibilité.

1. *Mean Time To Failure*

Considérons l'Expression 3.1 qui exprime le temps de complétion d'une manière générale. L'espérance du temps de complétion peut alors être exprimée comme suit :

$$\mathbb{E}\left[T(\omega)\right] = \mathbb{E}\left[\left(\sum_{i=1}^{S_\omega} U_i + D_i\right) + \omega - U_{S_\omega} - D_{S_\omega}\right]. \tag{3.5}$$

Par indépendance nous obtenons :

$$\mathbb{E}\left[T(\omega)\right] = \mathbb{E}\left[\sum_{i=1}^{S_\omega} U_i + D_i\right] + \omega - \mathbb{E}\left[U_{S_\omega}\right] - \mathbb{E}\left[D_{S_\omega}\right]. \tag{3.6}$$

Telle que, U_{S_ω} est une $v.a.r$ qui a la même distribution de U conditionnée par l'événement $\{U > \omega\}$, ceci se traduit aussi par $\mathbb{P}\{U_{S_\omega} = u\} = \mathbb{P}\{U = u | U > \omega\}$. L'espérance mathématique de cette $v.a.r$ conditionnelle est donc donnée par :

$$\mathbb{E}\left[U_{S_\omega}\right] = \frac{1}{\overline{F}(\omega)} \int_\omega^\infty x f(x) dx. \tag{3.7}$$

Par contre, la $v.a.r$ D_{S_ω} a la même distribution que D puisqu'elle est totalement indépendante de la réalisation de la $v.a.r$ S_ω. Donc nous obtenons $\mathbb{E}\left[D_{S_\omega}\right] = \mathbb{E}\left[D\right]$. Finalement, en utilisant le théorème de Wald [96] la somme aléatoire jusqu'à S_ω des $v.a.r$ U_i et D_i est donnée comme suit :

$$\mathbb{E}\left[\sum_{i=1}^{S_\omega} U_i + D_i\right] = \mathbb{E}\left[S_\omega\right]\left(\mathbb{E}\left[U\right] + \mathbb{E}\left[D\right]\right). \tag{3.8}$$

En injectant les Expressions 3.7 et 3.8 dans l'Expressions 3.6 l'espérance du temps de complétion est exprimée ainsi :

$$\mathbb{E}\left[T(\omega)\right] = \frac{1}{\overline{F}(\omega)}\left(F(\omega)\mathbb{E}\left[D\right] + \omega\overline{F}(\omega) + \int_0^\omega x f(x) dx\right).$$

$$\mathbb{E}\left[T(\omega)\right] = \frac{1}{\overline{F}(\omega)}\left(F(\omega)\mathbb{E}\left[D\right] + \omega\overline{F}(\omega) + \omega F(\omega) - \int_0^\omega (1 - \overline{F}(\omega)) dx\right).$$

$$\mathbb{E}\left[T(\omega)\right] = \frac{1}{\overline{F}(\omega)}\left(F(\omega)\mathbb{E}\left[D\right] + \int_0^\omega \overline{F}(x) dx\right). \tag{3.9}$$

Nous soulignons que cette expression a une forme analytique pour la plupart des distributions de panne comme par exemple pour la distribution exponentielle. Sinon dans le cas où l'intégrale de $\overline{F}(x)$ n'admet pas de forme analytique nous proposons deux encadrements (inférieur et supérieur). Pour exprimer la borne inférieure donnée ci-dessous, nous nous basons sur la propriété de décroissance de $\overline{F}(x)$.

$$\frac{F(\omega)}{\overline{F}(\omega)}\mathbb{E}\left[D\right] + \omega \leq \mathbb{E}\left[T(\omega)\right]. \tag{3.10}$$

Toutefois, cette borne représente un modèle de panne idéal où la quantité de travail qui est faite entre deux intervalles de pannes n'est pas perdu. En conséquence, seul le terme $\frac{F(\omega)}{\overline{F}(\omega)}\mathbb{E}\left[D\right]$ reste, puisqu'il représente le surcoût lié à la durée totale d'indisponibilité. Par

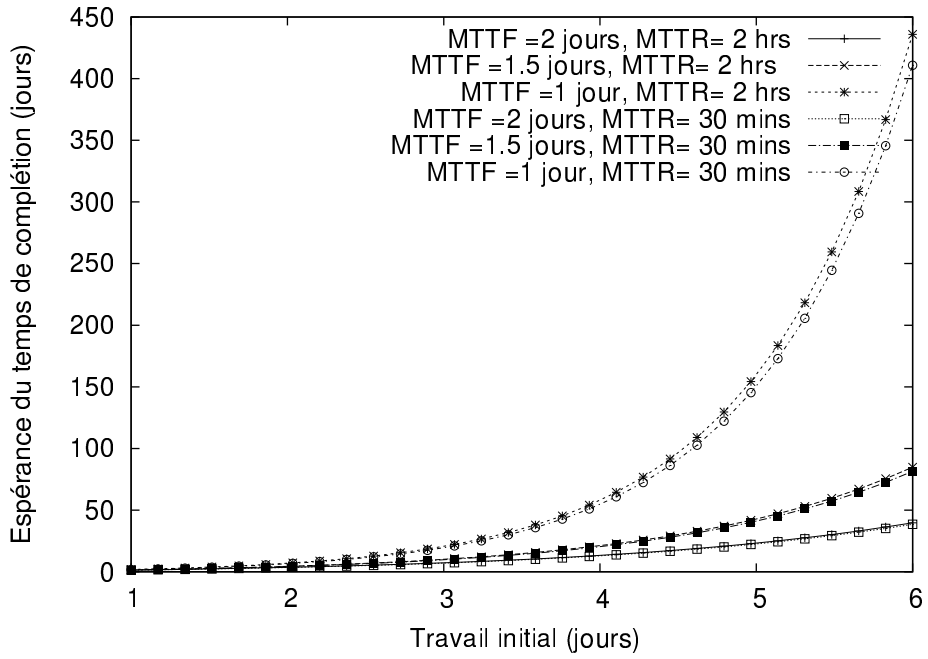


FIGURE 3.3 – Variation de l'espérance du temps de complétion en fonction de la quantité de travail initial, le taux de panne et d'indisponibilité

contre, pour exprimer la borne supérieure nous utilisons l'identité $\mathbb{E}[U] = \int_0^\infty \overline{F}(x)dx$. En conséquence l'espérance du temps de complétion est bornée supérieurement par :

$$\mathbb{E}[T(\omega)] \leq \frac{1}{\overline{F}(\omega)}(F(\omega)\mathbb{E}[D] + \mathbb{E}[U]) \leq \frac{\mathbb{E}[D] + \mathbb{E}[U]}{\overline{F}(\omega)}. \tag{3.11}$$

Considérons par exemple un processus de Poisson pour modéliser à la fois la distribution de panne avec un taux λ et la distribution d'indisponibilité avec un taux γ. L'espérance du temps de complétion est simplifiée comme le montre l'Expression 3.12. En utilisant cette expression nous traçons dans la Figure 3.3 la variation de l'espérance du temps de complétion en jours pour trois configurations de MTTF et deux configurations de MTTR [2].

$$\mathbb{E}[T(\omega)] = (\exp^{\lambda\omega} - 1)(\frac{1}{\lambda} + \frac{1}{\gamma}). \tag{3.12}$$

3.2.4 Bilan

Nous notons que la Figure 3.3 et l'Expression 3.12 de l'espérance du temps de complétion ainsi que les bornes inférieures et supérieures du temps de complétion, montrent très clairement que l'espérance du temps de complétion a une croissance exponentielle dans le cas de la distribution exponentielle que se soit en fonction de w ou en fonction de λ et ceci est lié au terme $\frac{F(\omega)}{\overline{F}(\omega)}$.

La méthode la plus simple qui permet de réduire cette croissance exponentielle en une croissance linéaire est la division de l'application en des parties indépendantes.

2. *Mean Time To Repair*

Formellement, supposons que nous pouvons fractionner l'application en n parties ρ_i tel que ces parties vérifient $\sum_{i=1}^{n} \rho_i = \omega$. Supposons que l'opération de fractionnement n'engendre aucun surcoût dans le temps de complétion de l'application. Donc par indépendance, l'espérance du temps de complétion et l'espérance du nombre de re-exécutions sont données dans les deux expressions suivantes dans le cas où $\lambda(t) = \lambda$ et $\gamma(t) = \gamma$:

$$\mathbb{E}\left[T(\omega)\right] = \sum_{i=1}^{n} \mathbb{E}\left[T(\rho_i)\right] = \left(\sum_{i=1}^{n} \exp^{\lambda\rho_i} - 1\right)\left(\frac{1}{\lambda} + \frac{1}{\gamma}\right) \tag{3.13}$$

$$\mathbb{E}\left[S_\omega\right] = \sum_{i=1}^{n} \mathbb{E}\left[S_{\rho_i}\right] = \sum_{i=1}^{n} \exp^{\lambda\rho_i}. \tag{3.14}$$

Ainsi, pour un n suffisamment grand qui vérifie $\lambda\omega/n \approx 0$. La croissance de l'espérance du temps de complétion et de l'espérance du nombre de re-exécutions deviennent linéaires en fonction de w ou λ puisque $\exp^{\lambda\omega/n} - 1 \approx \lambda\omega/n$. Dans la Section 3.3 nous montrons que moyennant la technique de sauvegarde et reprise nous pouvons appliquer cette idée, si ce n'est que chaque sauvegarde a un surcoût. Donc, il faut bien gérer le compromis entre rajouter d'avantage de sauvegardes pour réduire l'impact des pannes tout en rajoutant le minimum de surcoût de sauvegarde dans le temps de complétion de l'application.
Dans le reste de ce chapitre, nous formalisons analytiquement la fonction objectif qui modélise le compromis entre le surcoût de sauvegarde et le travail perdu à cause des pannes.

3.3 Modélisation avec tolérance aux pannes

Apres avoir présenté dans la Section 3.2 le modèle de performance sans mécanisme de tolérance aux pannes, dans cette section, nous couplons ce modèle avec le modèle du mécanisme de sauvegarde et reprise coordonnée. Ceci va permettre à la fois de quantifier l'apport du mécanisme de tolérance aux pannes et de servir comme modèle de performance pour exprimer le compromis entre l'impact des pannes sur la quantité de travail perdu et l'impact du surcoût accumulé dû aux points de sauvegarde.

Nous présentons dans un premier temps le modèle adopté pour modéliser le mécanisme de sauvegarde et reprise. Ensuite, nous montrons comment ce mécanisme améliore l'espérance du temps de complétion de l'application. Finalement nous décrivons, la stratégie d'ordonnancement des points de sauvegarde pour optimiser les critères de performance considérés.

3.3.1 Modèle du mécanisme de sauvegarde et reprise

En se basant sur la présentation du mécanisme de sauvegarde et reprise coordonnée dans la Section 2.4.3.a nous exhibons dans cette partie la description formelle adoptée pour mécanisme. Cette description contient essentiellement les trois points suivants. En premier lieu nous donnons la description formelle utilisée pour décrire une stratégie d'ordonnancement des points de sauvegarde. Ensuite nous décrivons le modèle du surcoût

des points de sauvegarde. Finalement, nous présentons les hypothèses sous-jacentes de cette abstraction.

Considérons une application qui contient w unités de travail. Sans perte de généralité supposons que $\nu = 1$. Rajouter des points de sauvegarde consiste à fractionner cette quantité de travail en n parties indépendantes qui seront exécutées dans l'ordre. Techniquement ceci implique que dans le cadre de la modélisation continue, l'exécution de l'application peut être préemptée à n'importe quel instant pour réaliser le point de sauvegarde. Comme nous l'avons indiqué dans le chapitre préliminaire, cette hypothèse est valide pour les applications qui sont couplées avec les mécanismes de sauvegarde et reprise implémentés au niveau système. Les frontières entre les n parties sont décrites dans une stratégie d'ordonnancement des points de sauvegarde notée par π. Deux méthodes de notation équivalentes sont utilisées dans ce travail pour décrire formellement π. Dans la première méthode, la stratégie d'ordonnancement des points de sauvegarde est décrite sous forme d'instants absolus où la quantité de travail absolu faite depuis le début est utilisée pour décrire les instants des points de sauvegarde. Dans ce cas π contient les instants absolus notés par $\{\tau_i\}_{i \leq n}$, où les points de sauvegarde sont déclenchés comme le montre la Figure 3.4. Plus précisément, soit $\Pi \subset \mathbb{R}_+^n$ l'ensemble des stratégies d'ordonnancement des points de sauvegarde valides, une stratégie d'ordonnancement des points de sauvegarde qui contient n points de sauvegarde $\pi \in \Pi$ vérifie nécessairement les contraintes suivantes :

$$\pi \in \Pi \text{ si et seulement si } \begin{cases} \forall (\tau_i, \tau_j) \in \pi,\ i < j \Rightarrow \tau_i < \tau_j, \\ \tau_n = \omega. \end{cases} \tag{3.15}$$

La première contrainte est intuitive puisque les τ_i sont des dates absolues dans le temps. Donc ils sont strictement positifs et ne se chevauchent pas. La dernière contrainte implique que forcément un point de sauvegarde est réalisé à la fin de l'exécution. Nous notons que ceci n'affecte pas la portée, ni la généralité de notre étude puisque ce dernier point de sauvegarde peut être considéré comme fictif et son surcoût comme nul.
La deuxième notation est à base d'intervalle, où nous utilisons la quantité de travail faite depuis le dernier point de sauvegarde pour décrire l'instant du prochain point de sauvegarde. Dans cette notation les intervalles inter sauvegarde sont notés par la séquence $\{\rho_i\}_{i \leq n}$. Nous tenons à souligner que les deux notations sont parfaitement équivalentes et reliées par la relation suivante :

$$\forall i \text{ tel que, } 1 \leq i \leq n, \text{ nous avons } \tau_i - \tau_{i-1} = \rho_i, \text{ avec } \tau_0 = 0 \text{ et } \sum_{i=1}^{n} \rho_i = \tau_n. \tag{3.16}$$

Nous entamons maintenant la description du modèle du surcoût associé aux points de sauvegarde. Bien évidemment, la durée du temps passé dans la phase de sauvegarde est considérée comme un surcoût qui est rajouté dans le temps de complétion global. Ceci est schématisé dans la Figure 3.4, tel que pendant cette phase, le temps global sur l'axe des abscisses augmente tandis que la quantité de travail ne décroit pas pendant cette durée. À notre connaissance, la totalité des travaux de modélisation existants se limitent à la considération d'un surcoût constant pour la modélisation des durées du temps passé dans la phase de sauvegarde. Dans ce travail, nous considérons que le surcoût d'un point de sauvegarde est variable et il dépend de l'état de l'avancement de l'application à cet instant. Cette variation de coût peut avoir plusieurs raisons. Par

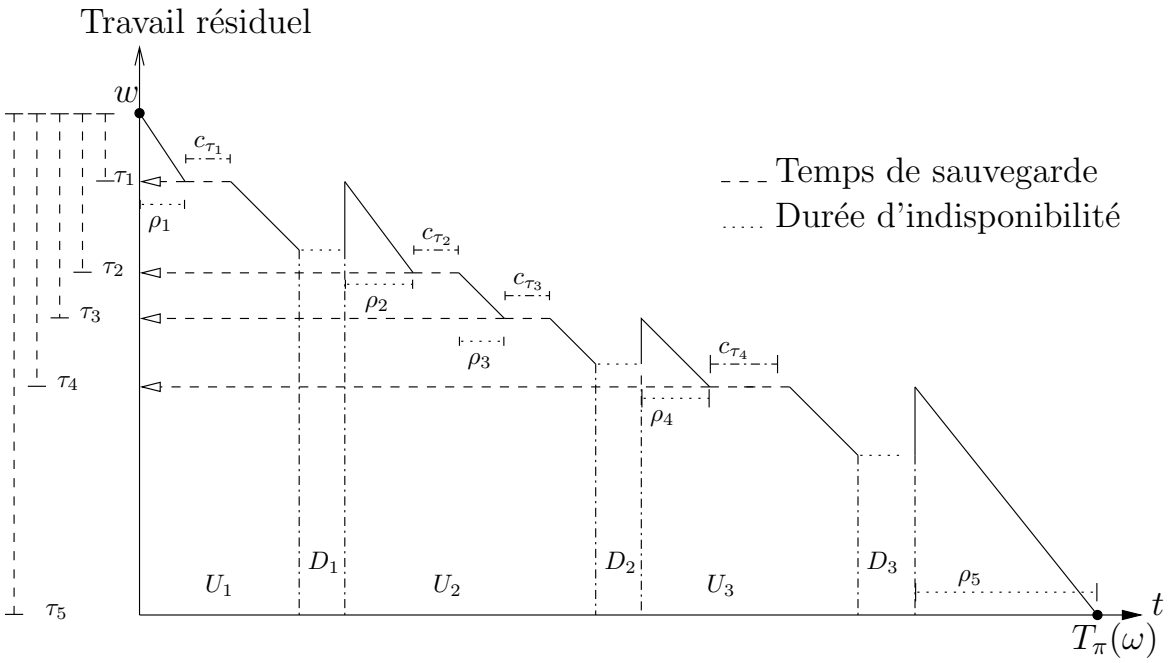


FIGURE 3.4 – Schéma d'exécution avec sauvegarde et reprise

exemple dans le contexte des applications HPC exécutées sur la plateforme BlueGene/P à *Argone National Laboratory* avec un mécanisme de sauvegarde et reprise coordonnée, Gupta *et al.* [52] indiquent que la variation des surcoûts induits par les points de sauvegarde dépend des points suivants :

- La quantité de mémoire totale utilisée par l'application à l'instant de sauvegarde. L'étude menée dans [52] indique que dans le cadre des applications HPC, la quantité de mémoire utilisée est en lien direct avec le nombre de processus impliqués dans le calcul et le volume des données d'entrée.
- Les performances I/O effectives entre les nœuds de calcul et la technique et l'architecture du support stable distribué utilisé pour stocker les sauvegardes.

Généralement les utilisateurs des applications peuvent estimer à l'avance la quantité de mémoire utilisée et assez souvent, ils utilisent une baie de stockage localisée physiquement dans le même cluster comme étant un support de stockage stable. Par exemple ceci est le cas des utilisateurs de la plateforme BlueGene/P. Ainsi, dans ce travail nous supposons que tous ces paramètres sont estimés à l'avance par l'utilisateur et encapsulés dans une seule fonction qui renvoie le surcoût du point de sauvegarde en fonction de la date de sauvegarde. Par convention cette fonction est notée par c_{τ_i} et elle représente le surcoût du $i^{ème}$ point de sauvegarde à l'instant τ_i. Dans le cas où c_τ est constante pour tout τ, le surcoût est noté par c.

Finalement comme hypothèses sous-jacentes, nous supposons que des pannes peuvent survenir également pendant la phase de sauvegarde. Nous considérons aussi que la durée de temps passée dans les phases de sauvegarde dégrade l'état du système et par conséquent, elle augmente nécessairement la probabilité de panne.

3.3.2 Formulation du problème d'optimisation

Ayant présenté précédemment le modèle de l'application, nous décrivons maintenant le modèle de la plateforme ainsi que le modèle du mécanisme de sauvegarde et reprise. Dans cette partie nous rassemblons ces différentes pièces du puzzle pour formaliser le compromis entre la quantité de travail perdu à cause des pannes versus le surcoût accumulé des points de sauvegarde. La méthode retenue dans ce chapitre pour exprimer ce compromis est de le formaliser à travers l'expression de l'espérance du temps de complétion. Comme première étape dans cette partie, nous présentons l'expression de l'espérance du temps de complétion en fonction des différents paramètres à savoir : le modèle de l'application, le modèle de la plateforme, le modèle de sauvegarde et la stratégie d'ordonnancement des points de sauvegarde. Ensuite, nous montrons comment cette formulation est utilisée pour définir formellement le problème d'optimisation sous contraintes. L'objectif est alors de calculer la stratégie d'ordonnancement optimale des points de sauvegarde qui minimise l'espérance du temps de complétion. Finalement, nous proposons une étude de cas, dans laquelle nous exhibons analytiquement la stratégie d'ordonnancement optimale des points de sauvegarde sous l'hypothèse du processus de Poisson et surcoût de sauvegarde constant.

a Espérance du temps de complétion avec sauvegarde et reprise

Pour exprimer l'espérance du temps de complétion, nous rappelons avant toute chose que l'insertion des points de sauvegarde selon une stratégie d'ordonnancement des points de sauvegarde revient à fractionner l'exécution de l'application en n parties. Ces parties sont exécutées une après l'autre d'une manière séquentielle, par contre en cas de panne seule la partie en cours d'exécution est perdue. Dans ce cas, l'espérance du temps de complétion de chaque partie est indépendante des autres et elle est donnée par l'Expression 3.9. En conséquence, l'espérance du temps de complétion de toute l'application est la somme des n termes.
Par contre, nous mentionnons que l'Expression 3.9 exprime l'espérance du temps de complétion de w unités de travail sachant que l'exécution commence à chaque fois à $t = 0$ autrement dit, quand l'âge de l'élément de calcul est 0 unité de temps. Or, il se peut que plusieurs fractions soient exécutées dans un seul cycle de panne. Donc, pour une partie quelconque qui ne commence pas au début du cycle (à $t = 0$), l'âge de l'élément de calcul est égal à la quantité de temps qu'il a fallu pour exécuter les autres fractions avec leurs sauvegardes respectives. Par conséquent pour adapter l'Expression 3.9 il suffit de conditionner la distribution de panne $F(x)$ par l'événement "l'élément de calcul est en vie depuis t unités de temps". Formellement, l'espérance du temps de complétion de la $i^{éme}$ partie ρ_i avec son point de sauvegarde c_{τ_i} est donnée par :

$$\mathbb{E}\left[T(\rho_i + c_{\tau_i})\right] = \frac{1}{\overline{F}_{a_i}(\rho_i + c_{\tau_i})}\left(F_{a_i}(\rho_i + c_{\tau_i})\mathbb{E}\left[D\right] + \int_0^{\rho_i + c_{\tau_i}} \overline{F}_{a_i}(x)dx\right), \tag{3.17}$$

où $a_i = \tau_{i-1} + \sum_{j=1}^{i-1} c_{\tau_j}$ est l'âge de l'élément de calcul (par convention $a_1 = 0$) et $F_{a_i}(x)$ est la distribution conditionnelle donnée par $F_{a_i}(x) = (F(x + a_i) - F(a_i))/\overline{F}(a_i)$. Nous tenons à préciser que l'Expression 3.9 est utilisable telle qu'elle dans le cas où le taux de panne est constant puisque la distribution exponentielle est sans mémoire. Donc en

se basant sur ces hypothèses, l'espérance du temps de complétion de l'application en fonction de π est exprimée de la manière suivante :

$$\mathbb{E}\left[T_\pi(w)\right] = \sum_{i=1}^{n} \mathbb{E}\left[T(\tau_i - \tau_{i-1} + c_{\tau_i})\right] = \sum_{i=1}^{n} \mathbb{E}\left[T(\rho_i + c_{\tau_i})\right]. \tag{3.18}$$

Nous tenons á signaler que cette expression de la quantité de travail résiduel et de la durée de vie du support d'exécution.

b Problème d'optimisation sous contraintes

Partant de l'Expression 3.18 qui exprime l'espérance du temps de complétion en fonction de la stratégie d'ordonnancement des points de sauvegarde π, nous formalisons dans cette partie le problème d'optimisation sous contraintes pour calculer la stratégie d'ordonnancement optimale des points de sauvegarde π^* qui minimise l'espérance du temps de complétion. Ce problème d'optimisation est défini formellement comme suit. Soit $\Psi : \mathbb{R}^n \rightarrow \mathbb{R}$ la fonction objectif et $\Pi \neq \emptyset$ un sous ensemble de $\mathbb{R}^n$ qui contient les solutions valides du problème appelé aussi ensemble des contraintes. Ce problème de minimisation est décrit formellement par :

$$\min_{\pi \in \mathbb{R}^n} \Psi(\pi) \text{ sous la contrainte } \pi \in \Pi. \tag{3.19}$$

En injectant l'Expression 3.17 de l'espérance du temps de complétion en fonction des unités de travail dans l'Expression 3.18, la définition de Ψ dans notre cas est donnée par :

$$\Psi(\pi) = \sum_{i=1}^{n} \left(\frac{\left(F_{a_i}(\tau_i - \tau_{i-1} + c_{\tau_i})\mathbb{E}\left[D\right] + \int_0^{\tau_i - \tau_{i-1} + c_{\tau_i}} \overline{F}_{a_i}(x)dx\right)}{\overline{F}_{a_i}(\tau_i - \tau_{i-1} + c_{\tau_i})} \right) \tag{3.20}$$

La définition de l'ensemble des contraintes Π est donnée dans l'Expression 3.15.

3.3.3 Résolution du problème d'optimisation

Deux méthodologies sont envisageables pour résoudre le problème décrit ci-dessus. La première consiste à résoudre le problème analytiquement en se basant sur des propriétés de type convexité ou autre. Mais ceci n'est pas toujours possible et parfois une solution analytique n'est pas aussi évidente à trouver. Dans ce cas nous pouvons utiliser les méthodes d'optimisation numérique.

a Solution analytique

Dans cette partie nous présentons une étude de cas dans laquelle nous montrons comment une stratégie d'ordonnancement des points de sauvegarde peut être exprimée analytiquement dans un cas ciblé. Ce cas est la configuration où le taux de panne ainsi que le surcoût de sauvegarde sont constants.

Ainsi, partant de la définition de Ψ dans l'Expression 3.20, l'espérance du temps de complétion en fonction de π dans ce cas est donnée par :

$$\mathbb{E}\left[T_\pi(w)\right] = \sum_{i=1}^{n} (\exp^{\lambda(\rho_i + c)} - 1)(\mathbb{E}\left[D\right] + 1/\lambda). \tag{3.21}$$

En se basant sur ces hypothèses nous soulignons que la fonction objectif obtenue est une fonction continue sur le domaine Π, d'ou l'existence de π^*. Partant de cette fonction objectif, nous démontrons que dans le cas où le taux de panne ainsi que le surcoût des points sauvegarde sont constants, il existe forcément une stratégie π^* dite périodique qui minimise l'espérance du temps de complétion et en plus π^* est un optimal global.

Théorème 3.1. *Si $\lambda(t) = \lambda$ et $c_\tau = c$, la stratégie d'ordonnancement des points de sauvegarde optimale π^* est périodique.*

Démonstration. Considérons une fonction $\Psi : \mathbb{R}^n \to \mathbb{R}$ définie par :

$$\Psi : \pi \to \kappa \sum_{i=1}^{n} \psi(\rho_i).$$

où κ est une constante positive et $\psi : \mathbb{R} \to \mathbb{R}$ est une fonction arbitraire.
Par définition Ψ est symétrique c'est-à-dire qu'elle est invariante par rapport à une rotation circulaire des variables :

$$\begin{aligned}\Psi(\rho_1, \rho_2, \cdots, \rho_n) \\ = \frac{\Psi(\rho_1, \rho_2, \cdots, \rho_n) + \Psi(\rho_n, \rho_1, \cdots, \rho_{n-1}) + \cdots + \Psi(\rho_2, \rho_3, \cdots, \rho_1)}{n}.\end{aligned} \tag{3.22}$$

Supposons maintenant que ψ est une fonction convexe[3] sur $\mathbb{R}$ ceci implique directement que Ψ est aussi une fonction convexe sur $\mathbb{R}^n$ puisque $\kappa \geq 0$, d'où :

$$\begin{aligned}\frac{1}{n}\left(\Psi(\rho_1, \rho_2, \cdots, \rho_n) + \Psi(\rho_n, \rho_1, \cdots, \rho_{n-1}) + \cdots + \Psi(\rho_2, \rho_3, \cdots, \rho_1)\right) \\ \geq \Psi\left(\frac{\rho_1 + \rho_2 + \cdots + \rho_n}{n}, \frac{\rho_n + \rho_1 + \cdots + \rho_{n-1}}{n}, \frac{\rho_2 + \rho_3 + \cdots + \rho_1}{n}\right).\end{aligned} \tag{3.23}$$

Ainsi en considérant $\psi(x) = \exp^{\lambda(x+c)} - 1$, $\kappa = \mathbb{E}[D] + 1/\lambda$ le résultat est immédiat puisque 3.22 et 3.23 impliquent :

$$\forall \pi \in \Pi,\ \Psi(\rho_1, \rho_2, \cdots, \rho_n) \geq \Psi\left(\frac{\omega}{n}, \frac{\omega}{n}, \cdots, \frac{\omega}{n}\right).$$

D'où :

$$\rho_1^* = \rho_2^* = \cdots = \rho_n^* = \frac{\omega}{n} = \rho^*$$

□

Le Théorème 3.1 implique que la valeur optimale de chaque intervalle ρ_i est égale à $\frac{\omega}{n}$. Pour définir complètement π^* il faut déterminer $\frac{\omega}{n}$ qui minimise l'espérance du temps de complétion. En s'appuyant sur ce théorème, l'expression de l'espérance du temps de complétion est exprimée en fonction de ρ comme suit :

$$\mathbb{E}[T_\rho(w)] = \frac{\omega}{\rho}(\exp^{\lambda(\rho+c)} - 1)(\mathbb{E}[D] + \frac{1}{\lambda}). \tag{3.24}$$

Nous soulignons que cette expression de l'espérance du temps de complétion est une fonction dérivable par rapport à ρ. Ainsi, l'intervalle optimal de sauvegarde vérifie la condition d'optimalité du premier ordre.

3. Une définition de la convexité est rappelée dans la Section 6.3 de l'annexe

Théorème 3.2. *Si $\lambda(t) = \lambda$ et $c_\tau = c$ la stratégie d'ordonnancement des points de sauvegarde π^* est périodique et la période optimale est $\rho^* = (1 + \mathcal{W}(-\exp^{-1-\lambda c}))/\lambda$ où $\mathcal{W}$ est la fonction de Lambert.*

Démonstration. Nous notons que $\mathbb{E}\left[T_\rho(w)\right]$ est dérivable par rapport à ρ, donc la valeur qui la minimise vérifie la condition d'optimalité du premier ordre, alors :

$$\frac{d\mathbb{E}\left[T_{\rho^*}(w)\right]}{d\rho} = 0$$

$$\left(1 + \exp^{\lambda(\rho^*+c)}(\rho^*\lambda - 1)\right)\frac{\omega}{(\rho^*)^2}(\mathbb{E}\left[D\right] + \frac{1}{\lambda}) = 0$$

Nous précisons que $\frac{\omega}{(\rho^*)^2}(\mathbb{E}\left[D\right] + \frac{1}{\lambda}) > 0$ alors :

$$\exp^{\lambda\rho^*-1}(\rho^*\lambda - 1) = -\exp^{-\lambda c-1}.$$

En utilisant la fonction Lambert [34] notée par $\mathcal{W}(x)$ nous obtenons :

$$\rho^* = \frac{1 + \mathcal{W}(-\exp^{-\lambda c-1})}{\lambda}.$$

□

Nous tenons à préciser que la fonction $\mathcal{W}(x)$ est utilisée pour résoudre l'identité suivante :

$$x = y\exp^y \iff y = \mathcal{W}(x).$$

Cette équation admet une solution unique si et seulement si $x \geq -\frac{1}{e}$ ce qui est le cas du terme $-\exp^{-1-\lambda c}$.

Pour clôturer cette partie d'étude de cas, nous notons que ces deux derniers résultats démontrent que, dans le cas où le taux de panne et le surcoût de sauvegarde sont constants, la stratégie d'ordonnancement des points de sauvegarde optimale est forcément périodique et elle est donnée analytiquement par le Théorème 3.2. Nous tenons à souligner que ce résultat publié dans [17] est une contribution importante puisque plusieurs travaux présentés dans l'état de l'art sont basés sur cette hypothèse sans aucune démonstration.

b Solution numérique

Malheureusement, une solution analytique exacte n'est pas toujours évidente à trouver. La deuxième approche qui reste toujours réalisable est la résolution numérique. Dans ce cas, avant d'entamer le calcul de la solution optimale, la première question à adresser est celle de l'existence d'un minimum global $\pi^* \in \Pi$ qui minimise la fonction objectif 3.20. Pour prouver l'existence de π^* dans ce travail nous nous basons sur le théorème des bornes atteintes (théorème de Weierstrass). Ce théorème indique que si la fonction est continue sur le domaine des contraintes et l'ensemble des contraintes est un borné fermé (*i.e.* compact) alors la fonction est bornée et atteint ses bornes. Par définition Π est un compact alors la condition suffisante pour que π^* existe est la continuité de Ψ. La continuité de la fonction objectif Π dépend de la définition des

fonctions $F(x)$ et c_τ. Les distributions de pannes typiques ne posent pas de problème de continuité. Concernant la fonction c_τ nous pouvons supposer qu'elle est exprimée par un polynôme. Si c'est le cas, nous pouvons conclure que dans la quasi totalité des cas π^* existe.
Une fois l'existence de π^* vérifiée la deuxième étape est de le calculer avec une méthode de résolution numérique. Dans notre cas l'ensemble des contraintes Π est un ensemble convexe. Ceci implique que la plupart des techniques basées sur la descente de gradient projeté [84] fonctionnent correctement dans la majorité des cas à condition que Ψ soit différentiable. Plusieurs techniques de résolution numérique sont disponibles. Cependant, le choix de la technique de minimisation et ses paramètres tel que le pas de descente est un art à part entière. Ceci implique qu'une solution générique n'existe pas mais il faut choisir une technique de résolution en fonction de $F(x)$ et c_τ. Dans cette thèse deux codes de résolution ont été développés. Le premier solveur [4] est codé en C et repose sur les routines de résolution numérique fournies par GSL [47]. Tandis que le deuxième repose entièrement sur la routine d'optimisation *trust-region-reflective algorithm* [33] disponible dans *Optimization Toolbox* de MATLAB. Cependant la bonne configuration des deux solveurs n'est pas du tout évidente. En effet dans les deux cas lors de la résolution du problème nous rencontrons des problèmes de stabilité numérique. Néanmoins quelques cas intéressants ont été résolus avec ces deux approches, tous les détails de résolution sont disponibles [16] [15]. Un autre inconvénient principal est à signaler. En effet, pour résoudre numériquement le problème il faut impérativement fixer n en avance. Ceci implique que la solution optimale calculée dépend du choix de n.

Pour résumer cette méthodologie nous rappelons qu'avant toute chose il faut vérifier impérativement que Ψ est différentiable. Ensuite choisir un algorithme de minimisation numérique en fonction de $F(x)$ et c_{τ_i} pour minimiser la fonction Ψ pour un n fixé. Finalement, injecter dans le solveur la fonction à optimiser Ψ définie dans 3.20 et les contraintes de résolution exprimées dans 3.15.

3.4 État de l'art

Ayant privilégié la continuité pour présenter la description de notre contribution, nous synthétisons dans cette section les principaux travaux existants pour calculer la stratégie d'ordonnancement des points de sauvegarde optimale. Nous trouvons dans l'état de l'art une multitude de modèles de performance conçus pour optimiser le compromis entre le surcoût des points de sauvegarde et la quantité de travail perdu à cause des pannes.

Ces approches de modélisation et résolution se basent sur différentes hypothèses de modélisation et s'intéressent à des fonctions objectif distinctes. Dans cette synthèse nous détaillons pour chaque contribution les points suivants : le modèle du couple application/plateforme d'exécution, le modèle du surcoût du mécanisme de sauvegarde, le modèle de panne, la fonction objectif considérée et finalement les propriétés de la stratégie d'ordonnancement des points de sauvegarde proposée.

4. disponible sur http ://ligforge.imag.fr/projects/ci-calc/

Young [102] : À notre connaissance, Young [102] a introduit le premier modèle de performance pour gérer ce compromis. Il exprime l'espérance du temps total perdu en fonction de la fréquence de sauvegarde et du taux d'inter-arrivées des pannes. Young considère un modèle d'application à horizon infini c'est-à-dire qu'il y a toujours un flux de travail à exécuter. Ensuite pour modéliser l'inter-arrivées des pannes, il opte pour le processus de Poisson en supposant que le régime de panne est permanent. Concernant le modèle de surcoût des points de sauvegarde. l'auteur suppose que le surcoût d'un point de sauvegarde est constant et noté par c. En se basant sur cette abstraction, Young s'intéresse plutôt à la minimisation de la somme du surcoût introduit par les points de sauvegarde et la quantité de travail perdu avant la première panne puisque le régime de panne est permanent. Nous appelons ce critère de performance le temps total perdu (*wasted time*) et il est noté par W. Pour exprimer le temps total perdu en fonction de la stratégie d'ordonnancement des points de sauvegarde. Young suppose que l'intervalle inter-sauvegarde est constant et noté par ρ. Ensuite, en conditionnant par l'événement "la panne arrive à l'instant t entre le $n^{\text{ème}}$ et le $(n+1)^{\text{ème}}$ intervalle de sauvegarde", le temps total perdu jusqu'à cet instant se compose de n points de sauvegarde dont le surcoût est donné par nc et le travail perdu entre l'instant de panne et la dernière sauvegarde est donné par $L = t - n(\rho + c)$.
En se basant sur cette modélisation, l'espérance de temps total perdu à cause des pannes et du surcoût des points de sauvegarde en fonction de ρ est donnée par l'expression suivante :

$$\mathbb{E}\left[W_\rho\right] = \sum_{n=0}^{\infty} \int_{n(\rho+c)}^{(n+1)(\rho+c)} (t - n\rho)\lambda \exp^{-\lambda t}\, dt. \tag{3.25}$$

Pour exprimer la valeur optimale de ρ^* en fonction de c et λ qui annule la dérivée par rapport à ρ de l'Expression 3.25. Young utilise l'approximation de premier ordre du terme $\exp^{\lambda(\rho+c)}$ en supposant que la valeur de $\lambda(\rho + c)$ est au voisinage de 0 c'est-à-dire $\rho + c << 1/\lambda$. Ainsi, l'intervalle de sauvegarde optimal est donné par l'expression suivante :

$$\rho^* = \sqrt{2c/\lambda}. \tag{3.26}$$

Dans ce modèle, Young ne s'intéresse pas au temps de complétion d'une application donnée mais plutôt au rendement du système de calcul en minimisant la totalité du temps perdu par rapport au travail effectif réalisé pendant un seul cycle de panne. Bien que la stratégie proposée a une forme analytique élégante, ce résultat est fondé sur deux hypothèses discutables. La première est la considération dès le départ que la stratégie d'ordonnancement des points de sauvegarde est périodique. Le deuxième point discutable est l'approximation de premier ordre de la fonction $\exp^x$.

