République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mouloud Mammeri de Tizi-Ouzou

Faculté de : Génie électrique et d'informatique
Département : Informatique

Mémoire de fin d'études

En vue de l'obtention du diplôme de

Master en Informatique

Spécialité : Conduite de Projets Informatiques

Présenté par :

Belfaked Rachida et **Tessa Samira**

Sujet : Implémentation et évaluation d'un modèle de Checkpointing pour la Tolérance aux Fautes dans des environnements avec une caractérisation incomplète des pannes

Proposé et dirigé par : SADI Samy

Soutenu le 27 Septembre 2018 devant le Jury composé de:

Mme. AMIROUCHE Fatiha	Présidente du Jury
Mr. RAMDANI Mohamed	Membre du Jury
Mr. SADI Samy	Directeur de mémoire

Remerciements

Nous remercions ALLAH qui nous aide et nous donne la patience et le courage durant ces longues années d'études.

Nous remercions notre encadreur le Dr. Samy Sadi pour son soutien, ses recommandations judicieuses, sa disponibilité, son suivi attentif, ses encouragements continus et c'est ce qui nous a permis de mieux nous exprimer et faire valoir nos connaissances.

Nos vifs remerciements vont également aux membres de jury pour l'intérêt qu'ils ont porté à notre travail en acceptant de l'examiner et de l'enrichir par leurs remarques.

Enfin, nous tenons également à remercier toutes les personnes qui ont participé de près ou de loin à la réalisation de ce mémoire

Samira et Rachida

Dédicaces

À mes très chers parents, source de tendresse, de noblesse et de sacrifice.

À mes chères sœurs Kahina, Zina, Mounira et mon frère Rezak pour leurs appui, soutiens et leurs encouragement.

À mon ami Amine qui m'a toujours aidé et soutenu.

À mes amies et mon binôme à qui je souhaite plus de succès.

À tous ceux que j'aime.

Rachida.

Dédicaces

A Ma mère, qui a œuvré pour ma réussite, de par son amour, son soutien, tous les sacrifices consentis et ses précieux conseils, pour toute son assistance et sa présence dans ma vie

À tous mes frères, ma sœur et son marie et mes ami(e)s que j'aime beaucoup ;

A mon binôme et mon amie Rachida, celle avec qui j'ai partagé mes années d'études et à toute sa famille.

Samira.

Résumé

Ce mémoire s'inscrit dans le cadre de l'amélioration de la Tolérance aux Fautes dans les systèmes informatiques. En effet, les développeurs d'applications font face à plusieurs défis parmi lesquels figure l'assurance de la continuité des services délivrés en présence de pannes. Cela passe par l'implémentation de techniques palliant à ce problème dans le système ciblé.

Dans ce mémoire, nous nous focalisons sur une des techniques de Tolérance aux Fautes permettant de minimiser l'impact des pannes à savoir : le Checkpointing. Nous étudions plus précisément la problématique de sélection de l'intervalle de Checkpointing dans les systèmes à caractérisation incomplète des pannes.

Nous répondons, tout au long de ce travail, à cette problématique. Pour cela, nous définissons et présentons les concepts les plus importants liés à la sûreté de fonctionnement et à la Tolérance aux Fautes. Ensuite, nous dressons un état de l'art concernant la problématique du choix de l'intervalle de Checkpointing. Comme contribution, nous proposons deux approches pour le calcul de cet intervalle dans un environnement à caractérisation incomplète des pannes. Enfin, nous implémentons ces approches dans un simulateur existant pour tester et vérifier leur efficacité. Ces tests nous ont permis de confirmer la pertinence de ces approches lorsque les propriétés des pannes dans le système considéré sont méconnues.

Mots clés: Tolérance aux Fautes, Checkpointing, Intervalle de Checkpointing, Simulation, Évaluation de performances.

Abstract

This report is made within the context of improvement of Fault Tolerance in computer systems. Indeed, application developers face several challenges, among which, the assurance of continuity of services delivered despite the presence of failures. This requires the implementation of Fault Tolerance techniques to overcome this problem in the targeted system.

In this report, we focus on one of the techniques of Fault Tolerance to minimize the impact of failures, namely: Checkpointing. In particular, we focus on the problem of the selection of the Checkpointing interval in systems with incomplete fault characterization.

Throughout this work, we respond to this problem. To do this, we define and present the most important concepts related to dependability and Fault Tolerance. Then, we draw up a state-of-the-art about the problem of the choice of the interval of Checkpointing. As a contribution, we propose two approaches for calculating this interval in an incomplete fault characterization environment. Finally, we implement these approaches in an existing simulator to test and verify their effectiveness. These tests enabled us to confirm the relevance of these approaches when the properties of the failures in the considered system are unknown.

Key-words: Fault Tolerance, Checkpointing, Checkpointing Interval, Simulation, Performance Evaluation.

Table des matières

Table des tableaux et des figures

Introduction générale

1 Contexte du travail

L'informatique s'affirme partout de nos jours, que ce soit dans les banques, les distributeurs, les écoles, les universités, les bibliothèques, les aéroports, chez-soi ou au travail plus rien ne fonctionne sans elle. Avec son évolution spectaculaire, et l'intérêt qu'elle suscite chez les gens, cette technologie a accumulé depuis sa mise en service au grand public une importance hors-pair. Certaines personnes la classent dans les nouvelles technologies. Pourtant, elle a plus d'un demi-siècle d'existence.

Malgré toute cette évolution qu'a connue l'informatique, l'homme n'a jamais cessé de rechercher la perfection. Ainsi, il a apporté de nouvelles évolutions et continue de tenter de combler les lacunes et les déficiences inévitables que rencontrent les systèmes informatiques et les applications en particulier. Car aucun système informatique et aucune application n'est à l'abri des fautes et des pannes.

Ainsi, comme tout service est sujet aux défaillances, et vu que c'est impossible d'empêcher l'arrivée des pannes dans un système, l'homme cherche à accroitre la qualité des services offerts aux utilisateurs en essayant d'assurer la continuité du service délivré en dépit des pannes et cela en faisant usage des techniques de Tolérance aux Fautes.

La Tolérance aux Fautes est un domaine vaste qui suscite beaucoup d'intérêt et connaît une énorme progression tout comme les autres domaines de l'informatique. Beaucoup d'effort de recherche a été mobilisé pour la conception et la mise en place de techniques logicielles et d'approches algorithmiques pour la Tolérance aux Fautes des systèmes informatiques.

Une des techniques les plus souvent utilisées dans ce sens est le Checkpointing. Cette technique consiste à sauvegarder périodiquement l'état d'une application sur un support de stockage fiable. L'état ainsi sauvegardé est appelé Checkpoint ou point de contrôle. En cas de panne, l'application redémarre à partir du dernier point de contrôle, évitant ainsi une perte importante des données et du temps d'exécution.

2 Problématique

La mise en place du Checkpointing soulève beaucoup d'interrogations. La plus importante est certainement celle relative à la fréquence de création des points de contrôle.

En effet, une fréquence trop élevée (ou un intervalle trop court entre les moments de création des points de contrôle) va pénaliser de façon importante l'exécution de l'application. La raison est que le Checkpointing a un coût en temps d'exécution et en transfert réseau qui se répercutent sur les performances de l'application. D'un autre côté, une fréquence trop faible (ou un intervalle trop long), va causer une perte importante de l'état de l'application à cause des pannes.

Ainsi, il faut déterminer un compromis entre ces deux critères pour avoir une fréquence ou intervalle de Checkpointing adéquat.

Ce problème a été partiellement résolu dans l'état de l'art lors de la connaissance des propriétés du système relativement à l'occurrence des pannes, en particulier, si le temps moyen entre deux pannes dans le système est connu. Cependant, ce n'est pas le cas lorsque cette dernière propriété n'est pas connue ou est mal évaluée. Ce qui est souvent le cas avec des systèmes neufs, ou vieillissants (le temps moyen entre deux pannes dans le système change avec le temps).

La problématique qui se pose alors est de comment choisir la fréquence (ou intervalle) de Checkpointing avec la non-connaissance ou la méconnaissance des propriétés de panne du système ?

Une problématique annexe est celle relative à l'évaluation de la solution proposée. En effet, comment savoir si le choix fait est pertinent ?

3 Contribution

Notre contribution dans le cadre de ce mémoire se situe à plusieurs niveaux:

1. Nous établissons un état de l'art relatif à la sûreté de fonctionnement et à la Tolérance aux Fautes dans les systèmes informatiques ;
2. Nous établissons un état de l'art relatif aux techniques de Checkpointing et au choix de l'intervalle de Checkpointing ;
3. Nous proposons deux approches pour la sélection de l'intervalle de Checkpointing lorsque la distribution et la fréquence des pannes n'est pas ou est mal caractérisée dans le système considéré ;

4. Nous implémentons nos approches sur un simulateur existant et nous les évaluons en réalisant plusieurs simulations.

4 Organisation du mémoire

Ce mémoire s'articule autour de la problématique précédente et il est répartit comme suit.

La première partie de ce mémoire comporte le premier et deuxième chapitre. Elle est dédiée à la présentation de l'état de l'art relatif à la sûreté de fonctionnement et à la Tolérance aux Fautes, puis au Checkpointing et au choix de l'intervalle de Checkpointing.

Dans la deuxième partie du mémoire, nous présentons nos contributions. Tout d'abord, dans le troisième chapitre nous revenons sur la problématique principale traitée, et nous proposons une solution. Ensuite, dans le quatrième et dernier chapitre, nous évaluons la solution proposée en implémentant nos approches puis en procédant à plusieurs simulations sur un simulateur existant.

Chapitre 1: La Tolérance aux Fautes

1 Introduction

Les systèmes sont conçus pour fournir à leurs utilisateurs un service qui répond à leurs attentes tout en respectant ses spécifications. L'impact de ces deux critères est ressenti via la qualité du service délivrée aux utilisateurs. Ainsi, le système en question doit être fiable, disponible, sécurisé et maintenable. L'ensemble de ces propriétés constitue les attributs clés pour bâtir un système sûr et fonctionnel. Toute déviation du système par rapport à ses spécifications est considérée comme une panne. Dès lors, le système ne délivre plus un service correct ou il cesse de fonctionner. Pour retrouver sa conformité et revenir à l'état normal, le système doit maitriser et s'affranchir de ces pannes par la mise en œuvre de différentes techniques de Tolérance aux Fautes, telle que le Checkpointing.

Dans ce chapitre, nous présentons d'abord le vocabulaire relatif à la sûreté de fonctionnement : ses attributs, ses moyens et ses entraves. Nous présentons également les deux catégorisations des pannes. Ensuite, nous décrivons les caractéristiques des pannes dans les systèmes informatiques, particulièrement en ce qui concerne leur fréquence, ce qui nous permet de souligner l'importance de leur prise en charge. Nous parlons aussi de la Tolérance aux Fautes comme d'un moyen pour assurer la sûreté de fonctionnement, puis nous présentons les techniques qui s'y rapportent. Enfin et avant de conclure, nous introduisons l'une des techniques pour la Tolérance aux Fautes la plus fréquemment utilisée qui est le Checkpointing, et qui est l'objet de notre mémoire.

2 Sûreté de fonctionnement

2.1 Définition

La sûreté de fonctionnement est apparue comme une nécessité au cours du XX^{ème}siècle, notamment avec la révolution industrielle. Le terme *dependability* est apparu dans une publicité sur des moteurs *Dodge Brothers* dans les années 1930. Selon J.C. Laprie la sûreté de fonctionnement est définie ainsi : « la sûreté de fonctionnement d'un système informatique est la propriété qui permet de placer une confiance justifiée dans le service qu'il délivre » (1).

La sûreté de fonctionnement est souvent appelée la science des défaillances. Autrement dit, la sûreté de fonctionnement signifie que le système va accomplir ses fonctions, et délivrer un service correct sans déviation ou défaillance. Il existe de nombreuses définitions,

de standards qui peuvent varier selon les domaines d'application : nucléaire, spatial, aéronautique, automobile, informatique, etc.

Dans la suite, nous présentons les concepts liés à la sûreté de fonctionnement (attributs, entraves et moyens) illustrés par la Figure 1 avant de s'intéresser à l'un de ses moyens en particulier qui est : la Tolérance aux Fautes.

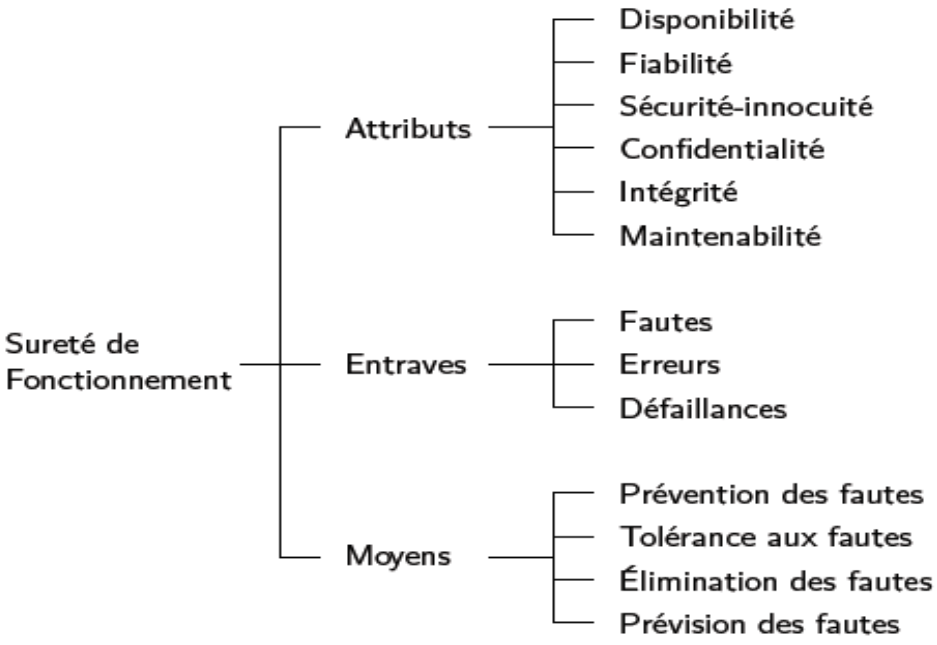

Figure 1:Les attributs, entraves et moyens de la sûreté de fonctionnement

2.2 Les attributs

Les attributs de la sûreté de fonctionnement d'un système permettent d'exprimer les propriétés qui sont attendues du système, afin d'apprécier la qualité du service délivré.

2.2.1 La disponibilité

La disponibilité mesure le degré avec lequel un système reste dans un état opérationnel. C'est la proportion du temps où le système reste dans un état fonctionnel. Ceci est souvent décrit par un pourcentage de disponibilité.

2.2.2 La fiabilité

La fiabilité ou la dependabilité décrit la capacité d'un système ou d'un composant à fonctionner dans des conditions correctes (préalablement spécifiées) pendant une période de temps. La fiabilité est étroitement liée à la disponibilité et à la maintenabilité d'un système. En effet, si le système a une faible fiabilité alors son pourcentage de disponibilité diminue. De même, si le système a une mauvaise maintenabilité alors certainement sa fiabilité l'est aussi.

Pour quantifier la fiabilité d'un système, on fait appel à deux mesures le *MTTF* et le *MTBF*. Le *MTTF* (de l'anglais : *Mean Time To Failure*) désigne le temps moyen avant la première panne. Quant au *MTBF* (de l'anglais : *Mean Time Between Failure*), il mesure la durée moyenne entre deux pannes. Il convient de noter que ces deux mesures sont étroitement liées à la fiabilité du système, ainsi plus ces deux mesures sont grandes plus le système est fiable.

Si le *Rel(t)* représente la probabilité que le système soit fiable pour une durée *t,* alors le *MTTF* du système est défini selon cette équation :

$$MTTF = \int_0^{\infty} Rel(t)$$

Lorsque le système est constitué de plusieurs composants indépendants et sans stratégies de Tolérance aux Fautes (cas d'un système distribué sur plusieurs machines), alors le *MTTF* du système est calculé ainsi :

$$MTTF_{système} = \frac{MTTF_{composant}}{nombre\ de\ composants}$$

2.2.3 La maintenabilité

La maintenabilité est une mesure décrivant la facilité avec laquelle un système peut être gardé dans un état correct (et donc réparé après une panne). Pour quantifier cet attribut, on se réfère à la durée moyenne avant la première panne ou le *MTTR* (en anglais : *Mean Time To Repair*). Lorsque le système a un *MTTR* faible cela implique que le système est doté d'une bonne maintenabilité. Si le *Mnt(t)* représente la probabilité qu'une réparation soit effectuée durant la durée *t*, alors l'équation qui définit le *MTTR du système est la suivante :*

$$MTTR = \int_0^{\infty} 1 - Mnt(t)d(t)$$

2.2.4 La confidentialité

La confidentialité mesure la capacité du système à ne pas divulguer d'informations censées être protégées durant un certain laps de temps.