Daly [35] : Dans la continuation du travail de Young, Daly [35] propose une approximation d'ordre supérieur de la valeur optimale de l'intervalle de temps qui sépare deux points de sauvegarde. Contrairement au modèle de Young, dans ce travail l'auteur considère le même modèle d'application présenté dans la Section 3.2.1 (temps de complétion à horizon fini). Par contre, Daly garde les mêmes hypothèses pour modéliser le temps d'inter-arrivées des pannes en supposant que le régime de panne est transitoire et que la durée d'indisponibilité est nulle. Cependant, dans ce travail pour chaque redémarrage il y a un surcoût de reprise associé, noté par R. Le modèle de sauvegarde lui aussi reste le même par rapport à celui de Young. Puisque le modèle d'application est à

horizon fini, dans ce travail l'auteur s'intéresse à la minimisation de l'espérance du temps de complétion. Ainsi, pour exprimer l'espérance du temps de complétion, l'auteur procède comme suit. Soit $S_{\rho+c}$ une $v.a.d$ qui représente le nombre de fois qu'il faut pour exécuter ρ unités de travail avec un point de sauvegarde sans interruption de panne. $L_{\rho+c}$ une $v.a.r$ qui représente la quantité de travail perdu à chaque fois que l'on exécute ρ unités de travail suivi d'un point de sauvegarde. L'espérance du temps de complétion en fonction de ρ est donnée par :

$$\mathbb{E}\left[T_\rho(\omega)\right] = \omega + c(\omega/\rho - 1) + \frac{\omega}{\rho}\mathbb{E}[S_{\rho+c}]\left(R + \mathbb{E}[L_{\rho+c}]\right). \tag{3.27}$$

Dans un premier temps, pour exprimer la valeur de ρ^* en fonction de λ, d et c, l'auteur suppose qu'en moyenne la panne arrivera à la moitié de l'intervalle, c'est-à-dire $\mathbb{E}\left[L_{\rho+c}\right] \approx (\rho + c)/2$. De plus il utilise une approximation du premier ordre pour estimer $\mathbb{E}\left[S_{\rho+c}\right]$ c'est-à-dire, $\exp^{\lambda(\rho+c)} - 1 \approx \lambda(\rho + c)$. En se basant sur ces hypothèses, la valeur optimale qui minimise l'Expression 3.27 est donnée par l'expression suivante :

$$\rho^* = \sqrt{2c(R + 1/\lambda)} \tag{3.28}$$

Dans un deuxième temps l'auteur utilise une approximation d'ordre supérieure des expressions de $\mathbb{E}\left[L_{\rho+c}\right]$ et $\mathbb{E}\left[S_{\rho+c}\right]$ pour calculer la valeur de l'intervalle optimal. Ensuite, en comparant analytiquement la solution d'ordre supérieur donnée par l'Expression 3.29 avec la solution exacte, il montre que cette solution a une erreur relative qui est inférieure à 5% dans toutes les configurations.

$$\rho^* = \begin{cases} \sqrt{2c/\lambda} - c & \text{si } c < \frac{1}{2\lambda} \\ 1/\lambda & \text{si } c \geq \frac{1}{2\lambda} \end{cases} \tag{3.29}$$

Un autre point important est à signaler, en effet dans la solution d'ordre supérieur la valeur optimale de l'intervalle de sauvegarde ne dépend plus du surcoût de reprise R. Pour valider cette solution d'approximation, Daly présente des simulations pour montrer que la solution d'approximation proposée est bonne. Nous notons que dans ces simulations Daly s'intéresse à la variation de l'espérance du temps de complétion en fonction de la variation du surcoût des points de sauvegarde et du taux de panne. Premièrement ces simulations, montrent que l'approximation de premier ordre de Young reste une très bonne approximation pour les configurations où le temps moyen des inter-arrivées des pannes est grand par rapport au surcoût d'un point de sauvegarde comme par exemple la configuration $MTTF = 1/\lambda = 24\ h$, $c = 5\ min$ et $R = 10\ min$ ou la configuration $MTTF = 1/\lambda = 6\ h$, $c = 5\ min$ et $R = 10\ min$. Deuxièmement, ces simulations montrent aussi que l'approximation de premier ordre proposée par Young s'écarte de l'approximation d'ordre supérieur quand le taux de panne devient important, comme par exemple la configuration $MTTF = 1/\lambda = 15\ min$, $c = 5\ min$ et $R = 10\ min$.

Ling *et al.* [101] : Contrairement aux travaux de Young et Daly, dans [101] Ling *et al.*, modélisent le compromis sous un angle complètement diffèrent en gardant quelques points communs. Comme hypothèses communes avec le modèle de Young, ils considèrent un modèle d'application à horizon infini et un modèle de surcoût constant pour les

points de sauvegarde. La fonction objectif reste aussi la même, c'est-à-dire le temps total perdu avant la première panne. Par contre, pour modéliser l'inter-arrivées des pannes, ils utilisent une distribution arbitraire notée par $F(x)$ en supposant que le régime de panne est permanent. Cependant, nous tenons à soulignons qu'il y a aussi deux différences subtiles qui concernent le modèle du mécanisme de sauvegarde et reprise. Premièrement, les auteurs supposent que durant la phase de sauvegarde il n'y a pas de pannes. Deuxièmement, ils considèrent que cette durée de sauvegarde n'a aucun impact sur la probabilité de panne dans le futur. Comme contribution majeure par rapport aux travaux de Young et Daly, ils ne considèrent pas que l'intervalle inter sauvegarde est forcément constant. Ainsi dans ce travail Ling *et al.* introduisent le concept de la fonction fréquence de sauvegarde $\phi(t)$ qui varie dans le temps. Formellement, l'instant du $i^{ème}$ point de sauvegarde noté par τ_i est la solution de l'équation ci-dessous avec $\tau_0 = 0$ comme cas de base :

$$\int_{\tau_{i-1}}^{\tau_i} \phi(u)du = 1.$$

En se basant sur cette modélisation, le temps total perdu avant la panne est composé comme d'habitude des deux termes. Le premier terme représente le temps perdu dû aux surcoûts de sauvegarde depuis le début de l'exécution jusqu'à l'instant de panne t, cette quantité, notée par σ_t est donné par :

$$\sigma_t = c\int_0^t \phi(u)du.$$

Le deuxième terme représente la quantité de travail perdu entre la dernière sauvegarde et l'instant de la panne. Pour exprimer le deuxième terme les auteurs utilisent le développement limité du premier ordre de la fonction de distribution de panne au voisinage de τ_{i+1}, en considérant l'égalité suivante :

$$\overline{F}(\tau_i) - \overline{F}(\tau_{i+1}) \approx f(\tau_{i+1})(\tau_{i+1} - \tau_i).$$

Nous soulignons que ce développement est équivalent à dire qu'en moyenne la quantité de travail perdu est la moitié de l'intervalle inter sauvegarde. Finalement, en conditionnant par l'événement apparition de panne à l'instant t. L'espérance du temps perdu en fonction de $\phi(t)$ est donnée par :

$$\mathbb{E}\left[W_\phi\right] = \int_0^\infty (c\int_0^t \phi(u)du + \frac{1}{2\phi(t)})f(t)dt.$$

En se basant sur cette modélisation ils donnent la forme analytique de fréquence optimale $\phi^*(t)$ qui minimise l'espérance du travail perdu est exprimée ci-dessous. Cependant, nous notons que la démarche mathématique de résolution repose fondamentalement sur l'hypothèse de non-décroissance du taux de panne (*IFR*).

$$\phi^*(t) = \sqrt{\frac{f(t)}{2c\overline{F}(t)}} = \sqrt{\frac{\lambda(t)}{2c}}. \tag{3.30}$$

Pour résumer, le résultat de Ling *et al.* est le même que celui de Young si le taux de panne $\lambda(t)$ est constant. Ainsi, théoriquement ce modèle affirme que la fréquence

optimale de sauvegarde est proportionnelle à la racine carrée de la fonction du taux de panne si cette dernière est croissante. Ce résultat implique aussi que la fréquence de sauvegarde optimale est constante seulement quand l'inter-arrivées des pannes suit un processus de Poisson. Nous rappelons que cette modélisation se restreinte aux fonctions de distribution avec un taux de panne constant ou croissant *IFR*.

Dohi *et al.* [36] : Sur la même ligne des travaux de Ling *et al.* [101] dans [36] Dohi *et al.* proposent de reformuler autrement le compromis. En effet, ils gardent exactement les mêmes hypothèses et méthodologie de modélisation présentées auparavant dans [101] à savoir : application à horizon infini, distribution générale pour modéliser l'inter-arrivées des pannes, durée constante pour la phase de sauvegarde, pas de pannes durant la sauvegarde et une fonction fréquence variable pour calculer les instants de sauvegarde. Partant de cette conception, les auteurs ont les mêmes expressions qui expriment respectivement le surcoût accumulé des points de sauvegarde et la quantité de travail perdu à cause des pannes. Par contre, dans ce travail ils ne minimisent pas la somme des deux termes mais, ils ont choisit de minimiser un terme sous contrainte de l'autre. Ainsi, dans un premier temps ils expriment la fréquence optimale qui minimise l'espérance du travail perdu avant la panne pour une valeur fixée du surcoût accumulé des points de sauvegarde et vice-versa. De cette façon, ils présentent une nouvelle expression de la fréquence optimale de sauvegarde sous contrainte du surcoût accumulé des points de sauvegarde ou la quantité de travail perdu à cause des pannes.

Okamura *et al.* [78], Ozaki *et al.* [81] : Les auteurs des deux travaux suivants [78, 81], proposent deux méthodologies équivalentes pour calculer la stratégie d'ordonnancement des points de sauvegarde qui minimise le temps total perdu durant un cycle de panne. Cette méthodologie repose principalement sur un résultat élégant produit par Barlow *et al.* [7] pour résoudre un problème très proche de celui traité ici. Comme hypothèses de modélisation, ils considèrent un modèle d'application à horizon infini, un surcoût constant pour les points de sauvegarde, une distribution générale pour modéliser l'inter-arrivées des pannes, pas de panne durant la phase de sauvegarde et la durée de temps écoulée dans la phase de sauvegarde ne détériore pas la probabilité de panne. Par contre, dans ce travail le régime de panne est supposé transitoire. Mais en se basant sur la théorie de renouvellement ils montrent que : minimiser la somme du temps total perdu pendant plusieurs cycles de pannes est équivalent à minimiser le temps total perdu pendant un seul cycle de panne si on suppose que le système est en état stationnaire. Ainsi, on se ramène au régime de panne permanent. En se basant sur ces hypothèses, l'espérance du temps total perdu est exprimée ainsi. Soit la séquence $\{\tau_i\}_{i\geq 1}$ qui représente la suite des instants absolus de sauvegarde durant un cycle. Tel que τ_i est l'instant du $i^{\text{ème}}$ point de sauvegarde avec $\tau_0 = 0$. En conditionnant par l'événement : la panne arrive à l'instant t entre le $i^{\text{ème}}$ et $i+1^{\text{ème}}$ point de sauvegarde, c'est-à-dire $\tau_i < t \leq \tau_{i+1}$. L'expression de l'espérance du temps total perdu durant un cycle en fonction de la séquence $\{\tau_i\}$ est donnée par :

$$\mathbb{E}\left[W_\pi\right] = \sum_{i=0}^{\infty} \int_{\tau_i}^{\tau_{i+1}} \left(ic + t - \tau_i\right) f(t)dt. \tag{3.31}$$

où le terme ic représente le surcoût accumulé des points de sauvegarde jusqu'à l'instant τ_i. Tandis que le terme $t - \tau_i$ représente la quantité de travail perdu si la panne arrive durant cet intervalle. Ainsi, la séquence optimale $\{\tau_i^*\}$ annule la dérivée partielle de l'Expression 3.31 par rapport à chaque τ_i. En conséquence, cette séquence vérifie l'expression suivante :

$$\forall i \geq 1,\ \tau_i^* - \tau_{i-1}^* = \frac{F(\tau_{i+1}^*) - F(\tau_i^*)}{f(\tau_i^*)} + c, \quad \text{avec } \tau_0 = 0. \tag{3.32}$$

Cependant, la relation de récurrence qui relie les éléments de la séquence rend cette expression quasi inutile vu la dépendance interne. C'est là où le travail de Barlow *et al.* intervient. En effet, dans [7] les auteurs montrent deux résultats importants. Dans le premier résultat (Théorème 5 dans [7]) ils démontrent que si la distribution d'inter-arrivées des pannes est de type *IFR*. La suite des intervalles d'inter sauvegarde est nécessairement décroissante et tous les termes sont positifs, tels que :

$$\forall i, j \text{ tel que } 0 \leq j < i \ \text{ nous avons, } \ 0 \leq \tau_{i+1}^* - \tau_i^* \leq \tau_{j+1}^* - \tau_j^*.$$

Ensuite dans un deuxième temps, ils établissent la proposition suivante : Considérant un τ_1 arbitraire positif et non nul pour calculer la séquence $\{\tau_i\}$ en utilisant l'Expression 3.32.

- Si τ_1 est sous-estimé (c'est-à-dire $\tau_1 < \tau_1^*$) : il existe forcément un rang i, tel que l'intervalle de temps entre le $i^{\text{ème}}$ et $i+1^{\text{ème}}$ est négatif ($\tau_{i+1} - \tau_i < 0$) ce qui contredit le théorème précédent.
- Dans le cas contraire si τ_1 est surestimé : il existe forcément un rang i, tel que la suite des intervalles inter sauvegarde est croisante à partir de i ce qui contredit aussi le théorème précédent.

En se basant sur cette proposition, calculer la séquence $\{\tau_i^*\}$ est équivalent à chercher τ_1 moyennant une dichotomie dans un intervalle borné. Cependant ce résultat n'est applicable que dans le cas où la distribution de l'inter-arrivées des pannes appartient à la classe *IFR* et quand l'opération de sauvegarde ne dégrade pas la probabilité de panne du système de calcul.

Bougeret *et al.* [14] : Un autre travail mené ultérieurement en 2011 par Bougeret *et al.* [14] présente un résultat en utilisant une modélisation continue et un autre résultat en utilisant une modélisation discrète. Le premier résultat est produit en considérant les hypothèses de modélisation suivantes. Un modèle d'application à horizon fini et un modèle de surcoût constant pour les points de sauvegarde. Concernant le modèle de panne, ils utilisent un processus de Poisson pour décrire l'inter-arrivées des pannes et ils supposent que le régime de panne est transitoire. Une durée de redémarrage de l'application après la panne est aussi intégrée dans ce modèle. Comme fonction objectif, ils s'intéressent à la minimisation de l'espérance du temps de complétion. Ainsi en adoptant une démarche de résolution mathématique différente de celle que nous présentons ici, ils exhibent un résultat qui confirme le résultat que nous avons présenté et publié dans [17].

Pour clôturer cette partie d'étude des contributions dans l'état de l'art. Nous rappelons que dans cette section trois grandes classes de modélisations ont été présentées.

- La première, dans laquelle il y a les travaux de Young [102] et Daly [35], représente la classe où les stratégies d'ordonnancement des points de sauvegarde sont supposées périodiques dès le départ sans aucune preuve formelle. En s'appuyant sur cette hypothèse ils fournissent une solution d'approximation qui est sans doute efficace pour minimiser l'espérance du temps de complétion dans le cas où la distribution de panne est un processus de Poisson et le surcoût des points de sauvegarde est constant.
- Ensuite, il y a la classe des travaux [101, 36] qui expriment la stratégie d'ordonnancement des points de sauvegarde sous forme de fonction fréquence. Les deux travaux dans cette classe reposent sur une approximation du premier ordre de la fonction distribution de panne. De plus ils supposent aussi que durant les phases de sauvegarde il n'y a pas de panne et que leurs durées ne dégradent pas la probabilité de panne.
- Finalement, la dernière classe est celle des travaux qui se basent sur le résultat de Barlow *et al.* [7]. Dans ces travaux il y a aussi quelques points qui sont notables. Telle que l'hypothèse de l'absence de pannes durant la phase de sauvegarde et que cette durée ne dégrade pas la probabilité de panne. De plus ces travaux ne sont applicables que pour les distributions de nature *IFR*.

3.5 Validation et comparaison expérimentale

Dans cette section nous présentons quelques simulations réalisées pour valider et comparer les contributions proposées avec les contributions existantes.

3.5.1 Paramètres et scénarios des simulations

Pour valider le modèle nous proposons différents scénarios de simulation. Dans tous les scénarios un processus de Poisson est utilisé pour modéliser l'inter-arrivées des pannes. Le modèle retenu comme modèle de comparaison est celui de Daly [35] en utilisant l'Expression 3.29 pour calculer la stratégie optimale. Les différents scénarios sont présentés dans la Table 3.1. Par exemple dans le scénario 1 le taux de panne par jour augmente dans l'intervalle $\lambda \in [1/2, 96]$ avec un pas de 5 tandis que le surcoût des points de sauvegarde et la quantité de travail initial sont constants. Nous notons que chaque scénario est simulé 10^4 fois avec des pannes générées aléatoirement selon les paramètres de la distribution de panne. Finalement, nous soulignons que les intervalles de confiance reportés sont à 95%.

Scenario	Taux de Panne (jours)	Suroût de sauvegarde (mins)	Travail Initial (jours)
1	$\lambda \in [1/2, 96]$ $(pas = 5)$	$C = 10$	$\omega = 10$
2	$\lambda = 1/2$	$C \in [1/2, 96]$ $(pas = 5)$	$\omega = 7$

Table 3.1 – Configuration des scénarios

3.5.2 Résultats des simulations

La Figure 3.5 reporte la variation de la moyenne du temps de complétion versus la variation du taux de panne dans le cadre du premier scénario. Nous soulignons avant

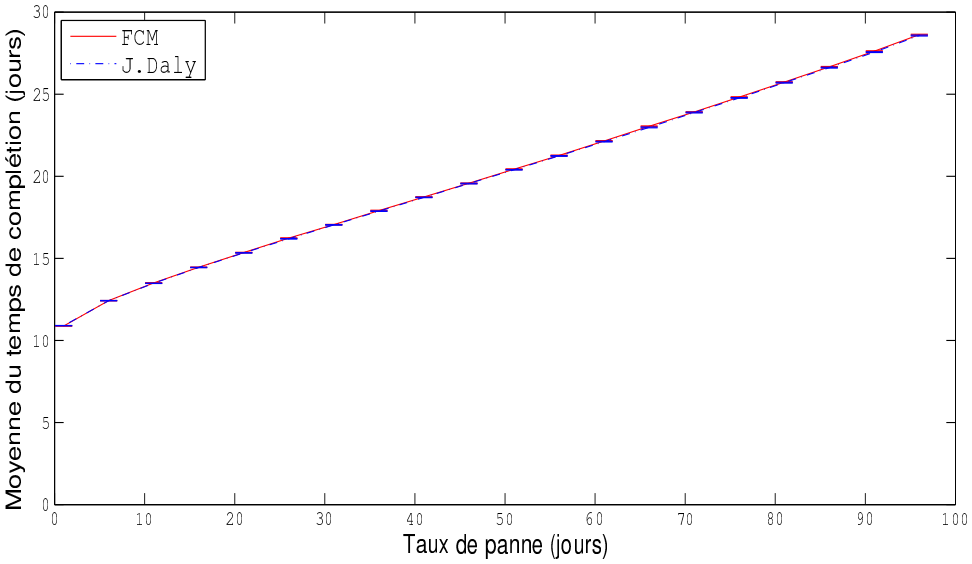


FIGURE 3.5 – Variation de la moyenne du temps de complétion (Scenario 1)

toute chose que cette figure montre bien que la croissance de la moyenne empirique du temps de complétion est linéaire en fonction de λ. Ceci affirme l'apport du mécanisme de tolérance aux pannes puisque nous rappelons que sans ce mécanisme la croissance est exponentielle. Nous constatons aussi que les deux stratégies de sauvegarde ont exactement les mêmes moyennes empiriques du temps de complétion. Cependant, dans la deuxième Figure 3.6 où nous traçons la variation du nombre de points de sauvegarde réalisés par chaque stratégie en fonction de la variation du taux de panne toujours dans le premier scénario. Nous voyons bien que la stratégie proposée réduit le nombre de sauvegardes par rapport a celle de Daly. En effet quand le taux de panne devient très élevé notre stratégie réduit le taux de sauvegarde de 20% par rapport à l'autre modèle.

Pour mieux comprendre la signification de ce résultat, nous conduisons une autre campagne de simulations en se basant sur le deuxième scénario cette fois. Ainsi la Figure 3.7 montre la variation de la moyenne du temps de complétion en fonction de la variation du surcoût des points de sauvegarde. Ce résultat confirme aussi le résultat reporté sur la figure précédente c'est-à-dire les deux stratégies ont exactement la même moyenne empirique du temps de complétion. Par contre, dans la Figure 3.8 où nous reportons la variation à la fois du surcoût accumulé des points de sauvegarde et la quantité de travail perdu versus la variation du surcoût des points de sauvegarde. Nous pouvons constater que notre stratégie réduit le surcoût accumulé des sauvegardes (courbe continue avec des ronds) par rapport à l'autre stratégie (courbe discontinue avec des ronds). En contre partie notre stratégie perd plus de travail à cause des pannes (courbe continue sans marqueur) par rapport à l'autre stratégie (courbe discontinue sans marqueur). En conséquence les deux stratégies ont la même moyenne empirique du temps de complétion.

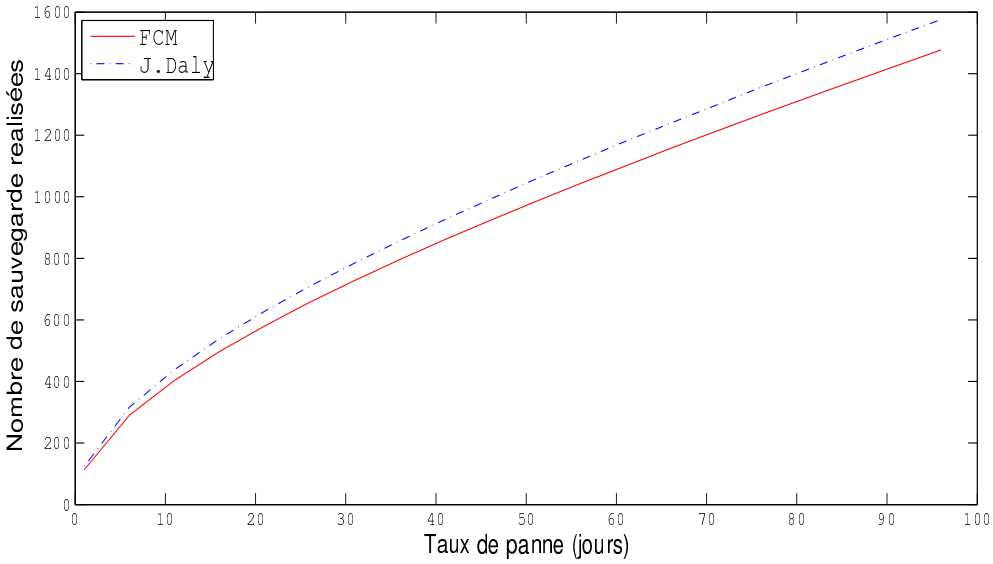


FIGURE 3.6 – Nombre optimal des points de sauvegarde (Scenario 1)

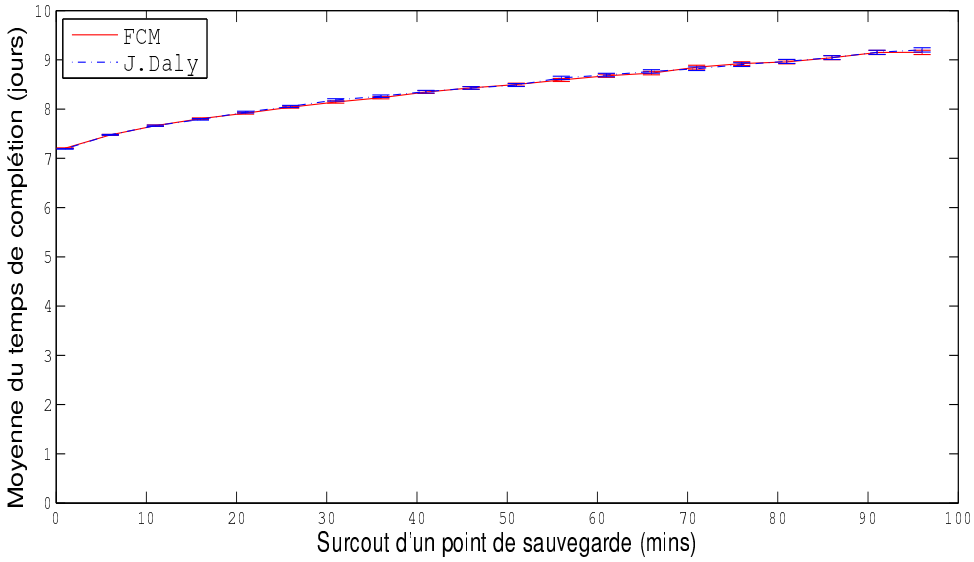


FIGURE 3.7 – Variation de la moyenne du temps de complétion (Scenario 2)

3.5.3 Bilan des simulations

Comme conclusions, nous tenons à préciser que minimiser le surcoût accumulé des points de sauvegarde tout en gardant le même temps de complétion. Cela est sans doute plus avantageux car nous rappelons qu'un point de sauvegarde dans un environnement de type HPC représente un énorme goulot d'étranglement entre les noeuds de calcul et le support stable. Nous notons que ce point est important car dans les simulations la congestion n'est pas pris en compte. Pour conclure cette section, nous insistons sur deux points très importants à mentionner :

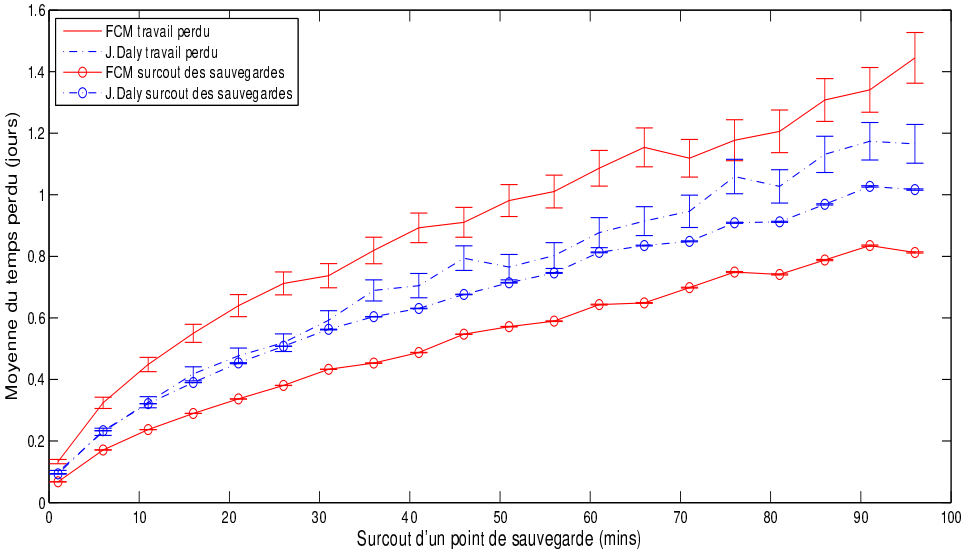


FIGURE 3.8 – Moyenne du surcoût accumulé de sauvegarde / Moyenne Travail perdu (Scenario 2)

- premièrement nous insistons sur le fait qu'il faut impérativement utiliser les mécanismes de tolérance aux pannes pour réduire l'espérance du temps de complétion sinon il croit exponentiellement en fonction de la quantité de travail ou le taux de panne.
- deuxièmement il faut utiliser une stratégie exacte qui se base sur une approximation d'ordre supérieur de la fonction Lambert [34] plutôt qu'une solution approchée qui a une forme analytique simple.

3.6 Conclusion

Dans ce chapitre, nous avons commencé par l'étude des performances d'une application en présence des pannes et sans aucun mécanisme de tolérance aux pannes. Cette étude a montré que l'intégration d'un mécanisme de tolérance aux pannes est quelque chose de primordiale puisque sans ce dernier, l'espérance du temps de complétion a une croissance exponentielle par rapport à l'augmentation de la quantité de travail initial ou par rapport à l'augmentation du taux de panne dans le cas de la loi exponentielle. À travers cette analyse, nous avons aussi illustré que le mécanisme de tolérance aux pannes basé sur le principe de sauvegarde et reprise est une solution efficace pour réduire cette croissance exponentielle. Comme tout mécanisme de tolérance aux pannes, lors de sa mise en place un problème d'optimisation de compromis est soulevé. Pour optimiser le compromis entre la quantité de travail perdu à cause des pannes versus le surcoût accumulé des points de sauvegarde, nous avons formulé un problème d'optimisation, dont l'objectif est de calculer une stratégie d'ordonnancement des points de sauvegarde qui minimise l'espérance du temps de complétion. Principalement, dans ce chapitre nous avons établi trois contributions. Premièrement, nous avons démontré que dans le cas où le taux de panne ainsi que le surcoût des points de sauvegarde sont constants,

la stratégie d'ordonnancement des points de sauvegarde optimale est nécessairement périodique. Alors que les travaux existantes partaient de cette hypothèse sans aucune démonstration. Deuxièmement, en considérant des fonctions arbitraires pour décrire la variation du taux de panne et des surcoûts des points de sauvegarde, nous avons proposé une approche de résolution numérique pour résoudre le problème. Finalement, nous avons montré à travers des simulations que la solution exacte proposée et la solution d'approximation proposée par Daly sont équivalentes si nous considérons seulement le temps de complétion comme critère de performance. Par contre, en présentant les résultats des simulations sous un autre angle, nous avons montré qu'il est possible de réduire le surcoût accumulé des points de sauvegarde tout en gardant le même temps de complétion. Ainsi, nous concluons qu'il vaut mieux utiliser une solution exacte plutôt qu'une solution analytique simple à l'image de l'Expression 3.29 même si celle si ne dévie que de 5% par rapport à la solution exacte.

Toutefois, la formulation présentée possède deux limitations importantes à souligner. Premièrement, nous notons que le choix de la bonne technique de résolution numérique et ses paramètres de configuration est une tâche délicate. Deuxièmement, nous n'avons aucune borne ni sur la qualité ni sur la vitesse de convergence de la méthode de résolution numérique. Ces éléments ont représenté la motivation principale des travaux présentés dans le chapitre suivant, dans lequel nous avons opté pour la modélisation discrète du problème.

Modélisation discrète 4

Sommaire

4.1 Introduction **47**
4.2 Modèle de l'environnement d'exécution **48**
4.2.1 Modèle de la plateforme d'exécution 48
4.2.2 Modèle d'application 49
4.2.3 Modèle du mécanisme de sauvegarde 50
4.3 Formulation du problème d'optimisation **50**
4.3.1 Introduction au processus de décision markovien 50
4.3.2 Problème d'ordonnancement des points de sauvegarde 51
4.3.3 Ordonnancement optimal des points de sauvegarde 55
4.4 Analyse de complexité **56**
4.4.1 Analyse de complexité de l'algorithme proposé 56
a Analyse de complexité dans le cas général 56
b Cas surcoût des points de sauvegarde constant 58
4.4.2 Analyse de complexité du problème 58
a Définition du problème décisionnel relaxé 59
b Définition de la famille d'instances 59
c La NP-complétude du problème 61
4.5 État de l'art **62**
4.6 Étude expérimentale **64**
4.6.1 Modèle de l'environnement d'exécution 65
a Modèle d'application 65
b Modèle de communication 65
c Modèle de panne 66
d La stratégie d'ordonnancement des points de sauvegarde . 66
4.6.2 Méthodologie d'expérimentation 67
a Critères de performance et stratégies d'ordonnancement : 67
b Modèle SimGrid 68
c Les scénarios : 69
4.6.3 Analyse des résultats 71
4.7 Conclusions **73**

4.1 Introduction

Nous continuons dans ce chapitre l'étude des techniques de modélisation du compromis entre le surcoût induit par le mécanisme de sauvegarde et la réduction de la

quantité de travail perdu due aux interruptions causées par les pannes. Contrairement au chapitre précédent, ici, les approches étudiées reposent sur une modélisation discrète de l'application.

Nous entamons ce chapitre par la description des nouvelles hypothèses introduites par rapport à celles décrites dans le chapitre précédent. Nous exposons dans la Section 4.3 le problème d'ordonnancement des points de sauvegarde dont l'objectif est la minimisation du temps total perdu. Ensuite, dans la même section, nous exhibons un schéma de programmation dynamique pour calculer la stratégie d'ordonnancement optimale des points de sauvegarde. La Section 4.4 est dédiée à la fois à l'analyse de la complexité de l'approche algorithmique proposée et de la complexité du problème d'ordonnancement des points de sauvegarde. Ensuite nous présentons dans la section de l'état de l'art les principales autres approches de modélisation existantes. Finalement, nous menons dans la dernière section une étude expérimentale dont l'objectif est la validation de la stratégie d'ordonnancement des points de sauvegarde proposée dans le contexte des plateformes de calcul distribué de type *desktop grid*.

4.2 Modèle de l'environnement d'exécution

Dans cette section nous mettons en évidence les différences, ainsi que les points communs, entre les modèles de la plateforme, de l'application et du mécanisme de sauvegarde qui ont été utilisés dans le chapitre précédent par rapport à ceux qui sont utilisés ici.

L'enchainement de cette section est le suivant. Premièrement, nous exhibons le mode de fonctionnement et le modèle de panne de la plateforme d'exécution. Ensuite, nous décrivons le nouveau modèle d'application ainsi que les hypothèses sous-jacentes à ce dernier. Nous exposons à la fin de cette section le modèle du mécanisme de sauvegarde lié au nouveau modèle d'application.

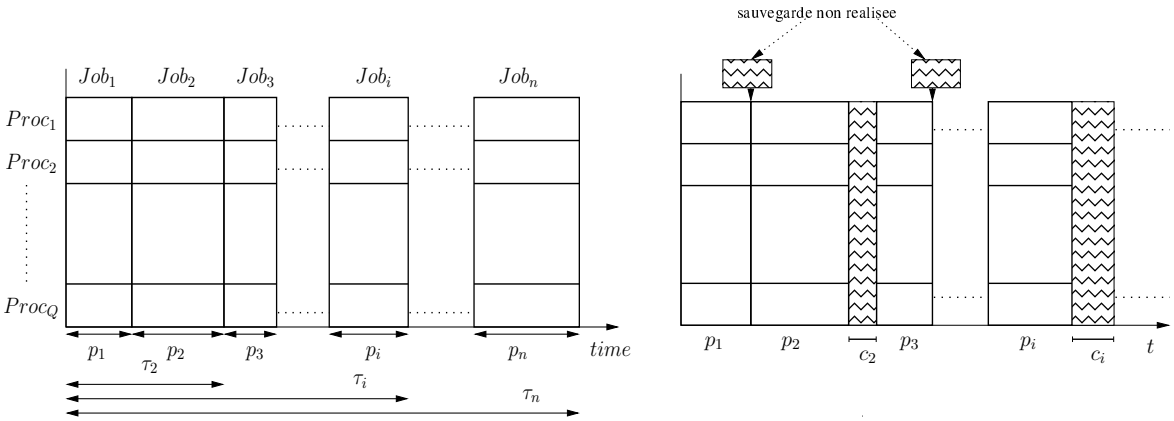
4.2.1 Modèle de la plateforme d'exécution

Pour caractériser la plateforme formellement, nous utilisons dans ce chapitre un modèle similaire à celui présenté auparavant. Nous rappelons que cette plateforme est assimilée à un seul élément de calcul composé de plusieurs processeurs et qui se caractérise par un taux de traitement d'instructions donné par $\nu = 1$. Cet élément de calcul est sujet à des pannes franches. Les instants d'arrivée des pannes sont modélisés par les fonctions $F(t), f(t)$ et $\lambda(t)$ qui représentent la fonction distribution, la fonction densité et le taux de panne. Typiquement, ces pannes sont détectées immédiatement moyennant un détecteur de panne. Dans le cas où le détecteur de panne n'est pas fiable ou absent, par exemple comme le cas des plateformes de type *desktop grid*. Nous supposons qu'au pire des cas la panne est détectée lors du prochain point de sauvegarde. Techniquement, ceci peut être implémenté de deux façons. Par exemple dans le contexte des environnements HPC, la réalisation d'un point de sauvegarde nécessite une synchronisation globale entre les processus. Ainsi si un processus est touché par une panne, ce processus sera absent au moment de la synchronisation et donc la panne est détectée à cet instant. Dans le contexte des environnements de type

desktop grid, le point de sauvegarde peut être considéré comme étant un point de rendez-vous ou un délai de garde (*time-out*), ainsi si le processus n'entame pas l'écriture de sa sauvegarde à l'instant prévu, il sera considéré comme étant en panne. Toutefois, nous notons que notre travail reste applicable même dans le cas où les pannes sont détectées immédiatement. En effet en cas de besoin, un paramètre de configuration est mis en place dans le modèle de performance pour négliger cette latence de détection.