2.2.5 L'intégrité

L'intégrité mesure la capacité d'un système à protéger ses données et applications contre des altérations ou des modifications non autorisées durant un certain laps de temps.

2.2.6 La sécurité-innocuité

La sécurité-innocuité mesure la capacité du système durant un certain intervalle à ne pas causer d'incidents catastrophiques sur son environnement.

2.3 Entraves à la sûreté de fonctionnement

Les entraves sont des évènements indésirables qui surviennent et qui peuvent affecter le système et dégrader la sûreté de fonctionnement.

2.3.1 La faute

Toute cause (événement, action, circonstance) pouvant provoquer une erreur. Exemple: faute dans la programmation.

2.3.2 L'erreur

C'est la manifestation d'une faute dans le système, elle est susceptible de provoquer une défaillance.

2.3.3 La défaillance (ou panne)

C'est la conséquence d'une erreur, la défaillance dénote l'incapacité du système à assurer le service spécifié par l'utilisateur.

Figure 2:La chaine fondamentale des entraves à la sûreté de fonctionnement

2.4 Les moyens de la sûreté de fonctionnement

Il s'agit des méthodes et techniques permettant d'améliorer la sûreté de fonctionnement d'un système en remédiant aux conséquences des fautes, erreurs et pannes dans ce système.

2.4.1 La prévention des fautes

Ensemble de moyens qui visent à empêcher l'apparition ou l'introduction de fautes dans le système. Elle se focalise sur les techniques permettant de réduire la présence, le nombre de fautes dans le système et leurs impacts. Ces techniques sont réalisées pendant la phase de développement ou lors des premières utilisations du système.

2.4.2 La prévision des fautes

Il s'agit d'une estimation ou d'une évaluation de la présence des fautes (temps, nombres, conséquences) dans le système, dans le but d'anticiper l'arrivée des pannes.

2.4.3 La Tolérance aux Fautes

Elle désigne la capacité du système à assurer la continuité du service en dépit des fautes. Cette continuité est obtenue par la mise en place des techniques de tolérance aux fautes qui passent par la détection des erreurs puis le rétablissement du système. L'ensemble de ces techniques sont suivies par des opérations de maintenance afin de corriger les composants fautifs. Nous revenons plus en détail sur l'ensemble de ces techniques dans la suite de ce chapitre.

3 Catégories de pannes

Les pannes peuvent être classées soit selon leur origine ou selon leur degré de gravité.

3.1 Selon l'origine

Les défaillances d'un système peuvent être d'origine matérielle, logicielle, humaine ou environnementale.

3.1.1 Les pannes d'origine matérielle

Les pannes d'origine matérielle sont causées par un dysfonctionnement d'un ou plusieurs composants matériels du système. La récupération du système nécessite la réparation ou le remplacement du composant défaillant.

3.1.2 Les pannes d'origine logicielle

En pratique, plus de 50 % des pannes sont d'origine logicielle (2). Les défaillances des logiciels sont causées par des fautes ou bugs dans les programmes, qui entrainent une défaillance dans certaines conditions d'exécution. Pour éliminer ces fautes, des tests logiciels peuvent être utilisés mais il est extrêmement couteux et difficile de détecter ce genre de fautes.

3.1.3 Les pannes humaines

Elles sont dues à une erreur de l'être humain lors de son intervention ou de son contact avec le système, quand il effectue des mises à jour logicielles ou quand il remplace des composants. Ces pannes sont souvent non intentionnelles et résultent d'un manque de compétence ou d'une négligence.

3.1.4 Les pannes naturelles

L'environnement dans lequel se trouve le système peut être affecté par des facteurs extérieurs telles que : les incendies, les séismes, les pannes d'alimentation, etc.

Ces contraintes environnementales peuvent entrainer une indisponibilité du système mais également des pannes.

3.2 Selon le degré de gravité

Les pannes peuvent être aussi classées selon leur degré de gravité. Nous distinguons alors quatre groupes de pannes.

3.2.1 Les pannes franches

Les pannes franches causent l'arrêt du système. Il s'agit du modèle de panne le plus simple. Le système a alors seulement deux états : soit il fonctionne et donne des résultats corrects soit il est en panne et ne fait rien.

3.2.2 Les pannes par omission

Les pannes par omission causent la perte de messages entrants (omission en réception), ou sortants (omission en émission).

3.2.3 Les pannes temporelles

Les pannes temporelles causent des comportements anormaux par rapport aux temps de réponse du système. Ainsi, le système délivre bien une réponse, mais celle-ci est soit trop tardive ou trop précoce.

3.2.4 Les pannes arbitraires ou byzantines

Tout comportement s'écartant des spécifications est qualifié de comportement byzantin. Le système fournit alors des résultats non conformes aux requêtes.

4 Caractéristiques des pannes

Pour bien cerner la notion de la Tolérance aux Fautes, il faut bien comprendre et analyser les caractéristiques des pannes dans le système cible. Dans le cadre de ce travail, nous avons abordé le problème de la Tolérance aux Fautes et envisagé les caractéristiques des pannes en général en dehors du contexte ou de l'environnement d'exécution.

En général, nous distinguons deux types de pannes classées selon l'origine de déclenchement. Le premier type inclut les pannes dont l'origine de déclenchement est un événement accidentel. Les pannes sont alors dues à l'usure physique du matériel et à la vétusté des composants. Par exemple, dans les environnements *HPC* comme dans le *Cloud,* les pannes arrivent à cause de l'usure du matériel, la complexité des composants, ou à cause d'une erreur humaine de manipulation (3) et (4).

Le deuxième type de pannes inclut les pannes d'origine malicieuse. Ces pannes sont causées par des programmes malveillants (malware).Très difficiles à détecter, elles sont introduites intentionnellement par l'être humain dans le but de nuire au système cible en exploitant les failles de ce dernier.

Dans le cadre des environnements *HPC,* les pannes sont généralement classées en deux catégories suivant leur origine. La première catégorie inclut les pannes dont l'origine principale est un problème matériel (en anglais : *hardware failures*). Tandis que la deuxième catégorie inclut les pannes dont l'origine principale est un problème logiciel (en anglais : *software failures*).

Ainsi, la Tolérance aux Fautes dans ces systèmes passe par la mise en place d'un ensemble de moyens distincts et des techniques différentes pour toucher à ces deux aspects de pannes : les pannes logicielles et matérielles.

Déduire l'origine des pannes la plus répandue est une tâche très difficile et les travaux existants faits à ce sujet ont des conclusions différentes. Par exemple Gibson et al (5)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 (6)constatent que 64% des pannes sont d'origine logicielle en analysant des traces provenant des centres de calcul.

5 Techniques de Tolérance aux Fautes

L'objectif de la Tolérance aux Fautes est d'éviter la défaillance du système malgré la présence de fautes. Il n'existe pas de techniques de Tolérance aux Fautes valables dans l'absolu mais il existe des mécanismes à mettre en œuvre pour développer des systèmes sûrs, voire, tolérants aux pannes. Ces techniques passent parla détection des erreurs et le rétablissement du système.

5.1 La détection des erreurs

La détection des erreurs est la première étape à franchir avant de rétablir le système. Cette technique a pour objectif d'identifier l'existence des erreurs ou des états incohérents dans le système. Pour y parvenir, la détection repose sur un ensemble de techniques qui sont la surveillance continue à base de *heart-beats* qui consiste à échanger régulièrement des messages de vie (en anglais : *heart-beat messages*) entre plusieurs machines du système. Lorsqu'une machine n'émet pas de signal et ne transmet plus de messages, elle est considérée comme défaillante.

Il y'a une autre technique qui permet de compléter et de corriger les failles observées dans la surveillance continue à base de *heart-beats* qui est la surveillance continu à base de tests fonctionnels. Cette technique consiste à envoyer fréquemment des requêtes de tests vers des machines cibles et observer la réponse de ces dernières à ces requêtes. Les réponses attendues doivent satisfaire certains critères prédéfinis à l'avance. Si une machine génère une réponse invalide qui ne remplit pas les spécifications, alors cette machine est considérée comme défaillante.

Une autre technique de détection utilise des codes détecteurs d'erreurs. Ils permettent de détecter les erreurs de transmission et de stockage de données en leurs associant différentes sommes de contrôle par exemple.

5.2 Le rétablissement du système

Le rétablissement du système vise à transformer l'état erroné d'un système en un état sans erreurs qui lui permette de délivrer un service correct. Le rétablissement du système peut être réalisé grâce à des techniques comme la redondance et la reprise.

5.2.1 La redondance

La redondance permet de masquer la présence de fautes, elle se présente en plusieurs types.

5.2.1.1 La redondance matérielle

Ce type de redondance est obtenu en dupliquant du matériel, que ce soit des ordinateurs ou des composants (processeurs, mémoires, disques durs, etc.). Dans le cas où le premier composant tombe en panne, l'autre composant de secours prend immédiatement le relais empêchant ainsi l'arrêt du système.

5.2.1.2 La redondance logicielle

Dans la redondance logicielle, on duplique des logiciels ou des programmes. En effet, plusieurs exemplaires et instructions d'un même programme sont créés. Ainsi la panne d'un logiciel ou d'un programme peut être masquée grâce aux autres répliques existantes de celui-ci.

5.2.1.3 La redondance temporelle

Ce type de redondance consiste à rajouter du temps en plus aux instructions. Ce temps supplémentaire servira à ré-exécuter les instructions plusieurs fois pour arriver à un résultat correct.

5.2.1.4 La redondance de données

La redondance de données consiste à rajouter aux données utiles des informations de redondance qui serviront à corriger les erreurs introduites lors du stockage, transmission et enregistrement de données.

5.2.2 La reprise avant

La reprise avant consiste à manipuler l'état courant d'un système contenant des erreurs, afin de générer un nouvel état dépourvu d'erreurs. A partir de ce nouvel état obtenu, le système peut continuer son exécution normale. La reprise avant s'avère utile lorsque la continuité du service passe avant l'exactitude des résultats délivrés.

5.2.3 Le Checkpointing ou la reprise arrière

La reprise arrière consiste à ramener le système dans un état antérieur ou sauvegardé survenant avant l'occurrence de l'erreur. Pour ce faire, cette technique crée, assez fréquemment, des points de sauvegarde aussi appelés points de contrôle ou tout simplement Checkpoints. Le processus de création de ces points de sauvegarde est le Checkpointing. Nous revenons plus en détail sur cette technique dans la section suivante, vu que notre travail s'intéresse précisément à cette approche de Tolérance aux Fautes.

6 Le Checkpointing

6.1 Définition

Le Checkpointing est un anglicisme désignant le processus de création de points de sauvegarde également appelés Checkpoints. C'est une technique permettant d'assurer la Tolérance aux Fautes d'un système en permettant le rétablissement d'applications en cas de pannes. Elle consiste à sauvegarder l'état d'une application en cours d'exécution sur une mémoire stable (ou mémoire secondaire) de manière assez fréquente afin de pouvoir restaurer cette application et continuer son exécution sur une autre machine (ou machine secondaire) en cas de panne.

La création d'un point de sauvegarde nécessite souvent la suspension de l'exécution de l'application considérée. Ainsi, le Checkpointing a un coût sur l'exécution des applications. De même, pour garantir la cohérence des applications, leurs communications réseau peuvent être journalisées ou mises en tampon mémoire à certains moments lors du processus du Checkpointing. Ainsi, ce processus a aussi un coût sur les communications des applications.

Le choix intelligent de la « fréquence de Checkpointing », ou fréquence avec laquelle l'état d'une application est sauvegardé, permet de minimiser ces coûts sans porter atteinte à la Tolérance aux Fautes du système. Cette question de recherche est des fois posée autrement : nous cherchons plutôt à déterminer « l'intervalle de Checkpointing » qui représente l'intervalle de temps entre les moments de création de deux Checkpoints successifs. Nous évoquons cette question de recherche dans les chapitres suivants.

Dans ce qui suit, nous présentons les types d'implémentation du Checkpointing. Ces types diffèrent au niveau de la transparence et de l'efficacité des mécanismes déployés pour l'implémentation.

6.2 Niveaux d'implémentation du Checkpointing

Le Checkpointing peut être implémenté au niveau de l'application elle-même, au niveau d'une bibliothèque logicielle ou au niveau du système d'exploitation.

6.2.1 Checkpointing au niveau de l'application

Ce Checkpointing est réalisé par le développeur. C'est lui qui implémente un ensemble de procédures qui assurent les opérations de sauvegarde et de reprise qu'il ajoute au code de l'application. L'avantage de cette approche est que le développeur maitrise parfaitement le code de l'application, donc il va sauvegarder uniquement les données pertinentes et il peut décider du meilleur moment pour enclencher le processus de sauvegarde. Ainsi, les coûts relatifs au Checkpointing se voient réduits.

6.2.2 Checkpointing au niveau d'une bibliothèque logicielle

Le Checkpointing est implémenté à travers des fonctions fournies par des bibliothèques logicielles. De même, son implémentation est moins lourde qu'une implémentation au niveau de l'application. En effet, le développeur n'a qu'à insérer des appels de fonctions au niveau de son application sans se soucier de l'implémentation-même de ces fonctions.

Bien évidemment, il faut avoir accès au code source de l'application pour pouvoir intégrer le Checkpointing, ce qui peut poser problème si l'application est à code source fermé ou propriétaire. Du reste, cette approche est moins performante qu'une implémentation au niveau de l'application. En effet, le code fourni par une bibliothèque logicielle n'est pas forcément optimisé pour l'application considérée.

6.2.3 Checkpointing au niveau du système d'exploitation

Cette approche d'implémentation aussi appelée Checkpointing au niveau du noyau, consiste à implémenter le Checkpointing directement au niveau du système d'exploitation grâce à des modules qui sont chargés dans le noyau du système d'exploitation ou avec du matériel spécifique dédié au Checkpointing. Cette approche ne touche pas au code source de l'application, ce qui la rend plus transparente. Cependant, elle est moins efficace car les seules optimisations possibles sont faites au niveau du système de Checkpointing.

6.3 Travaux sur le Checkpointing

Dans cette section, nous présentons quelques-unes des approches de Checkpointing les plus proéminentes existantes dans l'état de l'art.

6.3.1 Remus

Remus (7) est une approche de Checkpointing au niveau du système d'exploitation. Elle se base sur la virtualisation pour extraire l'état du système d'exploitation dans une machine virtuelle primaire et le transférer vers une machine virtuelle secondaire. Une machine virtuelle ou *VM,* est une instance ou réplique de la machine physique et elle donne illusion aux utilisateurs d'avoir un accès direct à la machine physique ou (machine primaire). Les *VM* sont utilisées pour permettre le partage de ressources d'un matériel très couteux. La virtualisation est donc utilisée pour réduire le coût du matériel et d'améliorer la productivité globale. Remus utilise un mécanisme de réplication de machine virtuelle implémenté sur l'hyperviseur *Xen* qui fournit une haute disponibilité et permet la migration de VMs. La migration est une technique avec laquelle une machine virtuelle est déplacée vers un autre hôte physique avec seulement une légère interruption de service.

Les étapes suivantes sont entreprises par Remus lors du Checkpointing :

1. L'exécution de la VM est toujours en mode spéculatif, où les sorties de la VM sont temporisées jusqu'à ce qu'un Checkpoint correspondant au moment de génération de ces sorties soit validé.

2. Le processus de Checkpointing commence par la copie de la mémoire de la VM primaire vers un nouvel emplacement (mémoire secondaire) pendant qu'elle continue de s'exécuter à l'ancien emplacement.

3. À un moment donné, lorsque le processus de Checkpointing ne fait plus de progrès quant au transfert de la mémoire de la VM (c.à.d. lorsque l'application modifie autant de pages mémoire que le processus en sauvegarde), Remus suspend alors l'exécution de la *VM* primaire.

4. Ensuite, l'état du processeur ainsi que l'ensemble des pages mémoires (celles restantes) sont copiées dans un tampon mémoire et transférées de manière asynchrone.

5. Puis, Remus reprend l'exécution de la machine virtuelle primaire en mode spéculatif.

6. Lorsque toutes les données temporisées sont envoyées vers la mémoire secondaire, et qu'un message de confirmation est reçu, alors le Checkpoint est considéré comme valide. Ainsi, les sorties de la VM temporisées avant la quatrième étape sont transférées. Les sorties temporisées après la quatrième étape, seront transférées au prochain Checkpoint.

7. L'exécution se poursuit ainsi (toujours en mode spéculatif) jusqu'au moment de création du prochain point de contrôle.

L'exécution spéculative est un mécanisme ingénieux de Remus durant lequel toutes les sorties réseau doivent être mises en mémoire tampon jusqu'à ce que l'état correspondant à la génération de ces sorties soit synchronisé sur la sauvegarde. Lorsqu'une image complète et cohérente de la VM (c.à.d. un Checkpoint valide) a été reçue au niveau de la machine secondaire, la sortie réseau mise en mémoire tampon est libérée.

6.3.2 BLobCR

BLobCR (8) présente un dépôt dédié à la sauvegarde des points de contrôles, qui est capable de prendre des instantanés de disques virtuels attachés aux instances de machines virtuelles pour ensuite les enregistrer dans l'entrepôt. Chaque instance de VM est stockée comme une grande séquence d'octets (*BLOB*) qui est divisée automatiquement en morceaux et stocké d'une manière distribuée et élastique.

Contrairement à Remus, l'approche BlobCR ne suspend pas l'exécution de la machine virtuelle pour copier la mémoire de cette dernière, mais fait appel à un autre mécanisme qui est la copie sur écriture ou (*copie-on-write*). En effet, pour une ressource qui est sollicitée par plusieurs machines ou programmes en même temps et qui ont besoin d'une copie de cette ressource pour y lire. Toutes les machines peuvent partager cette ressource en commun mais si l'une d'elles veut accéder en écriture à cette ressource, alors celle-ci doit être dupliquée avant la modification afin que les changements ne soient pas visibles par les autres qui partagent cette ressource. Autre différence avec Remus, BlobCR ne garantit pas la cohérence des communications.

L'approche de BlobCR peut être mise à profit, soit au niveau de l'application en demandant directement des instantanés de disque virtuel, soit au niveau du système d'exploitation en effectuant une modification sur les protocoles de points de contrôle pour

qu'ils deviennent transparents. BlobCR est largement utilisé du fait de sa comptabilité avec plusieurs machines et sa transparence.

6.3.3 Diskless Checkpointing

L'approche nommée Diskless Checkpointing (9) est une approche de Checkpointing dans un système distribué qui propose de remplacer la sauvegarde sur support stable (disque dur) par la mémoire vive de certaines machines du système. Cette technique permet de supprimer le surcoût de l'écriture sur disque et permet de tolérer certain types de pannes très rapidement. Toutefois, elle ne permet pas de s'affranchir des pannes franches. Il est donc conseillé de la coupler avec un système de Checkpointing sur mémoire stable.

6.3.4 Libckpt

Libckpt (10) est une librairie de Checkpointing au niveau utilisateur. Le programmeur doit modifier certaines parties du code et recompiler son application pour l'utiliser.

Libckpt peut s'utiliser de deux manières différentes : le développeur peut déclencher le Checkpointing à certains points qu'il aura fixés dans l'application. Alternativement, le Checkpointing peut aussi avoir lieu de manière transparente grâce un signal périodique émis par d'autres applications (ou services) durant l'exécution de l'application.

6.3.5 Condor

Le projet Condor (11) développé à l'Université de *Wisconsin*, offre une bibliothèque de sauvegarde et de reprise. Cette bibliothèque de niveau utilisateur, porte également le nom de Condor, tout comme le projet. Condor est conçu pour la sauvegarde et la reprise des applications avec un seul thread d'exécution. Pour créer un Checkpoint, le développeur doit compiler son code source en le liant avec la bibliothèque de Checkpointing de Condor.

Cette bibliothèque installe tout d'abord un manipulateur de signaux pour la sauvegarde de l'image d'un processus de façon asynchrone, ensuite elle fournit au noyau un ensemble d'appels systèmes pour la restauration ou la migration de processus. La sauvegarde est engagée lorsque le processus reçoit le signal SIGSTP. A la réception de ce signal, un processus fils est créé pour enregistrer le contexte d'exécution du processus pendant que ce dernier continue son exécution. Condor peut être configuré pour effectuer des Checkpoints périodiques.

Cette approche peut fonctionner sur certaines architectures car elle n'est pas compatible avec toutes les machines.

6.4 Les défis du Checkpointing

La Tolérance aux Fautes est un domaine vaste, L'implémentation du Checkpointing dans tout type d'application relève plusieurs défis.

6.4.1 La transparence

La majorité des applications ne permettent pas l'accès à leur code source pour effectuer des modifications ou pour l'ajout d'éventuelles fonctionnalités. Cela oblige les développeurs à mettre en place des techniques du Checkpointing transparentes qui visent à être indépendantes de l'application. Cette approche permet aux développeurs de concevoir leurs applications et de mettre en œuvre un programme de point de contrôle qui fonctionne sans être adapté à une application spécifique et qui sera en dehors du code source. Cela implique que le Checkpointing soit implémenté au niveau du système d'exploitation ou au niveau du matériel (12).

6.4.2 La performance

La finalité du Checkpointing est de pouvoir sauvegarder correctement le contexte d'une application en vue de sa reprise. Ainsi l'implémentation du Checkpointing ne doit pas avoir un impact indésirable qui pénalise la fiabilité des applications en entrainant des surcoûts sur leur temps d'exécution, ne pas délivrer les réponses attendues ou rentrer en conflit avec les communications réseau existantes.

6.4.3 La sécurité et la confidentialité

Lors de la mise en œuvre du Checkpointing, il faut faire attention à ne pas entraver la sécurité des applications. Pour éviter le risque d'introduire des failles non intentionnelles de sécurité ou des problèmes de confidentialité, qui peuvent entrainer la perte de messages, la divulgation des données protégées d'une application, la réexécution de transactions ou leur perte.

6.4.4 L'architecture logicielle

La mise en œuvre du Checkpointing nécessite de prévoir une architecture logicielle spécifique tolérante aux pannes qui passe par la conception des composants qui prévoient l'arrivée de pannes, faire adapter d'autres composants pour supporter les mécanismes du Checkpointing ou qui décident de l'emplacement des points de sauvegarde.

6.4.5 Le choix de l'intervalle du Checkpointing

Cette question est souvent mise en avant et figure parmi les défis majeurs à surmonter lors de l'implémentation du Checkpointing. Car le choix de la fréquence ou de l'intervalle entre les moments de création de Checkpoints successifs est d'une importance déterminante sur le temps d'exécution des applications ainsi que sur leur performance. Ainsi, l'intervalle de Checkpointing peut affecter les applications soit positivement en diminuant les coûts d'exécution et en augmentant leur fiabilité, ou négativement en engendrant des coûts d'exécution importants et en pénalisant la fiabilité des applications.

7 Conclusion

Dans ce chapitre nous avons introduit la sûreté de fonctionnement des systèmes informatiques en présentant les notions qui lui sont associées à savoir : ses attributs, ses moyens et ses entraves. Nous avons parlé des contraintes auxquelles est exposée la sûreté de fonctionnement qui sont les pannes, en évoquant les caractéristiques et les catégories de ces dernières. Cela, dans le but de comprendre la nécessité d'avoir des techniques efficaces pour la Tolérance aux Fautes.

Nous avons présenté une technique de Tolérance aux Fautes efficace et largement utilisée qui est le Checkpointing, en citant quelques travaux relatifs au Checkpointing et ses différents niveaux d'implémentation. Nous avons vu que l'un des défis les plus important concernant le Checkpointing est celui de la sélection de l'intervalle de Checkpointing. C'est cette problématique que nous abordons dans notre mémoire. Mais avant de présenter notre contribution. Nous commençons par dresser un état de l'art décrivant les principaux travaux relatifs à la sélection de l'intervalle de Checkpointing.

Chapitre 2: Sélection de l'intervalle de Checkpointing

1 Introduction

La sélection de l'intervalle de Checkpointing ou de la fréquence de création de points de contrôle dans une application n'est pas une tâche facile mais bien complexe qui affecte grandement les performances et la fiabilité de l'application. Dans l'idéal, ce choix doit se faire de façon optimale de sorte à : réduire les coûts sur l'exécution des applications et en même temps offrir un niveau de fiabilité élevé pour celles-ci. Ainsi, lasélection de cet intervalle n'est pas un choix arbitraire résultant d'un placement (ou d'une création) aveugle de points de contrôle lors de l'exécution de l'application, mais c'est le fruit d'un travail laborieux qui constitue un domaine de recherche à part entière.

Dans ce chapitre, nous commençons d'abord par définir la problématique relative à la sélection de l'intervalle de Checkpointing.Ensuite, nous exposons les diverses approches de sélection de l'intervalle de Checkpointing existantes dans l'état de l'art et qui ont traité de cette problématique. Ces approches sont classées en deux catégories : en premier lieu nous présentons les approches basées sur une caractérisation complète des pannes en citant les travaux faits par les auteurs dans l'état de l'art. Ensuite, nous passons en revue les autres approches basées sur une caractérisation incomplète des pannes. Nous détaillons pour chaque contribution les points suivants : les caractéristiques de chaque contribution, le modèle du surcoût du mécanisme de sauvegarde, le modèle de panne, les propriétés de la stratégie d'ordonnancement des points de contrôle proposée et enfin les points communs et les divergences entre ces deux catégories dans les sections qui suivent. Enfin, nous terminons par présenter une méthodologie d'évaluation inspirée des travaux existants et qui peut être adoptée pour évaluer ces approches.

2 Problématique

Une des préoccupations centrale des développeurs lors de la mise en œuvre du Checkpointing est le choix de l'intervalle de Checkpointing, en d'autres termes la sélection de la fréquence de création des points de contrôle lors de l'exécution de l'application. Notons qu'une fréquence de Checkpointing excessive (ou un intervalle trop court) permet de garantir une haute fiabilité des applications mais, d'un autre côté, elle entraine aussi un coût significatif sur l'exécution de l'application rallongeant de ce fait le temps nécessaire pour qu'elle accomplisse une tâche donnée. D'un autre côté, une fréquence de Checkpointing trop faible (ou un intervalle trop grand) n'est également pas un choix judicieux. En effet,

malgré le fait que ce choix minimiserait les coûts du Checkpointing, il induirait aussi une dégradation importante de la fiabilité. Ainsi, pour terminer une tâche donnée, il faudrait allouer un long délai pour l'application car celle-ci devra s'affranchir des pannes en recommençant le traitement de la tâche à chaque panne. Par conséquent, un compromis doit être fait entre la fiabilité et le coût du Checkpointing sur l'exécution des applications pour minimiser le temps nécessaire à l'application pour compléter différentes tâches. Cette problématique est illustrée dans la figure suivante.

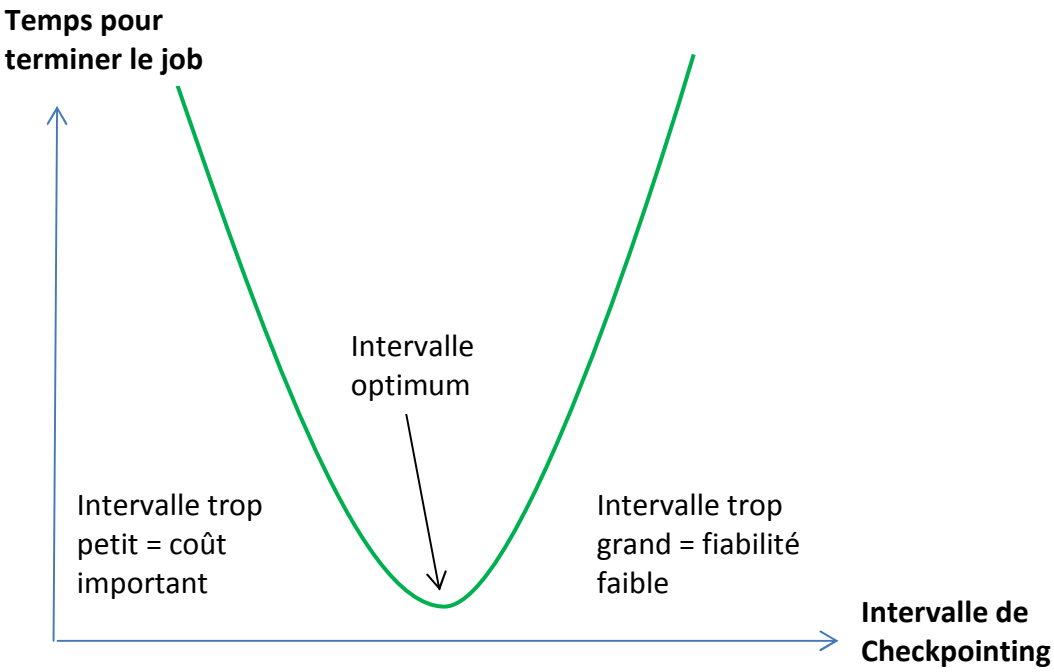

Figure 3: Problématique de sélection de l'intervalle de Checkpointing

Hormis ce compromis, une autre contrainte se pose lors de l'implémentation du Checkpointing dans l'application. En effet, tous les systèmes ne disposent pas souvent d'assez d''informationsconcernant les pannes (MTTF), ou bien les caractéristiques des techniques de prévision de pannes (rappel et précision) sont inconnues ou mal évaluées dans le système. A titre d'exemple, nous pouvons trouver des systèmes neufs qui n'ont pas eu d'antécédents de panne, ou encore des systèmes anciennement conçus dont nous avons mal évalué le MTTF (après vieillissement ou mise à jour logicielle ou matérielle du système). Pour ces systèmes dont la caractérisation de panne est incomplète, la sélection de l'intervalle du Checkpointing devient une tâche encore plus complexe. D'ou l'intérêt d'étudier d'autres approches existantes pour la sélection de l'intervalle du Checkpointing.

3 Approches basées sur une caractérisation complète des pannes

Le problème de sélection de l'intervalle de Checkpointing a profondément été étudié dans l'état de l'art. Nous détaillons dans cette section les principaux travaux existants dans l'état de l'art pour le calcul de l'intervalle de Checkpointing optimal lorsqu'il y a suffisamment d'informations à propos des pannes. Parmi ces informations, il y a le MTTF, le MTTR possiblement, ou encore, si le système dispose de moyens de prévision de pannes, alors nous connaissons la précision, le rappel et la fenêtre de prédiction de ces moyens.

3.1 Approche de Young

La première approche de calcul de l'intervalle de Checkpointing a été proposée par Young en 1974 (13), qui a réussi à donner une approximation de premier ordre de l'intervalle de point de contrôle optimal. Il a considéré que le surcoût dû au Checkpointing est constant et indépendant de la phase de calcul, et qu'il est négligeable par rapport au MTTF du système. De plus, l'occurrence des défaillances est supposée être dépendante et exponentiellement distribuée suivant un MTTF bien connu. Young considère un modèle d'application à horizon infini, c'est-à-dire qu'il y'a toujours un flux de travail à exécuter. La formule de Young détermine l'intervalle de contrôle Δ_{Young}, qui est basée sur deux quantités : le temps moyen avant la défaillance du système MTTF et le coût de création d'un point de contrôle C : $\Delta young = \sqrt{2 * C * MTTF}$

Pour modéliser le processus de panne, Young a opté pour le processus de Poisson en supposant que le régime de panne est permanent et en considérant les hypothèses suivantes :

- Les pannes suivent un processus de Poisson.
- Les pannes ne se produisent pas pendant les points de contrôle.
- Les pannes se produisent en moyenne à mi-chemin entre deux points de contrôle.
- Le *MTTF* et le coût sur l'exécution dû à la création d'un point de contrôle sont connus avec précision et à l'avance, et ne changent pas au cours du temps.
- Le coût du Checkpointing sur l'exécution de l'application C est négligeable par rapport au *MTTF* du système.
- Le temps nécessaire à une application pour redémarrer (sur une machine secondaire) après une panne est négligeable par rapport au MTTF du système.