4.2.2 Modèle d'application

Le modèle d'application considéré dans ce travail est complètement diffèrent du modèle décrit dans le chapitre précédent. En effet la différence entre la modélisation continue et discrète est fondamentalement liée au modèle d'application. Dans ce travail, l'application qui peut être de nature séquentielle ou parallèle comme celle reportée sur la Figure 4.1(a), est composée de n tâches non-préemptives. La $i^{ème}$ tâche notée par job_i contient p_i unités de travail. Ces tâches ont un ordre d'exécution prédéfini à l'avance, donné par l'indice des tâches. En conséquence, l'exécution de la $i^{ème}$ tâche ne peut commencer qu'après la complétion des tâches dont les indices sont dans l'intervalle $[1..i-1]$.
Donc, à la différence du modèle continu, ici l'application ne peut pas être préemptée à n'importe quel instant pour réaliser un point de sauvegarde.



(a) Schéma d'exécution général.

(b) Schéma d'exécution avec des points de sauvegarde.

FIGURE 4.1 – modèle d'exécution sans sauvegarde (a) modèle d'exécution avec sauvegarde (b)

Cette modélisation représente deux grandes classes d'application de calcul scientifique où le mécanisme de sauvegarde est plutôt intégré au niveau applicatif. La première classe contient les applications HPC qui sont basées sur le paradigme de programmation parallèle BSP [98], dans lequel les super-pas représentent les tâches. Nous citons par exemple les applications parallèles de calcul numérique, comme celles décrites dans [31], dont l'exécution peut être vue comme une exécution d'un ensemble de blocs d'itérations (bloc d'itération ≡ tâche) et le résultat intermédiaire produit à la fin de chaque bloc d'itération peut être utilisé comme étant un point de sauvegarde. La seconde classe représente la classe dominante des applications exécutées dans les grilles de calcul

distribué [61] qui sont sous forme de campagne d'exécution (*i.e. bag of tasks*). Une description plus en détail de cette classe est présentée dans la Section 4.6, puisque c'est le modèle utilisé pour conduire les expérimentations.
Pour résumer, nous tenons tout d'abord à signaler qu'il est toujours possible d'utiliser ce modèle avec les application qui sont préemptables. Il suffit de choisir un pas de discrétisation pour transformer l'application en une chaine de tâches unitaires. Finalement, nous notons que dans ce chapitre, l'application est décrite formellement par le vecteur suivant : $\{p_1, p_2, \cdots, p_n\}$ avec $\tau_i = \sum_{j=1}^{i} p_j$.

4.2.3 Modèle du mécanisme de sauvegarde

Conformément à la description du modèle d'application, les seuls endroits où les points de sauvegarde peuvent être ordonnancés sont les n frontières entre les tâches. En effet dans ce cadre, le point de sauvegarde correspond à l'enregistrement du résultat intermédiaire sur le support de stockage stable. Dans ce cas, le surcoût engendré par le point de sauvegarde après la complétion de la $i^{ème}$ tâche noté par c_i dépend de la durée de temps qu'il faut pour enregistrer le $i^{éme}$ résultat intermédiaire sur le support stable. Nous considérons aussi que durant la phase de sauvegarde des pannes peuvent arriver et que cette durée est incluse dans le temps de complétion de l'application d'où elle dégrade la probabilité de panne. Ceci est schématisé sur la Figure 4.1(b) qui reporte un exemple d'exécution avec sauvegarde.

4.3 Formulation du problème d'optimisation

Dans cette section, nous présentons la formulation du modèle de performance, dans lequel nous exprimons le compromis entre l'impact des pannes sur l'exécution versus le surcoût accumulé introduit par le mécanisme de sauvegarde. À la différence du chapitre précédent où le compromis à gérer est encapsulé dans l'expression de l'espérance du temps de complétion, ici nous gérons le compromis en minimisant le temps total perdu noté par W (*Wasted time*). Le temps total perdu est composé de deux termes. Le premier terme représente le temps perdu engendré par les points de sauvegarde. Le deuxième terme représente le temps perdu à cause des pannes.

Le plan de cette section est le suivant. Nous introduisons dans un premier temps les notions de bases, les notations et la terminologie utilisées pour formuler le problème sous la forme d'un processus de décision markovien (MDP *i.e. Markovian Decision Process*) [83]. Finalement, nous exhibons une approche algorithmique pour calculer la stratégie d'ordonnancement des points de sauvegarde qui minimise l'espérance du temps total perdu.

4.3.1 Introduction au processus de décision markovien

Un processus de décision markovien est généralement utilisé pour modéliser des systèmes décisionnels dont l'état évolue dans le temps selon un processus probabiliste. Le fonctionnement de ce système est décrit de la manière suivante : à chaque instant dans le temps le système est amené à choisir une action parmi un ensemble d'actions possibles. Cette décision est prise en fonction de l'état occupé par le système à cet instant.

L'objectif de cette décision est la minimisation d'une fonction perte (ou respectivement la maximisation d'une fonction de gain). Ensuite, une fois que le système choisit une action, son état évolue généralement selon un processus probabiliste vers un nouvel état. Cette évolution est dite de nature Markovienne, c'est-à-dire que les effets de l'action à un instant donné ne dépendent pas de l'historique des états visités mais seulement de l'état que le système occupe à cet instant de décision. Suite à cette transition d'état, le système encaisse une perte (reçoit un gain) en fonction de l'état actuel, l'action choisie et le prochain état du système. Les ingrédients d'un modèle MDP sont les cinq éléments suivants :

1. L'ensemble des instants de décision noté par $I = \{\tau_1, \tau_2, \cdots, \tau_i, \cdots, \tau_n\}$. Dans le cas où n est fini ($n < \infty$), le problème est à horizon fini, sinon dans le cas contraire ($n \leq \infty$) le problème est à horizon infini. Ces instants correspondent soit à des instants continus dans le temps et dans ce cas $\tau_i \in \mathbb{R}_+$, ou à des indices d'intervalles quand le temps est discrétisé. Nous tenons à préciser que dans que le cas où l'ensemble I est fini, par convention le système ne choisit pas d'action à l'instant n, mais à cet instant le système passe forcement à l'état final dans lequel la perte totale (ou le gain total) est évaluée.
2. L'ensemble des états que le système occupe dans le temps est noté par E. Principalement E est un ensemble fini ou un ensemble infini et dénombrable (d'autres définitions de topologie de cet ensemble sont décrites dans [83]).
3. L'ensemble d'actions possibles noté par A, qui est l'union de tous les ensembles d'actions possibles en fonction de l'état actuel donné par $A = \cup_{e \in E} A_e$.
4. La fonction de transition entre les états notée par $\eta_i(e|e', a)$. Cette fonction représente la probabilité de se retrouver à l'état $e \in E$ à l'instant $i + 1$ sachant que l'état actuel à l'instant de décision i est e' et que l'action choisie est a.
5. La fonction de perte reçue à chaque instant notée par $\delta_i(e', a)$. Cette fonction représente la perte dans le cas d'un problème de minimisation ou le gain dans le cadre d'un problème de maximisation. Cette perte (ou gain) dépend de l'état $e' \in E$ que le système occupe et l'action $a \in A_{e'}$ qu'il choisit à l'instant i. Nous mentionnons que généralement cette fonction dépend aussi du prochain état du système. Dans ce cas la fonction de perte est liée à la fonction de transition par l'identité suivante :

$$\delta_i(e', a) = \sum_{e \in E} \Delta_i(e, a, e') \eta_i(e|e', a).$$

   La fonction $\Delta_i(e, a, e')$ représente la perte sachant que le système occupe à l'instant i l'état e' ensuite l'état e l'instant $i + 1$ en choisissant l'action a.

Nous tenons à signaler que par convention l'indice i' ou l'état e' précède l'indice i ou l'état e.

4.3.2 Problème d'ordonnancement des points de sauvegarde

Nous abordons dans cette partie la présentation du problème d'ordonnancement des points de sauvegarde sous la forme d'un processus de décision markovien. Dans ce travail, nous modélisons le couple application/plateforme d'exécution par une chaîne de

Markov à laquelle nous ajoutons une composante décisionnelle qui représente la stratégie d'ordonnancement des points de sauvegarde. Cette application est amenée à choisir après la complétion de chaque tâche entre l'action "prendre le point de sauvegarde" ou l'action "ignorer ce point de sauvegarde". Cette suite de décisions est déterminée avant même que l'exécution ne commence et définie dans une stratégie d'ordonnancement des points de sauvegarde notée par $\pi \in \{0,1\}^n$.

Contrairement au chapitre précédent où nous considérons l'espérance du temps de complétion comme étant la fonction objectif à optimiser. Dans ce chapitre nous nous intéressons à la minimisation (respectivement la maximisation) de l'espérance du temps total perdu (respectivement temps total utile) pendant un cycle de panne. Typiquement, l'exécution de l'application est assez souvent repartie sur plusieurs cycles de panne. Cependant, ici, nous optimisons le temps total perdu pendant un cycle de panne. En effet l'idée est de minimiser le temps total perdu dans chaque cycle de panne de telle façon que l'application se termine le plutôt possible. Donc pour utiliser notre stratégie, il faut calculer une stratégie d'ordonnancement des points de sauvegarde au début de chaque cycle de panne après le redémarrage de l'appliction. La question qui se pose : "est ce que la minimisation de l'espérance du temps de complétion est équivalente à la minimisation du temps total perdu cycle par cycle ?". Nous procédons dans la suite de cette section à l'identification et à la description des cinq éléments fondamentaux du modèle MDP.

1. Ensemble des instants de décision :
   Nous rappelons que dans notre cas l'application est amenée à choisir après la complétion de chaque tâche entre sauvegarder le résultat intermédiaire ou l'ignorer. Donc l'ensemble des instants de décision est $I = \{\tau_1, \tau_2, \cdots \tau_i, \cdots, \tau_n\}$ tel que $\tau_i = \sum_{j=1}^{i} p_j$. I est aussi équivalent à l'ensemble $\{1, 2, \cdots, i, \cdots, n\}$ que nous utilisons dans la suite pour alléger la notation.
2. Ensemble des états :
   Pour décrire l'état de l'application à chaque instant de décision deux informations sont nécessaires. La première information est l'indice du dernier point de sauvegarde noté par i'. Tandis que la deuxième information représente le surcoût accumulé des points de sauvegarde depuis le début que nous notons par σ'. Cet ensemble est défini formellement comme suit. Soit c^{max} le surcoût maximal qu'un point de sauvegarde peut avoir $c^{max} = \max_{1 \leq i \leq n} c_i$. Sans perte de généralité, supposons que $c^{max} \in \mathbb{N}$, l'ensemble des états possibles est défini par $E = \{e_{i'}^{\sigma'},\ 1 \leq i' \leq n,\ 0 \leq \sigma' \leq nc^{max}\} \cup e_0^0 \cup e^{\infty}$. Plus précisément, si l'application occupe l'état $e_{i'}^{\sigma'}$ à l'instant i. Ceci implique que le point de sauvegarde le plus proche de l'instant i est réalisé après $job_{i'}$ et σ' représente le surcoût accumulé des points de sauvegarde depuis le début jusqu'à l'instant i sans prendre en compte le surcoût du $i^{éme}$ (c_i) s'il est jamais fait. Finalement nous désignons par l'état e^{∞} l'état final et par e_0^0 l'état initial en supposant que par convention l'application commence à $\tau_0 = 0$ et $\sigma = 0$.
3. Ensemble des actions :
   L'ensemble des actions est donné par $A = \{0, 1\}$. D'où dans ce cadre de modélisation discrète la stratégie d'ordonnancement des points de sauvegarde $\pi = \{a_1, a_2, \cdots, a_n\}$ est un vecteur de taille n avec $a_i = 1$ si le $i^{ème}$ point de sauvegarde est réalisé, sinon $a_i = 0$ dans le cas contraire.

4. Fonction de transition :
   En se basant sur la description des états que l'application est susceptible d'occuper. La fonction de transition $\eta : E \times E \times A \to [0,1]$ est donnée par :

$$\forall (e_j^{\sigma}, e_{i'}^{\sigma'}) \in E \times E, \eta_i(e_j^{\sigma} | e_{i'}^{\sigma'}, a) = \begin{cases} 1 & \text{si } i < n \text{ et } j = i \text{ et } a = 1 \text{ et } \sigma = \sigma' + c_i. \\ 1 & \text{si } i < n \text{ et } j = i' \text{ et } a = 0 \text{ et } \sigma = \sigma'. \\ 1 & \text{si } i = n \text{ et } \sigma = \sigma' + c_n. \\ 0 & \text{si aucune condition est vérifiée.} \end{cases} \tag{4.1}$$

   Le premier cas où $i < n$ et $a = 1$, l'application choisit l'action sauvegarder dans ce cas l'état passe $e_{i'}^{\sigma'}$ à $e_i^{\sigma'+c_i}$ avec une probabilité 1. Dans le deuxième cas où $i < n$ et $a = 0$, l'action choisie est d'ignorer la sauvegarde donc l'état reste invariant avec une probabilité 1. Le dernier cas où $i = n$, l'application sauvegarde systématiquement le résultat final d'où la transition vers l'état $e_n^{\sigma'+c_n}$. Nous notons que dans ce cas la fonction de transition est déterministe puisque l'action détermine avec probabilité 1 le prochain état de l'application.

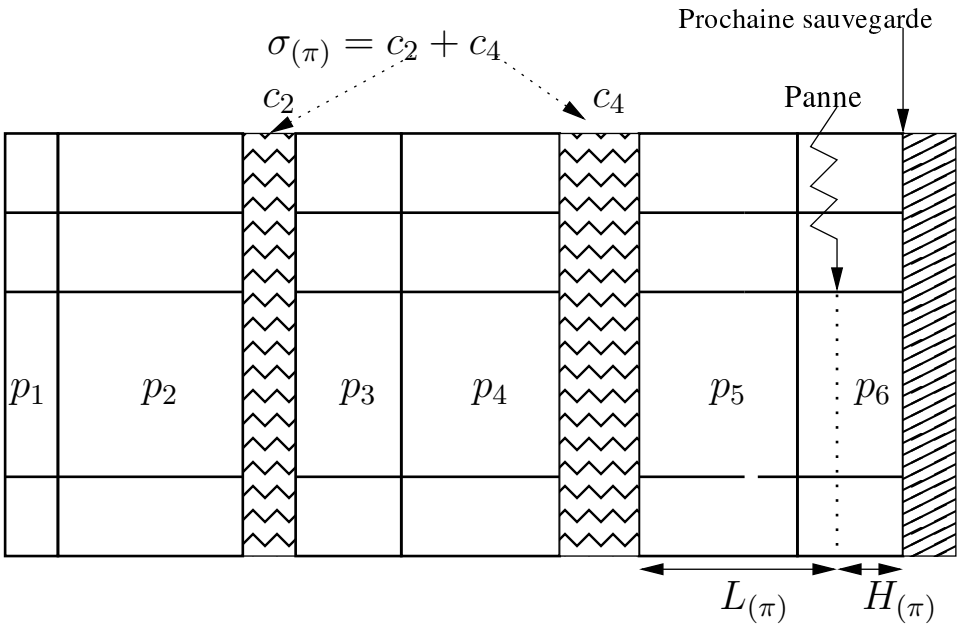


FIGURE 4.2 – Schéma d'exécution avec panne.

5. Fonction de perte :
   Nous passons maintenant à la description de la fonction de perte qui représente la fonction objectif à optimiser. Nous rappelons que dans ce chapitre nous nous intéressons à la minimisation du temps total perdu pendant un cycle de panne. La Figure 4.2 reporte les trois sortes de temps perdu que nous considérons.
   (a) Le premier type de temps perdu est celui écoulé durant les phases de sauvegarde. Cette durée est notée par σ et elle représente la somme des surcoûts des points de sauvegarde réalisés.
   (b) Le deuxième type de temps perdu représente la quantité de travail perdu à cause des pannes qui est la durée de temps qui sépare l'instant de panne du dernier point de sauvegarde notée par L.

(c) Finalement, dans ce chapitre nous supposons qu'au pire des cas la panne est détectée lors du prochain point de sauvegarde. Donc si une panne arrive à l'instant t elle ne sera détectée que lors de la prochaine sauvegarde prévue après t et cette durée de temps perdu est notée par H. Nous tenons à souligner que cette durée de détection est une extension dans le modèle qui peut être négligée si les pannes sont détectées instantanément.

En se basant sur cette abstraction la fonction perte $\delta : E \times A \to \mathbb{R}$ est définie par :

$$\delta_i(e_{i'}^{\sigma'}, a) = \begin{cases} \int_{\tau_{i'}+\sigma'}^{\tau_i+\sigma'+c_i} \left[\sigma' + \kappa_L(t - \tau_{i'} - \sigma') + \kappa_H(\tau_i + \sigma' - t)\right] f(t)dt & \text{si } \begin{cases} i < n \\ a = 1 \end{cases} \\ 0 & \text{si } \begin{cases} i < n \\ a = 0 \end{cases} \\ \int_{\tau_{i'}+\sigma'}^{\tau_n+\sigma'+c_n} \left[\sigma' + \kappa_L(t - \tau_{i'} - \sigma') + \kappa_H(\tau_n + \sigma' - t)\right] f(t)dt & \text{si } i = n \end{cases} \tag{4.2}$$

- Le premier cas où $i < n$ et $a_i = 1$: cette expression exprime l'espérance du temps total perdu entre les tâches $job_{i'}, job_i$ et elle est composée des trois termes du temps perdu présentés ci-dessous :
  (a) Le surcoût accumulé des points de sauvegarde noté par σ' qui représente le surcoût accumulé des points de sauvegarde depuis le début jusqu'à l'instant de décision i.
  (b) La quantité de travail perdu si jamais une panne surgit à l'instant t. Elle représente la durée de temps entre le dernier point de sauvegarde et l'instant de panne t. Cette quantité est donnée par $\kappa_L(t - \tau_{i'} - \sigma')$ avec $\kappa_L \in \mathbb{R}_+$ un coefficient arbitraire qui peut être utilisé pour pondérer cette durée par rapport aux autres pertes. Par exemple dans le domaine des bases de données le coût de re-exécutions du travail perdu est généralement inférieur au coût de la première exécution donc $\kappa_L \in]0, 1]$.
  (c) La durée de temps nécessaire à la détection qui est donnée par le dernier terme $\kappa_H(\tau_i + \sigma' - t)$ avec $\kappa_H \in \mathbb{R}_+$ un coefficient arbitraire de pondération. Par exemple $\kappa_H = 0$ si nous voulons négliger la durée de détection.

  D'où l'espérance du temps total perdu est obtenue en pondérant tous ces termes par la probabilité qu'une panne surgisse à l'instant t qui vérifie $\tau_{i'} + \sigma' < t \leq \tau_i + \sigma' + c_i$.
- Le deuxième cas où $i < n$ et $a_i = 0$ l'application choisit l'action "ignorer la $i^{éme}$ sauvegarde" dans ce cas l'application ne change pas d'état et la perte est nulle.
- Dans le dernier cas où $i = n$ par convention l'état de l'application évolue directement vers l'état final où le dernier point de sauvegarde est forcément réalisé et la perte est calculée de la même manière que le premier cas de figure.

En se basant sur cette abstraction le problème d'ordonnancement des points de sauvegarde noté par CS_F (*i.e. Checkpoint Scheduling*) est défini formellement comme suit :

- ENTRÉE : $2n$ nombres positifs notés par $\{p_1, p_2, \cdots, p_n\}, \{c_1, c_2, \cdots, c_n\}$.

– FONCTION OBJECTIF : $W : \{0,1\}^n \to \mathbb{R}$ définie mathématiquement par :

$$W : \pi \to \sum_{i=1}^{n} \delta_i(e_{i'}^{\sigma_{i'}}, a_i),$$

Nous notons que i' représente l'indice du dernier point de sauvegarde avant l'instant i qui est donné par la fonction $i' = \arg\max_{j<i} a_j \tau_j$ et $\sigma_{i'} = \sum_{j=1}^{i'} a_j c_j$.

– SORTIE : Une stratégie d'ordonnancement des points de sauvegarde optimale notée par $\pi^* \in \{0,1\}^n$ qui vérifie :

$$\forall \pi \in \{0,1\}^n,\ W(\pi^*) \leq W(\pi).$$

4.3.3 Ordonnancement optimal des points de sauvegarde

Nous exhibons dans cette partie la description de l'algorithme proposé pour calculer la stratégie d'ordonnancement des points de sauvegarde π^*. L'optimalité de cette solution découle directement du modèle MDP et elle s'appuie sur les trois éléments suivants. Le premier élément est la cardinalité de l'ensemble des instants de décision qui est finie. Le deuxième élément est lié à la relation entre l'action choisie et la fonction de transition. Nous notons que dans le modèle présenté la décision à l'instant i détermine avec probabilité 1 l'état du système à l'instant $i+1$. Finalement, nous rappelons que la fonction objectif du problème est la somme des pertes que le système récolte à chaque instant de décision. Par conséquent en s'appuyant sur ces trois éléments, calculer π^* est équivalent à calculer un chemin avec un coût minimal dans un graphe orienté [83]. Les nœuds de ce graphe sont les états que l'application est susceptible d'occuper et les arcs entre les nœuds sont définis à travers les fonctions $\eta_i(e_{i'}^{\sigma'}, a)$ et $\delta_i(e_{i'}^{\sigma'}, a)$. Dans la suite de cette partie nous présentons la transformation du modèle MDP en un problème de recherche de plus court chemin dans un graphe orienté.

Soit $G(E,V)$ un graphe orienté, où E représente l'ensemble des nœuds et V représente l'ensemble des arcs dans ce graphe. Le description formelle de E est assez évidente, puisque E est exactement le même ensemble que celui décrit précédemment tel que chaque nœuds est étiqueté par le couple (i,σ). La définition de l'ensemble V repose sur l'évaluation des fonctions η et δ à chaque instant de décision. En effet chaque arc dans V représente une transition possible entre deux états du système et le coût associé à cet arc est donné par la fonction perte. Plus précisément la fonction de transition donnée par l'Expression 4.1 stipule qu'à chaque instant de décision i, soit l'état du système reste le même, soit il passe de l'état $e_{i'}^{\sigma'}$ à l'état $e_i^{\sigma'+c_i}$ tel que i' vérifie $i' < i$. Dans ce cas, il y a forcément un arc sortant du nœud $e_{i'}^{\sigma'}$ vers le nœud e_i^{σ}, si les nœuds vérifient $i' < i \leq n$ et $\sigma' + c_i = \sigma$. Le coût associé à cet arc est donné par la fonction $\delta_i(e_{i'}^{\sigma'}, 1)$.

L'algorithme de création de graphe est le suivant. Dans un premier temps le graphe contient seulement le nœud initial $G(E,V) = (\{e_0^0\}, \emptyset)$. Dans un deuxième temps à partir du nœud initial, nous procédons à la création de tous les nœuds e_i^{σ} qui vérifient $i \leq n$ et $\sigma = c_i$ avec un arc à partir de l'état initial vers ce dernier dont le coût associé est donné par $\delta_i(e_0^0, 1)$. Ceci représente le cas de base. Une fois que tous les successeurs du nœud initial sont créés, nous passons à l'exploration d'un autre nœud non exploré. Formellement, soit $e_{i'}^{\sigma'}$ un nœud pas encore exploré avec une étiquette d'indice qui vérifie $i' < n$. Pour chaque valeur de i qui vérifie $i' < i \leq n$ créer le nœud $e_i^{\sigma'+c_i}$ et le

marquer comme inexploré s'il n'existe pas. Ensuite, créer un arc à partir de $e_{i'}^{\sigma'}$ vers $e_i^{\sigma'+c_i}$ dont le coût est donné par $\delta_i(e_{i'}^{\sigma'}, 1)$. Une fois que tous les successeurs possibles de $e_{i'}^{\sigma'}$ sont visités le nœud est marqué comme totalement exploré. Finalement, il reste la création des arcs avec un coût nul vers le nœud terminal e^∞ à partir des nœuds qui ont n comme étiquette d'indice et n'importe quelle valeur pour σ. Par conséquent il est évident que $\text{Card}(E) = \mathcal{O}\left(n^2 c^{max}\right)$ et $\text{Card}(V) = \mathcal{O}\left(n^3 c^{max}\right)$, puisque les étiquettes des nœuds varient dans les intervalles $1 \leq i \leq n$ et $0 \leq \sigma \leq nc^{max}$ et à partir de chaque nœud il y au plus n arcs sortant. Donc pour calculer la stratégie d'ordonnancement des points de sauvegarde optimale qui minimise le temps total perdu. Il faut tout d'abord créer le graphe $G(E, V)$ avec les règles définies précédemment en fonction des entrées du problème. Ensuite, il suffit de calculer le plus court chemin entre les nœuds e_0^0 et e^∞ dans ce graphe. Nous donnons dans la section suivante le schéma de programmation dynamique que nous proposons pour calculer π^* en s'appuyant sur cette modélisation.

4.4 Analyse de complexité

Nous analysons dans cette section à la fois la complexité de l'algorithme proposé pour calculer π^* et la complexité du problème CS_F dans le cadre général.

4.4.1 Analyse de complexité de l'algorithme proposé

a Analyse de complexité dans le cas général

Nous analysons dans cette partie l'algorithme dans le cadre général où les entrées du problème sont les deux vecteurs arbitraires $\{p_1, p_2, \cdots p_n\}$, $\{c_1, c_2, \cdots c_n\}$ qui décrivent l'application. Nous tenons à préciser que la fonction $F : \mathbb{R}_+ \to [0, 1]$ qui représente la distribution d'inter-arrivées des pannes ne fait pas partie de l'instance et elle est considérée comme une fonction pre-codée dans le problème CS_F.

Nous commençons par l'introduction des notations utilisées par la suite. Soient Π_i^σ un sous ensemble de $\{0, 1\}^i$ et $\pi_i^\sigma = \{a_1, a_2, \cdots, a_i\} \in \{0, 1\}^i$ une stratégie d'ordonnancement des points de sauvegarde pour une application qui contient i tâches. L'ensemble Π_i^σ est défini formellement par les contraintes suivantes :

$$\pi_i^\sigma \in \Pi_i^\sigma \text{ si et seulement si } \begin{cases} a_i = 1. \\ \sum_{j=1}^{i} a_j c_j = \sigma. \end{cases} \tag{4.3}$$

Plus précisément l'ensemble Π_i^σ contient toute stratégie d'ordonnancement des points de sauvegarde pour une application composée de i tâches avec forcément un point de sauvegarde après la dernière tâche ($a_i = 1$). De plus la deuxième contrainte stipule que le surcoût accumulé des points de sauvegarde dans cette stratégie est exactement σ. Nous rappelons que i' représente l'indice de dernier point de sauvegarde avant i donné par $i' = \arg\max_{j<i} a_j \tau_j$. En se basant sur la définition de i', soit $\pi_{i'}^{\sigma'}$ un sous vecteur de π_i^σ qui contient exactement les mêmes valeurs des a_j pour $1 \leq j \leq i'$ avec $\sigma' = \sigma - c_i$. En s'appuyant sur les définitions de π_i^σ, i' et $\pi_{i'}^{\sigma'}$, la fonction objectif qui représente le

temps total perdu sous forme récursive est exprimée ci-dessous :

$$W(\pi_i^\sigma) = W(\pi_{i'}^{\sigma'}) + \delta_i(e_{i'}^{\sigma'}, 1) = W(\pi_{i'}^{\sigma'}) + \int_{\tau_{i'}+\sigma'}^{\tau_i+\sigma} (\sigma' + \kappa_L(t - \tau_{i'} - \sigma') + \kappa_H(\tau_i + \sigma' - t))\, f(t)dt. \quad (4.4)$$

Nous rappelons que Π_i^σ est un ensemble fini d'où l'existence d'une stratégie d'ordonnancement optimale appartenant à cet ensemble qui minimise la fonction $W(\pi_i^\sigma)$ définie par :

$$\forall \pi_i^\sigma \in \Pi_i^\sigma,\ W(\pi_i^{\sigma *}) = W_{i,\sigma}^* \leq W(\pi_i^\sigma). \quad (4.5)$$

En se basant sur cette construction nous établissons la proposition suivante :

Proposition 4.1. *Si π_i^σ est un ordonnancement optimal dans Π_i^σ alors $\pi_{i'}^{\sigma'}$ est un ordonnancement optimal dans $\Pi_{i'}^{\sigma-c_i}$.*

Démonstration. Avant toute chose nous notons que $\pi_{i'}^{\sigma'} \in \Pi_{i'}^{\sigma-c_i}$ puisque par construction $\sigma' = \sigma - c_i$ et $a_{i'} = 1$.
Considérons un ordonnancement optimal $\pi_i^\sigma \in \Pi_i^\sigma$ tel que $W(\pi_i^\sigma) = W_{i,\sigma}^*$.
Supposons par contradiction que le sous vecteur de π_i^σ noté $\pi_{i'}^{\sigma'}$ n'est pas un ordonnancement optimal

$$W(\pi_{i'}^{\sigma'}) > W_{i',\sigma'}^*$$

Cette hypothése implique l'existence d'un autre ordonnancement $\tilde{\pi}_{i'}^{\sigma'} \neq \pi_{i'}^{\sigma'}$ qui vérifie :

$$\tilde{\pi}_{i'}^{\sigma'} \in \Pi_{i'}^{\sigma'} \text{ et } W(\tilde{\pi}_{i'}^{\sigma'}) < W(\pi_{i'}^{\sigma'}).$$

D'où la contradiction.

$$W(\tilde{\pi}_{i'}^{\sigma-c_i}) + \delta_i(e_{i'}^{\sigma'}, 1) < W(\pi_i^\sigma) < W_{i,\sigma}^*.$$

□

En s'appuyant sur la Proposition 4.1 et en utilisant la récursivité sur l'indice du dernier point de sauvegarde comme l'indique l'Expression 4.6, le schéma de programmation dynamique pour calculer $W_{i,\sigma}^*$ est donné par l'Algorithme 1 :

$$\forall i \leq n, \forall \sigma \leq ic^{max},\ W_{i,\sigma}^* = \min_{1 \leq i' < i} \{W_{i',\sigma-c_i}^* + \delta_i(e_{i'}^{\sigma-c_i})\} \quad (4.6)$$

Donc soit W_n^* la valeur optimale de l'espérance du temps total perdu qu'une stratégie peut réaliser pour une application composée de n tâches. Alors cette valeur est donnée par :

$$W_n^* = \min_{0 \leq \sigma \leq nc^{max}} W_{n,\sigma}^*.$$

En conséquence la complexité algorithmique en temps et en espace est une complexité pseudo-polynomiale puisque la complexité en temps est en $\mathcal{O}(n^3 c^{max})$ tandis que la complexité mémoire est en $\mathcal{O}(n^2 c^{max})$.

Algorithme 1 Calculer $W(i, \sigma)$

```
1 : function DP_WASTED_TIME(i, σ)
2 :     if wasted[i][σ] is defined then                ▷ Si la valeur est déjà dans le cache
3 :         return wasted[i][σ]
4 :     end if
5 :     if σ < c_i then                                ▷ Configuration impossible
6 :         wasted[i][σ] ← ∞
7 :         return wasted[i][σ]
8 :     end if
9 :     if σ = c_i then                                ▷ Le cas de base
10 :        wasted[i][σ] ← δ_i(e_0^0, 1)
11 :        return wasted[i][σ]
12 :    end if
13 :    wasted[i][σ] ← min_{0<i'<i}{DP_WASTED_TIME(i', σ − c_i) + δ_i(e_{i'}^{σ−c_i}, 1)}
14 :    return wasted[i][σ]
15 : end function
```

b Cas surcoût des points de sauvegarde constant

En utilisant une analyse similaire nous montrons dans cette partie que la solution donnée précédemment peut être simplifiée dans le cas où les points de sauvegarde ont le même surcoût, pour aboutir à une complexité polynomiale en fonction de n. Ainsi si tous les points de sauvegarde ont un surcoût constant σ est nécessairement un multiple de c tel que $\exists l \in \mathbb{N} |\, l = \sigma/c$. Donc il est évident que le surcoût accumulé des points de sauvegarde peut être décrit seulement moyennant le nombre des sauvegardes réalisées (noté par l) qui varie dans $[1..n]$.

Soient Π_i^l l'ensemble des stratégies d'ordonnancement des points de sauvegarde pour i tâches et qui contiennent exactement l sauvegardes et $W_{i,l}^*$ la valeur minimale qu'une stratégie de cet ensemble peut atteindre. Par analogie avec l'analyse précédente le schéma de programmation dynamique est donné par l'expression suivante :

$$\forall i \leq n, \forall l \leq i, \;\; W_{i,l}^* = \min_{l \leq i' < i} \{W_{i',l-1}^* + \delta_i(e_{i'}^{(l-1)c}, 1)\}. \tag{4.7}$$

Dans ce cas, la complexité en temps de la solution proposée est en $\mathcal{O}(n^3)$ au lieu de $\mathcal{O}(n^3 c^{max})$. De même pour la complexité en mémoire qui est simplement bornée par $\mathcal{O}(n^2)$. En s'appuyant sur cette analyse nous établissons le corollaire suivant.

Corollaire 1. Le problème CS_F appartient à la classe de complexité P si l'instance vérifie la contrainte suivante $\forall i \leq n, \; c_i = c$.

4.4.2 Analyse de complexité du problème

Nous étudions dans cette partie la complexité du problème d'ordonnancement des points de sauvegarde en général. Nous commençons avant toute chose par la présentation de la version décisionnelle du problème d'optimisation défini précédemment. Ensuite, nous décrivons une famille d'instances ainsi que quelques hypothèses sur la distribution de panne qui seront utilisées pour simplifier le problème. Finalement, en utilisant cette

famille d'instances nous démontrons que le problème CS_F appartient à la classe de complexité *NP-complet*.

a Définition du problème décisionnel relaxé

Cette analyse est menée sur la version décisionnelle du problème relaxé défini formellement comme suit :

- ENTRÉE : $2n$ nombres rationnels notés par $\{p_1 \cdots p_n\}$; $\{c_1 \cdots c_n\}$ et un objectif G.
- QUESTION : existe-t-il $\pi \in \{0,1\}^n$ qui vérifie $W_r(\pi) \leq G$?

$W_r : \{0,1\}^n \to \mathbb{R}$ représente la fonction objectif relaxée de l'espérance du temps total perdu. Deux relaxations sont introduites dans cette fonction par rapport à la fonction objectif du départ.

La première relaxation concerne les deux variables de pondération κ_L et κ_H. Dans la suite, nous considérons la formulation classique du problème où les pannes sont détectées immédiatement $\kappa_H = 0$ et le temps nécessaire pour la re-exécutions du travail perdu est exactement le même $\kappa_L = 1$. Nous notons par $\sigma_{i-1} = \sum_{j=1}^{i-1} a_j c_j$ le surcoût accumulé des points de sauvegarde depuis le début jusqu'à la $i-1^{éme}$ sauvegarde. Ainsi, en se basant sur la modélisation décrite dans la section précédente et les deux hypothèses ($\kappa_H = 0$ et $\kappa_L = 1$), la quantité de travail perdu si jamais une panne surgit dans l'intervalle $\tau_{i-1} + \sigma_{i-1} < t \leq \tau_i + \sigma_i$ est donnée par $t - \max_{i'<i}(a_{i'}\tau_{i'} + \sigma_{i'})$. Donc la fonction W_r est définie formellement par :

$$W_r : \pi \to \sum_{i=1}^{n} \int_{\tau_{i-1}+\sigma_{i-1}}^{\tau_i+\sigma_i} \left(\sigma_{i-1} + t - \max_{0\leq i'<i}(a_{i'}\tau_{i'} + \sigma_{i'})\right) f(t)dt. \tag{4.8}$$

Ensuite comme seconde relaxation nous considérons que la distribution d'inter-arrivées des pannes $F(t)$ est la distribution uniforme continue sur l'intervalle $[\alpha, \beta]$ définie par $F : t \to (t-\alpha)/(\beta-\alpha)$. Pour alléger la notation nous supposons que $\alpha = 0$ d'où $F(t)$ et $f(t)$ sont définies respectivement par $F : t \to t/\beta$ et $f : t \to 1/\beta$. En se basant sur cette simplification la fonction objectif $W_r : \{0,1\}^n \to \mathbb{R}$ est définie comme suit :

$$W_r : \pi \to \frac{(\tau_n + \sigma_n)^2}{2\beta} - \frac{1}{\beta}\sum_{i=1}^{n}\left((p_i + a_i c_i)(\max_{i'<i} a_{i'}\tau_{i'})\right). \tag{4.9}$$

Sans perte de généralité nous notons que la minimisation de la fonction objectif 4.9 est équivalente á la minimisation de la fonction objectif 4.10.

$$W_r : \pi \to \frac{(\tau_n + \sigma_n)^2}{2} - \sum_{i=1}^{n}\left((p_i + a_i c_i)(\max_{i'<i} a_{i'}\tau_{i'})\right). \tag{4.10}$$

Dans la suite du document nous notons cette variante du problème par CS_U (U pour distribution uniforme) au lieu de CS_F.

b Définition de la famille d'instances

Nous présentons dans cette partie une description d'une famille d'instances particulières décrite formellement par une assignation bien déterminée des surcoûts des points de sauvegarde. Cette assignation a pour objectif la simplification de la fonction

objectif en remplaçament l'opérateur max par un opérateur linéaire. L'idée intuitive de cette assignation est d'imposer la prise d'une sauvegarde sur deux, en supposant que les points de sauvegarde avec un indice pair ont un surcoût nul. En s'appuyant sur cette hypothèse sur l'instance d'entrée, nous démontrons dans le lemme suivant que toute stratégie d'ordonnancement optimale est forcée de prendre la $i^{éme}$ sauvegarde si $c_i = 0$.