Ainsi, l'utilisation de la formule de Young dans la pratique nécessite deux types d'informations: le coût C d'un point de contrôle et le MTTF du système. Nous ne retrouvons donc pas certaines informations telles que le délai total d'exécution d'une application, ou le temps nécessaire pour redémarrer une application après une panne.

3.2 Approche de Daly

Daly (14)s'est engagé à poursuivre le travail de base initié par Young. Dans la continuité du travail de ce dernier, l'auteur propose une approximation d'ordre supérieur de la valeur optimale de l'intervalle de temps qui sépare deux points de contrôle. Il garde les mêmes hypothèses faites par Young notamment en ce qui concerne la distribution des pannes en supposant que le régime de panne est transitoire, le modèle de sauvegarde lui aussi reste le même par rapport à celui de Young. Mais en revanche Daly considère que le temps de complétion d'une application (ou temps nécessaire pour finir les tâches d'une application) est à horizon fini. Ainsi, l'auteur s'intéresse à la minimisation de l'espérance du temps de complétion.

Daly présente des simulations pour montrer que la solution d'approximation proposée est bonne. Nous notons qu'à travers ces simulations l'auteur s'intéresse à la variation de l'espérance du temps de complétion en fonction de la variation du coût de création des points de contrôle et du taux de panne. Comme résultats ces simulations, montrent que l'approximation de premier ordre de l'intervalle donné par Young reste une très bonne approximation pour les configurations où le temps moyen des inter-arrivées des pannes est grand par rapport au coût de création d'un point de contrôle (c.-à-d. le MTTF du système est suffisamment grand en comparaison avec le coût de création d'un point de sauvegarde).

L'approche de Daly nécessite également deux informations : le MTTF du système et le coût du Checkpointing sur l'exécution de l'application. Ces travaux confirment également la non-pertinence du délai de redémarrage de l'application suite à une panne dans le calcul de l'intervalle de Checkpointing optimal.

3.3 Approche de Ling et al

Ling et al (15)introduisent une nouvelle technique appelée : approche de calcul variationnel qui permet de donner une approximation de l'intervalle optimal de Checkpointing. Cette méthode est basée sur le premier ordre du développement de Taylor pour la distribution des pannes sous l'hypothèse que le coût du point de contrôle est

constant et sûr (c.-à-d. n'est pas affecté par les pannes). Contrairement aux travaux de Young et Daly, 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. Comme fonction objectif, ils s'intéressent à la minimisation de l'espérance du temps de complétion, c'est-à-dire le temps total perdu avant la première panne. Par contre, pour modéliser l'inter-arrivée des pannes, ils utilisent une distribution arbitraire notée par *F(x)* en supposant que le régime de panne est permanent. Cependant, 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 des pannes. Deuxièmement, ils considèrent que cette durée de sauvegarde n'a aucun impact sur la probabilité des pannes 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 $\varphi(t)$ qui varie dans le temps.

Ainsi, cette approche nécessite d'avoir des informations sur le MTTF du système, ainsi qu'une description du coût du Checkpointing avec une fonction $\varphi(t)$.

3.4 Approches hybrides

Vu que le Checkpointing est une solution largement utilisée, il existe de nombreuses études sur la définition de son intervalle optimal. Cependant, la plupart de ces études sont axées sur la modélisation de systèmes qui adoptent uniquement des stratégies de Tolérance aux Fautes. En pratique, nous trouvons également des systèmes dits « hybrides » qui adoptent en plus de la Tolérance aux Fautes, un autre moyen de la sûreté de fonctionnement qui est la prévision des pannes.

Le calcul de l'intervalle de Checkpointing optimum dans ce contexte peut donc être adapté (en réduisant la fréquence de Checkpointing) pour prendre en compte cet aspect hybride afin d'améliorer encore plus les performances des applications sans heurter négativement leur fiabilité. Ces techniques de prévision de panne peuvent être utilisées en plus des points de contrôle pour déclencher des phases de sauvegarde supplémentaires lorsqu'une défaillance est prévue. Après la phase de sauvegarde, le travail est

immédiatement poursuivi sur la machine secondaire et aucun délai de reprise n'est nécessaire, une autre caractéristique importante de ce processus est que la fréquence des points de contrôle peut être réduite.

Les auteurs L. Zhu et al (16) modélisent les systèmes qui adoptent la méthode de Tolérance aux Pannes hybride, le but est de définir l'intervalle de point de contrôle optimal et de minimiser le temps de travail total. Dans (17), l'équipe de C. Wang a conçu une méthode de Tolérance aux Pannes hybrides qui se base sur un mécanisme efficace de migration en direct (en anglais : *live migration*) au niveau des processus. Ensuite, ils combinent le mécanisme avec la méthode du Checkpointing. Les évaluations entreprises indiquent que leur méthode peut garantir efficacement la fiabilité du système. Par ailleurs, des nouvelles recherches ont été entreprises pour la Tolérance aux Pannes hybrides en particulier pour prédire l'apparition des pannes avec des résultats justes.

Tout comme les approches précédentes (non hybrides), les approches décrites dans cette section nécessitent d'avoir des informations sur le MTTF du système, ainsi qu'une description du coût du Checkpointing. Toutefois, elles nécessitent aussi de connaître les caractéristiques du système de prévision des pannes, à savoir : sa précision, son rappel et la fenêtre de prédiction.

La précision *p* : est la fraction des vraies prédictions positives (Cvp) qui sont faites par le système par rapport à toutes les prédictions faites. Ces dernières incluent les prédictions justes (Cvp) est les prédictions erronées ou fausses (Cfp). Aussi, la précision peut être calculée ainsi :

$$\mathrm{P} = \frac{\mathrm{Cvp}}{(\mathrm{Cvp} + \mathrm{Cfp})}$$

Le rappel *r* : la valeur de rappel représente la capacité du système à prédire tous les futurs échecs. C'est la fraction des vraies prédictions positives sur le nombre total des échecs. Les prédictions d'échec manquées sont désignées comme faux négatifs (Cfn) et sont utilisées lors du calcul du rappel. Ce dernier est calculé ainsi :

$$\mathrm{R} = \frac{\mathrm{Cvp}}{(\mathrm{CVP} + \mathrm{Cfn})}$$

La fenêtre de prédiction : est l'intervalle de temps qui reste avant qu'une prédiction de l'échec arrive. C'est le temps entre le moment où la panne est prédite et le moment où elle se produit. Une bonne fenêtre de prédiction est assez grande pour qu'une phase de sauvegarde puisse être engagée et terminée avant que la panne ne se produise, si la fenêtre de prédiction est trop petite, alors la prédiction de défaillance est inutile car la phase de sauvegarde ne sera pas achevée.

4 Approches basées sur une caractérisation incomplète des pannes

Ces approches traitent le problème de sélection de l'intervalle du Checkpointing lorsque les informations concernant l'historique des pannes tel que le MTTF du système sont absentes ou imprécises.

Nous présentons quelques-unes de ces approches dans ce qui suit.

4.1 Okamura et al. Ozaki et al.

Les auteurs de ces deux travaux suivants (18) et (19), proposent deux méthodologies équivalentes pour calculer la stratégie de placement[1] des points de contrôle lorsque les informations sur les pannes sont incomplètes et lorsque la distribution des défaillances est inconnue. La stratégie proposée minimise le temps total perdu durant un cycle de panne. Cette méthodologie repose principalement sur un résultat similaire produit par Barlow et al (20).

Comme hypothèses de modélisation, ils considèrent un modèle d'application à horizon infini, un coût de Checkpointing constant pour les points de sauvegarde, une distribution générale pour modéliser l'inter-arrivées des pannes, ainsi 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 n'est pas permanant mais il est supposé transitoire. En effet en se basant sur la théorie de renouvellement ces auteurs 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.

[1] Nous entendons par placement : le placement au cours de (ou du temps alloué à) l'exécution de l'application. En d'autres termes, le placement des points de contrôle équivaut à déterminer l'intervalle ou la fréquence de Checkpointing.C'est une formulation souvent adoptée dans l'état de l'art.

4.2 Bouguerra et al.

Dans ce travail, l'auteur (21)s'intéresse à des implémentations des applications parallèles sur les plates-formes *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. Pour l'exécuter, ces processus coopèrent entre eux en échangeant des messages via les canaux de communication ou par une mémoire partagée.

Pour minimiser le temps d'exécution de ces applications, les auteurs se sont intéressés à la réduction du temps perdu à cause des pannes comme une mesure de fiabilité. Plus précisément, le temps perdu est dû à ces trois choses : tout d'abord aux frais généraux induit par les points de contrôle eux-mêmes ce qui augmente le temps de complétion des applications. D'autre part, la quantité de travail perdu après une panne entre le dernier point de contrôle et le temps de défaillance réelle. Enfin, comme les pannes ne sont pas nécessairement détectées immédiatement, il y a un autre coût à considérer qui correspond au temps écoulé depuis l'arrivée de la panne dans le système jusqu'au moment où il a été détecté par le système.

Les auteurs ont construit un système qui enregistre les traces de la disponibilité et la période d'indisponibilité de chaque ressource du système. Un modèle statistique est développé grâce aux traces recueillies à l'aide des techniques d'estimation bien connues, à savoir, Estimation du maximum de vraisemblance ou Expectation Maximisation. Puis, en utilisant le modèle de distribution des défaillances induites ils ont fourni une méthode numérique basée sur l'algorithme d'optimisation pour calculer l'intervalle optimale de Checkpointing. Il est bien utile de mentionner le fait que les auteurs ne fournissent aucune preuve technique sur l'existence ou l'unicité de la solution ou la complexité de calcul de l'optimisation numérique en ce qui concerne la distribution de panne.

Bouguerra et al ont calculé les intervalles optimaux et ils ont prouvé que la politique de point de contrôle optimale est périodique lorsque les frais généraux de ces points de contrôle sont constants, que ce soit pour des distributions de pannes exponentielles ou diverses. Mais leurs résultats reposent sur l'hypothèse que tous les processeurs sont rajeunis après chaque panne et après chaque point de contrôle.

5 Méthodologie d'évaluation

Pour évaluer l'efficacité d'une approche de sélection de l'intervalle du Checkpointing par rapport à d'autres existantes, nous faisons appel à des métriques. Dans notre cas, la métrique la plus importante à prendre en compte est celle du temps. Cette métrique est fondamentale mais elle n'est pas unique car qu'il existe bien d'autres mesures qui peuvent être prises en compte en plus du temps comme le coût sur les communications réseau induites par le Checkpointing, le temps de réponse ajouté aux applications etc. De ce fait nous pouvons évaluer une approche en étant meilleure qu'une autre si elle permette aux applications qui s'exécutent de terminer dans un temps minimal par rapport aux autres. Car le but à atteindre par le choix de l'intervalle de Checkpointing est la réduction du coût d'exécution d'une application, ou le temps nécessaire à un job pour terminer, nous désignons par un job « la charge de travail ».

Pour évaluer les approches proposées, nous listons un ensemble de pratiques et d'observations, aussi utilisées dans l'état de l'art :

- Évaluer le temps d'exécution ou de complétion pour un seul job (ou application) n'est pas fiable, il faut tester avec plusieurs jobs simultanément et faire la moyenne.
- Évaluer l'efficacité d'une approche avec une seule valeur de MTTF n'est pas une bonne pratique. Pour des résultats plus précis et corrects, il faut évaluer avec plusieurs valeurs de MTTF.
- Si l'approche dépend de plusieurs paramètres, il faut évaluer plusieurs combinaisons de valeurs et ne pas se contenter de quelques-unes seulement.
- Si deux approches sont comparées, nous pouvons calculer le gain d'une approche par rapport à une autre. Ceci permet de plus facilement distinguer l'approche qui a le plus grand apport.

6 Conclusion

Dans ce chapitre, nous avons souligné l'importance de la sélection de l'intervalle de Checkpointing qui peut influencer la fiabilité et le coût d'exécution des applications. Un bon choix d'intervalle nécessite d'établir un compromis d'optimisation entre la fiabilité et le coût.

Nous avons d'abord donné un aperçu des principaux travaux faits par les auteurs qui ont déjà traité cette problématique de sélection de l'intervalle. Ces travaux sont des approches

réparties en deux catégories. Premièrement, nous avons exposé les approches basées sur une caractérisation complète des pannes où le MTTF est bien connu. Ou bien le système dispose des moyens de prédiction de panne. Parmi les approches les plus éminentes nous avons vu l'approche de Young, celle de Daly, etc.

Deuxièmement, nous avons présenté les approches basées sur une caractérisation incomplète des pannes, faites parles auteurs comme Ozaki, Bouguerra, etc. Enfin nous avons proposé une méthodologie d'évaluation pour comparer et estimer ces approches en se basant sur la métrique du temps.

Dans le chapitre suivant, nous proposons une approche de sélection de l'intervalle de Checkpointing qui rentre dans le cadre des approches à caractérisation incomplète des pannes.

Chapitre 3: Approches proposées

1 Introduction

Comme vu précédemment, une mise en œuvre performante du Checkpointing nécessite un choix judicieux de l'intervalle de Checkpointing, et ceci afin de minimiser son coût en temps d'exécution et de maximiser sa fiabilité.

Dans ce chapitre, nous étudions la mise en œuvre d'une approche pour le calcul de l'intervalle de Checkpointing lorsque le MTTF du système est inconnu. Pour ce faire, nous proposons une approche de calcul de l'intervalle similaire à celle de Young, mais contrairement à cette dernière qui prend en compte un MTTF réel issue du système pour le calcul de l'intervalle, nous basons notre approche sur un MTTF arbitraire que nous désignons par MTTF à priori. Celui-ci est ajusté tout au long de l'exécution des applications afin de se rapprocher du MTTF réel du système (qui est inconnu du système).

Nous avons adopté l'organisation suivante dans ce chapitre. Nous commençons par définir la problématique du travail entrepris. Puis, nous donnons les motivations et les objectifs qui ont orienté notre contribution. Ensuite, nous présentons notre approche et nous décrivons son principe de fonctionnement. Enfin, nous terminons par une conclusion en donnant une vue générale de notre travail.

2 Problématique

Les approches existantes calculent l'intervalle de Checkpointing en employant la mesure du MTTF réel du système. Connaître celui-ci permet donc d'estimer le temps d'arrivée de la prochaine panne, ce qui facilite le choix de l'intervalle de Checkpointing.

Cependant, lorsque le MTTF du système n'est pas connu, il est difficile de fixer un intervalle de Checkpointing optimal. Il serait en effet inenvisageable de fixer un intervalle de Checkpointing arbitraire qui serait maintenu tout au long de l'exécution de l'application. Néanmoins, il est tout à fait possible de paramétrer un MTTF à priori arbitraire au début de l'application, et qui sera ensuite ajusté selon la fréquence des pannes. La problématique qui se pose alors est de savoir comment ajuster cet intervalle, que ce soit au niveau des formules à utiliser ou l'architecture du système.

3 Objectifs

Nous nous sommes fixés plusieurs objectifs quant à l'approche proposée. Ces objectifs sont résumés dans ce qui suit :

- L'approche doit pouvoir opérer sans connaître au préalable le MTTF du système.
- L'approche doit minimiser l'impact des pannes, en d'autres termes maximiser la fiabilité du système.
- L'approche doit minimiser le coût du Checkpointing pendant l'exécution de l'application.
- L'approche doit pouvoir s'intégrer facilement dans un système de Checkpointing existant.

4 Approches proposées

Dans les travaux présentés précédemment dans le deuxième chapitre, nous avons constaté que la sélection de l'intervalle de Checkpointing était plus simple à réaliser quand deux paramètres du système sont préalablement connus et qui sont : le coût du Checkpointing et le MTTF réel du système. Ces deux champs entrent dans la formule de Young destinée au calcul de cet intervalle. Une importance particulière a été attribuée au MTTF qui est le seul paramètre permettant de présager la survenue d'une panne dans le système, et sans ce dernier, nous perdons le seul repère indiquant le moment propice pour effectuer un point de contrôle de l'application.

Les approches que nous proposons pour le calcul de l'intervalle du Checkpointing entrent dans le cadre des approches à caractérisation incomplète de panne, par le biais de cette contribution, nous avons essayé de résoudre la problématique de sélection de l'intervalle de Checkpointing lorsque le MTTF réel du système est inconnu.