Lemme 4.1. *Si $c_i = 0$ alors $a_i = 1$ dans toute stratégie d'ordonnancement des points de sauvegarde optimale.*

Démonstration. Considérons une instance particulière où le surcoût du $i^{éme}$ point de sauvegarde est nul $c_i = 0$.
Supposons par contradiction que $\tilde{\pi} \in \{0,1\}^n$ est une stratégie optimale pour cette instance particulière tel que $\tilde{a}_i = 0$.
Nous introduisons la stratégie $\hat{\pi} \in \{0,1\}^n$ qui contient les mêmes décisions que celles de $\tilde{\pi}$ sauf pour l'indice i tel que $\hat{a}_i = 1 - \tilde{a}_i$.
Soit i' l'indice du dernier point de sauvegarde avant la $i^{ème}$ tâche. Après quelques manipulations mathématiques sur l'Expression 4.10, $W_r(\hat{\pi})$ devient :

$$W_r(\hat{\pi}) = W_r(\tilde{\pi}) + \frac{c_i^2}{2} + c_i(\tau_n + \sum_{j \neq i} a_j c_j) - (\tau_i - \tau_{i'})(p_{i+1} + a_{i+1}c_{i+1}).$$

Nous rappelons que $c_i = 0$ et $(\tau_i - \tau_{i'})(p_{i+1} + a_{i+1}c_{i+1}) \geq p_i(p_{i+1} + a_{i+1}c_{i+1}) > 0$.
D'où la contradiction :

$$W_r(\pi') - W_r(\tilde{\pi}) < 0.$$

□

Sans perte de généralité, dans la suite de cette analyse nous supposons que n est pair et I_o dénote l'ensemble des entiers impairs dans l'intervalle $[1..n]$, la famille d'instances est définie par :

$$\forall i \leq n,\ i \notin I_o \Rightarrow c_i = 0.$$

Dans la suite du document nous notons cette variante du problème par $CS_{U\&O}$ (*i.e. Checkpoint Scheduling problem under Uniform and Odd assumptions*).
En s'appuyant sur le Lemme 4.1 et cette famille d'instances. Le terme $\max_{i'<i} a_{i'}\tau_{i'}$ est donné par :

$$\max_{i'<i} a_{i'}\tau_{i'} = \begin{cases} \tau_{i-1} & si \quad i \in I_o \\ a_{i-1}\tau_{i-1} + \overline{a}_{i-1}\tau_{i-2} & si \quad i \notin I_o \end{cases} \tag{4.11}$$

Comme conséquence directe, la fonction objectif relaxée est définie comme suit :

$$W_r : \pi \rightarrow \frac{(\tau_n + \sigma_n)^2}{2} - \sum_{i \in I_o} \left(\tau_{i-1}(p_i + p_{i+1}) + a_i(\tau_{i-1}c_i + p_i p_{i+1})\right). \tag{4.12}$$

Nous isolons le terme $\tau_n^2/2 - \sum_{i \in I_o} \tau_{i-1}(p_i + p_{i+1}) = K$ qui est un terme constant pour n'importe quelle stratégie une fois que l'instance est fixée, la fonction objectif W_r est alors donnée par :

$$W_r : \pi \rightarrow K + \sigma_n^2/2 - \sum_{i \in I_o} a_i c_i(\tau_{i-1} - \tau_n + \frac{p_i p_{i+1}}{c_i}). \tag{4.13}$$

c La NP-complétude du problème

Nous exhibons dans cette partie la réduction utilisée pour démontrer que le problème $CS_{U\&O}$ se réduit à un autre problème qui appartient déjà à la classe *NP-complet*. Le problème retenu dans cette démonstration est le problème *Subset Sum* [43] formellement défini par :

- ENTRÉE : Un ensemble $M = \{s_1, s_2 \cdots s_m\}$ de m nombres positifs, et un entier S.
- QUESTION : Existe-t-il un sous ensemble de $\{1, 2, \cdots, m\}$ noté par Q qui vérifie $\sum_{i \in Q} s_i = S$?

Théorème 4.1. *La variante $CS_{U\&O}$ du problème d'ordonnancement des points de sauvegarde appartient à la classe de complexité NP-complet.*

Démonstration. Nous notons que $CS_{U\&O} \in NP$, puisque un algorithme déterministe en $\mathcal{O}(n)$ peut être utilisé pour vérifier que $W_r(\pi) \leq G$.
Nous entamons la description de la réduction entre les instances de $CS_{U\&O}$ et *Subset Sum* :

- Assignation des surcoûts des points de sauvegarde :
  Pour chaque s_i dans M, nous associons deux points de sauvegarde définies par $c_{2i-1} = s_i$ et $c_{2i} = 0$, ceci implique que $n = 2m$.
- Assignation des durées des tâches :
  Nous tenons à souligner que l'idée principale de cette réduction repose l'exhibition de l'égalité ci-dessous dans la fonction objectif donnée par l'Expression 4.13 :

$$\sum_{i \in I_o} a_i c_i (\tau_{i-1} - \tau_n + \frac{p_i p_{i+1}}{c_i}) = \sum_{i \in I_o} a_i c_i S. \tag{4.14}$$

  D'où les valeurs des p_i sont choisies de telle façon qu'elles vérifient cette égalité. Ainsi pour chaque i dans l'intervalle $[1..m]$, nous créons deux tâches telles que :

$$\forall i \in [1..m],\ p_{2i} = 2c_{2i-1} \text{ et } p_{2i-1} = S + \sum_{j=2i}^{2m} p_j$$

- Assignation des objectifs S et G :
  Finalement la relation entre les deux objectifs est donnée par : $G = K - S^2/2$.
- $\Rightarrow$
  Supposons que le problème *Subset Sum* a une instance positive notée par Q qui vérifie $\sum_{i \in Q} s_i = S$.
  Considérons la stratégie d'ordonnancement π définie de la manière suivante : $\forall i \leq m,\ a_{2i} = 1$, tous les points de sauvegarde avec indice pair sont forcément réalisés.
  Ensuite $\forall i \leq m$ si $i \in Q$ alors $a_{2i-1} = 1$ sinon $a_{2i-1} = 0$.
  Par construction les deux égalités suivantes sont valides :

$$\forall i \in I_o,\ \tau_{i-1} - \tau_n + \frac{p_i p_{i+1}}{c_i} = S \text{ et } \sigma_n = \sigma_{2m} = S.$$

En remplaçant ces termes dans la fonction objectif donnée dans l'Expression 4.13 nous obtenons.

$$W_r(\pi) = K + \sigma_n^2/2 - S\sigma_n = G.$$

Donc $CS_{U\&O}$ a une instance positive.

– $\Leftarrow$
Pour démontrer le sens inverse, supposons que $CS_{U\&O}$ a une instance positive π qui vérifie $W_r(\pi) \leq G$.
D'où par construction l'inégalité suivante est vraie :

$$K + \sigma_n^2/2 - \sum_{i \in I_o} a_i c_i (\tau_{i-1} - \tau_n + \frac{p_i p_{i+1}}{c_i}) = K + \sigma_n^2/2 - S\sigma_n \leq K - S^2/2.$$

Comme l'inégalité $(\sigma_n - S)^2 \leq 0$ a une seule solution qui est $S = \sigma_n$ alors le problème *Subset sum* a aussi une instance positive.

□

En s'appuyant sur cette analyse nous concluons que le problème CS_F est *NP-complet* au sens faible puisque il existe un algorithme optimal de résolution qui a une complexité pseudo-polynomiale en la taille de l'instance.

4.5 État de l'art

Dans cette section nous présentons les principales autres approches de modélisation discrète du problème d'ordonnancement des points de sauvegarde. Par rapport au nombre des travaux qui sont fondés sur une approche de modélisation continue, très peu de travaux ont été menés dans cette direction.

Sam *et al.* [97] : À notre connaissance, la première formulation du problème sous une forme discrète est proposée par Sam *et al.* [97]. Dans ce travail, les auteurs supposent que l'application est composée de n tâches. Chaque tâche a une durée d'exécution notée par p_i, un surcoût de sauvegarde c_i et un coût de redémarrage r_i si jamais l'application redémarre à partir de cette tâche après une panne. Pour modéliser l'inter-arrivées des pannes ils considèrent deux modèles. Dans un premier temps, ils proposent d'associer une probabilité de succès à chaque tâche notée par q_i. Tandis que dans le deuxième modèle la probabilité de panne est tous simplement décrite par une fonction de distribution arbitraire F. Moyennant une analyse légèrement différente à celle présentée dans le chapitre précédent pour exprimer l'Expression 3.9 de l'espérance du temps de complétion en fonction de la quantité de travail. Les auteurs expriment l'espérance du temps de complétion des tâches entre les indices i' et i de la façon suivante :

$$\mathbb{E}\left[T_{i':i}\right] = \begin{cases} \frac{1}{q_i}(\mathbb{E}\left[T_{i':i-1}\right] + p_i) + r_i(\frac{1}{q_i} - 1) & \text{Modèle avec } q_i \\ \dfrac{r_i F(\tau_i - \tau_{i'}) + \int_0^{\tau_i - \tau_{i'}} \overline{F}(t)}{(1 - F(\tau_i - \tau_{i'}))} & \text{Modèle distribution continue } F(t) \end{cases} \tag{4.15}$$

Les expressions obtenues dans les cas des deux modèles de panne sont fondées sur l'hypothèse de la loi sans mémoire : l'expression de l'espérance du temps de complétion

des tâches entre les indices i' et i ne dépend pas des durées des tâches réalisées avant la tâche i' dans le même cycle de panne. En se basant sur cette modélisation, les auteurs expriment l'espérance du temps de complétion des tâches entre l'indice 0 et i notée par $\mathbb{E}\left[T_{0:i}^{k}\right]$ en fonction du nombre des points de sauvegarde k de la manière suivante :

$$\mathbb{E}\left[T_{0:i}^{k}\right] = \mathbb{E}\left[T_{0:i'}^{k-1}\right] + \mathbb{E}\left[T_{i':i}\right] + c_{i'}. \tag{4.16}$$

Pour calculer la stratégie d'ordonnancement optimale pour exactement k points de sauvegarde $\mathbb{E}\left[T_{0:n}^{k}\right]^{*}$, ils proposent un schéma de programmation dynamique basé l'expression récursive suivante :

$$\mathbb{E}\left[T_{0:i}^{k}\right]^{*} = \min_{k-1 \leq i' < i} \{\mathbb{E}\left[T_{0:i'}^{k-1}\right]^{*} + \mathbb{E}\left[T_{i':i}\right] + c_{i'}\}. \tag{4.17}$$

Ainsi il est évident que le coût algorithmique de cette stratégie est en $\mathcal{O}\left(n^3\right)$. De plus, les auteurs montrent que si l'ordre entre les surcoûts des points de sauvegarde est le même ordre entre les coûts de redémarrage autrement dit si $c_i \leq c_j \Rightarrow r_i < r_j$. L'algorithme proposé est en $\mathcal{O}\left(n^2\right)$. Ensuite pour étendre le modèle et prendre en compte le fait qu'une panne peut surgir durant la sauvegarde, les auteurs rajoutent cette durée dans l'expression $\mathbb{E}\left[T_{0:i'}^{k-1}\right]$ en considérant $c_{i'}$ comme une tâche supplémentaire. Pour résumer, le travail proposé par Sam *et al.* [97] exprime l'espérance du temps de complétion en fonction de la stratégie d'emplacement des points de sauvegarde. En se basant sur cette modélisation, ils proposent un schéma de programmation dynamique pour calculer la stratégie d'ordonnancement des points de sauvegarde qui minimise l'espérance du temps de complétion. Par contre : Ils considèrent que la distribution de panne est sans mémoire ce qui implique que l'expression de la quantité $\mathbb{E}\left[T_{i':i}\right]$ ne dépend pas du nombre des sauvegardes réalisées dans l'expression $\mathbb{E}\left[T_{0:i'}^{k-1}\right]$ d'où la complexité polynomiale de l'algorithme du calcul de l'ordonnancement optimal.

Bougeret *et al.* [14] : Une autre contribution dans la même philosophie, est proposée tout récemment en 2011 dans [14] et où les auteurs se focalisent sur les applications parallèles en faisant le lien entre les approches de modélisation continue et celles fondées sur une description discrète. Ils considèrent que l'application est composée par un nombre fini d'instructions w qui représente la quantité de travail. Ensuite, pour se ramener au modèle d'application dont les tâches sont unitaires, ils discrétisent la quantité de travail en un ensemble de pas fini où chaque tâche est de taille $p_i = \frac{w}{u}$, tel que u représente le pas d'échantillonnage (pas de discrétisation). Ainsi ils passent d'une modélisation continue à une modélisation discrète. Concernant le modèle de sauvegarde ils supposent que les points de sauvegarde ont un surcoût constant. Pour modéliser les inter-arrivées des pannes une distribution arbitraire est utilisée. En se basant sur cette abstraction ils proposent dans un premier temps un schéma de programmation dynamique dont la complexité est bornée $\mathcal{O}\left(\frac{w^3}{u}(1 + c/u)\right)$ pour calculer la stratégie d'ordonnancement des points de sauvegarde qui minimise l'espérance du temps de complétion. Dans un deuxième temps pour améliorer cet algorithme qui est très couteux en temps de calcul vue sa complexité pseudo-polynomiale et résoudre le problème avec des grandes valeurs de w. Ils considèrent une fonction objectif qui est équivalente à celle que nous considérons dans ce travail dont l'objectif est la minimisation de l'espérance du temps total perdu

durant un cycle de panne. Par conséquent ils proposent une solution algorithmique en $\mathcal{O}\left(\frac{w^3}{u}\right)$ qui est équivalente à celle présentée dans cette thèse mais avec des restrictions sur le modèle d'application (tâches unitaires) ainsi que le modèle du surcoût des points de sauvegarde (surcoût constant). Un point très important est soulevé dans ce travail en comparant à travers des simulations les performances de la solution conçue pour la minimisation de l'espérance du temps total perdu avec celle conçue pour la minimisation de l'espérance du temps de complétion. Ainsi, moyennant des simulations, ils concluent que les deux algorithmes ont des performances équivalentes si nous nous intéressons à la minimisation de l'espérance du temps de complétion. Autrement dit, ces simulations indiquent que minimiser l'espérance du temps de complétion global revient à minimiser l'espérance du temps total perdu dans chaque cycle de panne.

Pour finir cette section d'étude de contributions existantes. Nous mentionnons que la plupart des travaux présentés contiennent des hypothèses qui sont discutables. La première hypothèse concerne la distribution de panne qui est supposée sans mémoire. La deuxième restriction concerne le modèle d'application dans lequel les durées des tâches ainsi que le surcoût des points de sauvegarde sont supposés constants.

4.6 Étude expérimentale

Nous présentons dans cette section une étude expérimentale dans le contexte des plateformes de calcul distribué fondées sur le principe du calcul volontaire (*i.e. volunteer computing*). Le modèle d'exécution ainsi que les paramètres sous-jacents utilisés pour conduire cette étude sont issus de la plateforme BOINC (*Berkeley Open Infrastructure for Network Computing*) [3]. Cette plateforme contient plusieurs projets de calcul scientifiques tels que SETI@home, climateprediction.net et EINSTEIN@home *etc.* Actuellement, BOINC fournit une puissance de calcul d'environ 7 PetaFLOPS [91, 4, 68, 41] pour 50 projets scientifiques de divers domaines (prédiction de climat, décodage de génome, détection de vie extra terrestre *etc*). Ces projets ont produit une centaine de contributions scientifiques [12] publiées dans des revues internationales très prestigieuses telles que *Science* ou *Nature*. Cependant, dans ce contexte de calcul bénévole, les applications doivent faire face à un défi majeur qui est la volatilité et l'indisponibilité des clients durant des intervalles de temps non négligeables. Nous citons l'exemple des clients volontaires qui participent au projet SETI@Home où la médiane des durées de disponibilité est d'environ 1 heure et la durée moyenne d'indisponibilité est d'environ 4.5 heures [64]. Alors que le temps de complétion des instances d'exécution du projet SETI@Home peut atteindre 5 jours [13]. Donc ces applications doivent faire face à ces défis et prendre en compte la volatilité et l'indisponibilité des ressources de calcul.

Deux objectifs sont visés par cette campagne de simulations. Premièrement, nous validons et nous comparons notre approche d'ordonnancement des points de sauvegarde avec les approches existantes. Deuxièmement, nous mettons en évidence à travers les simulations, qu'il est possible de faire face à ces challenges en utilisant le mécanisme de sauvegarde et reprise. L'utilisation du mécanisme de sauvegarde et reprise dans BOINC n'est pas une idée nouvelle. Par contre dans ces simulations nous mettons en place une autre approche de tolérance aux pannes en couplant le mécanisme de sauvegarde et reprise avec et le mécanisme de duplication. Le principe de notre approche est le suivant. Apres avoir reçu du travail au fur et à mesure de l'avancement de l'exécution le client

envoie au serveur des points de sauvegarde qui contiennent des résultats intermédiaires du calcul. Dans le cas où le client subit une panne, le serveur procède à la duplication de l'exécution sur un autre client en lui envoyant la sauvegarde la plus récente de l'application.

Dans la suite de cette section, nous présentons le modèle d'environnement d'exécution considéré pour reproduire la plateforme BOINC. Ensuite, nous décrivons la méthodologie expérimentale adoptée pour conduire les expériences. Finalement, nous analysons les résultats obtenus à l'issue de cette étude expérimentale.

4.6.1 Modèle de l'environnement d'exécution

Dans cette partie nous présentons tout d'abord le modèle d'application sur lequel repose les applications de type *desktop grid*. Ensuite, nous décrivons le modèle du réseau à travers lequel les clients et le serveur communiquent puis, nous passons à la description du modèle de panne utilisé pour modéliser la volatilité des clients. Finalement, nous finissons avec une courte description de l'outil développé pour simuler la plateforme BOINC.

a Modèle d'application

Typiquement, les applications de type *desktop grid* reposent sur le principe Client/-Serveur. Dans ce modèle les clients sont les entités volontaires disponibles durant une durée de temps aléatoire pour exécuter bénévolement du travail et récupérer des crédits virtuels. Le serveur est le responsable de la distribution du travail et de la récolte des résultats, demandés ou calculés par les clients [3, 13]. Typiquement, les projets exécutés sur la plateforme BOINC, sont des applications de type *bag of tasks* que nous appelons campagne d'exécution. Comme l'indique Estrada *et al.* [39], les scientifiques qui utilisent BOINC, soumettent une campagne d'exécution du même code exécutable mais avec des données d'entrée différentes. Ainsi, une campagne d'exécution contient plusieurs instances d'exécution indépendantes, c'est-à-dire elles peuvent être exécutées dans n'importe quel ordre sur plusieurs clients. Dans notre modélisation, chaque instance d'exécution est considérée comme étant une application indépendante composée d'une chaine de n tâches. Donc par analogie avec la granularité su modèle d'environnement présenté dans la Section 4.2.2, une instance d'exécution correspond à l'application exécutée sur un client qui représente l'élément de calcul.

b Modèle de communication

La communication entre les clients et le serveur est souvent dans un seul sens. En effet, à cause des par-feux sur les clients, il est souvent impossible au serveur de contacter les clients. Par conséquent ce sont toujours les clients qui contactent le serveur pour demander du travail. Une fois que le client termine son travail, il contacte de nouveau le serveur pour rendre les résultats du calcul et demander plus de travail. Ces communications sont transportées par le réseau Internet. Typiquement un serveur BOINC a un nombre maximum entre $200 - 300$ connexions simultanées. Dans ces simulations nous supposons qu'une fois le client connecté, la bande passante est égale

à la sienne. Nous considérons aussi que si le serveur a atteint le nombre maximal de connexions, il retarde les requêtes qui arrivent le temps qu'une connexion se libère.

c Modèle de panne

Nous notons que le modèle de panne utilisé pour conduire ces simulations repose sur les deux études statistiques menées sur des traces provenant de la plateforme BOINC par Javadi *et al.* [64] et par Heien *et al.* [55]. Les traces de panne utilisées dans ces analyses sont disponibles sur le site *failure trace archive* [67]. Elles ont été obtenues en instrumentant les clients BOINC pour qu'ils enregistrent chaque évènement qui cause une panne. Dans BOINC, la panne représente n'importe quel évènement qui empêche l'exécution de l'application sur le client. Donc une panne ne représente pas nécessairement une défaillance mais peut être aussi l'utilisation du processeur de la machine en question par d'autre processus, la mise hors tension de la machine par le propriétaire *etc.* Ces études indiquent que les clients BOINC exhibent les deux régimes de panne à savoir le régime transitoire ainsi que le régime permanent. En effet dans ces plateformes il a été constaté qu'un nombre important des clients quittent définitivement la plateforme après une durée de temps aléatoire. Ceci représente la durée de vie du client qui correspond à la panne permanente. Cette durée de vie a fait l'objet d'une étude dans laquelle Heien *et al.* [55], indiquent que cette durée de vie est modélisée avec une distribution de Weibull dont les paramètres sont reportés dans la Table 4.1. Durant cette période de vie les clients exhibent un régime de panne transitoire. En effet les pannes transitoires surgissent quand un des évènements évoqués auparavant surgit (utilisation du CPU, machine hors tension *etc*). Ainsi la durée de vie du client est composée des deux intervalles qui s'alternent à savoir un intervalle de disponibilité suivie d'un intervalle d'indisponibilité et ainsi de suite. Pour modéliser ces deux intervalles, Javadi *et al.* [64] présentent une méthodologie de modélisation où, ils commencent par l'identification de la distribution de disponibilité et de la distribution d'indisponibilité de chaque client indépendamment et par la suite ils regroupent les clients qui représentent des similarités en terme de distribution de disponibilité et indisponibilité pour en faire un groupe de client homogène. Cette étude a abouti à l'identification de 6 groupes homogènes qui contiennent 20% du nombre total des clients dont les durées de disponibilité et d'indisponibilité sont de nature $i.i.d$, donc parfaitement modélisables avec des lois de probabilité.

d La stratégie d'ordonnancement des points de sauvegarde

Conformément à la nature des plateformes de type *desktop grid*. Le problème d'ordonnancement des points de sauvegarde ne peut qu'être considéré localement par chaque client. Chaque client reçoit une instance d'exécution qui est décrite par le nombre de tâches n, la quantité de travail de chaque tâche p_i et la taille des données à envoyer pour chaque point de sauvegarde o_i. Les clients sont hétérogènes en terme de puissance de calcul et liens de communication, ceci implique que la durée de temps nécessaire pour calculer la même quantité de travail ou envoyer la même quantité de données varie d'un client à un autre. Ainsi, en se basant sur les données d'entrée et en fonction de ses capacités à savoir son taux de traitement, sa distribution de disponibilité et son taux de transfert. Chaque client calcule localement une stratégie d'ordonnancement des points

de sauvegarde dont l'objectif est la minimisation de son propre temps total perdu. Ce critère découle du fait que dans la plateforme BOINC un crédit virtuel est attribué aux clients en fonction du travail utile qu'ils réalisent.

4.6.2 Méthodologie d'expérimentation

Si les plateformes *desktop grid* reposent sur un modèle de calcul distribué simple elles sont extrêmement complexes à modéliser ou à expérimenter. Cette complexité est due à la nature même de la plateforme. En effet, cette plateforme est composée d'un nombre très important de clients volontaires avoisinant une centaine de milliers. Les clients ont des ressources de calcul complètement hétérogènes en terme de puissance de calcul, d'architectures *etc*. De plus ces clients ne sont pas disponibles tout le temps puisqu'ils sont complètement volatiles. Rajoutons à cela la couche communication entre les clients et le serveur qui repose sur le réseau Internet qui est complètement imprévisible. En conséquence, nous tenons à préciser que la difficulté majeure pour évaluer et mesurer l'apport des mécanismes de tolérance aux pannes dans un tel contexte, réside principalement dans la construction d'un simulateur qui reproduit fidèlement le comportement de cette plateforme. Pour mettre en oeuvre cette étude expérimentale nous utilisons le simulateur SimGrid [24] pour reproduire le comportement des entités clients et serveur.

Dans la suite de cette partie, nous présentons tout d'abord une description des critères de performance et des ordonnancements des points de sauvegarde retenus dans cette étude. Dans un deuxième temps, nous exposons le modèle expérimental utilisé pour modéliser les entités serveur et client moyennant les routines SimGrid. Finalement, nous exhibons les paramètres et les configurations utilisées dans les simulations.

a Critères de performance et stratégies d'ordonnancement :

Les objectifs de cette étude expérimentale sont la validation, la comparaison et l'évaluation de la stratégie d'ordonnancement des points de sauvegarde proposée dans le contexte des plateformes *desktop grid*. Les critères de performance que nous considérons dans cette étude sont les suivants :

1. Le temps de complétion maximal d'une campagne d'exécution. Ce critère représente le temps de complétion maximal parmi toutes les instances d'exécution.
2. Le temps total perdu durant une campagne d'exécution. Ceci représente les trois sortes de temps total perdu présentés dans la section précédente.

Trois stratégies d'ordonnancement des points de sauvegarde sont retenues dans cette étude expérimentale.

- La première stratégie l'ordonnancement des points de sauvegarde notée par **PCkP** est calculée en se basant sur notre algorithme. Nous rappelons que cet algorithme prend en entrée les éléments suivants :
    1. Les durées des tâches qui seront calculées en fonction de la quantité de travail dans chaque tâche et du taux de traitement du client.
    2. La durée qu'il faut pour envoyer un point de sauvegarde. Cette durée est aussi évaluée en fonction des capacités du client.

3. La distribution de panne du client en question. Nous tenons à signaler que chaque client connait sa propre distribution de panne qui décrit ses durées de disponibilité.
4. Les coefficients de pondération $\kappa_L = 1$ et $\kappa_H = 1$ qui correspondent à la configuration où les points de sauvegarde sont aussi utilisés comme étant un délai de garde pour détecter les pannes.

– La deuxième stratégie l'ordonnancement des points de sauvegarde notée par **Periodic** est calculé en utilisant l'intervalle proposé par Daly [35]. Nous rappelons que ce modèle fournie une période optimale de sauvegarde notée par ρ en fonction du surcoût des points de sauvegarde c et le taux de panne λ. Ainsi pour rendre ce modèle fonctionnel dans ce cadre c et λ sont donnés par :

$$c = \sum_{i=1}^{n} c_i/n \text{ et } \lambda = \int_0^{\infty} tf(t)dt.$$

La valeur de c_i est estimée avec le modèle linéaire de réseau et $f(t)$ la densité des pannes transitoires. Ensuite pour calculer les emplacements des points de sauvegarde en utilisant ρ nous adoptons la stratégie suivante : Soit i' l'indice du dernier point de sauvegarde qui est égale à zéro si aucun point de sauvegarde n'a pas été réalisé encore. Le prochain instant de sauvegarde noté par i est donné par :

$$i = \arg\min_{i'<j\leq n} |\rho + \tau_{i'} - \tau_j|$$

– La dernière stratégie d'ordonnancement des points de sauvegarde notée par **WithoutFT** est la politique standard des clients dans laquelle aucun point de sauvegarde distant n'est enclenché.

b Modèle SimGrid

Pour simuler la plateforme considérée nous utilisons le modèle client/serveur fournie par *SimGrid* [24]. Moyennant les routines de *SimGrid* nous mettons en œuvre la stratégie d'ordonnancement globale du serveur BOINC pour distribuer le travail et la stratégie des requêtes de demande de travail (*pull policy*) adoptée par les clients. La stratégie d'ordonnancement globale du serveur adoptée dans cette étude est basée sur le principe du premier arrivé premier servie (FCFS). Ainsi, quand le serveur reçoit une requête de demande de travail trois réponses sont envisageables.

1. Il y a des instances d'exécution qui ne sont pas encore envoyées à aucun client. Alors le serveur choisit une instance au hasard pour l'envoyer à un client.
2. Toutes les instances de la campagne d'exécution sont déjà affectées à des clients mais elles ne sont pas encore terminées. Nous soulignions que le serveur BOINC ne peut pas savoir si le client est encore en vie ou s'il a décidé de quitter la plateforme définitivement. Donc le serveur n'a aucune information sur l'avancement de l'exécution des instances. Dans ce cas, le serveur choisit au hasard une instance pour la dupliquer et l'envoyer au client en question. L'exécution de cette copie commence depuis le dernier point de sauvegarde disponible sur le serveur. Nous notons aussi qu'aucune synchronisation entre les exécutions dupliquées est mise

en œuvre. En effet le serveur ignore les résultats ainsi que les points de sauvegarde qui ne sont pas à jours.

3. Si tout le travail est fait. Alors un message d'arrêt est envoyé au client.

Concernant la stratégie de demande de travail chez les clients. Deux évènements peuvent déclencher cette requête.

1. Le premier évènement c'est la terminaison de l'instance d'exécution courante chez le client. Dans ce cas, le client renvoie les résultats de cette instance et il en demande une autre.
2. Le deuxième évènement c'est la transition de l'état indisponible vers l'état disponible après une panne transitoire. En effet, en cas de panne durant l'exécution d'une instance tout le travail fait et qui n'est pas sauvegardé sur le serveur distant est perdu. Donc, une fois que le client est de nouveau disponible il demande systématiquement une nouvelle instance.

Nous tenons à préciser que ces deux évènements correspondent tout à fait à la stratégie par défaut des clients BOINC.

c Les scénarios :

Nous entamons maintenant la description des configurations et des scénarios utilisés dans les simulations. Dans cette étude nous considérons que le nombre des clients dans la plateforme varie dans l'ensemble $\{5000, 10000, 20000\}$ avec seulement un seul serveur à chaque fois. Typiquement dans SimGrid, chaque entité (client au serveur) est caractérisée par : sa puissance de calcul, la capacité de son propre lien réseau et avec qui elle peut communiquer. De plus, SimGrid nous offre la possibilité de définir les intervalles de disponibilité et indisponibilité pour chaque entité.

Les propriétés du serveur sont les suivantes : une puissance de calcul de 2GFlops, un lien de communication Internet de 1 Gb/s. Nous tenons à signaler que le serveur est supposé comme 100% fiable c'est-à-dire qu'il ne subit pas de panne. Le support stable utilisé pour stocker les points de sauvegarde est hébergé sur le même serveur. Pour reproduire les performance des serveurs actuels des projets BOINC, nous limitons la capacité du serveur simulé à 100 connexions à la fois.

Les propriétés des clients sont issues des diverses analyses et études menées sur la plateforme BOINC. Ces propriétés sont générées aléatoirement selon des distributions de probabilité qui sont reportées dans la Table 4.1. Les distributions et leurs paramètres respectifs utilisés pour générer aléatoirement les propriétés des clients, reposent sur des études de statistiques menées sur des traces d'exécution provenant du projet SETI@Home [64, 55], à l'exception de la distribution utilisée pour générer les propriétés des liens réseau. Plus précisément les puissances de calcul des clients sont générées en s'appuyant sur le travail mené dans [55]. Dans ce dernier les auteurs montrent que la puissance de calcul des clients est distribuée selon une loi normale dont les paramètres sont reportés dans la Table 4.1. Les distributions de disponibilité et d'indisponibilité proviennent des travaux présentés dans [64]. Dans ces travaux, les auteurs identifient 6 groupes de clients tels que dans chaque groupe les clients ont plus ou moins les mêmes distributions de disponibilité et d'indisponibilité. Dans cette étude nous nous

intéressons seulement aux clients du groupe numéro 3 qui contient pus de 20000 clients. Les distributions ainsi que les paramètres respectifs des intervalles de disponibilité et d'indisponibilité sont reportées dans la Table 4.1. De même pour la distribution des durées de vie des clients qui est étudiée aussi dans [55], où il est indiqué que cette durée est distribuée selon une distribution de Weibull.

Table 4.1 – Propriétés des clients

Nombre des clients $\{5000, 10000, 20000\}$	Puissance de calcul (MIPS) Distribution normal $moyenne = 1771, std = 669.5$
Bande passante réseau (Mb/s) uniforme $[1, 2]$	Distribution de la durée de vie (jours) Weibull $\beta_w = 0.58$, $\alpha_w = 137$
Distribution de disponibilité (hrs) Weibull $\beta_w = 0.431, \alpha_w = 1.682$	Distribution d'indisponibilité (hrs) Hyper-exponential $p_i \in \{0.398, 0.305, 0.298\}$, $\mu_i \in \{0.031, 11.566, 1.322\}$

Table 4.2 – Propriétés de la campagne d'exécution

Nombre d'instances	Nombre de tâches par instance
$\{1000, 3000, 5000, 7000, 10000\}$	Distribution uniforme $[1, 50]$
Nombre d'instructions par tâche	Taille des sauvegardes (Mb)
Distribution uniforme $[36^9, 36^{10}]$	Distribution uniforme $[10, 512]$

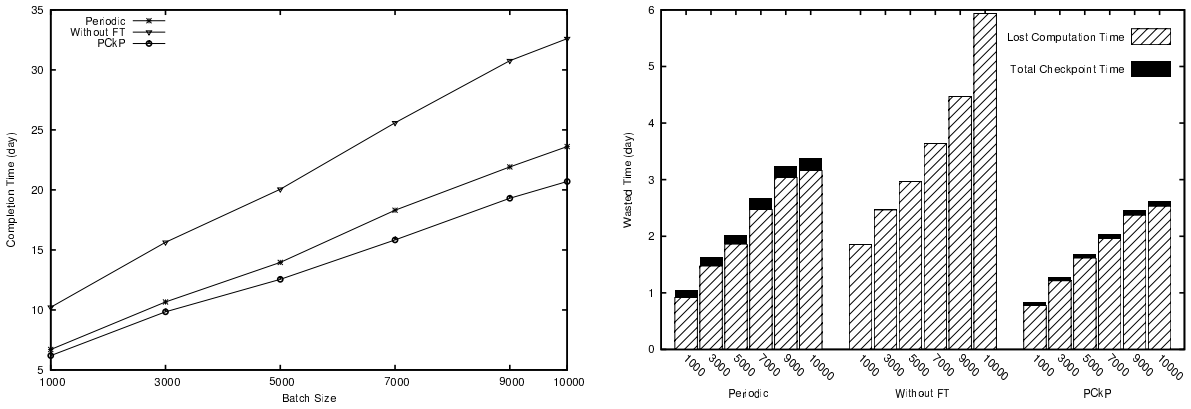
Table 4.2 reporte les différents paramètres utilisés pour reproduire les campagnes d'exécution. Dans cette étude, nous nous intéressons seulement à l'exécution d'une seule campagne d'exécution (un seul projet). Le nombre des instances dans cette campagne d'exécution varie dans l'ensemble $\{1000, 3000, 5000, 7000, 9000, 10000\}$ ceci correspond aux propriétés des campagnes reportées dans *Grid Workload Archive* [62] ou BOINC [39]. Chaque instance de la campagne est indépendante des autres instances. Ainsi chaque instance est considérée comme une application indépendante qui est composée de n tâches. Le nombre des tâches n dans chaque instance est généré uniformément dans l'intervalle $[1, 50]$ indépendamment des autres instances. La quantité d'instructions dans chaque tâche (p_i) est générée uniformément dans l'intervalle $[36^9, 36^{10}]$. Cet intervalle correspond à un temps d'exécution qui varie entre 1 heure et 5 heures en considérant la puissance moyenne des clients. Par conséquent le temps de complétion d'une instance d'exécution varie entre 1 heure et 250 heures. La borne inférieure 1 correspond au cas où l'instance d'exécution est n'composée que d'une seule tâche qui dure 1 heure tandis que la borne supérieure correspond au cas où l'instance d'exécution est composée de 50 tâches et chaque tâche dure 5 heures. Le temps de complétion d'une campagne composée de 5000 instances d'exécution sur un seul processeur varie entre 1/2 ans et 146 ans, toujours en considérant la puissance de calcul moyenne des clients. Ceci correspond à la durée d'exécution des tâches dans le projet *climate prediction* [13]. Finalement, nous supposons que la taille de chaque point de sauvegarde notée précédemment par o_i entre deux tâches d'une application donnée est générée uniformément dans l'intervalle

$[10, 512]$Mb. Cet intervalle est représentatif des tailles des exécutables des applications que les clients BOINC téléchargent.

4.6.3 Analyse des résultats

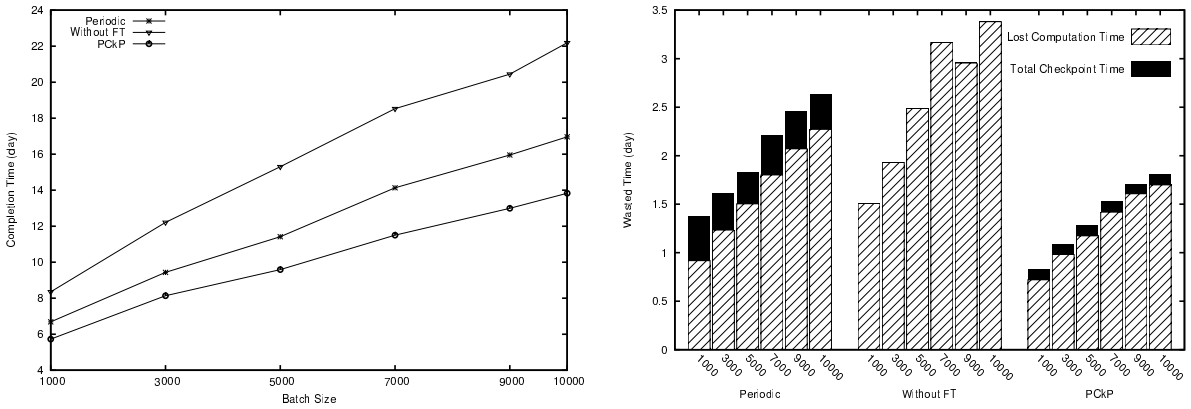
Nous présentons dans cette partie les résultats et les analyses à l'issue de cette étude expérimentale. Dans ces simulations nous, comparons les trois stratégies décrites auparavant en utilisant 3 configurations différentes pour le nombre des clients et 6 différentes configurations pour le nombre des instances dans une campagne d'exécution. Dans le premier scénario, le nombre d'instances dans la campagne est deux fois plus grand que le nombre des clients. Ceci est obtenu en fixant le nombre des clients à 5000 et en variant le nombre des instances d'exécution dans l'ensemble $\{1000, 3000, 5000, 7000, 9000, 10000\}$. Tandis que dans le deuxième scénario, le nombre d'instances est dans le même ordre que le nombre des clients bénévoles. Dans le dernier scénario, nous augmentons le nombre des clients à 20000 pour qu'il soit deux fois plus grand que le nombre maximal des instances d'exécution. En effet nous notons que ces configurations ont été choisies de telles façon que nous puissions observer l'impact de la duplication des instances sur les critères de performance.

Les résultats des simulations en utilisant le premier scénario sont reportés sur les Figures 4.3(a) and 4.3(b). La Figure 4.3(a) reporte la variation du temps de complétion maximal de la campagne en fonction de la variation du nombre des instances dans la campagne. Comme nous pouvons le constater le temps de complétion des instances exécutées avec des mécanismes de tolérance aux pannes sont presque équivalents. Par contre, nous notons que le temps de complétion des campagnes exécutées avec le mécanisme de sauvegarde est réduit d'un facteur de 34% (dans le cas de 10000 instances) par rapport aux instances exécutées sans mécanisme de tolérance aux pannes. Ceci montre dans un premier temps l'apport des mécanismes de tolérance aux pannes dans un tel contexte. Dans la deuxième Figure 4.3(b) nous reportons la variation de la moyenne du temps total perdu. Cette moyenne représente la somme du temps total perdu des clients divisée par le nombre total des clients durant l'exécution. Nous reportons avec les bars sans hachures le surcoût accumulé des points de sauvegarde, tandis que les bars avec hachures représentent la quantité de travail perdu à cause des pannes. Cette figure confirme bien que notre stratégie notée par (PckP) est légèrement meilleure à la stratégie périodique et que la moyenne du temps total perdu est réduite d'un facteur de 50% par rapport à la stratégie sans sauvegarde quand la campagne d'exécution est composée de 10000 instances. Ceci confirme bien les résultats précédents. Pour le deuxième scénario nous gardons le même ensemble de variation des instances d'exécution et nous fixons le nombre des clients à 10000. Les résultats des expérimentations avec ces paramètres sont présentés dans les Figures 4.4(a),4.4(b). Ces résultats indiquent que la stratégie proposée donne des résultats meilleurs que les deux autres stratégies. Nous constatons aussi à partir de la Figure 4.4(b) que notre stratégie réalise deux fois moins de sauvegardes par rapport à la stratégie périodique de Daly tout en ayant un temps de complétion meilleur. Ce qui implique que la stratégie proposée réduit la quantité de travail perdu toute en réalisant moins de sauvegardes. Ceci peut être expliqué par le fait que le modèle périodique ne prend pas en compte la variation possible dans les durées des tâches ainsi que les durées des points de sauvegarde.



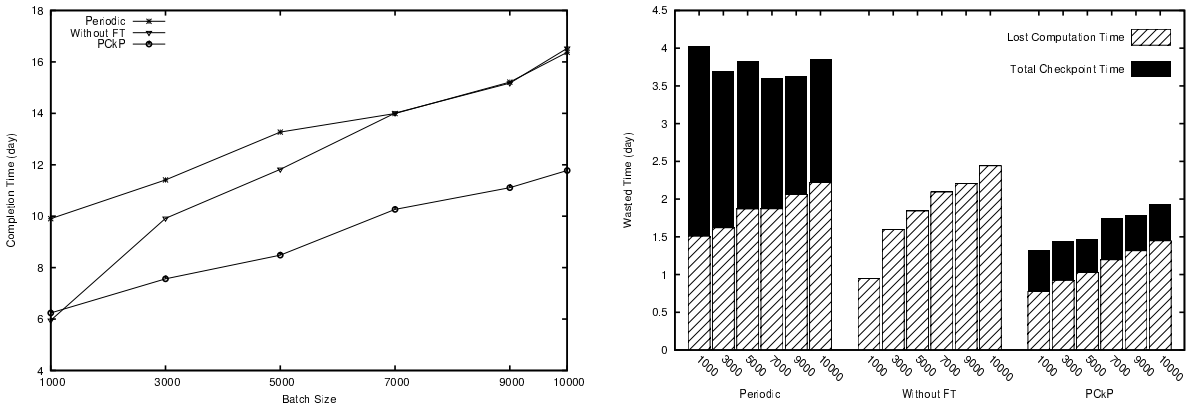
(a) Variation du temps de complétion (jours) (b) Variation du temps total perdu (jours)

FIGURE 4.3 – Variation du temps de complétion (a) et du temps total perdu (b) avec 5000 clients en fonction du nombre d'instances.



(a) Variation du temps de complétion (jours) (b) Variation du temps total perdu (jours)

FIGURE 4.4 – Variation du temps de complétion (a) et du temps total perdu (b) avec 10000 clients en fonction du nombre d'instances.



(a) Variation du temps de complétion (jours) (b) Variation du temps total perdu (jours)

FIGURE 4.5 – Variation du temps de complétion (a) et du temps total perdu (b) avec 20000 clients en fonction du nombre d'instances.

Finalement dans la dernière configuration le nombre des clients est fixé à 20000 ce qui représente deux fois le nombre maximal des instances à exécuter. Dans cette configuration chaque instance est dupliquée au moins deux fois. Le premier résultat associé à cette configuration est tracé sur la Figure 4.5(a) qui reporte la variation du temps de complétion en fonction du nombre d'instances dans la campagne. Comme nous pouvons le constater, la stratégie proposée minimise le temps de complétion de la campagne par rapport aux autres stratégies candidates. Un autre constat important est à signaler. En effet sur cette figure nous constatons que le temps de complétion de la campagne exécutée avec la stratégie périodique s'est dégradé dans le sens où il est nettement supérieur au temps de complétion de la campagne exécutée sans mécanisme de sauvegarde. Ceci et d'autant plus visible sur la Figure 4.5(b) qui apporte une réponse à ce constat. En effet cette figure montre bien que les campagnes exécutées avec le mécanisme de sauvegarde périodique et sans mécanisme de sauvegarde ont presque la même quantité de travail perdu. Par contre, le surcoût accumulé des points de sauvegarde change la donne et dégrade le temps de complétion. La seule explication qui nous semble raisonnable pour expliquer ce constat c'est la congestion du lien du réseau au niveau du serveur. En effet nous notons que dans ce cas le nombre des clients a augmenté considérablement tandis que la quantité de travail initial est la même. Ce qui implique que plusieurs clients qui exécutent les mêmes instances essayent d'accéder en même temps au serveur stable pour écrire les sauvegardes ce qui crée de la congestion. Cette congestion pénalise plus la stratégie périodique puisqu'elle en fait plus de sauvegardes par rapport à la stratégie que nous proposons.

Pour conclure, nous notons que la stratégie d'ordonnancement des points de sauvegarde proposée réduit considérablement le temps de complétion de la campagne d'exécution par rapport à la stratégie périodique ainsi que la stratégie sans sauvegarde. Cette amélioration est due au fait que notre modélisation prend en compte la variabilité des surcoûts des points de sauvegarde ainsi que la variabilité des emplacements possibles de ces points. Un élément très important est mis en évidence à travers ces simulations est l'impact de la congestion sur les performances de l'application et la plateforme. Donc pour intégrer le mécanisme de sauvegarde et reprise distant sur le serveur du projet, il faut étudier avant toute chose la congestion pour ne pas nuire et dégrader les performances de l'environnement d'exécution.

4.7 Conclusions

Dans ce chapitre nous avons proposé de prime abord une étude théorique du problème d'ordonnancement des points de sauvegarde en utilisant un modèle d'application discret. Dans ce dernier, nous avons modélisé l'application par une suite de tâches, où chaque tâche est défini par la quantité de travail et la quantité de données à sauvegarder. En s'appuyant sur cette modélisation discrète de l'application nous avons posé le problème sous forme d'un problème d'optimisation combinatoire dont l'objectif est la minimisation du temps total perdu durant un cycle de panne. Nous avons modélisé ce problème formellement sous forme d'un processus de décision markovien. Ensuite, en se basant sur les propriétés du modèle MPD formulé, nous avons exhibé une stratégie optimale des points de sauvegarde qui minimise l'espérance du temps total perdu. À notre

connaissance cet algorithme est le premier qui résout le problème en considérant des durées de tâche variables, des surcoûts pour les points de sauvegarde variables et en supposant que la distribution de panne est arbitraire sans hypothèse de loi sans mémoire. De plus, une analyse du coût algorithmique nécessaire pour calculer la stratégie optimale a été menée. Dans cette analyse nous avons démontré que la solution proposée a une complexité pseudo-polynomiale en $\mathcal{O}\left(n^3 c^{max}\right)$ dans le cadre général. Cependant, nous avons exhibé un cas particulier du problème dans lequel les surcoûts des points de sauvegarde sont tous constants. Ainsi sous cette hypothèse raisonnable nous avons prouvé que le problème d'ordonnancement des points de sauvegarde appartient à la classe de complexité P en exhibant un algorithme en $\mathcal{O}\left(n^3\right)$. Dans un deuxième temps nous avons aussi justifié la complexité de l'algorithme proposé dans le cadre général en démontrant que le problème d'ordonnancement des points de sauvegarde appartient à la classe *NP-Complet*. Cette réduction a mis en évidence que l'élément qui rend le problème d'ordonnancement des points de sauvegarde *NP-Complet* est la variabilité des surcoûts des points de sauvegarde et non pas la distribution de panne arbitraire puisque le problème se réduit à *Subset Sum* même quand la loi de panne est décrite avec la fonction la plus élémentaire donnée par $F(t) = t$ et il appartient à la classe de complexité P simplement quand $c_i = c$ pour n'importe quelle $F(t)$.

La deuxième étude complémentaire qui a été menée dans ce chapitre est de nature expérimentale. Nous nous sommes intéressés à l'étude des apports du mécanisme de tolérance aux pannes dans le contexte des plateformes de type *desktop grid*. Pour réaliser cette étude nous avons utilisé le modèle d'exécution des applications dans la plateforme BOINC en considérant le problème d'ordonnancement des points de sauvegarde localement par rapport à chaque client. Ainsi chaque client est autonome et il calcule sa propre stratégie d'ordonnancement des points de sauvegarde optimale en fonction de ses propriétés et des paramètres de l'application qu'il exécute. Par conséquent, l'objectif de chaque client est la minimisation de son propre temps total perdu. Trois stratégies d'ordonnancement des points de sauvegarde ont été retenues pour mener cette étude. Dans la première stratégie chaque client calcule localement son propre ordonnancement des points de sauvegarde en utilisant la solution proposée. La deuxième est une adaptation de la stratégie de sauvegarde périodique proposée par Daly [35]. Tandis que dans la dernière configuration nous avons considéré une exécution sans mécanisme de sauvegarde. Pour mettre en oeuvre les simulations, nous nous sommes basés sur les routines du simulateur SimGrid. L'analyse a mis en évidence trois constants à travers ces simulations. Dans un premier temps nous avons montré que moyennant le mécanisme de sauvegarde, le temps de complétion ainsi que le temps total perdu sont nettement améliorés. Ensuite, nous avons montré que notre stratégie améliore les deux critères de performance considérés par rapport à la stratégie périodique en réalisant moins de sauvegardes. Ceci s'explique par le fait que notre solution prend en compte la variabilité des durées des tâches et des surcoûts des points de sauvegarde. Le dernier constat à retenir de cette étude est l'impact de la congestion réseau sur les performances de la plateforme.

Pour conclure ce chapitre nous tenons à souligner que premièrement il est primordial de prendre en compte la congestion réseau comme critère de performance supplémentaire dans le contexte des plateformes de calcul volontaire. Le deuxième point à considérer

est lié au modèle de l'application qui est le modèle chaine de tâches. Donc deux questions naturelles se posent. La première est "comment peut-on étendre cette analyse en considérant que les tâches sont indépendantes avec des durées arbitraires ?". La deuxième question qui vient toute suite après à l'esprit est, "comment faire si l'application est décrite par un graphe acyclique de précédences entre les tâches ?".

II

Ordonnancement

Ordonnancement tolérant aux pannes 5

Sommaire

5.1 Introduction **79**
5.2 Formulation du problème d'optimisation **80**
5.3 État de l'art **82**
5.3.1 Quelques rappels en théorie d'ordonnancement 82
5.3.2 Optimisation multi-objectifs 84
5.3.3 Résultats dans le cadre de la tolérance aux pannes 87
5.4 Optimisation du *makespan* et de la fiabilité de l'application **93**
5.4.1 Optimisation de $Q||(C_{max}, \mathcal{R}_l)$ pour une loi DFR 93
a Tâches indépendantes et unitaires 95
b Tâches indépendantes et arbitraires 99
5.4.2 Optimisation de $P||(C_{max}, \overline{\mathcal{R}}_l)$ avec $\lambda(t)$ IFR 102
a Taches unitaires et indépendantes 102
b Tâches indépendantes et arbitraires 104
5.5 Conclusions **105**

5.1 Introduction

Ce chapitre est consacré à l'analyse des approches d'ordonnancement conçues pour les nouveaux environnements de calcul parallèle et distribué où la tolérance aux pannes est incontestablement primordiale. L'objectif de cette étude est la conception de nouvelles approches ordonnancement de tâches qui sont tolérantes aux pannes et qui répondent aux nouvelles spécificités des plateformes de calcul parallèle et distribué actuelles.

Le plan de ce chapitre est le suivant. Dans un premier temps, nous présentons dans la Section 5.2 la formulation du problème en décrivant un nouveau modèle d'exécution ainsi que les critères de performances retenus.
Dans un deuxième temps nous synthétisons dans la section 5.3 les différentes notions et notations essentielles dans cette étude. À savoir : les résultats classiques en théorie d'ordonnancement et les résultats fondamentaux dans la branche d'optimisation multi-objectifs. Ensuite nous présentons les approches d'ordonnancement tolérant aux pannes qui sont en lien avec l'objet d'étude de ce travail.
Ensuite, nous traitons dans la Section 5.4 la problématique d'optimisation formulée où nous nous focalisons sur l'optimisation des deux objectifs considérés dans cette étude : la minimisation du temps de complétion ainsi que la maximisation de la probabilité de

succès de l'application. Finalement, nous présentons dans la dernière section quelques conclusions et perspectives envisageables à l'issue de cette étude.

5.2 Formulation du problème d'optimisation

Nous introduisons dans cette section la description formelle de la problématique d'ordonnancement étudiée. Nous utilisons dans cette description la célèbre notation à trois champs $\alpha|\beta|\gamma$ introduite par Graham *et al.* [50] pour noter les différentes variantes des problèmes d'ordonnancement. Dans cette notation, le champ α est utilisé pour décrire la caractéristique de la plateforme utilisée. Le champ β est dédié à la description des contraintes que toute solution valide du problème doit respecter et les caractéristiques de l'application. Finalement le dernier champ γ exprime les fonctions objectifs de la problématique.

Deux modèles de plateformes sont considérés dans cette étude. Dans un premier temps nous ciblons une plateforme d'exécution composée d'un ensemble de processeurs noté par $\mathcal{Q}$ et qui contient m processeurs uniformes. Chaque processeur j de cet ensemble possède son propre taux de traitement noté par ν_j. Ceci correspond au modèle Q dans la notation de Graham. Ensuite, nous considérons le modèle de plateforme noté par P où tous les processeurs sont supposés identiques. Nous tenons à signaler que dans les deux cas, les processeurs ont la même fonction de taux de panne notée par $\lambda(t)$ et le régime de panne est supposé permanent.

Nous envisageons aussi deux modèles d'application. Dans le premier, l'application est composée de n tâches indépendantes. Chaque tâche contient p_i unités de travail. Ainsi la durée d'exécution de la tâche i sur le processeur j est donnée par p_i/ν_j. Le deuxième modèle d'application considéré est un cas particulier du premier où toutes les tâches ont la même quantité de travail ($p_i = 1$). Notez que dans ce chapitre nous utilisons par convention i comme indice des tâches et j comme indice de processeurs.

Ainsi la problématique classique de ordonnancement de l'ensemble des tâches $\mathcal{T}$ sur l'ensemble des processeurs $\mathcal{Q}$ correspond à l'affectation d'un processeur et d'une date de début d'exécution pour chaque tâche. Ceci peut être défini formellement de plusieurs façons. Dans ce manuscrit un ordonnancement de tâches noté par $\mathcal{S}$ est défini par les deux applications notées respectivement par ξ et ζ. La fonction allocation spatiale $\xi : \mathcal{T} \to \mathcal{Q}$ alloue à une tâche i le processeur $\xi(i)$. La fonction allocation temporelle $\zeta : \mathcal{T} \to \mathbb{R}_+$ alloue à une tâche i une date de début d'exécution $\zeta(i)$. Nous notons par C_i la date de complétion de la tâche i qui est donnée par $C_i = \zeta(i) + p_i/\nu_{\xi(i)}$. Nous notons de la même manière par C_j le temps de complétion de la dernière tâche sur processeur j. Cette quantité est donnée par $C_j = \sum_{i\in\mathcal{T}|\xi(i)=j} p_i/\nu_j$ dans le cas où l'ordonnancement est compact et par $C_j = \max_{\forall i\in\mathcal{T}|\xi(i)=j}\{C_i\}$ dans le cas général. Pour qu'un ordonnancement de tâches soit valide, il doit au moins respecter les contraintes suivantes : Les tâches sont exécutées sans préemption, un processeur n'exécute qu'une seule tâche à la fois et chaque tâche est nécessairement affectée à un processeur. Nous notons par V l'ensemble des ordonnancements qui vérifient ces contraintes.

Les problèmes d'ordonnancement sont associés à une fonction d'optimisation. Dans ce chapitre la fonction considérée est multi-objectifs. Le premier objectif est la minimisation du critère classique à savoir la date de complétion de l'application. Cet objectif est appelé communément le *makespan* et noté par C_{max}. Soit un ordonnancement $\mathcal{S} = (\xi, \zeta)$

le *makespan* de cet ordonnancement est donné par :

$$C_{max}^{\mathcal{S}} = \max_{i \in \mathcal{T}} \left(\zeta(i) + p_i / \nu_{\xi(i)} \right). \tag{5.1}$$

Le deuxième objectif retenu est la maximisation de la fiabilité de l'application. Cet objectif noté par $\mathcal{R}$, est donné formellement par la probabilité de succès de l'exécution. Etant donné le régime permanent des pannes, la probabilité de succès de l'application est donnée par la probabilité qu'aucune panne ne survienne durant l'exécution des tâches. Cette quantité est donnée formellement par :

$$\mathcal{R}(\mathcal{S}) = \prod_{j=1}^{m} \overline{F}(C_j). \tag{5.2}$$

Dans la suite de cette étude nous nous intéressons à la maximisation du logarithme de la fiabilité noté par $\mathcal{R}_l$. Nous rappelons que la fonction logarithme est une fonction croissante et donc la maximisation de $\mathcal{R}$ est équivalente à la maximisation de $\log \mathcal{R}$. Cette transformation clé qui distingue notre contribution des travaux existants est mise en œuvre principalement pour deux raisons. Premièrement la fonction logarithme nous permet de passer de la forme produit à la forme somme et deuxièmement la fonction $\log \overline{F}(x)$ a des propriétés de concavité ou convexité très utiles dans l'analyse du problème. Ainsi par convention la fonction objectif à maximiser qui représente la fiabilité est donnée par :

$$\max_{\mathcal{S} \in V} \mathcal{R}(\mathcal{S}) = \prod_{j=1}^{m} \overline{F}(C_j) \equiv \max_{\mathcal{S} \in V} \mathcal{R}_l(\mathcal{S}) = \sum_{j=1}^{m} \log \overline{F}(C_j). \tag{5.3}$$

Nous introduisons d'une manière similaire la variante de minimisation du manque de fiabilité de l'application.

$$\min_{\mathcal{S} \in V} \overline{\mathcal{R}}(\mathcal{S}) = 1 - \prod_{j=1}^{m} \overline{F}(C_j) \equiv \min_{\mathcal{S} \in V} \overline{\mathcal{R}}_l(\mathcal{S}) = \sum_{j=1}^{m} - \log \overline{F}(C_j). \tag{5.4}$$

Nous introduisons aussi le paramètre ρ pour noter le ratio de performance d'un ordonnancement $\mathcal{S}$ par rapport à un ordonnancement optimal $\mathcal{S}^*$ [57]. Nous signalons que dans ce manuscrit nous considérons par convention $\rho \geq 1$, par exemple $\rho = \mathcal{R}(\mathcal{S}^*)/\mathcal{R}(\mathcal{S})$ dans le cas d'une problématique de maximisation et $\rho = \overline{\mathcal{R}}(\mathcal{S})/\overline{\mathcal{R}}(\mathcal{S}^*)$ dans le cas de minimisation.

En s'appuyant sur ces concepts et sur cette nouvelle mesure de fiabilité, nous présentons dans la suite de ce chapitre une analyse détaillée de chacun des points suivants. Premièrement, nous analysons en fonction de la nature de la distribution de panne la complexité de la problématique d'optimisation mono-objectif où nous nous intéressons seulement à l'optimisation de la fiabilité de l'application. Deuxièmement nous analysons la relation qui gouverne les deux objectifs à savoir la fiabilité et le *makespan* toujours en fonction de la nature de la distribution de panne. Finalement à la lumière des deux premières analyses nous proposons des approches d'ordonnancement pour optimiser les deux objectifs à la fois.

5.3 État de l'art

5.3.1 Quelques rappels en théorie d'ordonnancement

Nous entamons cette section d'état de l'art par une présentation des différents résultats techniques en ordonnancement à la fois classiques et essentiels dans le cadre de cette étude.

$\mathbf{P||C_{max}}$: Nous commençons cette synthèse avec la présentation du problème d'ordonnancement classique $P||C_{max}$. Ici, la plateforme est composée de m processeurs identiques. L'application est donnée par un ensemble de tâches où chacune à une durée d'exécution arbitraire notée par p_i. Etant donnée cette instance d'entrée ($p_1, p_2, \cdots p_n$ et m) l'objectif est de calculer une stratégie ordonnancement qui minimise le temps de complétion de la dernière tâche C_{max}. Cependant comme tout problème d'optimisation combinatoire intéressant, résoudre $P||C_{max}$ optimalement avec un algorithme qui a un temps de calcul polynomial en la taille de l'instance est impossible puisque ce problème élémentaire est $\mathcal{NP}$-complet [42] sauf si $P = NP$. Par contre ce problème reste approximable à un facteur constant aussi proche qu'on veut de l'optimal $(1 + \epsilon)$ moyennant un schéma d'appoximation polynomial ($\mathcal{PTAS}$) [58]. Toutefois les approches de type $\mathcal{PTAS}$ sont généralement très couteuses en temps de calcul puisque le terme constant augmente exponentiellement en fonction de la précision ϵ souhaitée. Une autre approche de résolution de type glouton appelée *List Scheduling* a été proposée par Graham [48]. Cet algorithme glouton trie les tâches dans un ordre particulier de priorité par exemple selon les durées des tâches. Ensuite, dès qu'un processeur est libre, la tâche en tête de liste est ordonnancée sur ce dernier. Ainsi, le principe de tout ordonnancement *list* est de ne pas avoir de processeur inactif tant qu'il existe des tâches prêtes à l'exécution. Donc en absence de relation de précédence entre les tâches, tout ordonnancement de ce type est un ordonnancement compact c'est-à-dire qu'il n'y a pas de temps libre entre les tâches. L'ordonnancement produit par cet algorithme est à un facteur 2 au pire des cas de l'ordonnancement optimal. Nous notons que ce ratio de performance est porté à $\frac{4}{3} - \frac{1}{3m}$ [49] en utilisant la variante LPT (*Largest Processing Time*) où les tâches sont triées par ordre décroissant selon leurs durées d'exécution, que l'on peut raffiner en 5/4 [66].

$\mathbf{Q|prec|C_{max}}$: La deuxième variante est une généralisation naturelle du problème précédent. En effet, à la différence de $P||C_{max}$ dans celle-ci les processeurs ne sont plus identiques mais uniformes. Chaque processeur a un son propre taux de traitement noté par ν_j. De plus, deux points changent dans le modèle d'application. Premièrement, les tâches ne sont plus indépendantes mais elles sont reliées par une relation de précédence arbitraire. Deuxièmement, dans ce cas de figure, p_i représente la quantité de travail dans la tâche i et la durée d'exécution cette tâche sur le processeur j est donnée par p_i/ν_j. Ainsi la première question intuitive qui vient à l'esprit est, "est ce que le résultat de Graham *et al.* [48] est encore valable si les processeurs sont uniformes ?". En considérant un graphe de précédence arbitraire entre les tâches Liu *et al.* [72] démontrent que le ratio de performance de l'algorithme *list* est en $\mathcal{O}\left(1 + \max_j \nu_j / \min_j \nu_j - \max_j \nu_j / \sum_{j=1}^{m} \nu_j\right)$ si les machines sont uniformes. En conséquent, il est évident que le ratio de performance

de l'ordonnancement produit par l'algorithme *list* peut être loin d'un facteur non borné de l'ordonnancement optimal puisque le ratio entre les deux taux de traitement du processeur le plus rapide et le plus lent peut être borné. Ainsi en supposant que ce ratio soit borné par $\mathcal{O}\left(1/\sqrt{m}\right)$, Jaffe [63] étend l'analyse faite par Liu *et al.* [72] pour en déduire que dans ce cas de configuration l'ordonnancement produit par l'algorithme de *list* est au plus à un facteur $\mathcal{O}\left(\sqrt{m}\right)$ de l'ordonnancement optimal. En utilisant une autre approche qui se base sur la résolution des programmes linéaires relaxés, Shmoys *et al.* [92] améliore le résultat précèdent en proposant un algorithme qui garantit que l'ordonnancement produit est au plus à un facteur $\mathcal{O}\left(m\right)$ où m est le nombre des taux de traitement différents. En effet dans ce travail, l'instance initiale avec m taux différents est transformée en une instance où il y a seulement $\mathcal{O}\left(\log m\right)$ taux différents. Ceci est fait en ignorant tous les processeurs qui ont un taux de traitement à plus d'un facteur $1/m$ du plus rapide et en arrondissant les taux de traitement des processeurs restant en la puissance de 2 la plus proche. Ensuite ils mettent en œuvre un ordonnancement de type *list* en utilisant cette fois le taux de traitement des processeurs pour définir l'ordre d'exécution des tâches. Plus précisément, à la différence de l'ordonnancement *list* classique où la tâche est affectée au premier processeur libre, dans cette approche la tâche est affectée au premier processeur libre et qui vérifie une contrainte calculée à l'avance sur le taux de traitement. Cet ensemble de contraintes est calculé moyennant la résolution d'un programme linéaire. Ainsi ils montrent qu'au pire des cas, l'ordonnancement est à un facteur $\log m$ de l'ordonnancement optimal. Cependant, bien que cette approche ait complexité théorique polynomiale, son utilité pratique reste limitée à cause du facteur constant relatif à la résolution du programme linéaire. Pour contourner cette limitation, Chekuri *et al.* [30] proposent une autre approche en s'inspirant des travaux de Shmoys *et al.* [92]. Dans ce travail ils gardent la même idée de transformation de l'instance et le même principe de l'ordonnancement *list* avec des contraintes sur le taux de traitement. Par contre pour calculer l'ensemble des contraintes ils mettent en œuvre une autre technique qui se base sur l'algorithme de décomposition du graphe de précédence en un ensemble de chaines de tâches de taille maximale. Ils montrent que leur approche a aussi le même ratio de performance, mais avec une complexité de seulement $\mathcal{O}\left(n^3\right)$.

$\mathbf{P}||\sum_{\mathbf{j=1}}^{\mathbf{m}} \mathbf{g}(\mathbf{C_j})$: La dernière variante présentée est une problématique très proche de notre cas d'étude. Dans cette problématique l'objectif est la minimisation de $\sum_{j=1}^{m} g(C_j)$ où C_j est le *makespan* du processeur j et une fonction $g : \mathbb{R} \to \mathbb{R}$ supposée croissante dans un cas de minimisation. Dans le cadre d'une problématique de placement de données sur une baie de stockage dont l'objectif est la minimisation de la latence de lecture Chandra *et al.* [28] formulent une variante particulière de $P||\sum_{j=1}^{m} g(C_j)$ où $g(x) = x^2$. Dans ce travail, ils montrent dans un premier temps que le ratio de performance de l'algorithme d'ordonnancement glouton LPT devient 25/24. Ensuite en considérant une fonction objectif plus générale donnée par $g(x) = x^h$ avec $h > 1$ ils montrent que LPT a un ratio de performance constant par rapport à l'optimal et ce ratio dépend de h.
Une variante *On-line* de cette problématique où les tâches arrivent à la volée une par une est analysée par Avidor *et al.* [5]. Dans un premier temps ils démontrent que dans le cas de figure où $g(x) = x^2$, le ratio de performance de l'algorithme glouton LPT tend asymptotiquement vers 4/3 quand m tend vers l'infini. Dans un deuxième temps ils

généralisent cette analyse en considérant $g(x) = x^h$ avec h fixé qui vérifie $h > 1$. Dans ce cas, ils démontrent que le ratio de performance de LPT est $(2 - \Theta((\ln h)/h))$.
Alon *et al.* [2] considèrent une variante encore plus générale où $g(x)$ est une fonction arbitraire. Dans cette contribution, ils proposent un schéma d'approximation polynomial qui approxime le problème à un facteur $(1 + \epsilon)$ dans le cas où $g(x)$ vérifie les deux propriétés suivantes

- $g(x)$ est une fonction croisante et convexe dans $\mathbb{R}_+^*$.
- $\forall \epsilon > 0, \exists \delta > 0$ qui peut dépendre de ϵ, la fonction $g(x)$ doit nécessairement vérifier :

$$\forall x, y \geq 0,\ (1-\delta)x \leq y \leq (1+\delta)x \Rightarrow (1-\epsilon)g(x) \leq g(y) \leq (1+\epsilon)g(x). \quad (5.5)$$

La complexité de cette $\mathcal{PTAS}$ est bornée par $\mathcal{O}(n)$ où la constant implicite est exponentielle en la précision ϵ. Toutefois, les auteurs démontrent aussi que dans ce cas de figure, l'ordonnancement *list* a un ratio de performance constant pour n'importe quelle fonction $g(x)$ qui vérifie les deux propriétés décrites ci-dessus.

5.3.2 Optimisation multi-objectifs

Plusieurs problématiques d'optimisation existent où les utilisateurs souhaitent optimiser plusieurs fonctions objectifs à la fois. C'est le cas ici où nous nous intéressons à la minimisation du temps de complétion et à la maximisation de la probabilité de succès de l'application. Dans ce cas où les objectifs sont multiples, diverses méthodes d'optimisation peuvent être utilisées pour résoudre le problème. Le choix entre une méthode ou une autre dépend intrinsèquement de la nature des fonctions objectifs. Nous présentons dans la suite de cette partie une synthèse des diverses approches existantes. Cette synthèse est inspirée de la thèse de Erik Saule [86].

Nous commençons par la présentation des approches de type agrégation. L'idée principale de cette approche est l'agrégation des fonctions objectifs dans une seule fonction mono-objectif à optimiser. Plusieurs formes d'agrégation sont envisageable. La première qui vient à l'esprit est la combinaison convexe des différents objectifs [53]. Ainsi, en passant de plusieurs objectifs vers une seule fonction objectif, la problématique est résolue via les approches classiques d'optimisation mono-objectif. Cependant cette approche est limitée par deux éléments. Premièrement, assez souvent les objectifs ont des échelles et des unités de mesure différentes (comme en ce qui nous concerne par exemple). De plus les fonctions objectifs peuvent avoir des domaines de définition différents. Donc il y aura forcément des solutions hors d'atteinte. Deuxièmement, même si les fonctions objectifs sont proches, une infinité de combinaisons convexes rend le choix des bons facteurs de pondération une autre problématique à part entière.

Une deuxième approche de hiérarchisation peut aussi être utilisée pour passer de la problématique multi-objectifs vers une problématique mono-objectif [69]. Dans cette approche, les objectifs sont optimisés séparément l'un après l'autre en hiérarchisant les fonctions objectifs selon un ordre de priorité défini à l'avance. Cet ordre est donné par exemple par l'utilisateur final. Toutefois il n'est pas toujours évident de trouver un ordre ou une règle de priorité entre les différents objectifs et on obtient aussi moins de solutions.

Finalement, la dernière approche que nous utilisons dans nos travaux est l'approche de résolution via l'ajout de contraintes. Dans cette approche, les différents objectifs

sont optimisés de façon coordonnée. Ceci est mis en œuvre en fixant des contraintes sur une partie des objectifs et en optimisant les autres objectifs sous ces contraintes. Ainsi, en variant les contraintes sur les différents objectifs, un ensemble de solutions est calculé. Cet ensemble contient des solutions qui sont à un ratio de performance garanti par rapport à un objectif en vérifiant des contraintes par rapport aux autres objectifs. Par exemple, nous pouvons imaginer dans notre cas l'optimisation du temps de complétion sous une certaine contrainte de fiabilité ou l'inverse. Par contre le calcul d'un tel ensemble de solutions peut être couteux en terme de coût de calcul. Par exemple rien que l'optimisation du problème $2||C_{max}$ sans contrainte est un problème $\mathcal{NP}$-complet. De plus, contrairement au cas de figure des problématiques mono-objectif où une relation d'ordre total est induite par la fonction objectif, dans le cas des problématiques multi-objectifs dans la majorité des cas les solutions ne sont pas comparables. Donc il est difficile de choisir les solutions les plus intéressantes sur cet ensemble en absence d'une relation d'ordre totale entre les éléments. Néanmoins dans certains cas une relation d'ordre partiel est utilisée pour choisir des solutions qualifiées de solutions Pareto-optimales. La notion de relation de Pareto-dominance introduite dans [99] se définie formellement comme suit. Considérons une problématique avec l fonctions objectifs notées respectivement par $g_1(x), g_2(x), \cdots, g_l(x)$. Supposons que l'objectif est la maximisation des gains.

Définition 5.1. La solution $\mathcal{S}$ Pareto-domine une autre solution $\mathcal{S}'$ si et seulement si $\forall h \in [1..l],\ g_h(\mathcal{S}') \leq g_h(\mathcal{S})$ et $\exists h \in [1..l]\ |g_h(\mathcal{S}') < g_h(\mathcal{S})$.

Toutefois cette relation d'orde reste partielle puisqu'il y a des cas où les solutions ne sont comparables. Considérons par exemple deux solutions $\mathcal{S}, \mathcal{S}'$ qui vérifient $\exists h \in [1..l]|g_h(\mathcal{S}) < g_h(\mathcal{S}')$ et $\exists h' \in [1..l]|g_{h'}(\mathcal{S}') < g_{h'}(\mathcal{S})$, dans ce cas ces solutions sont Pareto-indépendantes. En se basant sur cette définition de Pareto-dominance, une solution $\mathcal{S}$ est qualifiée de solution Pareto-optimale si et seulement si elle n'est Pareto-dominée par aucune autre solution valide.

Définition 5.2. $\mathcal{S} \in V$ est une solution Pareto-optimale si et seulement si : $\forall \mathcal{S}' \in V\ |\mathcal{S}' \neq \mathcal{S},\ \exists h \in [1..l] \mid g_h(\mathcal{S}) < g_h(\mathcal{S}') \Rightarrow \exists h' \in [1..l] \mid g_{h'}(\mathcal{S}') < g_{h'}(\mathcal{S})$.

Les solutions qui vérifient cette définition forment l'ensemble des solutions Pareto-optimales noté par $\mathcal{P}^*$, cet ensemble est aussi appelé le front des solutions Pareto-optimales. Finalement soit $g_1^*, g_2^*, \cdots g_l^*$ les valeurs optimales qu'une solution du front Pareto-optimal peut avoir, nous appelons zénith $\mathcal{Z}$, le point qui peut être non valide qui vérifie $\forall S \in V,\ \forall h \in [1..l],\ g_h(\mathcal{S}) \leq g_h^* = g_h(\mathcal{Z})$.

Toutefois dans la majorité des cas la cardinalité de l'ensemble des solutions Pareto-optimales est exponentiel en la taille de l'instance. Ceci implique que l'énumération des solutions de cet ensemble est une tâche très couteuse en temps de calcul. Donc dans ce cas nous utilisons des méthodes d'approximation pour calculer un autre ensemble de solutions qui approche à un facteur costant toute solution valide du front Pareto-optimal. Dans ce travail nous utilisons l'approche générique proposée par Papadimitriou *et al.* [82] pour calculer un autre ensemble qui contient un nombre de solutions raisonnable (polynomial en la taille de l'instance) tel que pour chaque point du front des solutions Pareto-optimales il existe forcément un point qui l'approxime à facteur constant. Formellement considérons une problématique de maximisation à deux objectifs.