4.1 Nos approches

Comme principale contribution de ce mémoire, nous avons proposé deux approches pour le calcul de l'intervalle de Checkpointing lorsque le MTTF du système n'est pas connu.

Afin d'implémenter ces approches, nous commençons par déclarer une valeur arbitraire du MTTF que nous avons nommé le $MTTF_{\text{à priori}}$, celui-ci remplacera le MTTF réel du système

qui est inconnu dans notre cas, cette valeur ayant été donnée aléatoirement, elle n'est pas fixe et peut changer à plusieurs reprises dans le système selon la fréquence des pannes.

Les approches proposées contiennent des formules permettant d'ajuster ce $MTTF_{à\ priori}$ tout au long de l'exécution de l'application et cela dans le but d'approcher le MTTF réel et ainsi d'avoir des intervalles de Checkpointing optimaux.

Dans cette approche, nous distinguons deux cas :

Le premier étant celui où la panne survient avant le $MTTF_{à\ priori,}$ il faut donc ajuster celui-ci en tenant compte du TTF qui est le temps écoulé entre la panne actuelle et la panne précédente.

Le deuxième est celui où le temps d'exécution dépasse le $MTTF_{à\ priori}$ sans connaître de pannes, un réajustement est là aussi nécessaire.

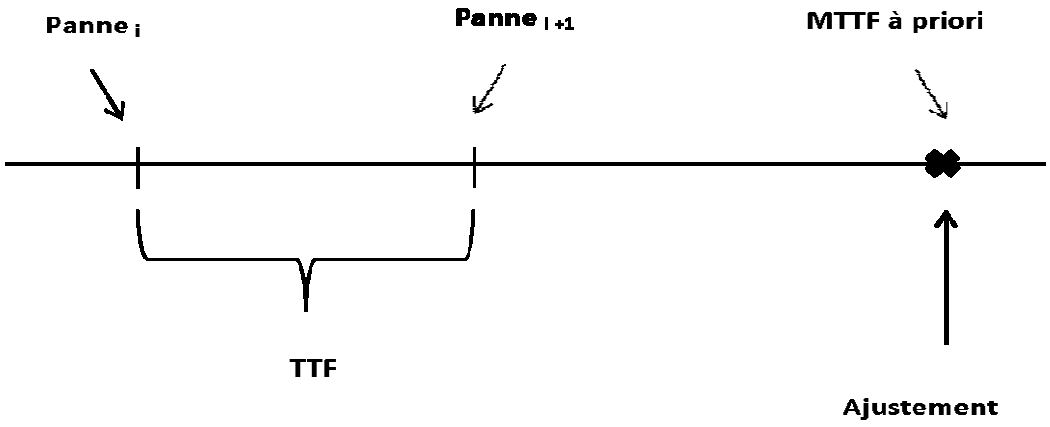

Figure 4 : Cas où la panne survient avant le MTTF à priori

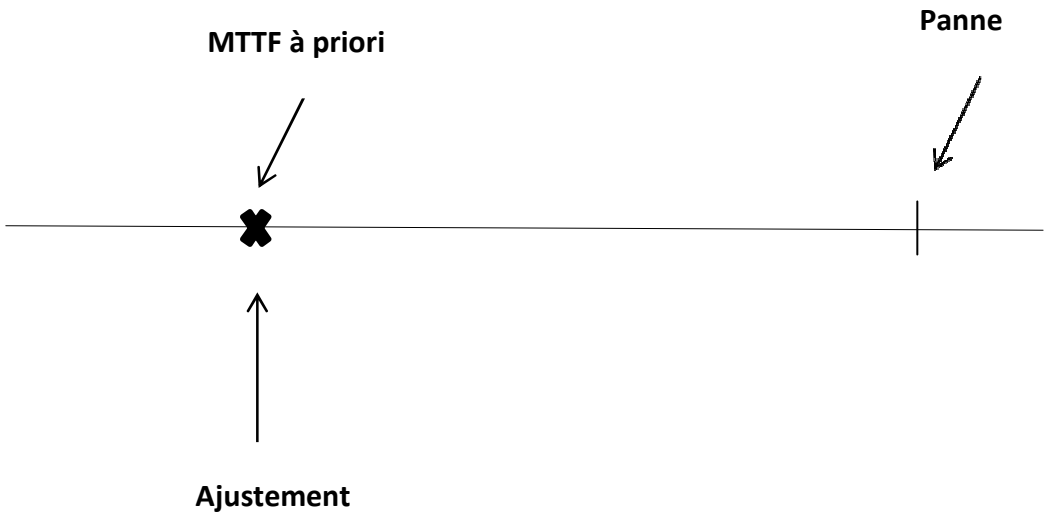

Figure 5 : Cas où la panne survient après le MTTF à priori

Nous proposons deux approches pour le calcul du MTTF à priori, l'une est une méthode additive et l'autre est multiplicative.

- Dans le premier cas le réajustement du MTTF à priori est réalisé après chaque panne selon l'une des formules suivantes:

  - Une formule additive :

$$\mathrm{MTTF}_{\mathrm{a\_priori}} = \mathrm{MTTF}_{\mathrm{à\ priori}} + \eta * (\mathrm{TTF} - \mathrm{MTTF}_{\mathrm{à\ priori}})$$

  - Une formule multiplicative

$$\mathrm{MTTF}_{\mathrm{à\ priori}} = \mathrm{MTTF}_{\mathrm{à\ priori}} * \alpha^{(\mathrm{TTF}-\mathrm{MTTF}_{\mathrm{a_{priori}}})/\mathrm{MTTF}_{\mathrm{a_{priori}}}}$$

Avec :

TTF : temps écoulé entre la panne actuelle et la panne précédente. Ce TTF est mis à jour à chaque fois qu'il y'a une nouvelle panne dans le système.

η: un taux d'ajustement pour la méthode additive compris dans l'intervalle] 0, 1]

α : un taux d'ajustement pour la méthode multiplicative, strictement supérieur à 1.

- Dans le deuxième cas où la panne n'est pas survenue dans le système après écoulement du $MTTF_{\text{à priori}}$, celui-ci est réajusté selon l'une des deux formules suivantes :

Pour la méthode additive :

$$\text{Ajuster } \mathrm{MTTF}_{\text{à priori}} = (\mathrm{MTTF}_{\text{à priori}} + \mathrm{n} * \mathrm{MTTF}_{\text{à priori}})$$

Pour la méthode multiplicative :

$$\text{Ajuster } \mathrm{MTTF}_{\text{à priori}} = \mathrm{MTTF}_{\text{à priori}} * \mathrm{a}$$

A chaque fois qu'une nouvelle valeur du $MTTF_{\text{à priori}}$ est obtenue à l'aide de l'une de ces deux approches, la fréquence du Checkpointing est recalculée selon la formule suivante, inspirée du modèle de Young pour le choix d'un intervalle de Checkpointing optimal :

$$d = \sqrt{2 * co\hat{u}t * MTTF_{\text{à } priori}}$$

5 Conclusion

Dans ce chapitre, nous avons exposé le problème du calcul d'un intervalle de Checkpointing quand le MTTF réel du système est inconnu. Nous avons proposé deux approches et avons cité l'objectif visé par celles-ci qui est le calcul de l'intervalle optimal de Checkpointing, ce dernier doit minimiser le coût du Checkpointing et augmenter la fiabilité du système.

Nous avons ensuite présenté les formules à implémenter pour l'ajustement du $MTTF_{\text{à priori}}$ qui sera utilisé dans une approche s'inspirant du modèle de Young, et qui est adaptée aux environnements à caractérisation incomplète de pannes, ce MTTF est recalculé grâce à deux méthodes, l'une additive et l'autre multiplicative soit à la survenue d'une panne, soit au dépassement du MTTF à priori actuel.

Dans le chapitre qui suit nous allons tester les deux approches et évaluer leurs performances, ceci en présentant les valeurs d'entrée et les résultats obtenus à la fin de la simulation.

Chapitre 4: Implémentation et expérimentations

1 Introduction

Dans ce chapitre, nous présentons l'implémentation réalisée pour simuler nos approches pour le calcul de l'intervalle du Checkpointing sans connaissance du MTTF réel du système. Le MTTF réel est alors approché grâce à deux formules l'une additive et l'autre multiplicative. Chacune des deux approches sont implémentées dans un simulateur existant. De plus, nous implémentons également l'approche de Young (avec connaissance du MTTF réel, et sans sa connaissance) et aussi une approche sans Checkpointing qui est pratique uniquement lorsque le MTTF est trop grand et qu'il n'y a pas de pannes dans le système.

L'organisation de ce chapitre est comme suit. Nous commençons par présenter l'environnement de simulation utilisé. Ensuite, nous présentons notre implémentation des différentes approches de Checkpointing. Après quoi, nous présentons les résultats de simulation. Enfin, nous terminons avec une conclusion quant à l'efficacité et la pertinence de ces approches.

2 Environnement de simulation

2.1 Présentation de quelque simulateurs

Il existe un nombre importants d'outils de simulation pour les systèmes distribués. Les plus connus sont GridSim, CloudSim, Simgrid. Dans ce qui suit, nous présentons quelques-uns de ces simulateurs.

2.1.1 CloudSim

C'est un cadre de modélisation et de simulation d'infrastructures et de services de Cloud Computing. Initialement conçu au laboratoire Cloud Computing and Distributed Systems (CLOUDS) à l'Université de Melbourne en Australie, CloudSim est devenu l'un des plus populaires simulateurs de Cloud open source dans la recherche et le monde universitaire. CloudSim est écrit en Java.

2.1.2 GridSim

C'est une boîte à outils qui fournit des fonctionnalités de base pour la simulation d'applications distribuées dans des environnements distribués hétérogènes. L'objectif spécifique du projet est de faciliter la recherche dans le domaine des systèmes à grande échelle parallèles et distribués.

2.1.3 SimGrid

C'est un instrument scientifique permettant d'étudier le comportement de systèmes distribués à grande échelle tels que les systèmes Grids, Clouds, HPC ou P2P. Il peut être utilisé pour évaluer des heuristiques (de routage, de distribution de charge, etc.), des applications prototypes ou même pour évaluer des applications MPI existantes. Tout cela en tant que logiciel libre.

2.1.4 ICanCloud

C'est une plate-forme de simulation destinée à modéliser et à simuler des systèmes Clouds. Le principal objectif d'ICanCloud est de prévoir des compromis entre le coût et les performances d'un ensemble donné d'applications exécutées sur un matériel spécifique, puis de fournir aux utilisateurs des informations utiles sur ces coûts.

2.2 Présentation du simulateur utilisé

ACS (Advanced Cloud Simulator) (22) est un simulateur à événements discrets qui permet de simuler différents aspects du Cloud Computing. Son architecture se compose de sept couches de composants comme il est montré dans la Figure 6. Pour le coté implémentation, ACS se compose d'environ 400 classes et 55 mille lignes de code, le tout en Java.

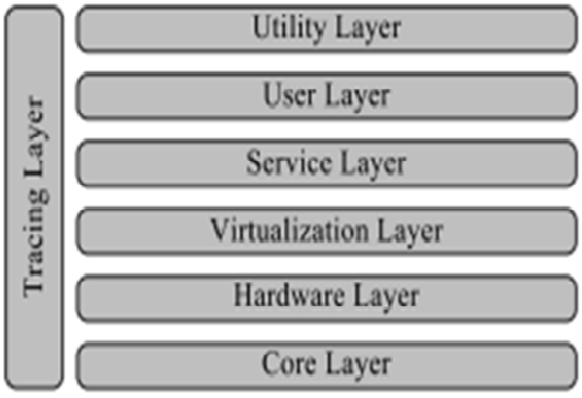

Figure 6: L'architecture en couches d'ACS

2.2.1 La couche noyau

Cette couche contient toutes les fonctionnalités de base du simulateur. Il définit les composants élémentaires du simulateur (entités, événements, etc.) et le code sous-jacent pour exécuter une simulation et extraire des résultats.

Ses principaux composants sont représentés dans la Figure 7 :

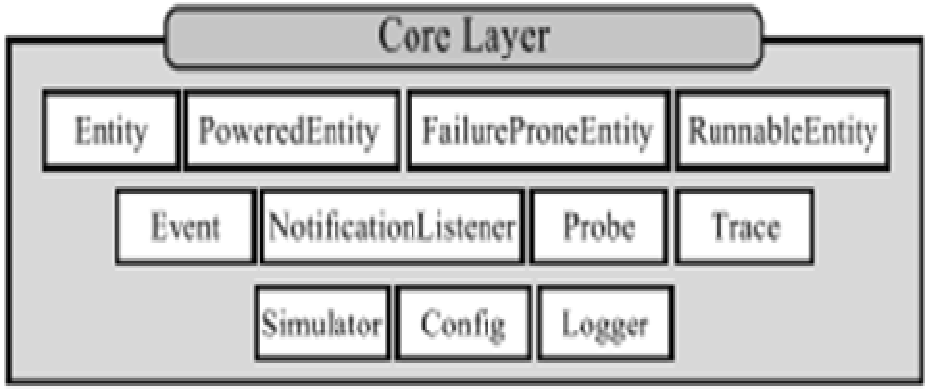

Figure 7: Architecture de la couche noyau d'ACS

- **Event**: Les événements sont définis par un code d'exécution et peuvent être planifiés à un moment de la simulation et être annulés si nécessaire. Il existe deux types d'événements dans ACS : des événements normaux qui sont toujours exécutés et des événements dont on peut se dispenser (*DispensableEvent).* Ces événements, dits dispensables, sont traités de la même manière que tous les autres événements du simulateur avec une petite différence: si seuls des événements dispensables restent dans la file d'attente du simulateur, la simulation se termine. Les événements dispensables ne sont exécutés par le simulateur que s'il y a des événements normaux de planifiés après eux. Ils sont utilisés par exemple pour générer des pannes, ou encore pour planifier la création d'un Checkpoint. En effet, une panne qui est planifiée à un moment donné, n'a d'intérêt que si la simulation atteint ce moment. De même, la création d'un Checkpoint pour une machine virtuelle par exemple n'a pas d'intérêt si l'exécution de la machine virtuelle est terminée.
- **Entity :** Les entités sont organisées de manière parent-enfant. Chaque entité a un parent et peut avoir une ou plusieurs entités enfants. Par exemple, une entité représentant un ordinateur peut être un parent de tous ses composants (processeurs, disques de stockage, etc.). Le simulateur est l'entité ancêtre de toutes les autres entités. C'est aussi la seule entité qui n'a pas de parent.

  La couche centrale d'ACS définit pour sa partie trois entités:

- **FailureProneEntity** : qui est l'entité de base pour toutes les entités qui peuvent échouer ou tomber en panne à un moment donné de la simulation. Par exemple, un disque de stockage peut tomber en panne à un moment donné dans la simulation.
- **PoweredEntity :** qui est l'entité de base pour toutes les entités qui peuvent être sous tension ou mise hors tension lors de la simulation. Par exemple, une machine peut être allumée et éteinte à certains moments de la simulation.
- **RunnableEntity** : qui est l'entité de base pour toutes les entités qui peuvent être mise en état d'exécution pour un certain délai ou suspendue durant la simulation. Par exemple, un job ou une VM sont des RunnableEntity.
- **Simulator :** est l'ancêtre de toutes les autres entités. C'est une entité spéciale qui n'a pas de parent. Elle définit des méthodes pour planifier des événements et les traiter en fonction de leur moment de traitement et de l'ordre dans lequel ils ont été ajoutés (exemple : si deux événements sont planifiés pour le même moment).
- **NotificationListener :** les notifications sont un mécanisme utilisé par ACS pour faire communiquer les entités entre elles. Par exemple, une notification est générée lorsqu'une machine tombe en panne. Une VM peut alors « écouter » et attendre de recevoir cette notification afin d'arrêter son exécution lorsque la panne survient. De même, d'autres entités qui sont intéressées par cette notification peuvent aussi l'écouter. Ainsi, les notifications permettent de simplifier la communication entre les entités.
- **Config :** c'est un composant associé à chaque entité dans le simulateur. La classe de configuration offre un ensemble de méthodes pour charger et stocker les valeurs de configuration. Cette dernière est identifiée par un nom et une valeur. Dans sa version actuelle, ACS prend en charge différents types de valeur (chaînes, booléens, nombres, définitions de classes, etc.).
- **Probe :** Ce composant définit une sonde qui peut contenir à n'importe quel moment une valeur représentative dans la simulation. La valeur de cette sonde change donc au cours de la simulation.
- **Trace :** Ce composant permet de contenir une trace des changements de la valeur d'une sonde au cours de la simulation. Contrairement au composant

Probe qui contient une valeur ponctuelle relative à un moment donné dans la simulation, ce composant permet donc de connaitre à la fin de la simulation le changement de la valeur d'une sonde au cours du temps.

2.2.2 La couche matérielle

La couche matérielle définit plusieurs entités liées à la simulation des composants physiques. Il comprend des entités pour la mise en réseau, la mémoire, le stockage et la simulation informatique.

Son architecture est illustrée dans la Figure 8 :

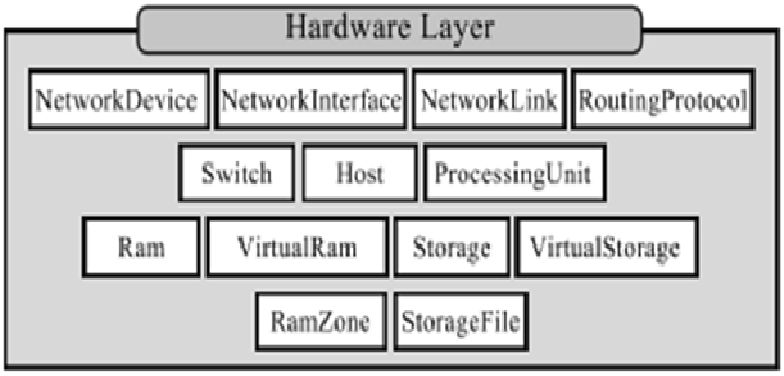

Figure 8 : Architecture de la couche matérielle d'ACS

Ses entités les plus importantes sont:

- L'entité **NetworkInterface** ou interfaces réseau qui peuvent être connectées avec d'autres interfaces grâce à un lien réseau.
- Le **NetworkDevice** définit un dispositif qui peut avoir une ou plusieurs interfaces réseau et qui peut communiquer via le réseau.
- Le **NetworkLink** ou lien réseau sert à connecter deux interfaces réseau.
- L'entité **Switch** définit un commutateur réseau qui est un périphérique réseau capable d'acheminer des données vers d'autres périphériques.
- L'entité **Host** ou hôte définit un périphérique réseau spécial pouvant comporter des disques de stockage, des unités de traitement, de la mémoire Ram et pouvant contenir une ou plusieurs machines virtuelles.
- L'entité **ProcessingUnit** qui définit une entité de calcul.
- L'entité **Ram** définit une mémoire vive.

- L'entité **Storage** définit une unité de stockage de données.

2.2.3 La couche de virtualisation

La couche de virtualisation contient les composants nécessaires pour simuler la virtualisation.

Ses principaux composants sont représentés dans la Figure 9 :

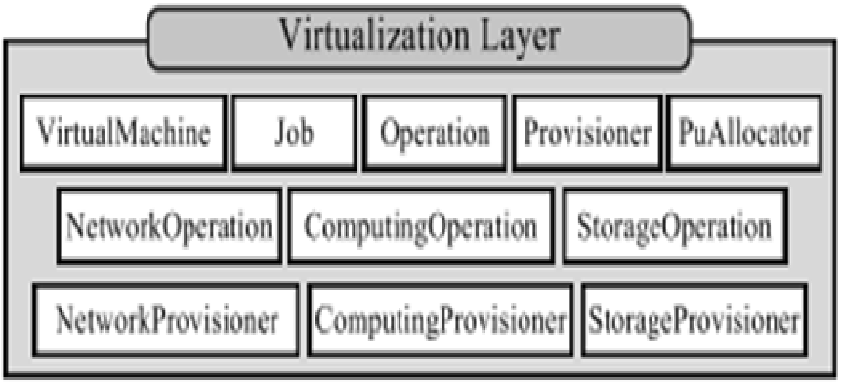

Figure 9: Architecture de la couche virtualisation d'ACS

Les entités les plus notables sont:

- L'entité **VirtualMachine** définit une VM qui utilise les ressources de son ordinateur parent pour exécuter un ou plusieurs jobs (charges de travail).
- L'entité **Job** qui possède et exécute plusieurs opérations (notées ***Operation***). Une opération est définit par une longueur et un état d'exécution. Il existe trois types d'opérations : les opérations réseau, les opérations de calcul et les opérations de stockage (c. à-d lecture / écriture sur disque).

2.2.4 La couche service

Cette couche contient les composants nécessaires à la simulation des services du Cloud Computing. Son architecture est illustrée dans la Figure 10.

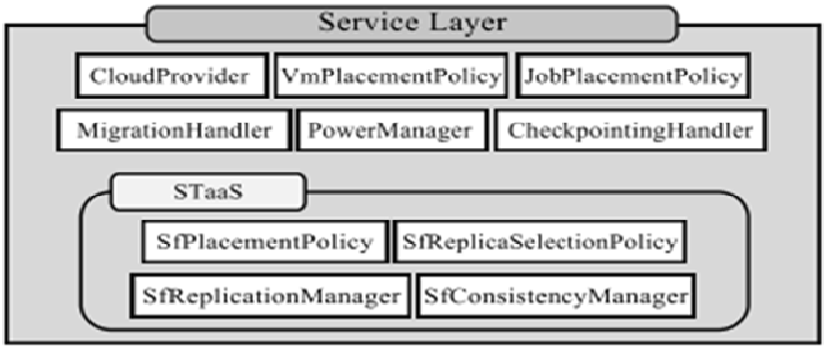

Figure 10: Architecture de la couche service d'ACS

Les entités les plus importantes sont :

- **CloudProvider** : définit un fournisseur Cloud en ce qui concerne son infrastructure ses utilisateurs et l'ensemble des services qu'il fournit.
- **VmPlacementPolicy** : c'est une entité qui traite les demandes des clients lors du déploiement de nouvelles machines virtuelles. Elle est chargée de trouver un hôte qui réponde aux besoins des clients.
- **JobPlacementPolicy** : se charge du placement des Jobs dans les VMs des clients.

2.2.5 La couche utilisateur

Cette couche contient deux éléments principaux :

- **User:** c'est une entité qui définit un client du Cloud. Elle contient des méthodes pour lister les machines virtuelles déployées, et les fichiers enregistrés par un utilisateur particulier.
- **ThinClient:** c'est une entité qui définit un client léger dont le rôle ressemble à celui des navigateurs web qui s'appuient sur les serveurs web pour fournir une fonctionnalité. Ainsi, ce composant ne peut, par exemple, pas effectuer d'opérations de calcul, mais il est utilisé par un utilisateur pour se connecter au Cloud et utiliser les ressources présentes dans le Cloud.

2.2.6 La couche utilitaire

Cette couche contient plusieurs classes d'utilitaires. En particulier, il offre un ensemble de classes et outils nécessaires à la configuration automatique et à la génération des scénarios de simulation.

2.2.7 La couche de traçage

Cette couche contient plusieurs définitions de sondes (l'implémentation de différents types de sondes). Les sondes implémentées dans cette couche permettent de mesurer (entre autre) la consommation d'énergie, les ressources utilisées (calcul, stockage et bande passante), et l'utilisation de la mémoire (Ram ou fichier sur disque). Tous ces paramètres sont accessibles à différents niveaux (job, VM, hôte, Cloud ou niveau de l'utilisateur). Par exemple, on peut ainsi mesurer la consommation d'énergie pour l'ensemble du Cloud, pour un hôte spécifique ou même pour un utilisateur.

2.3 Environnement de simulation

Les simulations ont été effectuées sur une station de travail avec un processeur 64 bits Intel® core ™ i3-2350M CPU à 2.30 Ghz, et 4.00 GO de RAM. Le système d'exploitation installé est Windows 7 64bits. L'environnement de développement utilisé est éclipse. Avec un langage de programmation java.

3 Implémentation

3.1 Description de notre implémentation

Nous avons utilisé le Simulateur ACS pour simuler nos approches et les comparer avec d'autres approches. Plus précisément, nous avons implémenté :

- notre approche additive, notée ci-après : OurIntervalAdditiveApproach ;
- notre approche multiplicative, notée ci-après : OurIntervalMultiplicativeApproach ;
- l'approche de Young avec connaissance du MTTF et sans connaissance du MTTF, notée : YoungCheckpoint ;
- une approche sans Checkpointing notée : NoCheckpoint.

Nous avons utilisé pour cela uniquement les composants de la couche noyau d'ACS. Ceci nous a permis d'avoir un plus grand contrôle sur la simulation en réalisant une implémentation de bas-niveau où nous avons directement manipulé (défini et planifié) les

événements de pannes, de Checkpointing et d'exécution des Jobs. Le code ainsi réalisé nous a permis de mesurer plusieurs aspects tel que la durée moyenne pour compléter les jobs, le temps ajouté (causé par les pannes et / ou le Checkpointing), son pourcentage et aussi le nombre de pannes.

Dans cette version nous avons utilisé la couche noyau du simulateur ACS. Nous avons utilisé en particulier le composant *Event* et *DispensableEvent*. Les événements sont traités par le simulateur après leur planification lorsque le temps de simulation est égal à l'heure (ou moment) programmée de l'événement. Les événements dispensables sont utilisés pour générer les pannes et pour initier la création d'un Checkpoint.

En outre, nous avons utilisé la classe *Simulator* pour créer une simulation et l'initialiser simulation, cela en : en créant l'objet simulateur, en créant des entités, en enregistrant des événements et en enregistrant éventuellement des écouteurs de notification (*EventListener*). Ensuite, nous lançons la simulation comme illustré dans la Figure 11 :

```
// Initialiser le Simulateur
Simulator simulator = new Simulator(new Config());
MainClass m = new MainClass(APPROCHE,
        // mttf_reel pour chaque job
        mttf * Simulator.HOUR,
        // mttf à priori pour chaque job
        ap_mttf * Simulator.HOUR,
        DELTA, RESTART, NB_JOBS, LENGTH_JOB,
        // randomSeed: il faut que cette valeur soit la meme pour tous
        // les tests pour pouvoir faire une comparaison cohérente
        0);
// démarre la simulation pour un delai max = taille d'un job * 100
simulator.start(LENGTH_JOB * 20);
```

Figure 11: Comment initialiser et démarrer la simulation

Nous avons aussi utilisé la classe de configuration (*Config*), qui offre un ensemble de méthodes pour charger et stocker les valeurs de configuration. Pour générer un nombre aléatoire de nombre de pannes, nous avons utilisé le générateur *Exponential*. Celui-ci permet de générer des nombres aléatoires selon une distribution exponentielle avec une certaine moyenne.

Les classes suivantes ont été écrites lors de notre implémentation :

- La classe principale *MainClass*, pour lancer les simulations avec différentes configurations (approche, MTTF différents, etc.).

- La classe *Job* : elle représente la structure du job. Elle contient des attributs comme la durée totale du job, la durée enregistrée dans le Checkpoint, les événements de fin de job et Checkpointing relatif au job, etc. Elle contient également un ensemble de méthodes pour démarrer le job, l'arrêter, etc. Ces méthodes, planifient ou annulent en amont différents événements de simulation

- La classe *CheckpointInterval* : c'est une interface qui définit une seule méthode *long getCheckpointInterval ()* qui nous retourne l'intervalle de Checkpointing.

- La classe *NoCheckpoint* : cette classe implémente l'interface *CheckpointInterval*, aucun Checkpoint n'est créé lorsque cette classe est utilisée. Pour cela elle retourne un intervalle de Checkpointing suffisamment pour qu'aucun Checkpoint ne soit créé.

- La classe *YoungCheckpoint* : elle implémente aussi l'interface *CheckpointInterval* et retourne un intervalle calculé selon la formule de Young.

- La classe *OurInterval* : est une classe abstraite qui implémente la classe *CheckpointingInterval*. Son constructeur prend en paramètre un MTTF à priori et évènement ajuster. À chaque panne on met à jour le MTTF A priori selon la méthode additive ou multiplicative, lorsqu'il n'y a pas de panne et qu'on dépasse le MTTF d'une trop grande valeur on met à jour le MTTF a priori.

- la classe *OurIntervalAdditive* : est une classe fille de la casse *OurInterval*. Elle calcule l'intervalle de Checkpointing en utilisant le MTTF à priori avec la méthode additive. Après chaque panne, ou lorsqu'il n'y a pas de panne pour une longue période, alors le MTTF à priori est modifié selon la méthode additive.

- La classe *OurIntervalMultiplicative* : une classe file de *OurInterval.* Elle calcule l'intervalle de Checkpointing en utilisant le MTTF à priori avec la méthode multiplicative. Après chaque panne, ou lorsqu'il n'y a pas de panne pour une longue période, alors le MTTF à priori est modifié selon la méthode multiplicative.

- La classe *Sortie* : une classe pour générer un fichier CSV (ouvrable avec Excel) pour visualiser les résultats de la simulation.

3.2 Les entrées de simulateur

Plusieurs simulations ont été réalisées avec différents paramètres en entrée. Pour chaque combinaison de valeurs d'entrée, plusieurs simulations ont été lancées.

Pour chaque simulation correspond un MTTF réel spécifique. Lors des simulations, nous avons utilisé différentes valeur de MTTF réel pour les effectuer les tests. Le MTTF réel est utilisé pour générer effectivement les pannes. Celui-ci n'est pas forcément connu par les approches utilisées. Dans notre cas, nous avons utilisé 28 valeurs pour MTTF réel. Aussi, nous avons fixés certains paramètres selon des valeurs empiriques trouvées dans l'état de l'art. Pour ce qui est de nos approches, nous avons utilisé plusieurs valeurs pour les paramètres a et n.

Voici donc les entrées du simulateur :

- MTTF réel : 28 valeurs : 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000, 100000 (en heures).
- MTTF à priori : 100000, 5000 et 500 (en heures).
- Nombres de jobs : 500.
- Durée initiale des jobs : 200, 500 et 1000 heures.
- Coût de création d'un Checkpoint : 5 minutes.
- Délai de détection de panne et de reprise : 5 minutes.
- Valeur de n pour l'approche additive : 0.8, 0.5 et 0.2.
- Valeur de a pour l'approche multiplicative : 1.5, 2 et 3.
- Approches : nos deux approches, l'approche de Young avec et sans connaissance du MTTF réel, et une approche sans Checkpointing.

Le MTTF réel est utilisé comme moyenne d'un générateur de nombre pour générer le moment effectif des pannes. Le générateur choisi est le générateur *Exponential.*

Notons enfin que l'aléatoire a été introduit pour ces valeurs, en les utilisant comme moyennes pour un générateur de nombres aléatoire suivant une distribution exponentielle. Ainsi par exemple, tous les jobs n'ont pas une durée initiale fixe de 200 h. Par contre, la moyenne des durées initiales de tous les jobs tend vers 200h (lorsque le nombre de jobs est suffisamment grand). Même chose pour le coût de création d'un Checkpoint et pour le délai de détection de panne et de reprise.

Pour s'assurer que les valeurs aléatoires soient les mêmes lorsque nous comparons deux approches, nous avons utilisé une même graine aléatoire pour toutes les simulations.

3.3 Les sorties de simulateur

Le simulateur donne en sortie différentes informations sur la simulation réalisée. Comme résultats de simulation nous avons :

- La durée initiale moyenne des jobs (en heures) calculée avec la formule suivante :

$$\mathrm{t_{initial\ moyen}} = \sum_{i=0}^{nbjobs-1} \frac{\mathrm{t_{initial}(i)}}{\mathrm{nbjobs}}$$

Où :

$\mathrm{t_{initial\ moyen}}$: est la durée initiale moyenne de tous les jobs.

$\mathrm{t_{initial}}$ (i) : la durée initiale du job i.

$nbjobs$: le nombre des jobs.

- La durée moyenne pour compléter les jobs (en heures) calculée avec la formule suivante :

$$\mathrm{t_{fin\ moyen}} = \sum_{i=0}^{nbjobs-1} \frac{\mathrm{t_{fin}(i)}}{\mathrm{nbjobs}}$$

Où :

$t_{\mathrm{fin\ \ moyen}}$: est la durée moyenne de complétion (terminaison) de tous les jobs.

$\mathrm{t_{fin}}(i)$: le temps de terminaison effectif du job i.

$nbjobs$: le nombre des jobs.