Définition 5.3. L'ensemble $\mathcal{P}_a^*$ est une $<\rho_1, \rho_2>$-approx de l'ensemble $\mathcal{P}^*$ si et seulement si $\forall \mathcal{S}^* \in \mathcal{P}^*$, $\exists \mathcal{S} \in \mathcal{P}_a^* \mid g_1(\mathcal{S}^*) \leq \rho_1 g_1(\mathcal{S})$ et $g_2(\mathcal{S}^*) \leq \rho_2 g_2(\mathcal{S})$ avec $1 \leq \rho_1$, $1 \leq \rho_2$.

Pour construire cet ensemble de solutions approchées Papadimitriou *et al.* proposent une méthodologie en deux étapes.
Dans un premier temps il faut concevoir une $<\overline{\rho}_1, \rho_2>$-approx pour calculer une solution $\mathcal{S}$ qui pour une contrainte w donnée sur le premier objectif vérifie à la fois les deux propriétés suivantes :

1. la solution calculée en fonction de la contrainte w est au plus à un facteur ρ_1 de w, formellement,
$$w \leq \rho_1 g_1(\mathcal{S}).$$
2. la solution calculée est au plus à un facteur ρ_2 de toute autre solution Pareto-optimale qui est à un facteur 1 de w, formellement,
$$\forall \mathcal{S}^* \in \mathcal{P}^* \mid w \leq g_1(\mathcal{S}^*) \Rightarrow g_2(\mathcal{S}^*) \leq \rho_2 g_2(\mathcal{S}).$$

Dans un deuxième temps en utilisant cette $<\overline{\rho}_1, \rho_2>$-approx une $<\rho_1 + \epsilon, \rho_2>$approx du front des solutions Pareto-optimales est calculée moyennant l'Algorithme 2 décrit ci-dessous. En effet cet algorithme calcule des solutions d'approximation du front Pareto-optimal moyennant la $<\overline{\rho}_1, \rho_2>$-approx en faisant varier à chaque fois la contrainte w. Plus précisément, après avoir fixé la borne supérieure R_b et la borne inférieure L_b que le premier objectif peut atteindre, w est incrémenté à chaque itération selon une suite géométrique et pour chaque valeur de w une solution d'approximation est calculée.

Algorithme 2 Algorithme d'approximation du front Pareto-optimal [82]

```
function <ρ1 + ε, ρ2>-APPROX(ε, ρ1, ρ2, Rb, Lb)
    k ← 0
    P*_a ← ∅                                  ▷ Ensemble des solutions Pareto-optimales
    while k ≤ ⌈log_{1+ε/ρ1}(Rb/Lb)⌉ do
        w_k ← (1 + ε/ρ1)^k Lb
        S_k ← <ρ̄1, ρ2>-APPROX(w_k)
        P*_a ← P*_a ∪ S_k
        k ← k + 1
    end while
    return P*_a
end function
```

Théorème 5.1. *L'Algorithme 2 est une $<\rho_1+\epsilon, \rho_2>$-approx de l'ensemble des solutions Pareto-optimales.*

Démonstration. Soit $\mathcal{S}^* \in \mathcal{P}^*$ une solution Pareto-optimale arbitraire.
Considérons $\epsilon \in \mathbb{R}_+^*$ un degré de précision fixé à l'avance.
Puisque $L_b \leq g_1(\mathcal{S}^*)$, $\exists k \in \mathbb{N}$ qui vérifie :
$$(1 + \epsilon/\rho_1)^k L_b \leq g_1(\mathcal{S}^*) \leq (1 + \epsilon/\rho_1)^{k+1} L_b.$$
Soit $\mathcal{S}_k$ la solution produite par la $<\overline{\rho}_1, \rho_2>$-approx sous la contrainte $w = (1+\epsilon/\rho_1)^k L_b$.
Nous obtenons :

– la propriété 2 de la fonction $<\overline{\rho}_1, \rho_2>$-approx garantit que :

$$g_2(\mathcal{S}^*) \leq \rho_2 g_2(\mathcal{S}_k).$$

– la construction et la propriété 1 de la fonction $<\overline{\rho}_1, \rho_2>$-approx garantissent que $(1+\epsilon/\rho_1)^k L_b \leq \rho_1 g_1(\mathcal{S}_k)$ et $g_1(\mathcal{S}^*) \leq (1+\epsilon/\rho_1)^{k+1} L_b$, alors,

$$g_1(\mathcal{S}^*) \leq (\rho_1 + \epsilon) g_1(\mathcal{S}_k).$$

□

Pour clôturer cette partie nous notons que la cardinalité de l'ensemble $\mathcal{P}_a^*$ dépend essentiellement du codage des valeurs de la fonction objectif. Plus précisément le nombre d'itérations k est borné par $\mathcal{O}\left(\log_{1+\epsilon/\rho_1}(R_b/L_b)\right)$, donc la cardinalité dépend de la taille du ratio R_b/L_b. Nous tenons à préciser que dans notre cas, la cardinalité de cet ensemble est bien polynomiale puisque le ratio R_b/L_b est donné par le ratio entre le *makespan* en pire des cas et le *makespan* optimal. Ainsi l'expression $\log_{1+\epsilon/\rho_1}(R_b/L_b)$ est bornée par un polynôme en la taille de l'instance ceci implique directement que la cardinalité de $\mathcal{P}_a^*$ est polynomiale en la taille de l'instance. Dans ce cas la complexité de l'Algorithme 2 est aussi polynomiale si la complexité de la fonction $<\overline{\rho}_1, \rho_2>$-APPROX invoquée à chaque itération est aussi bornée par un polynôme.

5.3.3 Résultats dans le cadre de la tolérance aux pannes

Nous présentons dans cette partie les différentes approches et techniques d'ordonnancement qui sont conçues et plus orientées vers les problématiques de tolérance aux pannes. Ces différentes approches présentées dans cette section se distinguent sur plusieurs plans. Dans cette synthèse nous distinguons essentiellement trois plans. Le premier plan concerne la technique de tolérance aux pannes considérée pour réduire l'impact des pannes sur l'application. Le deuxième point qui diffère d'un travail d'un autre est essentiellement l'ensemble des hypothèses utilisées pour modéliser le couple modèle d'application et modèle de plateforme d'exécution considérées. Par exemple, les processeurs ont ils le même taux de traitement d'instructions ? Ont-ils le même taux de panne, les tâches sont-elles indépendantes ? Ont-elles la même durée d'exécution ou sont-elles reliées par une relation de précédence particulière ? Le dernier point qui distingue un travail à un autre, concerne l'ensembles des critères de performance à optimiser.
Dans la suite de cette synthèse nous regroupons les différentes approches et contributions existantes dans deux grandes classes selon la technique de tolérance aux pannes utilisée. Ainsi, nous présentons tout d'abord dans la première classe, les approches de tolérance aux pannes qui sont destinées à des applications où aucun mécanisme supplémentaire de tolérance aux pannes est mis en œuvre. En effet, dans ce cas de figure, le seul levier d'optimisation que ces approches peuvent manipuler pour tolérer les pannes, est l'allocation spatiale et temporelle des tâches sur la plateforme d'exécution. Ensuite nous décrivons dans un deuxième temps la seconde classe qui contient les approches et techniques d'ordonnancement qui ont été développées pour les applications où la duplication active des tâches est mise en œuvre pour tolérer les pannes.

Nous entamons la présentation des travaux proposés dans les contextes d'exécution où aucun mécanisme supplémentaire de tolérance aux pannes n'est mis en œuvre.

Wei *et al.* [70] : Pour commencer nous présentons les travaux de Wei *et al.* [70] qui considèrent un ensemble de tâches arbitraires et indépendantes. Chaque tâche i a une durée d'exécution probabiliste distribuée selon une loi générale qui est la même pour toutes les tâches, un facteur de priorité w_i et une date d'échéance d_i de nature probabiliste distribuée selon une loi exponentielle avec un paramètre identique pour toutes les tâches. Cet ensemble de tâches est ordonnancé sur un seul processeur qui est sujet à des pannes de nature transitoire. Les durées de disponibilité de ce processeur sont distribuées selon une loi type phase (*i.e. phase-type distribution*). Par contre, les durées d'indisponibilité sont distribuées selon une distribution arbitraire. Dans ce travail les auteurs s'intéressent à l'optimisation de trois fonctions objectif différentes telles que la minimisation de l'espérance de la somme des temps de terminaison pondérés $\mathbb{E}\left[\sum_{i\in\mathcal{T}} w_i C_i\right]$, la minimisation de la somme des espérances des durées de dépassement pondérées $\mathbb{E}\left[\sum_{i\in\mathcal{T}|C_i>d_i} w_i(C_i-d_i)\right]$ et la minimisation de l'espérance du nombre pondéré des tâches en retard par rapport à leurs date d'échéance $\mathbb{E}\left[\sum_{i\in\mathcal{T}|C_i>d_i} w_i\right]$. Ainsi la question de cette problématique est la suivante : quelle est la permutation optimale des tâches qui minimise ces différents critères. Pour y répondre les auteurs considèrent quelques distributions particulières pour la loi de panne de type phase telles que la distribution Erlang et hyper-exponentielle. En s'appuyant sur les propriétés de la distribution de panne, la distribution des durées de tâches les auteurs exhibent des stratégies d'ordonnancement optimales pour chaque fonction objectif. Cependant nous tenons à signaler que dans ce travail les auteurs considèrent un modèle d'exécution totalement différent de celui de notre contexte car ils supposent que l'interruption des pannes ne cause pas la perte du travail entamé avant la panne mais, cause seulement un retard d'exécution.

Jeannot *et al.* [65] : Dans le même classe où aucun mécanisme de tolérance aux pannes est mis en œuvre. Jeannot *et al.* [65] étudient la problématique d'ordonnancement de tâches tolérant aux pannes sur une plateforme d'exécution parallèle. Dans ce travail les auteurs supposent que l'application est modélisée par un graphe orienté et acyclique où chaque sommet i représente une tâche à laquelle ils associent une quantité de travail p_i. Les arcs entre les tâches représentent la relation de précédence qui définit l'ordre d'exécution. La plateforme d'exécution est composée de m processeurs où chaque processeur j a un taux de traitement d'instructions ν_j, un taux de panne constant λ_j (*i.e.* processus de Poisson) et le régime de panne de chaque processeur est supposé permanent. En considérant différents modèles de précédence entre les tâches les auteurs s'intéressent à l'optimisation d'une fonction multi-objectifs. Plus précisément dans ce travail les auteurs s'intéressent à la minimisation à la fois de la date de complétion de la dernière tâche dans l'application C_{max} et à la maximisation de la fiabilité de l'application. Dans ce cadre d'analyse, ils démontrent que la relation entre les deux objectifs considérés est une relation antagoniste c'est-à-dire qu'on ne peut pas améliorer un objectif sans dégrader l'autre et de plus on ne peut pas approcher les deux objectifs simultanément à un facteur constant. Donc en absence de toute relation d'ordre totale entre les différentes stratégies d'ordonnancement où d'un ordonnancement optimal

absolu qui optimise les deux critères à la fois les auteurs s'intéressent plutôt au calcul de l'ensemble des solutions Pareto-optimales. Toutefois, ils démontrent que l'ensemble des solutions Pareto-optimales a une cardinalité exponentielle en la taille de l'instance même dans le cas où les tâches sont indépendantes et la plateforme est composée de deux processeurs un rapide mais pas fiable et un autre très lent mais fiable. Donc étant donnée cette cardinalité exponentielle, la solution qui reste envisageable est l'approximation de l'ensemble des solutions Pareto-optimales en un temps de calcul raisonnable.

Pour construire un ensemble de solutions de taille polynomiale qui approxime à un facteur constant les solutions du front Pareto-optimal, les auteurs mettent en place une approche pour des instances particulières en s'appuyant sur la méthodologie proposée par Papadimitriou *et al.* [82] décrite dans la section 5.3.2. En effet, dans un premier temps ils considèrent une famille d'instances où les tâches sont unitaires et indépendantes. Dans ce cas de figure ils proposent une $(1+\epsilon, 1)$-approximation du front Pareto-optimal. Le coût algorithmique de cette approche proposée est borné par $\mathcal{O}\left(m \log_{1+\epsilon}(n\nu_1)\right)$ avec $\epsilon \in \mathbb{R}^+$ et ν_1 est le taux de traitement du processeur le plus rapide. Ensuite dans un deuxième temps ils s'intéressent à une famille d'instances plus générale où les durées d'exécution des tâches sont arbitraires. En utilisant la même approche ils proposent dans ce cas une $(2+\epsilon, 1)$-approximation du front Pareto-optimal. Cette approche a aussi une complexité polynomiale bornée par $\mathcal{O}\left(n \log n + \log_{1+\epsilon/2}(R_b/L_b)(n + m \log m)\right)$ tel que R_b représente le temps de complétion de l'application sur le processeur le moins rapide et L_b est le temps de complétion minimal qu'un ordonnancement valide peut atteindre. Finalement en s'appuyant sur ces deux approches proposées ils constatent que le critère le plus important à considérer pour optimiser la fiabilité est le produit entre le taux de panne et le taux de traitement de chaque processeur. Ainsi en se basant sur cette observation il proposent une méthodologie pour convertir les différentes heuristiques d'ordonnancement des applications avec un graphe de précédence arbitraire dans le contexte des plateformes d'exécution parallèles et hétérogènes en des heuristiques qui prennent en considération la fiabilité de l'application comme critère supplémentaire d'optimisation.

Nous passons maintenant à la présentation des travaux qui utilisent le mécanisme de duplication active des tâches [88] pour tolérer l'arrivées des pannes et réduire leur impact sur l'application.

Malewicz [76] : Nous commençons par citer le travail de Malewicz [76] qui considère une application composée de n tâches séquentielles reliées entre elles par une relation de précédence acyclique. Cette application est exécutée sur une plateforme composée de m processeurs où la technique de duplication active des tâches [88] est utilisée pour tolérer les pannes. Chaque tâche i a une durée d'exécution unitaire et une probabilité de succès q_{ij} si elle est exécutée sur le processeur j. Dans ce travail Malewicz s'intéresse à l'analyse de la complexité du problème en considérant la minimisation de l'espérance du temps de complétion de la dernière tâche comme fonction objectif. Il montre que quand le nombre des machines ainsi que la largeur du graphe de précédences sont constants, un ordonnancement optimal est calculable en un temps polynomial. Par contre si un de ces paramètres est non borné, il démontre que le problème est dans la classe des problèmes $\mathcal{NP}$-complet même dans le cas où l'application est un ensemble de tâches

indépendantes. De plus le problème général est inapproximable avec un facteur de 5/4 sauf si $P = NP$.

Rajaraman *et al.* [71] : Dans la continuité des travaux de Malewicz [76], Rajaraman *et al.* [71] présentent des algorithmes d'approximation avec un coût algorithmique polynomial en la taille de l'instance en considérant des graphes particuliers de précédences entre les tâches. Dans un premier temps, ils présentent une $\mathcal{O}(\log n)$-approximation quand les tâches sont indépendantes. Ensuite en se basant sur ce résultat ils considèrent trois cas particuliers de graphes de tâches, où ils proposent une $\mathcal{O}(\log n \log m \log(n+m)/\log\log(n+m))$-approximation quand le graphe des tâches est une collection de chaînes disjointes, une $\mathcal{O}\left(\log^2 n \log m\right)$-approximation si le graphe est une collection de *out-in-trees* et finalement une $\mathcal{O}\left(\log^2 n \log m \log(n+m)/\log\log(n+m)\right)$-approximation dans le cas où le graphe est une forêt orientée.

Girault *et al.* [46] : Dans ce travail les auteurs s'intéressent à l'optimisation simultanément de la probabilité de succès et du temps de complétion de l'exécution dans l'environnement des applications temps réel et critiques utilisées dans les systèmes embarqués comme par exemple les ordinateurs de bords des machines aérospatiales ou encore dans le domaine de l'avionique. Dans cet environnement de système critique la réplication des tâches est mise en œuvre pour augmenter la probabilité de succès de l'application mais en contre partie comme tous les problèmes de tolérance aux pannes il y a un compromis à gérer puisque cette réplication détériore forcément le temps de complétion de l'application. Donc dans ce travail les auteurs ont deux leviers d'optimisation qui sont le degré de réplication et l'allocation spatiale et temporelle de chaque tâche. Formellement la problématique considérée est la suivante. Étant donnée une application composée de n tâches liées par une relation de précédence acyclique où chaque tâche i a une durée d'exécution arbitraire et qui dépend du processeur sur lequel elle est exécutée notée par p_{ij}. Cette application est exécutée sur une plateforme d'exécution composée de m processeurs qui sont sujets à des pannes transitoires. Chaque processeur a son propre taux de panne qui est supposé constant en fonction du temps (*i.e.* processus de Poisson) noté par λ_j et un taux d'indisponibilité négligeable ($\gamma_j \approx 0$). L'objectif de cette problématique est le calcul d'une allocation spatiale et temporelle des tâches qui garantit une certaine probabilité de succès de l'application et que la date de complétion de la dernière tâche ne dépasse pas une certaine échéance. Pour commencer, les auteurs analysent en premier lieu la relation entre les deux objectifs considérés. Cette analyse aboutit à la même conclusion présentée auparavant dans le cas sans réplication active et qui stipule que les deux critères sont antagonistes par nature. Ensuite en s'appuyant sur le formalisme *RBD* (*i.e. Reliability Block Diagram*) [73] ils indiquent que le calcul de la probabilité de succès de l'application en tenant compte de la réplication est un problème $\mathcal{NP}$-complet. Pour dépasser cette problématique ils adoptent un modèle d'exécution de type série parallèle, où chaque tâche ne peut commencer son exécution que lorsque toutes les copies des tâches de son ensemble de prédécesseurs sont terminées. Ainsi en s'appuyant sur cette modélisation, le calcul de la probabilité de succès peut être fait en un temps polynomial en la taille de l'instance [93]. Ensuite en s'appuyant sur la nature sans mémoire de la distribution de panne qui implique que la probabilité de succès d'une tâche ne dépend pas de l'instant où la tâche commence, en conséquence la probabilité de

succès de l'application dépend seulement de l'allocation spatiale des tâches. Ils proposent dans un premier temps un algorithme glouton de nature aléatoire pour calculer une allocation spatiale des tâches qui garantit que la probabilité de succès de l'application est supérieure à un certain seuil fixé à l'avance. Cet algorithme duplique à chaque itération la tâche critique dont la probabilité de panne est inférieure au seuil fixé et si une telle tâche n'existe pas une autre tâche est choisie au hasard. Cette procédure de duplication est répétée tant que le seuil de probabilité visé n'est pas atteint. Une fois que le seuil de probabilité de succès est atteint ils abordent la phase du calcul de l'allocation temporelle qui minimise le temps de complétion de la dernier tâche sous la contrainte de l'allocation spatiale calculée auparavant. Cependant il est bien connu que résoudre optimalement le problème d'ordonnancement de tâches sous des contraintes spatiales est un problème $\mathcal{NP}$-complet au sens fort [32]. Pour contourner ce problème dur, ils mettent en œuvre une stratégie gloutonne d'ordonnancement de type *list* pour calculer une allocation temporelle sous la contrainte de l'allocation spatiale. Finalement, ils montrent qu'aucune borne supérieure à un facteur constant ne peut être établie pour cet algorithme glouton et au pire des cas il peut être à un facteur m de l'optimal.

Girault et Kalla [45] : Dans le même environnement des systèmes embarques critiques Girault et Kalla [45] formulent aussi un problème d'optimisation multi-objectifs avec des différences de modélisation et des critères d'optimisation différents. Plus précisément ils considèrent une application composée de n tâches liées par une relation de précédence qui représente aussi le schéma de communication entre eux. Cette relation est décrite moyennant un graphe orienté et acyclique. Les noeuds de ce graphe sont les tâches et les arcs représentent les messages de communication entre les tâches. Cette application est exécutée sur une plateforme parallèle composée de m processeurs hétérogènes où chaque processeur j possède son propre taux de traitement ν_j. Tout processeur est sujet à des pannes transitoires qui arrivent selon un processus de Poisson et le taux d'indisponibilité est toujours supposé négligeable ($\gamma_j \approx 0$). Cet ensemble de processeurs est lui aussi représenté par un graphe biparti tel que les noeuds de ce graphe sont les processeurs et les arcs sont les liens de communication entre les processeurs qui sont eux aussi sujets à des pannes transitoires distribuées selon un processus de Poisson. De même dans ce travail pour éviter l'explosion combinatoire lors du calcul de la probabilité de succès en tenant compte de la duplication, les auteurs proposent un ordonnancement de type série parallèle entre les différentes copies. Par contre dans ce travail les auteurs s'intéressent à un nouveau critère de fiabilité appelé le taux de panne global du système au lieu de l'optimisation de la probabilité de succès de l'application. Plus précisément le taux de panne global est donné par le ratio entre le logarithme de la probabilité de succès de toute l'application et la quantité de travail totale exécutée sur l'ensemble des processeurs y compris les tâches dupliquées. Ainsi le taux de panne est exprimé par unité de temps, ceci implique que cet objectif n'est plus directement lié à l'allocation spatiale et temporelle de l'application sur les processeurs mais il est exprimé par unité de temps. Ensuite, pour optimiser ces deux critères simultanément ils proposent une stratégie d'ordonnancement de tâches qui à la fois vérifie une certaine contrainte sur le taux de panne global du système et minimise le temps de complétion de la dernière tâche. Cependant, l'utilisation de cette stratégie est limitée aux cas des instances avec un nombre des processeurs relativement petit puisque sa complexité est

en $\mathcal{O}(n2^m)$.

Saule *et al.* [87] : Un autre travail dans le thème de l'optimisation multi-objectifs a été mené par Saule *et al.* [87] dans un autre cadre d'exécution où la fonction objectif est toujours l'optimisation à la fois de la probabilité de succès de l'application et le temps de complétion de la dernière tâche. Dans ce travail les auteurs considèrent principalement deux couples différents pour modéliser l'application et la plateforme d'exécution où la duplication active est utilisée comme mécanisme de tolérance aux pannes. Dans un premier temps ils supposent que l'application est une chaîne de n tâches où chaque tâche i a une durée d'exécution p_{ij} sur le processeur j. Cette chaîne de tâches est exécutée sur une plateforme composée de m processeurs. Chaque processeur est sujet à des pannes transitoires qui surviennent selon un processus de Poisson avec un taux λ_j tandis que la durée d'indisponibilité est supposée négligeable. En considérant ce modèle d'exécution et en s'appuyant sur la méthodologie d'approximation de l'ensemble des points Pareto-optimaux [82] ils proposent une $((1+\epsilon_1)(1+\epsilon_2), 1)$-approximation du problème. Ainsi chaque point du front Pareto-optimal est au plus à un facteur $(1+\epsilon_1)(1+\epsilon_2)$ pour le critère temps de complétion et à un facteur 1 pour le critère de la probabilité de succès. Toutefois, la complexité de cette stratégie est relativement élevée puisque l'algorithme d'approximation utilisé à une complexité pseudo-polynomiale en $\mathcal{O}(n^2m/\epsilon_2)$ ainsi la complexité global est en $\mathcal{O}\left(\log_{1+\epsilon_1}(R_b/L_b)n^2m/\epsilon_2\right)$.
Ensuite dans un deuxième temps ils considèrent un modèle d'application où les tâches sont indépendantes et de durées arbitraires notées par p_i. La plateforme d'exécution est composée de m processeurs identiques sur le plan fiabilité et puissance de calcul. Ainsi, dans ce cas de figure aussi un résultat similaire au précédent est développé, dans lequel un schéma de programmation dynamique est proposé pour calculer un facteur de réplication optimal de chaque tâche sous la contrainte d'une quantité de travail totale à ne pas dépasser. Finalement, en couplant le schéma de programmation dynamique avec la méthodologie de Papadimitriou [82] ils aboutissent à une $((2+\epsilon_1)(1+\epsilon_2), 1)$-approximation du front Pareto-optimal avec un coût algorithmique en temps borné par $\mathcal{O}\left(\log_{1+\epsilon_1}(\frac{C^{max}}{C^{min}_{max}})n^2m/\epsilon_2\right)$.

Pour récapituler et placer notre contribution par rapport aux travaux existants, nous tenons à préciser que tous les travaux présentés dans cette section adoptent deux approches pour modéliser l'inter-arrivées des pannes. Dans la première approche, un processus de Poisson avec un taux λ_j qui dépend du processeur est utilisé pour modéliser l'inter-arrivées des pannes. Tandis que dans la seconde approche qui est plus ou moins une généralisation de la première, une probabilité de succès q_{ij} qui dépend du processeur j est associée aux tâches. Cette approche est une généralisation de la première en considérant $q_{ij} = \exp^{\lambda_j p_i/\nu_j}$. Ces deux modèles impliquent que quel que soit l'instant dans le temps où la tâche i est ordonnancée la probabilité de succès ne dépend que du processeur grâce à la propriété sans mémoire de la distribution de panne. Cependant, nous tenons à signaler que ces deux hypothèses ne sont plus tout à fait valables dans les plateformes d'exécution actuelles. Nous citons l'exemple des deux études menées récemment [89, 56] dans le contexte des plateformes HPC où la distribution de panne suit une loi de Weibull avec un taux de panne décroissant. Nous tenons à signaler que ceci est aussi le cas de la plateforme de calcul volontaire BOINC où la majorité des

distributions de disponibilité sont de type Weibull avec un taux de panne décroissant. Ces études montrent que le taux de panne n'est pas constant, donc la probabilité de succès d'une tâche donnée dépend forcément de l'allocation temporelle de cette tâche.

Ainsi dans ce chapitre nous proposons des extensions et des analyses plus détaillées des approches d'ordonnancement tolérant aux pannes en considérons une distribution de panne arbitraire avec un taux de panne variable et qui est représentatif des plateformes de calcul actuelles. Plus précisément, dans la suite de ce chapitre nous proposons une extension de l'approche proposée par Jeannot *et al.* [65], en considérant une distribution de panne arbitraire au lieu du processus de Poisson.

5.4 Optimisation du *makespan* et de la fiabilité de l'application

Dans cette section nous nous intéressons à l'optimisation à la fois de la probabilité de succès et le *makespan* de l'application. Nous notons tout d'abord que la totalité des résultats présentés dans ce chapitre reposent sur la théorie de la majoration [77]. Une introduction d'un ensemble de notions et notations mathématiques qui sont utilisées pour démontrer la majorité des résultats sont présentés dans l'annexe (Section 6.3). Le plan de cette section est le suivant. Nous analysons dans un premier temps le probléme d'optimisation multi-objectifs $Q||(C_{max}, \mathcal{R}_l)$. Dans ce problème nous nous intéressons à l'optimisation du *makespan* et du logarithme de la fiabilité en considérant une distribution de panne arbitraire avec un taux de panne décroissant DFR et des processeurs uniformes. Dans un deuxième temps nous analysons le deuxième problème $P||(C_{max}, \overline{\mathcal{R}}_l)$ où nous gardons les mêmes objectifs mais cette fois dans le cas de figure où les processeurs sont identiques et le taux de panne $\lambda(t)$ est supposé croissant *IFR*.

5.4.1 Optimisation de $Q||(C_{max}, \mathcal{R}_l)$ pour une loi *DFR*

Nous considérons dans cette partie le cas de figure où les processeurs sont uniformes et leur distribution de panne est identique de nature DFR tels que le processus de Poisson ou la distribution Weibull avec un paramètre de forme inférieur à 1 ($\beta_w \leq 1$). Pour commencer nous introduisons les notations suivantes. Soit $\mathcal{C}^{\mathcal{S}} = \{C_1^{\mathcal{S}}, \cdots, C_m^{\mathcal{S}}\}$ le vecteur des temps de complétion d'une stratégie d'ordonnancement $\mathcal{S} = (\xi, \zeta)$. Nous tenons à préciser que l'indice $\mathcal{S}$ est omis en notant $\mathcal{C}^{\mathcal{S}}$ par $\mathcal{C}$ dans le cas où aucune confusion n'est possible. Dans le cas où l'ordonnancement est compact C_j est donné par ω_j/ν_j où ω_j est la quantité de travail totale affectée au processeur j qui a un taux de traitement ν_j. Soit $E \subset \mathbb{R}_+^m$ l'ensemble qui contient les vecteurs des temps de complétion possibles qui vérifient la contrainte suivante :

$$\mathcal{C} \in E \text{ si et seulement si } \sum_{j=1}^{m} C_j \nu_j = \sum_{i=1}^{n} p_i. \tag{5.6}$$

Ainsi E contient forcément tous les ordonnancements possibles et valides où la totalité de la quantité de travail est forcément ordonnancée sur les processeurs disponibles. Donc l'ensemble des ordonnancements valides V vérifie nécessairement $V \subset E$. Nous considérons par convention que $\nu_m \leq \cdots \leq \nu_1$, en conséquent le processeur 1 possède

le taux de traitement le plus élevé. Nous notons par convention $C_{[k]}$ (respectivement $C_{(k)}$) le $k^{éme}$ temps de complétion en considérant un ordre décroissant (respectivement croissant) tel que $C_{[m]} \leq C_{[m-1]} \leq \cdots \leq C_{[1]}$ (respectivement $C_{(1)} \leq C_{(2)} \leq \cdots \leq C_{(m)}$).

Ayant introduit les notations utilisées dans cette section, nous entamons tout d'abord l'analyse de la problématique $Q||\mathcal{R}_l$ où nous abordons l'optimisation de la fiabilité de l'ordonnancement sans se soucier du *makespan*. Pour commencer nous exhibons quelques propriétés mathématiques importantes de la fonction objectif $\mathcal{R}_l$ quand le taux de panne est décroissant.

Lemme 5.1. *Soit la fonction $g(x) = \log \overline{F}(x)$, la fonction $\mathcal{R}_l(\mathcal{C}) = \sum_{j=1}^{m} g(C_j)$ est Schur-convexe et décroissante si $F(x)$ est de nature* DFR.

Démonstration. Nous rappelons que si la fonction $\lambda(t)$ est décroissante alors $g : x \to \log \overline{F}(x)$ est une fonction convexe en x d'après le Théorème 6.2.
Nous notons aussi que

$$g(x) = \log \overline{F}(x) = -\int_0^x \lambda(t)dt$$

est décroissante en x.
Alors dans ce cas en se basant la Proposition 6.1 de l'annexe, la fonction,

$$\mathcal{R}_l(\mathcal{C}) = \sum_{j=1}^{m} \log \overline{F}(C_j),$$

est Schur-convexe et décroissante. □

Ainsi d'après cette propriété de la fonction objectif, nous concluons que l'optimisation de la probabilité de succès est un problème simple quand le taux de panne est décroissant. En effet une probabilité de succès maximale est obtenue moyennant la stratégie d'ordonnancement notée par $\mathcal{S}_1$ dans laquelle toutes les tâches sont affectées au processeur le plus rapide. Ceci est une conséquence directe de la propriété *DFR* de la distribution de panne sur la fonction objectif, puisque quand le taux de panne est décroissant la fonction objectif $\mathcal{R}_l(\mathcal{C})$ est Schur-convexe et décroissante.

Théorème 5.2. *Soit $\mathcal{S}_1$ un ordonnancement où toutes les tâches sont affectées au processeur le plus rapide, alors $\mathcal{S}_1$ est une solution optimale du problème $Q||\mathcal{R}_l$ si $F(x)$ est* DFR.

Démonstration. Supposons que l'application contienne w unités de travail : $\sum_{i=1}^{n} p_i = w$.
D'où par construction $\mathcal{C}^{\mathcal{S}_1} = \{\frac{w}{\nu_1}, 0, \cdots, 0\}$.
Considérons un vecteur arbitraire des temps de complétion valide noté par $\mathcal{C} \in E$.
Par construction la somme partielle des éléments dont leurs rangs k est entre 1 et $m-1$ en considérant un ordre croissant sur les C_j vérifie bien :

$$\forall k < m, \sum_{j=1}^{k} C_{(j)} \geq \sum_{j=1}^{k} C_{(j)}^{\mathcal{S}_1} = 0.$$

Pour $k = m$ il est évident de vérifier que

$$\mathcal{C} \in E \Rightarrow \sum_{j=1}^{m} C_j \geq \sum_{j=1}^{m} \frac{\omega_j}{\nu_1} = \frac{w}{\nu_1}$$

Donc $\forall \mathcal{C} \in E$ nous obtenons $\mathcal{C} \prec^w \mathcal{C}^{\mathcal{S}_1}$(voir l'annexe pour la définition de la sur-majoration $\prec^w$).
La fonction objectif $\mathcal{R}_l(\mathcal{C})$ est Schur-convexe et décroisante alors en s'appuyant sur le résultat 6.4 du Théorème 6.1, $\mathcal{R}_l(\mathcal{C})$ préserve toute relation de sur-majoration.

$$\forall \mathcal{C} \in E,\ \mathcal{C} \prec^w \mathcal{C}^{\mathcal{S}_1} \Rightarrow \mathcal{R}_l(\mathcal{C}) \leq \mathcal{R}_l(\mathcal{C}^{\mathcal{S}_1})$$

□

Donc $\mathcal{S}_1$ est un ordonnancement valide dont la fiabilité majore tout autre ordonnancement valide puisque $V \subset E$. Toutefois, la stratégie d'ordonnancement $\mathcal{S}_1$ qui maximise la probabilité de succès de l'application peut être loin d'un facteur m de l'ordonnancement optimal pour le *makespan*. Ainsi il est évident qu'en aucun cas nous ne pouvons avoir un ordonnancement que l'on qualifie d'optimal global pour les deux critères. Par conséquent pour gérer le compromis entre ces deux critères antagoniste nous pouvons calculer l'ensemble des ordonnancements Pareto-optimaux et choisir parmi cet ensemble l'ordonnancement le plus convenable aux critères de l'utilisateur final. Donc après avoir étudier la problématique $Q||\mathcal{R}_l$, la deuxième question soulevée est : "est-t-il possible d'énumérer tous les ordonnancements du front Pareto-optimal moyennant un algorithme dont la complexité en temps est raisonnable pour la problématique multi-objectifs $Q||(C_{max}, \mathcal{R}_l)$?".

a Tâches indépendantes et unitaires

Nous considérons dans cette partie une version relaxée du problème où l'application est composée de n tâches unitaires. L'instance d'entrée du problème est un entier $n \in \mathbb{N}^*$, un ensemble de processeurs $\mathcal{Q} = \{\nu_1, \cdots, \nu_m\}$ et la fonction de distribution de panne notée par $F(x)$ et supposée de nature *DFR*. Pour commencer nous analysons la cardinalité de l'espace des solutions Pareto-optimales.

Théorème 5.3. *La cardinalité de l'ensemble des solutions Pareto-optimales est exponentielle en la taille de l'instance pour certaines instances du problème* $2|p_i = 1|(C_{max}, \mathcal{R}_l)$.

Démonstration. Considérons une instance particulière avec $2n$ tâches unitaires et deux processeurs identiques.
Supposons que la fonction $g(x) = \log \overline{F}(x)$ soit une fonction strictement convexe (ceci est obtenu en considérant une distribution de panne dont le taux est décroissant monotone par exemple une distribution Weibull telle que $\overline{F}(x) = \exp^{-x^{1/2}}$ d'où $g(x) = -\sqrt{x}$).
Ainsi l'inégalité décrite dans la Définition 6.1 est stricte.
Puisque les tâches sont unitaires alors toute stratégie d'ordonnancement est définie par le nombre de tâches qu'elle affecte au processeur 1 ou 2.
Soit, $\mathcal{S}_i$ la stratégie qui affecte $2n - i + 1$ au processeur 1 et $i - 1$ sur le deuxième processeur.
Ainsi pour chaque valeur de i et δ qui vérifient, $1 \leq i \leq n+1$ et $1 \leq \delta \leq n - i + 1$ nous obtenons les inégalités suivantes :
Le *makespan* de $\mathcal{S}_{i+\delta}$ est strictement inférieur à celui de $\mathcal{S}_i$.

$$C_{max}^{\mathcal{S}_{i+\delta}} < C_{max}^{\mathcal{S}_i}. \tag{5.7}$$

La fiabilité $\mathcal{S}_{i+\delta}$ est strictement inférieure à celle de $\mathcal{S}_i$.

$$\mathcal{R}_l(\mathcal{S}_i) = g(2n-i+1) + g(i-1) > g(2n-i-\delta+1) + g(i-1+\delta) = \mathcal{R}_l(\mathcal{S}_{i+\delta}) \quad (5.8)$$

Donc les inégalités 5.7,5.8 impliquent que l'ensemble des solutions Pareto-optimales est en $\mathcal{O}(n)$, puisque pour tout i dans $[1..n+1]$, $\mathcal{S}_i$ est une solution Pareto-optimale. □

Le Théorème 5.3 indique que l'énumération de tous les ordonnancements Pareto-optimaux est une solution coûteuse puisque la cardinalité de l'ensemble des solutions Pareto-optimales est exponentielle en la taille de l'instance. Pour dépasser l'énumération de toutes les solutions du front Pareto-optimal $\mathcal{P}^*$ nous utilisons la méthodologie proposée par Papadidimitriou *et al.* [82] pour calculer un ensemble d'approximation $\mathcal{P}_a^*$ dont la cardinalité reste polynomiale en la taille de l'instance. L'ensemble $\mathcal{P}_a^*$ vérifie que pour toute solution $\mathcal{S}^* \in \mathcal{P}^*$ il existe une solution $\mathcal{S} \in \mathcal{P}_a^*$ qui approche $\mathcal{S}^*$ à un facteur constant.

Pour commencer la conception de cette approche, nous rappelons que dans ce cas de figure où les tâches sont unitaires, toute stratégie d'ordonnancement est définie par le nombre de tâches ω_j qui sont affectées au processeur j. Donc, calculer un ordonnancement dont le *makespan* est inférieur à une certaine borne est un problème simple. Par conséquent nous pouvons approcher les points du front Pareto-optimal moyennant une $<\bar{1},\rho_2>$-approx qui calcule un ordonnancement dont le $C_{max} \leq w$ et en garantissant que la fiabilité est à un facteur ρ_2 sous cette contrainte sur le *makespan*.