- Le temps ajouté (en heures) calculé avec la formule suivante :

$$t_{ajouté} = \mathrm{t_{initial\ moyen}} - \mathrm{t_{fin\ moyen}}$$

- Le temps ajouté (pourcentage) obtenu grâce à la formule suivante :

$$p = \frac{t_{ajouté} * 100}{\mathrm{t_{initial\ moyen}}}$$

- Nous avons aussi comme sortie du simulateur le nombre de pannes.

3.4 Résultats et discussion

Nous avons d'abord réalisé une simulation de l'approche sans Checkpoints (*NoCheckpoint*). Nous avons remarqué que cette approche est pratique uniquement lorsque le MTTF est trop grand et qu'il n'y a pas ou peu de pannes dans le système. Nous avons également simulé l'approche de Young pour calculer l'intervalle du Checkpointing en connaissance du MTTF réel du Système. Nous présentons dans ce qui suit le graphe illustré dans la Figure 12 qui montre le pourcentage % du temps ajoutés par l'approche de Young et celle sans Checkpointing sur le temps d'exécution initial :

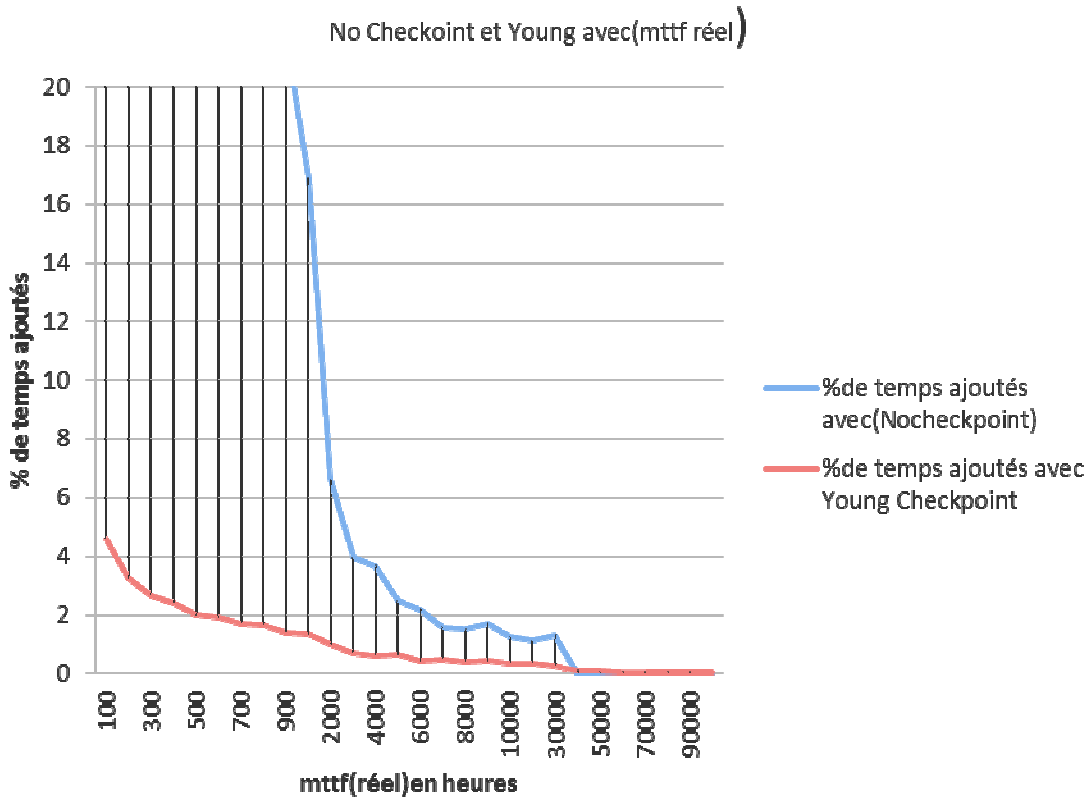

Figure 12: Pourcentage du temps ajouté en fonction de MTTF réel pour les approches de Young et sans Checkpointing

Nous constatons que l'approche sans Checkpointing est pratique uniquement lorsque MTTF réel est supérieur à 4000 heures.

Nous avons aussi testé not approche additive avec la combinaison de paramètre n=0.5, n=0.8 et n= 0.2 avec une valeur du $MTTF_{\text{à priori}}$=1000h. Les résultats sont décrits dans la Figure 13 :

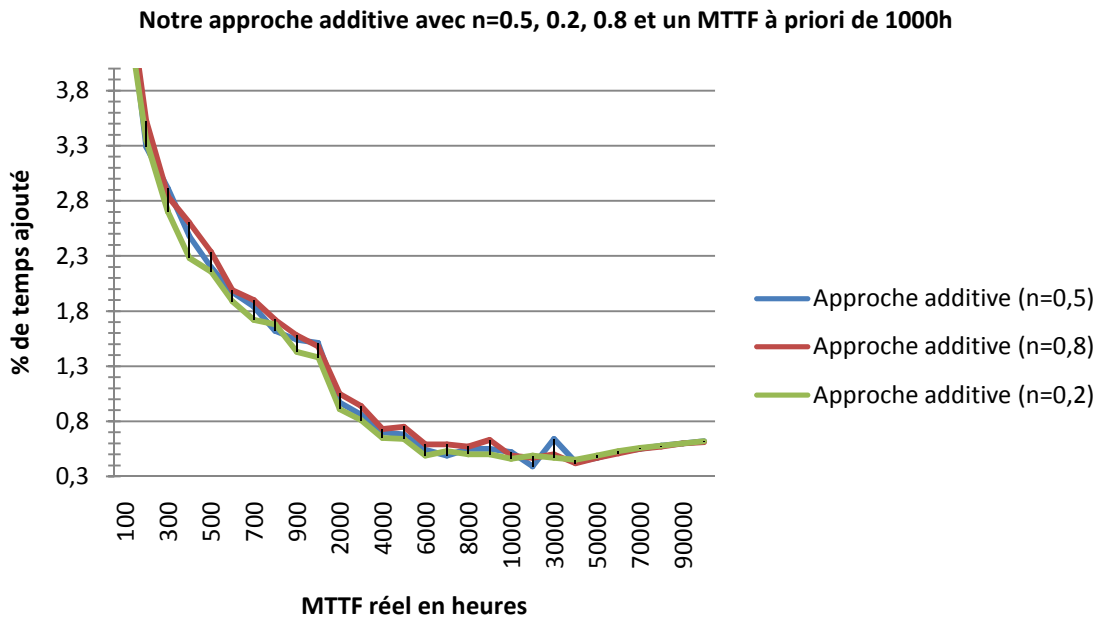

Figure 13: Pourcentage du temps ajoutés avec notre approche additive avec n=0.2, 0.5, 0.8 et un MTTF à priori de 1000h

Nous remarquons que pour l'approche additive le paramètre n n'influe pas beaucoup sur le temps ajouté en pourcentage.

Nous avons aussi testé notre approche multiplicative avec des combinaisons de paramètre a=1.5, a=2, a=3 et comme valeur de $MTTF_{\text{à priori}}$=1000h. Les résultats sont illustrés par la Figure 14 :

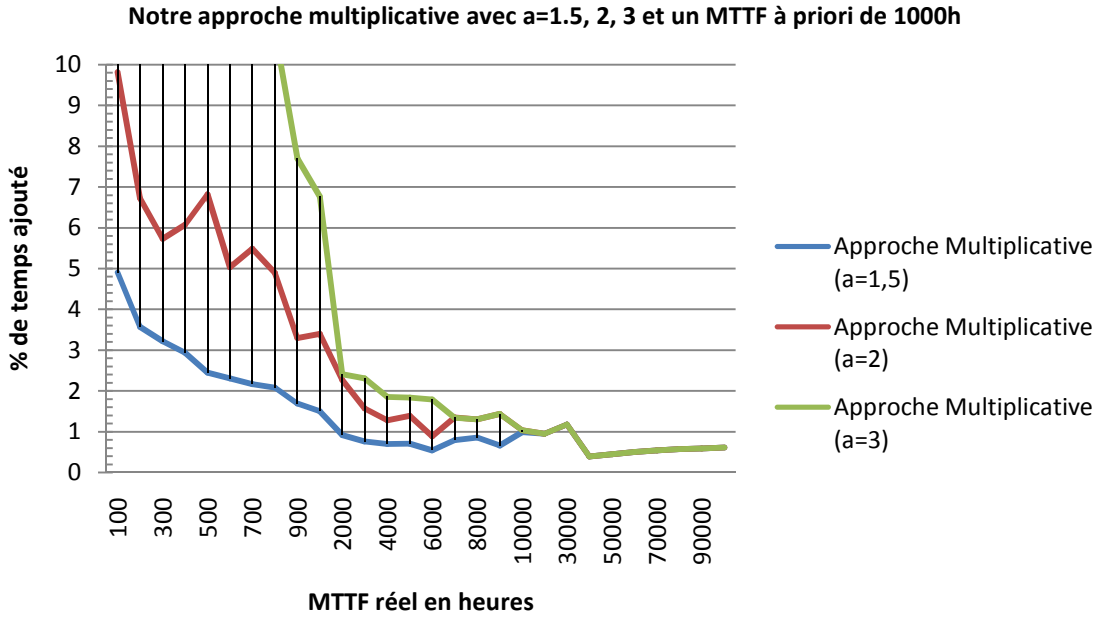

Figure 14: Pourcentage du temps ajoutés avec notre approche Multiplicative avec a=1.5, 2, 3 et un MTTF à priori de 1000h

Nous remarquons que contrairement au paramètre n dans l'approche additive, dans l'approche multiplicative le paramètre a influe sur les résultats. Lorsque la valeur du paramètre a augmente, le pourcentage du temps ajoutés augmente également. Nous constatons que le meilleur résultat est obtenu avec une valeur de a=1.5. Ainsi, dans les tests qui suivent nous fixons la valeur de a à 1.5 :

Nous avons également testé et comparé les approches de Young (avec et sans MTTF réel), et nos deux approches. Les résultats ci-dessous (Figure 15) sont réalisés avec un MTTF à priori de 100000 heures :

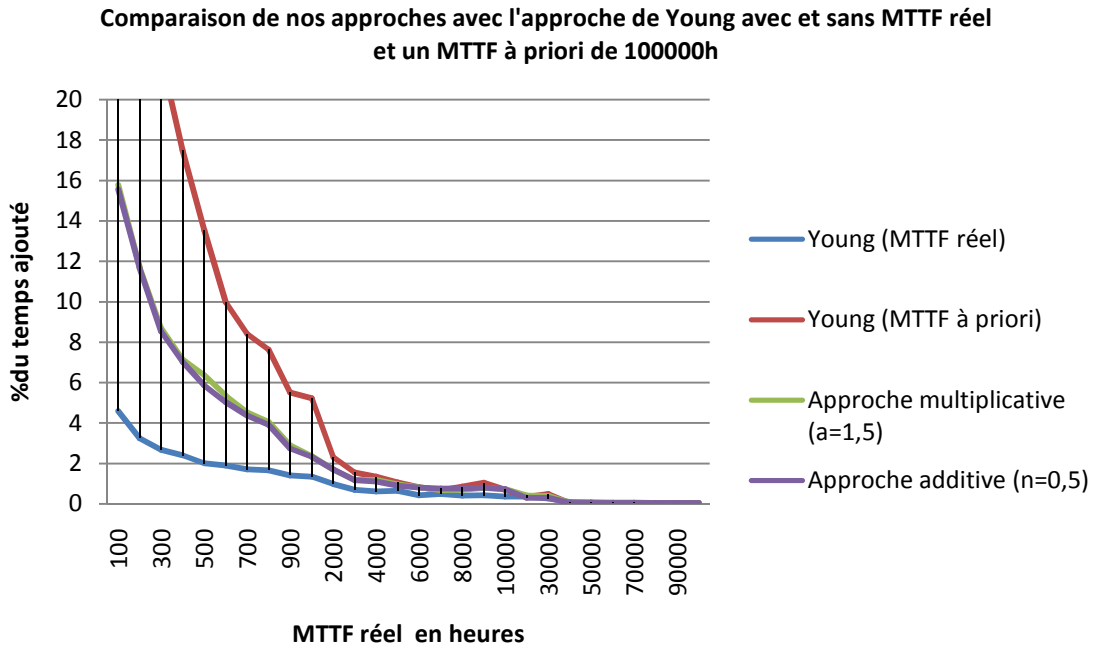

Figure 15: Pourcentage du temps ajoutés de nos approches et de celle de Young et un MTTF à priori de 100000h

Nous constatons que pour l'approche de Young avec le MTTF réel nous donne les meilleurs résultats (ce qui était prévisible). Nos approches font mieux que l'approche de Young sans connaissance du MTTF réel, avec un léger avantage de notre approche additive

L'évaluation suivante compare les mêmes approches que précédemment mais avec un MTTF à priori de 5000h. La Figure 16 montre les résultats obtenus.

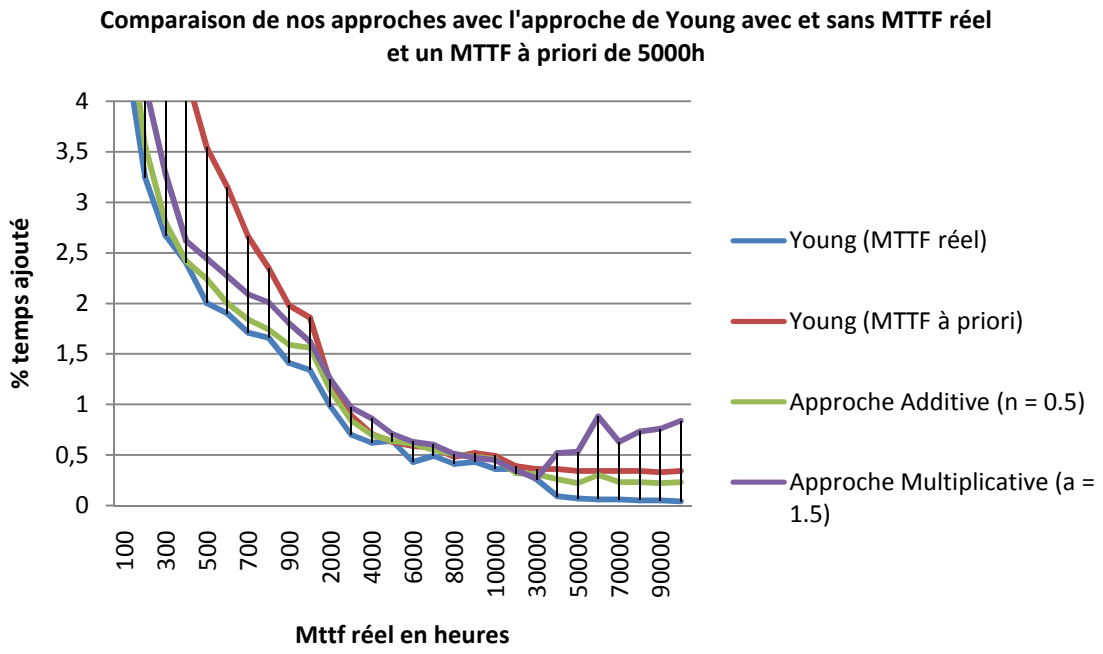

Figure 16: Pourcentage du temps ajoutés de nos approches et de celle de Young et un MTTF à priori de 5000h

Nous constatons que pour l'approche de Young avec le MTTF réel nous donne toujours les meilleurs résultats (ce qui est encore une fois prévisible). Nos approches font mieux que l'approche de Young sans connaissance du MTTF réel pour la plus grosse partie des simulations. En effet, au voisinage du MTTF réel = MTTF à priori = 5000h, l'ensemble des approches semblent donner les mêmes résultats. Lorsque le MTTF est grand, l'approche multiplicative ne semble pas aussi performante que l'approche additive.