Formellement étant donnée une contrainte w, l'objectif est de construire un ordonnancement noté par $\hat{\mathcal{S}}$ qui vérifie $\hat{C}_{max} \leq w$, $\hat{\mathcal{C}} \in V$ et finalement la fiabilité de $\hat{\mathcal{S}}$ doit vérifier

$$\forall \mathcal{C} \in V \mid C_{max} \leq w,\ \mathcal{R}_l(\mathcal{C}) \leq \rho_2 \mathcal{R}_l(\hat{\mathcal{C}})$$

Avant d'entamer la conception de l'approche algorithmique qui permet d'atteindre cet objectif, nous proposons tout d'abord une règle de transformation d'ordonnancement qui en effectuant des transferts de charge de travail entre les processeurs optimise nécessairement la fiabilité du nouvel ordonnancement obtenu.

Théorème 5.4. *Considérons un $\mathcal{C} \in E$ et deux processeurs arbitraires j et j' qui vérifient $\nu_{j'} < \nu_j$, soit l'operation de transfert de δ_j unités de travail du processeur j' vers j qui engendre le vecteur $\hat{\mathcal{C}} = \{C_1, \cdots, C_j + \delta_j/\nu_j, \cdots, C_{j'} - \delta_j/\nu_{j'}, \cdots, C_m\}$ tel que $C_{j'} - \delta_j/\nu_{j'} \leq C_j + \delta_j/\nu_j$ alors nécessairement $\mathcal{C} \prec^w \hat{\mathcal{C}}$.*

Démonstration. Soit l (respectivement h) le rang du processeur j (respectivement j') dans le vecteur $\mathcal{C}$ trié par ordre croissant.
Supposons qu'après l'opération de transfert les nouveaux rangs de j et j' sont l^+ et h^-. La différence des sommes partielles entre $\mathcal{C}$ et $\hat{\mathcal{C}}$ dans le premier cas de figure où $h < l$ est donnée par.

1. $\forall k \mid 1 \leq k < h^-,\ \sum_{j=1}^{k} C_{(j)} - \sum_{j=1}^{k} \hat{C}_{(j)} = 0.$
2. $\forall k \mid h^- \leq k < h,\ \sum_{j=1}^{k} C_{(j)} - \sum_{j=1}^{k} \hat{C}_{(j)} = C_{(k)} - \hat{C}_{(h^-)} \geq 0$ (puisque $\hat{C}_{(k+1)} = C_{(k)}$).

3. $\forall k \mid h \leq k < l,\ \sum_{j=1}^{k} C_{(j)} - \sum_{j=1}^{k} \hat{C}_{(j)} = \delta_j/\nu_{j'}$.
4. $\forall k \mid l \leq k \leq l^+,\ \sum_{j=1}^{k} C_{(j)} - \sum_{j=1}^{k} \hat{C}_{(j)} = C_{(l)} + \delta_j/\nu_{j'} - \hat{C}(k) \geq 0$ puisque $C_{(l)} + \delta_j/\nu_{j'} \geq C_{(l^+)} \geq \hat{C}_{(k)}$.
5. $\forall k \mid {}^+l \leq k \leq m,\ \sum_{j=1}^{k} C_{(j)} - \sum_{j=1}^{k} \hat{C}_{(j)} = \delta_j(1/\nu_{j'} - 1/\nu_j) \geq 0$.

Dans le deuxième cas de figure où $l < h$ et $h < l^+$.

1. $\forall k \mid 1 \leq k < h^-,\ \sum_{j=1}^{k} C_{(j)} - \sum_{j=1}^{k} \hat{C}_{(j)} = 0$
2. $\forall k \mid h^- \leq k < l,\ \sum_{j=1}^{k} C_{(j)} - \sum_{j=1}^{k} \hat{C}_{(j)} = C_{(k)} - \hat{C}_{(h^-)} \geq 0$ (puisque $C_{(k)} = \hat{C}_{(k+1)}$).
3. $\forall k \mid l \leq k < h,\ \sum_{j=1}^{k} C_{(j)} - \sum_{j=1}^{k} \hat{C}_{(j)} = C_{(l)} - \hat{C}_{(h^-)} \geq 0$.
4. $\forall k \mid h \leq k < l^+,\ \sum_{j=1}^{k} C_{(j)} - \sum_{j=1}^{k} \hat{C}_{(j)} = C_{(l)} + \delta_j/\nu_{j'} - \hat{C}_{(k)} \geq 0$.
5. $\forall k \mid l^+ \leq k \leq m,\ \sum_{j=1}^{k} C_{(j)} - \sum_{j=1}^{k} \hat{C}_{(j)} = \delta_j(1/\nu_{j'} - 1/\nu_j) \geq 0$.

Dans le dernier cas de figure où $l < h$ et $l^+ < h$ l'analyse reste la même sauf pour les sous cas 3 et 4.

1. $\forall k \mid 1 \leq k < h^-,\ \sum_{j=1}^{k} C_{(j)} - \sum_{j=1}^{k} \hat{C}_{(j)} = 0$
2. $\forall k \mid h^- \leq k < l,\ \sum_{j=1}^{k} C_{(j)} - \sum_{j=1}^{k} \hat{C}_{(j)} = C_{(k)} - \hat{C}_{(h^-)} \geq 0$ (puisque $C_{(k)} = \hat{C}_{(k+1)}$).
3. $\forall k \mid l \leq k < l^+,\ \sum_{j=1}^{k} C_{(j)} - \sum_{j=1}^{k} \hat{C}_{(j)} = C_{(l)} - \hat{C}_{(h^-)} \geq 0$.
4. $\forall k \mid l^+ \leq k < h,\ \sum_{j=1}^{k} C_{(j)} - \sum_{j=1}^{k} \hat{C}_{(j)} = C_{(k)} + C_{(l)} - (\hat{C}_{(l^+)} + \hat{C}_{(h^-)}) \geq 0$ (puisque $C_{(k)} = \hat{C}_{(k+1)}$).
5. $\forall k \mid h \leq k \leq m,\ \sum_{j=1}^{k} C_{(j)} - \sum_{j=1}^{k} \hat{C}_{(j)} = \delta_j(1/\nu_{j'} - 1/\nu_j) \geq 0$.

Dans les trois cas de figures nous obtenons :

$$\sum_{j=1}^{m} C_{(j)} - \sum_{j=1}^{m} \hat{C}_{(j)} \geq 0.$$

Alors les deux vecteurs vérifient :

$$C \prec^w \hat{C}$$

□

Ce théorème indique que la fiabilité de tout ordonnancement obtenu à partir d'un autre ordonnancement avec des opérations de transfert qui vérifient les deux contraintes à savoir $\nu_{j'} < \nu_j$ et $C_{j'} - \delta_j/\nu_{j'} \leq C_j + \delta_j/\nu_j$ est nécessairement supérieure si la fonction objectif préserve la sur-majoration. En s'appuyant sur ce théorème et en s'inspirant de l'idée de l'Algorithme 3 proposé par Jeannot *et al.* [65], nous exhibons une $<\bar{1}, 1>$approximation pour approximer l'ensemble des solutions Pareto-optimales. Cette $<\bar{1}, 1>$approximation garantit pour un w donné, que l'ordonnancement produit $\hat{\mathcal{S}}$ est valide, le *makespan* de $\hat{\mathcal{S}}$ vérifie $\hat{C}_{max} \leq w$ et $\hat{\mathcal{S}}$ a une fiabilité optimale par rapport à tout autre ordonnancement valide dont le *makespan* vérifie $C_{max} \leq w$. Cette approche proposée repose sur un algorithme glouton qui affecte à chaque processeur par ordre décroissant de ν_j une quantité de travail proportionnelle à son taux de traitement. Ainsi ceci permet de remplir le plus possible les processeurs les plus rapides tout en respectant la contrainte w.

Algorithme 3 Allocation fiable pour des tâches unitaires

```
function <1̄, 1>-APPROX(w, n, Q)
    A ← 0                                   ▷ L'ensemble des tâches allouées
    for j ← 1 to m  do   ▷ Les processeurs sont triés par ordre décroissant selon ν_j
        ω[j] ← 0
        if A < n then
            ω[j] ← Min(n − A, ⌊w/ν_j⌋)
        end if
        A ← A + ω[j]
    end for
    if A==n then
        return  ω
    end if
    return Error          ▷ Aucun ordonnancement ne peut valider la contrainte w
end function
```

Théorème 5.5. *Soit $\hat{\mathcal{S}}$ l'ordonnancement produit par L'Algorithme 3, alors cet ordonnancement vérifie $\hat{C}_{max} \leq w$, $\hat{\mathcal{C}} \in V$ et $\forall \mathcal{C} \in V \mid C_{max} \leq w,\ \mathcal{R}_l(\mathcal{C}) \leq \mathcal{R}_l(\hat{\mathcal{C}})$*

Démonstration. Nous notons avant toute chose que par construction cet algorithme vérifie nécessairement que $\hat{C}_{max} \leq w$, $\sum_{j=1}^{m} \hat{C}_j \nu_j = n$ (toutes les tâches sont executées) et $\forall j,\ \hat{C}_j \in \mathbb{N}_+$ alors $\hat{\mathcal{C}} \in V$.
Considérons un ordonnancement arbitraire noté par $\mathcal{S} \in V$.
Supposons que le *makespan* du processeur j dans les deux ordonnancements $\mathcal{S}$ et $\hat{\mathcal{S}}$, vérifie la relation suivante, $C_j < \hat{C}_j$.
Ceci implique que le processeur j a reçu moins de travail dans l'ordonnancement $\mathcal{S}$.
Nous rappelons que par construction, tous les processeurs qui sont plus rapides que j sont remplis au maximum.
Donc cette quantité de travail de moins est allouée à un processeur j' qui est plus lent que j.
Donc en s'appuyant sur le Théorème 5.4, tout transfert de j' vers j tel que $\hat{C}_j = C_j + \delta_j/\nu_j = \lfloor w/\nu_j \rfloor$ optimise la fiabilité du nouvel ordonnancement en respectant la

contrainte w.
Nous notons finalement que cet algorithme est déterministe, alors effectuer une série de transferts de ce type en commençant par les processeurs les plus rapides, engendre forcément l'ordonnancement $\hat{\mathcal{S}}$. □

En utilisant cette $<\bar{1}, 1>$-approx, nous mettons en œuvre une approche d'approximation $(1+\epsilon, 1)$ des points du front Pareto-optimal en couplant l'Algorithme 3 avec la méthodologie proposée par Papadimitriou [82] que nous avons rappelée au début de ce chapitre. Ceci est obtenu en incrémentant le seuil w selon une suite géométrique entre la borne supérieure qui représente le *makespan* de l'ordonnancement le plus fiable donnée par $R_b = n/\nu_1$ et la borne inférieure $L_b = C^*_{max}$ qui représente le *makespan* optimal (nous notons que cette quantité est calculée en un temps polynomial [25] puisque les tâches sont unitaires et indépendantes).

Ainsi en s'appuyant sur le Théorème 5.1, nous garantissons que pour chaque point de l'ensemble des solutions Pareto-optimales $\mathcal{P}^*$ il existe nécessairement un point dans l'ensemble $\mathcal{P}^*_a$ qui est à un facteur $1+\epsilon$ du *makespan* optimal et dont la probabilité de succès est maximale.
Finalement, la complexité en temps de calcul de cette approche est linéaire en la taille de l'instance et donnée par $\mathcal{O}\left(m \log m + m \log_{1+\epsilon}(n/\nu_1)\right)$ avec ϵ constante positive qui sera fixée en fonction du choix de la précision demandée. Nous notons que cette complexité est obtenue moyennant un algorithme de tri rapide dont le coût en temps est en $\mathcal{O}(m \log m)$ qui sera exécuté une seule fois.

b Tâches indépendantes et arbitraires

Nous analysons dans cette partie le problème général $Q||(\mathcal{R}, C_{max})$ où les tâches ont des durées arbitraires. L'instance d'entrée dans ce cas de figure est un ensemble de n tâches $\mathcal{T} = \{p_1, \cdots, p_n\}$, un ensemble de m processeurs uniformes $\mathcal{Q} = \{\nu_1, \cdots, \nu_m\}$ et une fonction de distribution de probabilité $F(x)$ supposée de nature *DFR*. Cette problématique générale hérite du résultat du Théorème 5.3 où il est montré que la taille des solutions sur le front Pareto-optimal est exponentielle en la taille de l'instance comme l'indique le théorème suivant.

Théorème 5.6. *La cardinalité de l'ensemble des solutions Pareto-optimales est exponentielle en la taille de l'instance pour certaines instances du problème $Q||(C_{max}, \mathcal{R}_l)$.*

Démonstration. Considérons l'instance suivant :
- n tâches définies par $\forall i,\ 1 \leq i \leq n,\ p_i = 2^{i-1}$.
- 2 processeurs identiques avec $\forall j \leq 2,\ \nu_j = 1$.
- Une distribution de panne dont le taux de panne est monotone décroissant (ceci implique une stricte convexité du $\log \overline{F}(x)$).

Nous notons que dans ce cas nous obtenons 2^{n-1} ordonnancements avec des C_{max} différents qui varient dans $2^{n-1} \leq C_{max} \leq 2^n$.
L'ordonnancement $\mathcal{S}_i$ alloue $2^n - i$ unités de travail au processeur 1 et i unités de travail au deuxième processeur avec $0 \leq i \leq 2^{n-1}$.
Chaque $\mathcal{S}_i$ est une solution Pareto-optimale, en utilisant le même argument développé dans le Théorème 5.3. □

Nous tenons à souligner qu'à la différence de la preuve présentée dans [65] où les auteurs exhibent un résultat équivalent en considérant une instance particulière qui contient un processeur très rapide mais pas fiable et un autre fiable mais très lent. Ce théorème présenté ici indique, que même dans le cas où les processeurs sont identiques la cardinalité de l'ensemble des solutions Pareto-optimales est exponentielle en la taille de l'instance. Donc il est hors de question de calculer tous les points sur le front Pareto-optimal. Dans ce cas de figure, nous exhibons aussi une extension de l'approche qui a été proposée par Jeannot *et al.* [65] pour le cas particulier du processus de Poisson. Cette approche est mise en œuvre en utilisant la méthodologie d'approximation de Papadimitriou *et al.* [82]. Cependant, dans ce cas de figure où les tâches ont des tailles arbitraires, il n'existe aucun algorithme dont la complexité est polynomiale en la taille de l'instance et qui calcule un ordonnancement dont le $C_{max} \leq w$ sauf si $P = NP$ [43]. Par conséquent, nous proposons dans ce cas une $<\bar{2}, 1>$-approx gloutonne qui repose sur le principe du *list scheduling*. Le principe de cette approche est toujours d'allouer le plus possible de travail au processeur le plus rapide. Donc il faut monter que si cet

Algorithme 4 Allocation fiable pour des tâches arbitraires

```
function <2̄, 1>-APPROX(w, T, Q)
    j ← 1
    i ← 1
    Trier les tâches par ordre décroissant des p_i
    Trier les processeurs par ordre décroisant des ν_j
    ∀j ∈ Q, C_j ← 0
    while i ≤ n do
        if C_j ≤ w then
            ξ[i] ← j
            ζ[i] ← C_j
            C_j ← C_j + p_i/ν_j
            i ← i + 1
        else j ← j + 1
        end if
        if j > m then
            return Error   ▷ Aucun ordonnancement ne peut valider la contrainte w
        end if
    end while
    return ξ, ζ
end function
```

algorithme renvoie un ordonnancement alors cet ordonnancement vérifie :

- Le *makespan* vérifie $\hat{C}_{max} \leq 2w$.
- La fiabilité est optimale par rapport a toute autre ordonnancement arbitraire valide dont le *makespan* vérifie $C_{max} \leq w$

Sinon, s'il ne renvoie pas de résultat il faut s'assurer qu'aucun ordonnancement valide dont le *makespan* vérifie $C_{max} \leq w$ existe.

Théorème 5.7. *S'il existe un ordonnancement valide qui vérifie $C_{max} \leq w$ alors $\hat{\mathcal{S}}$ l'ordonnancement produit par L'Algorithme 4 vérifie $\hat{C}_{max} \leq 2w$, $\hat{\mathcal{C}} \in V$ et $\forall \mathcal{C} \in$*

$V \mid C_{max} \leq w,\ \mathcal{R}_l(\mathcal{C}) \leq \mathcal{R}_l(\hat{\mathcal{C}})$

Démonstration.
- Pour commencer nous montrons que s'il existe un ordonnancement valide qui vérifie $C_{max} \leq w$ alors cet algorithme respecte la contrainte $\hat{C}_{max} \leq 2w$.
  Supposons par contradiction que $\hat{C}_{max} > 2w$. Alors il existe une une tâche i' qui vérifie : "i' est ordonnancée sur un processeur noté par j' à l'instant w tel que $\hat{C}_{j'} > 2w$".
  Cette hypothèse implique que la quantité de travail dans i' vérifie $p_{i'}/\nu_{j'} > w$.
  Notons que cet algorithme alloue tout d'abord les plus grosses tâches.
  Alors dans tout ordonnancement qui vérifie $C_{max} \leq w$, les tâches dont le rang $i \in [1..i']$ (toujours par ordre décroissant de p_i) doivent être allouées aux processeurs dont le rang j vérifie $1 \leq j < j'$. Or le *makespan* de tous les processeurs dans cet intervalle vérifie $C_j > w$.
  Donc, il n'existe aucun ordonnancement valide qui vérifie dont le *makespan* vérifie $C_{max} \leq w$.
- L'algorithme proposé ne renvoie pas de solution si et seulement si lors de l'ordonnancement de la tâche $i' \leq n$ tous les processeurs vérifient $\hat{C}_j > w$, $1 \leq j \leq m$.
  Donc dans ce cas de figure il n'existe aucun ordonnancement valide qui vérifie la contrainte $C_{max} \leq w$.
- Nous notons que par construction, cet algorithme renvoie un ordonnancement où les $\hat{C}_j$ des $k-1 \leq m$ processeurs les plus rapides (toujours par ordre décroissant de ν_j) vérifient $\hat{C}_j > w$, un seul processeur dont le rang k vérifie $\hat{C}_k \leq w$ et finalement $\hat{C}_j = 0$ pour tous les processeurs dont le rang j vérifie $k < j \leq m$.
  Supposons qu'il existe un ordonnancement arbitraire $\mathcal{S}$ tel que $C_{max} \leq w$.
  Ainsi à partir de cet ordonnancement nous pouvons construire les configurations des $k-1$ processeurs plus rapides dans $\hat{\mathcal{S}}$ en effectuant des transferts de charge à partir de l'ensemble des $m-k$ processeurs les moins rapides. En s'appuyant sur le Théorème 5.4 cela ne peut qu'augmenter la fiabilité.
  Ensuite pour le processeur k soit $C_k < \hat{C}_k$, alors dans ce cas aussi un transfert est fait depuis un processeur moins rapide que lui.
  Sinon si $C_k > \hat{C}_k$ alors la quantité de travail de moins est allouée à un processeur plus rapide que k puisque par construction $\hat{C}_j = 0$ pour tout $j > k$.

□

Dans ce cas aussi nous couplons cette $<\bar{2}, 1>$-approx avec la méthodologie de Papadimitriou [82] pour en construire une $<2+\epsilon, 1>$-approx du front Pareto-optimal. Cette fois la borne inférieure de w est donnée par $\max(\sum_{i=1}^{n} p_i / \sum_{j=1}^{m} \nu_j, p_{max}/\nu_1)$ tandis que la borne supérieure reste la même $\sum_{i=1}^{n} p_i/\nu_1$. Par conséquent la cardinalité de l'ensemble d'approximation du front Pareto-optimal reste bornée par $\mathcal{O}\left(\log_{1+\epsilon}(\sum_{i=1}^{n} p_i/\nu_1)\right)$. Par contre la complexité de l'approche change légèrement puisqu'il faut aussi gérer le tri des tâches. La complexité globale de cette méthodologie est alors $\mathcal{O}\left(m \log m + n(\log n + \log_{1+\epsilon}(\sum_{i=1}^{n} p_i/\nu_1))\right)$.

5.4.2 Optimisation de $P||(C_{max}, \overline{\mathcal{R}}_l)$ avec $\lambda(t)$ *IFR*

Nous analysons dans cette partie la problématique dans le cas de figure où le taux de panne est croissant *IFR*. La classe des distributions *IFR* contient par exemple la distribution uniforme, la distribution exponentielle ou encore la distribution Weibull avec un paramètre de forme supérieur à 1 ($\beta_w > 1$). La première question qui vient à l'esprit est : "est t il possible de résoudre la problématique $P||\mathcal{R}_l$ en calculant un ordonnancement qui maximise la fiabilité de l'application dans un temps raisonnable comme c'est le cas pour les distributions *DFR* ?".

Théorème 5.8. *Le problème $P||\overline{\mathcal{R}}_l$ est dans la classe des problèmes $\mathcal{NP}$-complet si le taux de panne est croissant.*

Démonstration. Nous rappelons qu'optimiser la fiabilité revient à minimiser la fonction $\overline{\mathcal{R}}_l(\mathcal{S}) = \sum_{j=1}^{m} g(C_j)$ avec $g(x) = -\log \overline{F}(x)$.
Le taux de panne est croissant alors $g(x) = -\log \overline{F}(x)$ est une fonction croissante et convexe.
Dans ce cas nous obtenons la même problématique étudiée dans [2] par exemple en considérant la distribution de Weibull avec $\beta_w = 2$ (d'où $g(x) = x^2$). □

Donc contrairement au cas *DFR*, dans ce cas de figure l'optimisation de la fiabilité de l'application est un problème $\mathcal{NP}$-complet même quand les processeurs sont identiques. Néanmoins, nous tenons à signaler que ceci n'est pas forcément une mauvaise nouvelle car en s'appuyant sur la Proposition 6.1 la fonction objectif $\sum_{j=1}^{m} -\log \overline{F}(C_j)$ est une fonction Schur-convexe et croissante quand le taux de panne est décroissant. Par conséquent elle est minimale quand tous les C_j sont égaux (Théorème 6.1). Donc dans ce cas les deux objectifs ne sont plus antagoniste puisque si les C_j sont égaux le *makespan* est nécessairement optimal.

a Taches unitaires et indépendantes

Pour commencer nous considérons dans cette section le cas de figure le plus élémentaire où les tâches sont supposées identiques $P|p_i = 1|(C_{max}, \overline{\mathcal{R}}_l)$. Nous rappelons que tout ordonnancement compact est décrit par la quantité de travail affectée à chaque processeur. En s'appuyant sur la propriété de Schur-convexité de la fonction objectif, tout ordonnancement qui équilibre le vecteur des C_j est un ordonnancement optimal pour les deux objectifs. Supposons que l'application contient n tâches, donc s'écrit $n = \alpha m + r$. Soit $\hat{\mathcal{S}}$ l'ordonnancement produit par l'algorithme suivant. Allouer $\alpha + 1$ tâches à r processeurs et α tâches à chacun des $m - r$ restant. Alors cet ordonnancement est optimal pour les deux objectifs.

Théorème 5.9. *Soit $\hat{\mathcal{C}}$ le vecteur résultant de l'ordonnancement produit par l'Algorithme 5, $\hat{\mathcal{C}}$ vérifie nécessairement*

$$\forall \mathcal{C} \in V,\ \hat{\mathcal{C}} \prec \mathcal{C}$$

Démonstration. Soit $\mathcal{C}$ un vecteur résultant d'un ordonnancement arbitraire et valide noté par $\mathcal{S}$.
Supposons par l'absurde que la relation $\hat{\mathcal{C}} \prec \mathcal{C}$ est fausse.

Algorithme 5 Allocation optimale pour des tâches unitaires

```
procedure ALLOCATION(n, Q)
    for j = 1 to n%m do
        w_j = ⌈n/m⌉
    end for
    for j = n%m + 1 to m do
        w_j = ⌊n/m⌋
    end for
end procedure
```

– Premier cas :

$$\exists k \leq r \mid \sum_{j=1}^{k} C_{[j]} < \sum_{j=1}^{k} \hat{\mathcal{C}}_{[j]}. \tag{5.9}$$

Alors $\sum_{j=1}^{k} \mathcal{C}_{[j]} < k(\alpha+1) \Rightarrow C_{[k]} \leq \alpha$ (puisqu'il y a forcément un processeur moins chargé que $\alpha+1$).
Donc,$\forall j \mid k < j \leq m,\ C_{[k]} \leq \alpha \Rightarrow C_{[j]} \leq \alpha$.
Dans ce cas, la somme de la quantité de travail sur les processeurs restants est :

$$\sum_{j=k+1}^{m} C_{[j]} \leq (m-k)\alpha. \tag{5.10}$$

Donc les inéquations 5.9,5.10 impliquent que :

$$\sum_{j=1}^{m} C_{(j)} < r - k + \alpha m.$$

Ceci est absurde car il y a $r-k-1$ tâches non allouées dans $\mathcal{S}$.

– Deuxième cas :

$$\exists k, |r < k \leq m,\ \sum_{j=1}^{k} C_{[j]} < \sum_{j=1}^{k} \hat{\mathcal{C}}_{[j]} \tag{5.11}$$

D'où par le même argument $C_{[k]} < \alpha$ (puisqu'il y a forcément un processeur moins chargé que α sinon on a l'égalité).
D'où la somme de la quantité de travail sur les processeurs restant est :

$$\sum_{j=k+1}^{m} C_{[j]} \leq (m-k)(\alpha-1) \tag{5.12}$$

Ceci est absurde car les inéquations 5.11,5.12 impliquent qu'il y a $m-k-1$ tâches non allouées dans $\mathcal{S}$.

□

Donc ce théorème implique que le vecteur $\hat{\mathcal{C}}$ engendré par l'ordonnancement $\hat{\mathcal{S}}$ minimise la fonction objectif $\overline{\mathcal{R}}_l$, puisqu'il est majoré par tout autre vecteur d'un ordonnancement valide. En conséquent, cet ordonnancement est un optimal global pour les deux objectifs puisque même la fonction objectif C_{max} est une fonction Schur-convexe en s'appuyant sur la Définition 6.4.

Finalement nous tenons à signaler que ce résultat n'est plus valable avec le modèle de plateforme à processeurs uniformes Q. Puisque nous ne retrouvons plus avec des relations de majoration entre les vecteurs où la somme $\sum_{j=1}^{m} C_j$ est constante quel que soit $\mathcal{S} \in V$. Mais nous nous retrouvons plutôt avec des relations de type faible majoration. D'où ceci implique que le vecteur des temps de complétion équilibrés n'est plus une solution optimal pour $\overline{\mathcal{R}}_l$. Par exemple, considérons l'instance suivante de la problématique $Q|p_i = 1|(C_{max}, \overline{\mathcal{R}}_l)$:

- $n = 11$, $\nu_1 = 10, \nu_2 = 1$
- Distribution de Weibull avec $\alpha_w = 1$ et $\beta_w = 2$ donc $-\log \overline{F}(x) = x^2$.

Considérons par exemple les deux ordonnancements suivants.

1. $\mathcal{S}_1 \equiv w_1 = 11,\ w_2 = 0$ donc $C_{max}^{\mathcal{S}_1} = 1.1$, $\overline{\mathcal{R}}_l(\mathcal{S}_1) = 1,21$.
2. $\mathcal{S}_2 \equiv w_1 = 10,\ w_2 = 1$ donc $C_{max}^{\mathcal{S}_2} = 1$, $\overline{\mathcal{R}}_l(\mathcal{S}_2) = 2$.

Ainsi nous nous retrouvons dans un cas où il n'existe aucun ordonnancement optimal global qui optimise les deux objectifs en même temps.

b Tâches indépendantes et arbitraires

Nous considérons dans cette partie la généralisation du problème précédent en considérant des tâches avec des durées arbitraires. Ainsi, nous rappelons que dans ce cas de figure les deux problèmes mono-objectif qui en découlent c'est-à-dire la minimisation du C_{max} seulement ou la minimisation du manque de fiabilité $\overline{\mathcal{R}}_l$ sont des problèmes $\mathcal{NP}$-complet au sens fort. Dans ce cas, deux approches restent envisageables pour résoudre cette problématique multi-objectifs.

La première approche que nous pouvons envisager, consiste en la conception d'un schéma d'approximation polynomial. Dans ce cas, la première idée qui vient à l'esprit est la réutilisation du résultat proposé dans [2] où les auteurs proposent un schéma d'approximation polynomial pour $P||\sum_{j=1}^{m} g(C_j)$ et $P||C_{max}$ à la fois. Par contre pour réutiliser ce résultat il faut que la fonction $g(x)$ vérifie les deux conditions nécessaires présentées auparavant dans la partie état de l'art. Nous notons que dans notre cas où la distribution de panne est de nature *IFR* la fonction $g(x) = -\log \overline{F}(x)$ vérifie la condition de convexité et la croissance monotone. Par contre nous n'avons aucune preuve formelle ni de contre exemple qui indique si la fonction $-\log \overline{F}(x)$ vérifie ou non la deuxième condition donnée auparavant à la section état de l'art dans l'Expression 5.5. Cependant nous notons que certaines distributions de panne très intéressantes vérifient cette condition comme par exemple la distribution exponentielle ou encore la distribution de Weibull dans le cas où $\beta_w > 1$ puisque dans ces deux cas $g(x)$ est de la forme αx^{β} avec $\alpha \in \mathbb{R}_+^*$ et $\beta \geq 1$.

La deuxième approche qui est envisageable et très intéressante consiste en l'utilisation de l'ordonnancement glouton *LPT* (à faible coût mais garantie). Nous notons par exemple si la distribution de panne est une distribution exponentielle, l'ordonnancement *LPT* est optimal par rapport à l'objectif de fiabilité et il est à un facteur 4/3 du deuxième objectif C_{max}. Dans le cas aussi où la distribution est de type Weibull en s'appuyant sur l'analyse proposée dans [28]. L'ordonnancement *LPT* est à un facteur 25/24 si le paramètre de forme de la loi $\beta_w = 2$ et il est à un facteur borné qui dépend de β_w dans le cas général. Toutefois, pour le cas d'une distribution arbitraire de nature *IFR*, nous n'avons aucune preuve ou contre exemple qui indiquent que l'ordonnancement *list* n'est pas à un facteur

constant d'un ordonnancement qui maximise la fiabilité. Ainsi nous conjecturons que si la distribution de panne est de nature *IFR*, l'ordonnancement *list* a un ratio de performance borné par un facteur constant par rapport à tout ordonnancement qui maximise la fiabilité dans le cadre de la problématique $P||(C_{max}, \overline{\mathcal{R}}_l)$.

5.5 Conclusions

Dans ce chapitre nous avons exhibé une analyse fine et en profondeur du problème multi-objectifs $Q||(C_{max}, \mathcal{R})$. À travers cette analyse, nous avons illustré le rôle déterminant que joue la fonction du taux de panne sur la complexité et la nature du problème. Cette étude a permis de montrer pour la première fois à notre connaissance que la minimisation du *makespan* et la maximisation de la fiabilité de l'application ne sont pas toujours antagonistes. Nous avons démontré aussi que selon le sens de croissance du taux de panne le problème de maximisation de fiabilité d'un ordonnancement est trivial dans le cas où le taux de panne est décroissant et il devient $\mathcal{NP}$-complet au sens fort dans le cas où le taux de panne est croissant.

À la lumière de cette analyse, nous avons proposé plusieurs stratégies d'ordonnancement qui sont essentiellement des extensions d'un ensemble de travaux existants qui ont été limité au cas du taux de panne constant ou qui ont été proposé pour résoudre d'autre problèmes. Principalement, cette analyse a permis premièrement d'élargir la portée des stratégies d'ordonnancement proposées par Jeannot *et al.* [65] pour prendre en considération diverses distributions de panne autres que la distribution exponentielle. Deuxièmement, elle a permis d'établir des liens entre le problème de départ et plusieurs autre stratégies d'ordonnancement proposées pour des problème différents. Essentiellement, en partant de l'analyse de Chandra *et al.* [28] dans le cadre du problème $P||\sum_{j=1}^{m}(C_j)^{\beta}$, nous avons mis en évidence que pour toute distribution Weibull de nature *IFR* ($\beta_w > 1$), tout ordonnancement de type *list scheduling* produit un ordonnancement dont le ratio de performance est garanti pour les deux objectifs du problème $P||(C_{max}, \mathcal{R})$.

Cette étude ouvre plusieurs perspectives et offre une nouvelle démarche d'analyse très puissante. Tout d'abord, nous conjecturons que l'ordonnancement LPT est a un facteur constant pour les deux objectifs si le taux de panne est croissant. Ensuite, il serait sans doute très intéressant d'analyser le ratio de performance de l'ordonnancement LPT quand la plateforme est composée d'un ensemble de processeurs uniformes. Finalement, nous tenons à souligner qu'il est aussi primordial de se focaliser sur l'étude des approches d'ordonnancement dans la cas où la duplication active est mise en place pour tolérer l'arrivée des pannes en s'appuyant sur les propriétés mathématiques de la fonction taux de panne.

Conclusions et travaux futurs 6

Sommaire

6.1 Bilan 107
6.2 Perspectives 108

6.1 Bilan

Dans la première partie de cette thèse nous nous sommes focalisés sur l'analyse du problème d'optimisation du compromis soulevé lors de la mise en place du mécanisme de sauvegarde et reprise dans les environnements de calcul parallèle et distribué. Tout d'abord, cette analyse a mis en évidence que l'exécution des applications sans mécanisme de tolérance aux pannes sur les plateformes d'exécution actuelles et futures, subira incontestablement d'énormes dégradations. Nous avons ensuite illustré que le mécanisme de tolérance aux pannes qui repose sur le principe de sauvegarde et reprise est une solution efficace pour assurer la complétion des applications dans ces environnements hautement perturbés par les pannes. Toutefois, la mise en place d'un mécanisme de tolérance aux pannes est une tâche délicate qui soulève de nombreux défis scientifiques. Parmi ces défis, nous nous sommes intéressés à la première question soulevée lors de la mise en place du mécanisme de sauvegarde et reprise qui est : "où doit-on ordonnancer les points de sauvegarde ?". Étant donnée son importance, ce problème a reçu beaucoup d'attention et elle a fait l'objet de nombreuses études. Toutefois, les travaux existants sont dépassés par l'évolution que subissent les applications et les plateformes de calcul actuelles. En effet, la quasi totalité des travaux existants considèrent une distribution exponentielle pour modéliser les inter-arrivées des pannes, alors que les pannes sur les plateformes actuelles suivent d'autres distributions telles que la loi Weibull. Ces travaux supposent que tous les points de sauvegarde ont le même surcoût, or ceci n'est plus vrai avec par exemple la technique de sauvegarde incrémentale. De plus, la quasi totalité des solutions fournies par ces travaux sont des approximations de premier ordre. Dans la première partie de cette thèse, nous avons essentiellement mené deux études différentes pour répondre à ces nouveaux besoins.

Dans la première étude, le problème d'ordonnancement des points de sauvegarde a été posé en utilisant une modélisation continue du temps. Dans cette modélisation, nous avons encapsulé le compromis à gérer dans la fonction objectif qui exprime l'espérance du temps de complétion en fonction de l'ordonnancement des points des sauvegarde. Cette première étude a permis de démontrer formellement que la politique optimale d'ordonnancement des points de sauvegarde est nécessairement périodique si le taux de panne et les surcoûts des points de sauvegarde sont constants.

Ensuite, nous avons mis en place une approche de résolution numérique du problème pour le cas où le taux de panne et les surcoûts des points de sauvegarde sont arbitraires. Finalement, nous avons mené une campagne de simulations pour comparer la solution exacte proposée avec la solution d'approximation d'orde supérieur proposée par Daly. Ces simulations ont mises en évidence que les deux solutions sont équivalentes si nous considérons le temps de complétion comme étant le seul critère de performance. Toutefois, en analysant plus finement les résultats, nous avons montré qu'il est possible de réduire le surcoût accumulé des points de sauvegarde tout en gardant le même temps de complétion. Étant donné le surcoût qu'un point de sauvegarde peut engendrer, ce résultat implique qu'il faut utiliser la solution exacte plutôt que la solution qui a une forme analytique simple.

Dans la deuxième étude nous avons opté pour une modélisation discrète du problème. Dans celle-ci, nous nous sommes focalisées sur l'optimisation du temps total perdu. Essentiellement, cette étude a permis d'établir deux contributions principales. Premièrement, nous avons proposé un schéma de programmation dynamique qui résout le problème général où la fonction de distribution de panne et les surcoûts des points de sauvegarde sont arbitraires. Deuxièmement étant donnée la complexité pseudo-polynomiale de l'algorithme proposé, nous avons mené une analyse de complexité qui montre l'appartenance du problème général à la classe $\mathcal{NP}$-complet et l'appartenance à la classe P d'une autre variante problème où les surcoûts des points de sauvegarde sont constants.

Dans la deuxième partie de cette thèse, nous nous sommes intéressés au second problème de compromis soulevé lors de la phase d'ordonnancement d'une application sur l'ensemble des ressources disponibles. Dans cette partie, nous avons présenté une analyse plus fine du problème d'ordonnancement multi-objectifs $Q||(C_{max}, \mathcal{R})$. Dans cette analyse, nous avons illustré le rôle important que joue la fonction du taux de panne. Principalement, cette analyse a permis de démontrer que la minimisation du *makespan* et la maximisation de la fiabilité de l'application ne sont pas toujours des objectifs antagonistes. Deuxièmement, en utilisant les propriétés mathématiques de la fonction objectif comme par exemple la Schur-convexité ou la Schur-concavité qui sont engendrées par la fonction taux de panne. Nous avons étendu le champ d'application de plusieurs travaux existants en particulier pour la résolution du problème bi-objectifs $Q||(C_{max}, \mathcal{R})$.

6.2 Perspectives

À l'issue de cette thèse plusieurs questions et problématiques restent ouvertes. Concernant le problème d'ordonnancement des points de sauvegarde, les questions sont liées essentiellement aux trois éléments suivants. Le premier concerne le modèle de panne. En effet, la modélisation adoptée pour décrire formellement l'arrivée des pannes repose sur l'hypothèse d'indépendance entre les occurrences de panne or les études de statistiques récentes révèlent l'existence de corrélation entre les pannes. Le deuxième point concerne l'impact du nombre de processeurs utilisés dans l'exécution sur le modèle de performance. En effet, tout au long de cette thèse nous avons négligé cet impact d sur le surcoût des points de sauvegarde ou sur le taux de panne global de la plateforme. Or il est évident que le nombre de processeurs utilisés a un impact direct sur le volume

de données à sauvegarder ou sur le taux de panne global auquel l'application doit s'attendre. Le troisième élément est le modèle d'application qui est limité pour l'instant au modèle de chaine de tâches ou au modèle de tâches indépendantes. Il conviendrait d'étendre l'étude à des graphes de tâches quelconques.

La deuxième problématique étudiée dans cette thèse a aussi donné naissance à plusieurs nouveaux problèmes importants. Le premier concerne la détermination du ratio de performance de l'ordonnancement *list* dans le cas où le taux de panne est croissant. Il serait aussi intéressant d'exploiter les diverses propriétés mathématiques engendrées par la fonction taux de panne dans le cadre des problèmes d'ordonnancement avec duplication.

Annexe

Nous tenons à signaler que la majorité des théorèmes d'analyse fonctionnelle et majoration sont extraits de l'ouvrage de Marshall et Olkin [77] tandis que tous les théorèmes concernant les propriétés mathématique de la distribution de panne sont extrait de l'ouvrage de Barlow *et al.* [8].

6.3 Schur-convexe

Pour commencer nous rappelons la définition de convexité (concavité) d'une fonction.

Définition 6.1. une fonction $g : \mathbb{R} \to \mathbb{R}$ est convexe (respectivement concave) si elle vérifie :

$$g(x+\delta) + g(y-\delta) \underset{(\geq)}{\leq} g(x) + g(y),\ \forall x \leq y \text{ avec } 0 \leq \delta \leq y - x. \tag{6.1}$$

Nous décrivons maintenant la notion de majoration entre les vecteurs qui est utilisée pour déterminer la similarité ou l'entropie entre les éléments des deux vecteurs. Considérons les deux vecteurs $x, y \in \mathbb{R}_+^n$, nous notons par $x_{[i]}$ (respectivement $x_{(i)}$) l'élément du $i^{éme}$ rang du vecteur x selon un ordre décroissant (respectivement croissant).

Définition 6.2. $\forall x, y \in \mathbb{R}^n$ y majore x (noté par $x \prec y$) si la condition suivante est vérifiée :

$$x \prec y \text{ si } \begin{cases} \sum_{i=1}^{k} x_{[i]} \leq \sum_{i=1}^{k} y_{[i]},\ k \in \{1, 2 \cdots n-1\}, \\ \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i. \end{cases} \tag{6.2}$$

Nous tenons à signaler que cette relation d'ordre partiel est définie pour les vecteurs qui ont la même somme $\sum_i^n x_i = s$ des éléments. Ainsi, nous pouvons intuitivement montrer que le vecteur uniforme $u_s = \{\frac{s}{n}, \frac{s}{n}, \cdots, \frac{s}{n}\}$ est majoré par tout autre vecteur $x \in \mathbb{R}_+^n$ qui vérifie $\sum_i^n x_i = s$. De la même façon nous introduisons la notion de faible majoration (*i.e. weak majorization*) notée par $\prec_w, \prec^w$ dans le cas où les deux vecteurs n'ont pas la même somme.

Définition 6.3. Pour tout $x, y \in \mathbb{R}^n$,

$$x \prec_w y \text{ si } \sum_{i=1}^{k} x_{[i]} \leq \sum_{i=1}^{k} y_{[i]},\ k = 1, 2, \cdots n, \tag{6.3}$$

et

$$x \prec^w y \text{ si } \sum_{i=1}^{k} x_{(i)} \geq \sum_{i=1}^{k} y_{(i)},\ k = 1, 2, \cdots n, \tag{6.4}$$

Donc dans les deux cas x est faiblement majoré par y, par contre dans le premier cas x est sous-majoré par y (*submajorized*) et tandis que dans le deuxième cas x est sur-majoré par y (*supermajorized*).

Nous caractérisons maintenant le concept de Schur-convexité et Schur-concavité des fonctions. Toute fonction $G : \mathbb{R}^n \to \mathbb{R}$ qui préserve l'ordre partiel défini dans la Définition 6.2 est dite Schur-convexe.

Définition 6.4. Soit $I \subset \mathbb{R}^n$ la fonction $G : I \to \mathbb{R}$ est Schur-convexe si et seulement si :

$$\forall x, y \in I,\ x \prec y \Rightarrow G(x) \leq G(y).$$

Nous tenons à noter que $g(x)$ est Schur-concave si et seulement si $-g(x)$ est Schur-convexe. Donc une fonction est Schur-convexe transforme l'ordre entre les vecteurs vers un ordre de type scalaire. Par exemple la fonction $\max(x) = x_{[1]}$ est une fonction Schur-convexe puisque si $x \prec y$ alors par définition $x_{[1]} \leq y_{[1]}$. En se basant sur la Définition 6.4 plusieurs propositions sont données pour caractériser les fonctions Schur-convexes. Les résultats présentés dans ce travail reposent principalement sur cette proposition. Le premier élément est la proposition [1] suivante.

Proposition 6.1. *Soit $I \subset \mathbb{R}$ si la fonction $g : I \to \mathbb{R}$ est convexe et croissante (respectivement décroisante) sur I alors $\forall x \in I^n$ la fonction $G : I^n \to \mathbb{R}$ définie par*

$$G(x) = \sum_{i=1}^{n} g(x_i),$$

est une fonction Schur-convexe croissante (respectivement décroissante).

Dans le cas où les vecteurs sont liés par une relation de majoration faible, la transformation de la relation d'ordre est définie par le théorème [2] suivant.

Théorème 6.1. *Soient $I \subset \mathbb{R}^n$ et $x, y \in I$ toute fonction $G : I \to \mathbb{R}$ vérifie,*

$$x \prec_w y \Rightarrow G(x) \leq G(y), \tag{6.5}$$

si et seulement si G est croissante et Schur-convexe. De la même façon G vérifie

$$x \prec^w y \Rightarrow G(x) \leq G(y), \tag{6.6}$$

si et seulement si G est décroisante et Schur-convexe

6.4 Distribution de probabilité *IFR/DFR*

Nous exhibons dans la suite de cette partie les propriétés mathématiques des distributions de probabilité qui vont être utilisées pour appliquer toutes ces notions et théorèmes de majoration. Nous tenons à signaler que la majorité de ces définitions sont introduites dans le Chapitre 2 de l'ouvrage de Balrow *et al.* [8]. Pour commencer nous rappelons que les fonctions de distribution de probabilité utilisées pour modéliser les durées aléatoires de disponibilité ou indisponibilité vérifient :

Définition 6.5. $F : \mathbb{R}_+ \to [0, 1]$, $\forall x, y \in \mathbb{R}_+,\ x < y \Rightarrow F(x) \leq F(y)$ (F est croissante).

Deuxièmement nous supposons que la fonction de densité notée par f existe. Ceci implique que toutes les distributions considérées sont aussi caractérisées par une fonction appelée taux de panne, notée par $\lambda(t) = f(t)/\overline{F}(t)$. Ainsi une distribution de panne est dite *IFR* (respectivement *DFR*) si la fonction $\lambda(t)$ est croissante (respectivement

1. (3.C.1)[77]
2. (3.A.8)[77]

décroissante) en t. Nous rappelons que toutes ces fonctions (F, f, λ) sont liées par l'identité suivante :

$$\overline{F}(x) = 1 - F(x) = 1 - \int_0^x f(t)dt = \exp^{-\int_0^x \lambda(t)dt}. \tag{6.7}$$

En s'appuyant sur cette identité le théorème[3] suivant est vérifié :

Théorème 6.2. *F est une distribution* IFR *(*DFR*) alors la fonction* : $t \to \log \overline{F}(t)$ *est concave (convexe) pour* $t \in \mathbb{R}_+$.

3. Théorème 4.1 page 25 [8]

Notations et terminologies

Nous présentons ici quelques notations et abréviations que nous utilisons tout au long de ce document de thèse. Certaines abréviations anglaises d'usage courant même en français ont été conservées.

Abréviations :

- $v.a.r$: variable aléatoire à valeur réelle.
- $v.a.d$: variable aléatoire à valeur discrete.
- *IFR* : *Increasing Failure Rate* (taux de panne croissant)
- *DFR* : *Decreasing Failure Rate* (taux de panne décroissant)
- $i.i.d$: independent and identically distributed (indépendantes et identiquement distribuées).
- MDP : *Markovian Decision Process* [83]
- $\mathcal{PTAS}$: *Polynomial Time Approximation Scheme*

Notations et terminologies :

- ν : taux de calcul où nombre d'opérations par seconde.
- U_i : le $i^{\text{ème}}$ $v.a.r$ intervalle du temps inter pannes.
- D_i : le $i^{\text{ème}}$ intervalle du temps inter réparation.
- λ : utilisé pour décrire le taux de pannes.
- $\lambda(t)$: taux de pannes en fonction du temps t.
- $\gamma(t)$: taux d'indisponibilité en fonction du temps t.
- $\arg\max(\arg\min)$: une fonction qui renvoie l'argument de l'élément maximal (minimal).
- w : quantité de travail initiale.
- L : quantité de travail perdu.
- W : temps total perdu.
- σ : surcoût accumulé des points de sauvegade.
- ω : temps de completion de ω unités de travail.
- τ : instant absolut de sauvegarde.
- ρ : periode de sauvegarde.
- π : stratégie d'ordonnancement des points de sauvegarde.
- $\{p_i\}$: vecteur de quantité de travail dans chaque tâche.
- $\{c_i\}$ vecteur des surcoûts des points de sauvegarde pour chaque tâche.
- *makespan* : date de complétion de l'application.
- $\mathcal{S}$: un ordonnancement.
- V : l'ensemble des ordonnancements valides
- $\mathcal{T}$: l'ensemble des tâches.
- $\mathcal{Q}$: l'ensemble des processeurs.
- $\mathcal{R}(\overline{\mathcal{R}})$ la probabilité de succés de l'application (la probabilité de panne de l'application).

Bibliographie

[1] S.L. Albin. On poisson approximations for superposition arrival processes in queues. *Management Science*, pages 126–137, 1982. ➪ *cité page 12*

[2] N. Alon, Y. Azar, G.J. Woeginger, and T. Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1(1) :55–66, 1998. ➪ *3 citations pages 84, 102, and 104*

[3] D. Anderson. Boinc : A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, Pittsburgh, USA, 2004. ➪ *2 citations pages 64 and 65*

[4] D. Anderson and G. Fedak. The computational and storage potential of volunteer computing. In *Proceedings of the IEEE/ACM International Symposium on Cluster Computing and Grid*, pages 73–80, 2006. ➪ *cité page 64*

[5] A. Avidor, Y. Azar, and J. Sgall. Ancient and new algorithms for load balancing in the l p norm. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 426–435, 1998. ➪ *cité page 83*

[6] A. Avizienis, J.C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1) :11–33, 2004. ➪ *cité page 8*

[7] R.E. Barlow, L.C. Hunter, and F. Proschan. Optimum checking procedures. *Journal of the society for industrial and applied mathematics*, 11(4) :1078–1095, 1963. ➪ *3 citations pages 39, 40, and 41*

[8] R.E. Barlow, F. Proschan, and L.C. Hunter. *Mathematical theory of reliability*. SIAM, 1996. ➪ *5 citations pages 4, 11, 112, 113, and 114*

[9] A.A. Bertossi, L.V. Mancini, and F. Rossini. Fault-tolerant rate-monotonic first-fit scheduling in hard-real-time systems. *IEEE Transactions on Parallel and Distributed Systems*, 10(9) :934 –945, 1999. ➪ *cité page 14*

[10] X. Besseron, M.S. Bouguerra, T. Gautier, E. Saule, and D. Trystram. *Fault tolerance and availability awarness in computational grids*, chapter 5. Numerical Analysis and Scientific Computing. Chapman and Hall/CRC Press, 2009. ISBN : 978-1439803677. ➪ *3 citations pages 3, 16, and 125*

[11] X. Besseron and T. Gautier. Optimised recovery with a coordinated checkpoint/-rollback protocol for domain decomposition applications. In *Proceedings of MCO*, pages 497–506, 2008. ➪ *3 citations pages 9, 17, and 18*

[12] BOINC Papers. http://boinc.berkeley.edu/trac/wiki/BoincPapers. ➪ *cité page 64*

[13] Catalog of boinc projects. http://www.boinc-wiki.info/Catalog_of_BOINC_Powered_Projects. ➪ *4 citations pages 2, 64, 65, and 70*

[14] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, and F. Vivien. Checkpointing strategies for parallel jobs. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 33 :1–33 :11, New York, NY, USA, 2011. ACM. ➪ *2 citations pages 40 and 63*

[15] M.S. Bouguerra and T. Gautier. Un modèle flexible pour la sauvegarde/reprise dans les systèmes parallèles. In *Proceedings of Renpar'19*, Toulouse, France, sept 2009. ➪ *2 citations pages 3 and 35*

[16] M.S. Bouguerra, T. Gautier, D. Trystram, and J.M. Vincent. A new flexible checkpoint/restart model. Technical report, RR-6751, INRIA, 2008. ➪ *cité page 35*

[17] M.S. Bouguerra, T. Gautier, D. Trystram, and J.M. Vincent. A Flexible Checkpoint/Restart Model in Distributed Systems. *Parallel Processing and Applied Mathematics*, pages 206–215, 2009. ➪ *3 citations pages 3, 34, and 40*

[18] M.S. Bouguerra, D. Kondo, and D. Trystram. On the scheduling of checkpoints in desktop grids. *Proceedings of the IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 305–313, 2011. ➪ *cité page 4*

[19] M.S. Bouguerra, D. Trystram, and F. Wagner. Complexity analysis of checkpoint scheduling with variable costs. *IEEE Transactions on Computers*, To appear, 2012. ➪ *cité page 4*

[20] A. Bouteiller, T. Hérault, G. Krawezik, P. Lemarinier, and F. Cappello. MPICH-V project : A multiprotocol automatic fault tolerant MPI. *The International Journal Of High Performance Computing Applications*, 20 :319–333, 2006. ➪ *cité page 9*

[21] F. Cappello. Fault tolerance in petascale/exascale systems : Current knowledge, challenges and research opportunities. *International Journal of High Performance Computing Applications*, 23(3) :212–226, 2009. ➪ *4 citations pages 2, 3, 7, and 13*

[22] F. Cappello, H. Casanova, and Y. Robert. Checkpointing vs. migration for post-petascale supercomputers. In *International Conference on Parallel Processing*, pages 168 –177, 2010. ➪ *cité page 2*

[23] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, and M. Snir. Toward exascale resilience. *International Journal of High Performance Computing Applications*, 23(4) :374, 2009. ➪ *3 citations pages 1, 7, and 8*

[24] H. Casanova, A. Legrand, and M. Quinson. SimGrid : a Generic Framework for Large-Scale Distributed Experiments. In *10th IEEE International Conference on Computer Modeling and Simulation*, March 2008. ➪ *2 citations pages 67 and 68*

[25] H. Casanova, A. Legrand, and Y. Robert. *Parallel Algorithms*. Chapman & Hall, 2008. ➪ *cité page 99*

[26] S. Chakravorty and L. V. Kalé. A fault tolerant protocol for massively parallel systems. *IEEE International Proceedings Parallel and Distributed Processing Symposium*, 12 :212–220, 2004. ➪ *cité page 9*

[27] S. Chakravorty, C. Mendes, and L. Kalé. Proactive fault tolerance in mpi applications via task migration. *High Performance Computing-HiPC 2006*, pages 485–496, 2006. ➪ *cité page 13*

[28] A.K. Chandra and CK Wong. Worst-case analysis of a placement algorithm related to storage allocation. *SIAM Journal on Computing*, 4 :249, 1975. ➪ *3 citations pages 83, 104, and 105*

[29] K.M. Chandy and L. Lamport. Distributed snapshots : Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1) :63–75, 1985. ➪ *cité page 16*

[30] C. Chekuri and M. Bender. An efficient approximation algorithm for minimizing makespan on uniformly related machines. *Journal of Algorithms*, 41(2) :212–224, 2001. ➪ *cité page 83*

[31] Z. Chen and J. Dongarra. Highly scalable self-healing algorithms for high performance scientific computing. *IEEE Transactions on Computers*, pages 1512–1524, 2009. ➪ *cité page 49*

[32] P. Chretienne, E.G. Coffman, J.K. Lenstra, Z. Liu, and P. Brucker. *Scheduling theory and its applications*, volume 149. John Wiley & Sons, 1995. ➪ *cité page 91*

[33] T.F. Coleman and Y. Li. An Interior Trust Region Approach for Nonlinear Minimization Subject to Bounds . *SIAM Journal on Optimization*, 6 :418–445, 1996. ➪ *cité page 35*

[34] R.M. Corless, D.J. Jeffrey, and D.E. Knuth. A sequence of series for the Lambert W function. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, pages 197–204. ACM New York, USA, 1997. ➪ *2 citations pages 34 and 44*

[35] J. T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Computer Systems*, 22(3) :303–312, 2006. ➪ *4 citations pages 36, 41, 68, and 74*

[36] T. Dohi, T. Ozaki, and N. Kaio. Optimal Checkpoint Placement with Equality Constraints. In *2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing*, pages 77–84, 2006. ➪ *2 citations pages 39 and 41*

[37] J. Dongarra, P. Beckman, T. Moore, P. Aerts, G. Aloisio, J.C. Andre, D. Barkai, J.Y. Berthou, T. Boku, B. Braunschweig, et al. The international exascale software project roadmap. *International Journal of High Performance Computing Applications*, 25(1) :3–62, 2011. ➪ *cité page 1*

[38] E. N. Elnozahy and J. S. Plank. Checkpointing for peta-scale systems : A look into the future of practical rollback-recovery. *IEEE Transactions on Dependable Secure Computing*, 1(2) :97–108, 2004. ➪ *cité page 2*

[39] T. Estrada, K. Reed, and M. Taufer. Modeling job lifespan delays in volunteer computing projects. In *Proceedings of the IEEE/ACM International Symposium on Cluster Computing and Grid*, pages 331–338, 2009. ➪ *2 citations pages 65 and 70*

[40] D.G. Feitelson. The supercomputer industry in light of the top500 data. *Computing in Science Engineering*, 7(1) :42 – 47, 2005. ➪ *cité page 1*

[41] Folding@home Papers. `http://folding.stanford.edu/English/Papers`. ➪ *cité page 64*

[42] M.R. Garey and D.S. Johnson. “strong”np-completeness results : Motivation, examples, and implications. *Journal of the ACM (JACM)*, 25(3) :499–508, 1978. ➪ *cité page 82*

[43] M.R. Garey and D.S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-completeness*. WH Freeman & Co. New York, USA, 1979. ➪ *2 citations pages 61 and 100*

[44] S. Genaud and C. Rattanapoka. P2p-mpi : A peer-to-peer framework for robust execution of message passing parallel programs on grids. *Journal of Grid Computing*, 5(1) :27–42, 2007. ➪ *cité page 14*

[45] A. Girault and H. Kalla. A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate. *IEEE Transactions on Dependable and Secure Computing*, 6(4) :241–254, 2009. ➪ *cité page 91*

[46] A. Girault, E. Saule, and D. Trystram. Reliability versus performance for critical applications. *Journal of Parallel and Distributed Computing*, 69(3) :326–336, 2009. ➪ *cité page 90*

[47] Brian Gough. *GNU Scientific Library Reference Manual - Third Edition*. Network Theory Ltd., 3rd edition, 2009. ➪ *cité page 35*

[48] R.L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9) :1563–1581, 1966. ➪ *cité page 82*

[49] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2) :416–429, 1969. ➪ *cité page 82*

[50] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.R. Kan. Optimization and approximation in deterministic sequencing and scheduling : a survey. *Annals of discrete Mathematics*, 5(2) :287–326, 1979. ➪ *cité page 80*

[51] William D. Gropp. Personal communication, May 2010. ➪ *cité page 1*

[52] R. Gupta, H. Naik, and P. Beckman. Understanding checkpointing overheads on massive-scale systems : Analysis of the ibm blue gene/p system. *International Journal of High Performance Computing Applications*, 25(2) :180, 2011. ➪ *3 citations pages 2, 15, and 30*

[53] M. Hakem and F. Butelle. A Bi-objective Algorithm for Scheduling Parallel Applications on Heterogeneous Systems Subject to Failures. In *RenPar2006*, pages 25–35. RenPar2006, 2006. ➪ *cité page 84*

[54] P.H. Hargrove and J.C. Duell. Berkeley lab checkpoint/restart (blcr) for linux clusters. In *Journal of Physics : Conference Series*, volume 46, page 494. IOP Publishing, 2006. ➪ *cité page 15*

[55] E. Heien, D. Kondo, and D. Anderson. Correlated resource models of internet end hosts. In *31st International Conference on Distributed Computing Systems (ICDCS)*, 2011. ➪ *4 citations pages 8, 66, 69, and 70*

[56] E. Heien, D. Kondo, A. Gainaru, D. LaPine, B. Kramer, and F. Cappello. Modeling and tolerating heterogeneous failures in large parallel systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 45. ACM, 2011. ➪ *5 citations pages 1, 7, 8, 10, and 92*

[57] D.S. Hochba. Approximation algorithms for np-hard problems. *ACM SIGACT News*, 28(2) :40–52, 1997. ➪ *cité page 81*

[58] D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM (JACM)*, 34(1) :144–162, 1987. ➪ *cité page 82*

[59] K.H. Huang and J.A. Abraham. Algorithm-based fault tolerance for matrix operations. *IEEE Transactions on Computers*, 100(6) :518–528, 1984. ➪ *cité page 14*

[60] J. Hursey, J. M. Squyres, T. I. Mattox, and A. Lumsdaine. The design and implementation of checkpoint/restart process fault tolerance for Open MPI. In *IEEE International Proceedings Parallel and Distributed Processing Symposium*, pages 1–8, 2007. ➪ *3 citations pages 9, 17, and 18*

[61] A. Iosup, C. Dumitrescu, D. H. J. Epema, H. Li, and L. Wolters. How are real grids used ? the analysis of four grid traces and its implications. In *GRID*, pages 262–269, 2006. ➪ *cité page 50*

[62] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. H. J. Epema. The grid workloads archive. *Future Generation Comp. Syst.*, 24(7) :672–686, 2008. ➪ *cité page 70*

[63] J.M. Jaffe. Efficient scheduling of tasks without full use of processor resources. *Theoretical Computer Science*, 12(1) :1–17, 1980. ➪ *cité page 83*

[64] B. Javadi, D. Kondo, JM. Vincent, and D.P. Anderson. Discovering statistical models of availability in large distributed systems : An empirical study of seti@home. *IEEE Transactions on Parallel and Distributed Systems*, 22(11) :1896 –1903, 2010. ➪ *5 citations pages 1, 8, 64, 66, and 69*

[65] E. Jeannot, E. Saule, and D. Trystram. Optimizing performance and reliability on heterogeneous parallel systems : Approximation algorithms and heuristics. *Journal of Parallel and Distributed Computing*, 2012. ➪ *5 citations pages 88, 93, 98, 100, and 105*

[66] H. Kellerer. Algorithms for multiprocessor scheduling with machine release times. *IIE transactions*, 30(11) :991–999, 1998. ➪ *cité page 82*

[67] D. Kondo, B. Javadi, A. Iosup, and D. Epema. The failure trace archive : Enabling comparative analysis of failures in diverse distributed systems. In *Proceedings of IEEE/ACM International Symposium on Cluster Computing and Grid*, pages 398–407, 2010. ➪ *cité page 66*

[68] S. M. Larson, C. D. Snow, M. Shirts, and V. S. Pande. Folding@Home and Genome@Home : Using distributed computing to tackle previously intractable problems in computational biology. *Computational Genomics*, 2003. ➪ *cité page 64*

[69] JYT Leung. *Handbook of scheduling*. Chapmann & Hall/CRC, New York, 2004. ➪ *cité page 84*

[70] W. Li and K.D. Glazebrook. On stochastic machine scheduling with general distributional assumptions. *European journal of operational research*, 105(3) :525–536, 1998. ➪ *cité page 88*

[71] G. Lin and R. Rajaraman. Approximation algorithms for multiprocessor scheduling under uncertainty. In *Proceedings of the nineteenth annual ACM symposium on parallel algorithms and architectures*, pages 25–34. ACM, 2007. ➪ *cité page 90*

[72] J.W.S. Liu and C.L. Liu. *Bounds on scheduling algorithms for heterogeneous computing systems*. Dept. of Computer Science, University of Illinois at Urbana-Champaign, 1974. ➪ *2 citations pages 82 and 83*

[73] D.K. Lloyd and M. Lipow. *Reliability : management, methods, and mathematics*. Prentice-Hall, 1962. ➪ *cité page 90*

[74] J. Long, W.K. Fuchs, and J.A. Abraham. Compiler-assisted static checkpoint insertion. In *Twenty-Second International Symposium on Fault-Tolerant Computing*, pages 58 –65, 1992. ➪ *cité page 15*

[75] C.D. Lu. *Scalable diskless checkpointing for large parallel systems*. PhD thesis, University of Illinois, 2005. ➪ *2 citations pages 1 and 7*

[76] G. Malewicz. Parallel scheduling of complex dags under uncertainty. In *Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 66–75. ACM, 2005. ➪ *2 citations pages 89 and 90*

[77] A.W Marshall and I. Olkin. Theory of majorization and its applications. *Academic, New York, USA*, 1979. ➪ *4 citations pages 4, 93, 112, and 113*

[78] H. Okamura and T. Dohi. Comprehensive evaluation of aperiodic checkpointing and rejuvenation schemes in operational software system. *Journal of Systems and Software*, 83(9) :1591–1604, 2010. ➪ *cité page 39*

[79] R.A. Oldfield, S. Arunagiri, P.J. Teller, S. Seelam, M.R. Varela, R. Riesen, and P.C. Roth. Modeling the impact of checkpoints on next-generation systems. In *24th IEEE Conference on Mass Storage Systems and Technologies*, pages 30–46, 2007. ➪ *cité page 1*

[80] A. Oliner and J. Stearley. What supercomputers say : A study of five system logs. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 575–584, 2007. ➪ *4 citations pages 1, 2, 7, and 8*

[81] T. Ozaki, T. Dohi, and N. Kaio. Numerical computation algorithms for sequential checkpoint placement. *Performance Evaluation*, 66(6) :311–326, 2009. ➪ *cité page 39*

[82] C.H. Papadimitriou and M. Yannakakis. On the approximability of tradeoffs and optimal access of web sources. In *IEEE Proceedings of the 41st Annual Symposium on Foundations of Computer Science.*, pages 86–92, 2000. ➪ *8 citations pages 85, 86, 89, 92, 96, 99, 100, and 101*

[83] M.L. Puterman. *Markov decision processes : Discrete stochastic dynamic programming*. John Wiley & Sons, Inc. New York, USA, 1994. ➪ *5 citations pages 11, 50, 51, 55, and 115*

[84] J.B. Rosen. The gradient projection method for nonlinear programming. part i. linear constraints. *Journal of the Society for Industrial and Applied Mathematics*, 8(1) :181–217, 1960. ➪ *cité page 35*

[85] S. Sankaran, J. M. Squyres, B. Barrett, A. Lumsdaine, J. Duell, P. Hargrove, and E. Roman. The LAM/MPI checkpoint/restart framework : System-initiated checkpointing. *International Journal Of High Performance Computing Applications*, 19(4) :479–493, 2005. ➪ *3 citations pages 9, 17, and 18*

[86] E. Saule. *Algorithmes d'approximation pour l'ordonnancement multi-objectif. Application aux systèmes parallèles et embarqués*. PhD thesis, Grenoble INP, 2008. ➪ *cité page 84*

[87] E. Saule and D. Trystram. Analyzing scheduling with transient failures. *Information Processing Letters*, 109(11) :539–542, 2009. ➪ *cité page 92*

[88] F.B. Schneider. Implementing fault-tolerant services using the state machine approach : A tutorial. *ACM Computing Surveys (CSUR)*, 22(4) :299–319, 1990. ➪ *2 citations pages 14 and 89*

[89] B. Schroeder and G.A. Gibson. A large-scale study of failures in high-performance computing systems. *IEEE Transactions on Dependable and Secure Computing*, 7(4) :337–351, 2010. ➪ *6 citations pages 1, 2, 7, 10, 12, and 92*

[90] S.L. Scott, C. Engelmann, G.R. Vallée, T. Naughton, A. Tikotekar, G. Ostrouchov, C. Leangsuksun, N. Naksinehaboon, R. Nassar, M. Paun, et al. A tunable holistic resiliency approach for high-performance computing systems. In *ACM SIGPLAN Notices*, volume 44, pages 305–306. ACM, 2009. ➪ *cité page 13*

[91] Boinc stats for seti. http://boincstats.com/stats/project_graph.php?pr=sah&view=hosts. ➪ *cité page 64*

[92] D.B. Shmoys, J. Wein, and D.P Williamson. Scheduling parallel machines on-line. *SIAM journal on computing*, 24(6) :1313–1331, 1995. ➪ *cité page 83*

[93] D.P. Siewiorek, R.S. Swarz, and D.P. Suewiorek. *Reliable computer systems : design and evaluation*, volume 3. AK Peters MA, 1998. ➪ *cité page 90*

[94] C. Sosa, B. Knudson, and International Business Machines Corporation. International Technical Support Organization. *IBM system Blue Gene solution : Blue Gene/P application development*. IBM International Technical Support Organization, 2009. ➪ *2 citations pages 17 and 18*

[95] R. Strom and S. Yemini. Optimistic recovery in distributed systems. *ACM Transactions on Computer Systems*, 3(3) :204–226, 1985. ➪ *cité page 17*

[96] H. C. Tijms. *A First Course in Stochastic Models*. John Wiley, 2003. ➪ *cité page 26*

[97] S. Toueg and Ö. Babaoğlu. On the optimum checkpoint selection problem. *SIAM Journal on Computing*, 13(3) :630–649, 1984. ➪ *2 citations pages 62 and 63*

[98] L.G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8) :103–111, 1990. ➪ *cité page 49*

[99] M. Voorneveld. Characterization of pareto dominance. *Operations Research Letters*, 31(1) :7–11, 2003. ➪ *cité page 85*

[100] C. Wang, F. Mueller, C. Engelmann, and S.L. Scott. Proactive process-level live migration in hpc environments. In *IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2008. ➪ *cité page 13*

[101] X.Lin Y.Ling, J.Mi. A variational calculus approach to optimal checkpoint placement. *IEEE Transactions on Computers*, 50(07) :699, July 2001. ➪ *3 citations pages 37, 39, and 41*

[102] J. W. Young. A first order approximation to the optimum checkpoint interval. *Communications of the ACM*, 17(9) :530–531, 1974. ➪ *2 citations pages 36 and 41*

Table des figures

2.1 Chaîne de Markov associée à l'élément de calcul 11
2.2 Exemple typique d'un point de sauvegarde cohérent C_1 et un autre point incohérent C_2 [10] 16

3.1 modèle d'exécution sans panne (a) avec pannes (b) 23
3.2 L'espérance du nombre de re-exécutions 25
3.3 Variation de l'espérance du temps de complétion en fonction de la quantité de travail initial, le taux de panne et d'indisponibilité 27
3.4 Schéma d'exécution avec sauvegarde et reprise 30
3.5 Variation de la moyenne du temps de complétion (Scenario 1) 42
3.6 Nombre optimal des points de sauvegarde (Scenario 1) 43
3.7 Variation de la moyenne du temps de complétion (Scenario 2) 43
3.8 Moyenne du surcoût accumulé de sauvegarde / Moyenne Travail perdu (Scenario 2) 44

4.1 modèle d'exécution sans sauvegarde (a) modèle d'exécution avec sauvegarde (b) 49
4.2 Schéma d'exécution avec panne. 53
4.3 Variation du temps de complétion (a) et du temps total perdu (b) avec 5000 clients en fonction du nombre d'instances. 72
4.4 Variation du temps de complétion (a) et du temps total perdu (b) avec 10000 clients en fonction du nombre d'instances. 72
4.5 Variation du temps de complétion (a) et du temps total perdu (b) avec 20000 clients en fonction du nombre d'instances. 72

Table des matières

Sommaire iii

1 Introduction 1
1.1 Contexte et motivations . . . 1
1.2 Positionnement et problématiques . . . 2
1.3 Guide de lecture et contributions . . . 3

2 Préliminaires 5
2.1 Introduction . . . 5
2.2 Caractéristiques et conséquences des pannes . . . 6
2.2.1 Caractéristiques des pannes . . . 6
2.2.2 Conséquences des pannes . . . 8
2.3 Modélisation formelle des pannes . . . 9
2.3.1 Modélisation stochastique des pannes . . . 9
a Collecte et analyse des traces . . . 10
b Inférence statistique . . . 10
2.3.2 Modèle de panne typique . . . 10
2.3.3 Distributions de panne typiques . . . 12
2.4 Approches de tolérance aux pannes . . . 13
2.4.1 Prévention des pannes . . . 13
a Migration . . . 13
2.4.2 Masquage des pannes . . . 13
a Duplication . . . 13
b Approches algorithmiques (ABFT) . . . 14
2.4.3 Recouvrement après panne . . . 14
a Sauvegarde et reprise coordonnée . . . 16
b Sauvegarde et reprise non coordonnée . . . 17
2.5 Bilan et conclusions . . . 18

I Modélisation Stochastique du Mécanisme de Sauvegarde et Reprise 19

3 Modélisation continue 21
3.1 Introduction . . . 21
3.2 Modélisation sans tolérance aux pannes . . . 22
3.2.1 Modèle de l'environnement d'exécution . . . 22
3.2.2 Distribution du nombre de re-exécutions . . . 24
3.2.3 Espérance du temps de complétion . . . 25
3.2.4 Bilan . . . 27

3.3 Modélisation avec tolérance aux pannes 28
3.3.1 Modèle du mécanisme de sauvegarde et reprise 28
3.3.2 Formulation du problème d'optimisation 31
a Espérance du temps de complétion avec sauvegarde et reprise 31
b Problème d'optimisation sous contraintes 32
3.3.3 Résolution du problème d'optimisation 32
a Solution analytique 32
b Solution numérique 34
3.4 État de l'art 35
3.5 Validation et comparaison expérimentale 41
3.5.1 Paramètres et scénarios des simulations 41
3.5.2 Résultats des simulations 41
3.5.3 Bilan des simulations 43
3.6 Conclusion 44

4 Modélisation discrète 47
4.1 Introduction 47
4.2 Modèle de l'environnement d'exécution 48
4.2.1 Modèle de la plateforme d'exécution 48
4.2.2 Modèle d'application 49
4.2.3 Modèle du mécanisme de sauvegarde 50
4.3 Formulation du problème d'optimisation 50
4.3.1 Introduction au processus de décision markovien 50
4.3.2 Problème d'ordonnancement des points de sauvegarde 51
4.3.3 Ordonnancement optimal des points de sauvegarde 55
4.4 Analyse de complexité 56
4.4.1 Analyse de complexité de l'algorithme proposé 56
a Analyse de complexité dans le cas général 56
b Cas surcoût des points de sauvegarde constant 58
4.4.2 Analyse de complexité du problème 58
a Définition du problème décisionnel relaxé 59
b Définition de la famille d'instances 59
c La NP-complétude du problème 61
4.5 État de l'art 62
4.6 Étude expérimentale 64
4.6.1 Modèle de l'environnement d'exécution 65
a Modèle d'application 65
b Modèle de communication 65
c Modèle de panne 66
d La stratégie d'ordonnancement des points de sauvegarde . . 66
4.6.2 Méthodologie d'expérimentation 67
a Critères de performance et stratégies d'ordonnancement : . . 67
b Modèle SimGrid 68
c Les scénarios : 69
4.6.3 Analyse des résultats 71
4.7 Conclusions 73

II Ordonnancement 77

5 Ordonnancement tolérant aux pannes 79
5.1 Introduction 79
5.2 Formulation du problème d'optimisation 80
5.3 État de l'art 82
5.3.1 Quelques rappels en théorie d'ordonnancement 82
5.3.2 Optimisation multi-objectifs 84
5.3.3 Résultats dans le cadre de la tolérance aux pannes 87
5.4 Optimisation du *makespan* et de la fiabilité de l'application 93
5.4.1 Optimisation de $Q||(C_{max}, \mathcal{R}_l)$ pour une loi *DFR* 93
a Tâches indépendantes et unitaires 95
b Tâches indépendantes et arbitraires 99
5.4.2 Optimisation de $P||(C_{max}, \overline{\mathcal{R}}_l)$ avec $\lambda(t)$ *IFR* 102
a Taches unitaires et indépendantes 102
b Tâches indépendantes et arbitraires 104
5.5 Conclusions 105

6 Conclusions et travaux futurs 107
6.1 Bilan 107
6.2 Perspectives 108

Annexe 111
6.3 Schur-convexe 112
6.4 Distribution de probabilité *IFR*/*DFR* 113

Bibliographie 117

Table des figures 125

Table des matières 127