L'évaluation suivante est réalisée pour les mêmes approches avec un MTTF à priori de 500 heures. Les résultats sont décrits dans la Figure 17 :

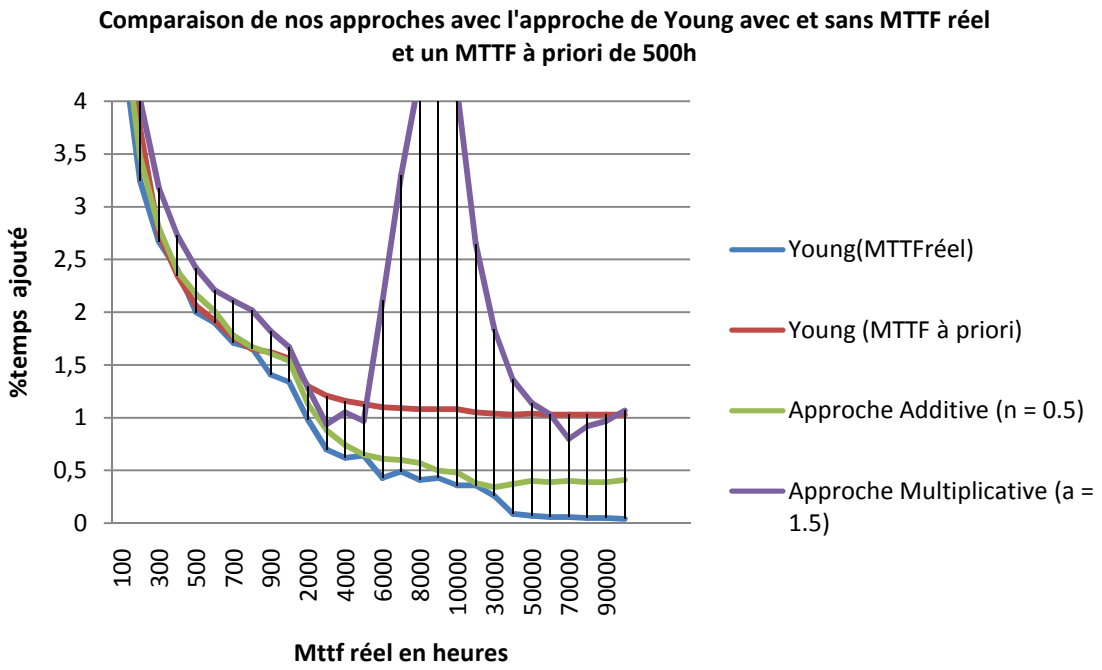

Figure 17: Pourcentage du temps ajoutés de nos approches et de celle de Young et un MTTF à priori de 500h

Nous constatons une nouvelle fois que pour l'approche de Young avec le MTTF réel nous donne les meilleurs résultats. Nos approches font mieux que l'approche de Young sans connaissance du MTTF réel jusqu'à un MTTF de 4000h. Après quoi notre approche additive continue de présenter de meilleurs résultats. Cependant, l'approche multiplicative ne semble plus ajuster correctement le MTTF et donne des résultats médiocres.

Pour les tests suivants, nous initialisons la longueur initiale d'un job à 200h au lieu de 1000h. Les résultats obtenus sont représenté dans les graphes suivants :

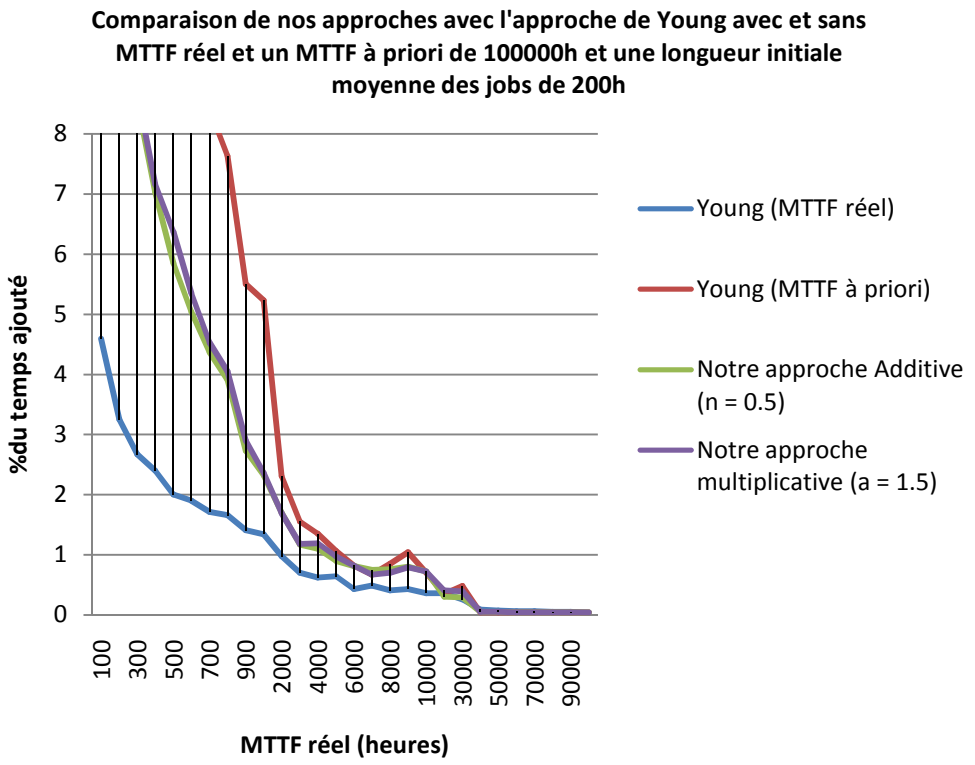

Figure 18: Pourcentage du temps ajoutés de nos approches et de celle de Young et un MTTF à priori de 100000h et une longueur initiale moyenne des jobs de 200h

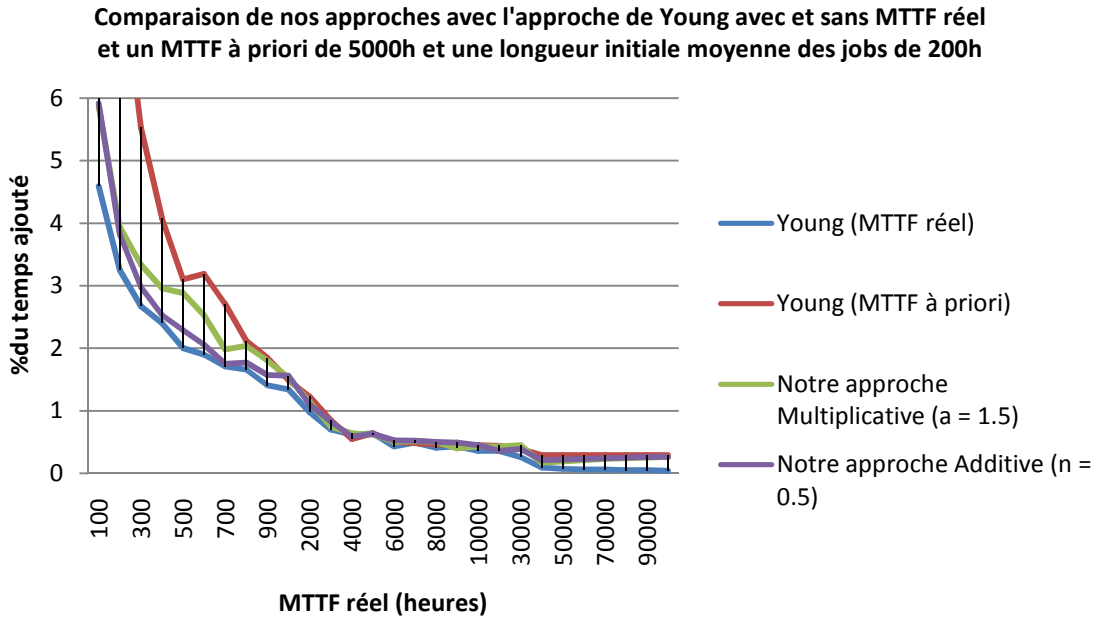

Figure 19: Pourcentage du temps ajoutés de nos approches et de celle de Young et un MTTF à priori de 5000h et une longueur initiale moyenne des jobs de 200h

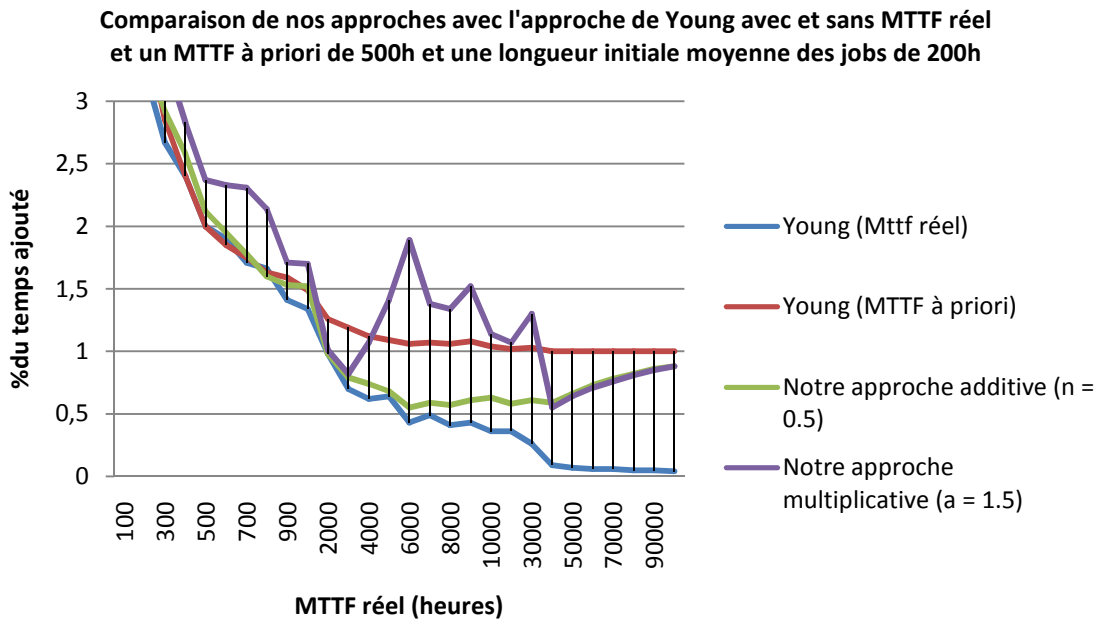

Figure 20: Pourcentage du temps ajoutés de nos approches et de celle de Young et un MTTF à priori de 500h et une longueur initiale moyenne des jobs de 200h

Nous constatons que pour une longueur initiale moyenne des jobs différente, notre approche additive reste la plus performante en comparaison des autres approches utilisant un MTTF à priori.

4 Conclusion

Dans ce chapitre, nous avons présenté notre implémentation des approches multiplicatives et additives en leur comparaison avec l'approche de Young et sans Checkpointing. Nous avons ainsi, présenté le simulateur utilisé (à savoir ACS) et détaillé notre implémentation. Nous avons réalisé en tout 756 simulations en utilisant différentes valeurs de MTTF et de paramètres d'entrée. Les résultats ont démontré l'intérêt des deux approches proposées, mais ils ont aussi démontré que ces approches peuvent avoir certaines faiblesses sous certaines circonstances.

Conclusion générale

1 Synthèse

L'ensemble des contributions de ce mémoire s'inscrivent dans le cadre de la Tolérance aux Fautes dans les systèmes informatiques. Ainsi, la première partie de ce mémoire est dédiée à la présentation des notions relatives à la sûreté de fonctionnement et à tous ce qui touche au domaine de la Tolérance aux Fautes en général. Nous avons présenté une technique de Tolérance aux Pannes bien particulière et largement utilisée qui est le Checkpointing. Toutefois, la mise en place de ce mécanisme de Tolérance aux Fautes est une tâche délicate qui soulève de nombreux défis scientifiques. Parmi ces défis, nous nous sommes intéressés au choix de l'intervalle du Checkpointing qui correspond aussi à la fréquence de création et de sauvegarde des points de contrôle dans une application. Étant donnée son importance, ce problème a reçu beaucoup d'attention et il a fait l'objet de nombreuses études que nous avons détaillées dans la seconde partie de ce mémoire.

Comme déjà souligné la deuxième partie du mémoire, est consacrée à la présentation de la problématique de sélection de l'intervalle du Checkpointing. Ce dernier doit pouvoir minimiser le coût du Checkpointing et maximiser la fiabilité des applications. Nous avons dressé un état de l'art décrivant les principales contributions faites par les auteurs à ce sujet en présentant deux types d'approches existantes pour le calcul de cet intervalle à savoir les approches à caractérisation incomplète des pannes et les approches à caractérisation complète des pannes.

Dans la troisième partie du mémoire, nous avons répondu à la problématique posée précédemment. Comme solution, nous avons proposé deux approches pour le calcul de l'intervalle du Checkpointing dans les environnements à caractérisation incomplète des pannes, plus précisément dans notre cas : lorsque le MTTF du système n'est pas connu. Pour ce faire, nous avons fixé un MTTF arbitraire désigné par « MTTF à priori ». Ensuite, cette valeur est ajustée pour s'approcher du MTTF réel avec deux approches : l'une additive et l'autre multiplicative.

La dernière partie du mémoire est consacrée à l'implémentation. Pour tester et évaluer l'apport de nos approches, nous avons procédé à une simulation avec un simulateur à évènement discret nommé ACS. Pour ce faire, nous avons lancé plusieurs simulations avec plusieurs combinaisons de paramètres d'entrée. Les résultats obtenus sont évalués avec des graphes comparatifs pour estimer et comparer les gains de nos deux approches avec

d'autres approches utilisées comme celle n'utilisant pas de Checkpointing et l'approche Young Checkpoint (avec et sans connaissance du MTTF réel). Nous avons ainsi vérifié l'apport de nos approches, plus particulièrement l'approche additive, qui permet de minimiser le temps d'exécution des jobs en présence de pannes et sans connaissance du MTTF réel.

2 Perspectives

Nous envisageons plusieurs suites à nos travaux :

- La première perspective serait d'améliorer la précision de nos deux approches proposées, pour proposer des formules plus rigoureuses et étudiées.
- Une deuxième perspective serait d'implémenter d'autres approches étudiées dans l'état de l'art et de proposer d'autres formules pour le calcul de l'intervalle du Checkpointing en plus de celle de Young.
- Une autre perspective consiste à implémenter les approches dans un système existant (non simulé) et les tester dans des conditions réelles.

Annexe : Résultats des simulations

1 Résultats sans Checkpointing

MTTF Réel (heures)	longueur initiale moyenne du job	temps ajouté (%)	nb pannes	jobs restants
100	1000	115,81	4669	66
200	1000	105,35	1255	16
300	1000	80,33	590	3
400	1000	60,24	399	2
500	1000	49,46	320	2
600	1000	40,2	238	1
700	1000	38,74	191	0
800	1000	27,92	145	0
900	1000	21,72	118	0
1000	1000	16,89	100	0
2000	1000	6,61	48	0
3000	1000	3,96	30	0
4000	1000	3,67	23	0
5000	1000	2,47	17	0
6000	1000	2,19	13	0
7000	1000	1,58	11	0
8000	1000	1,52	10	0
9000	1000	1,71	10	0
10000	1000	1,26	8	0
20000	1000	1,12	5	0
30000	1000	1,32	4	0
40000	1000	0	0	0
50000	1000	0	0	0
60000	1000	0	0	0
70000	1000	0	0	0
80000	1000	0	0	0
90000	1000	0	0	0
100000	1000	0	0	0

2 Résultats avec l'approche de Young et la connaissance du MTTF réel

MTTF Réel(heures)	longueur initiale moyenne du job	temps ajouté (%)	nb pannes	jobs restants
100	1000	4,59	943	0
200	1000	3,25	475	0
300	1000	2,67	336	0
400	1000	2,4	259	0
500	1000	2	196	0
600	1000	1,9	159	0
700	1000	1,71	140	0
800	1000	1,66	119	0
900	1000	1,41	99	0
1000	1000	1,34	89	0
2000	1000	0,98	45	0
3000	1000	0,7	29	0
4000	1000	0,62	20	0
5000	1000	0,64	16	0
6000	1000	0,43	12	0
7000	1000	0,49	11	0
8000	1000	0,41	10	0
9000	1000	0,43	10	0
10000	1000	0,36	8	0
20000	1000	0,36	5	0
30000	1000	0,26	4	0
40000	1000	0,09	0	0
50000	1000	0,07	0	0
60000	1000	0,06	0	0
70000	1000	0,06	0	0
80000	1000	0,05	0	0
90000	1000	0,05	0	0
100000	1000	0,04	0	0
100	200	4,59	943	0
200	200	3,25	475	0
300	200	2,67	336	0
400	200	2,4	259	0
500	200	2	196	0
600	200	1,9	159	0
700	200	1,71	140	0
800	200	1,66	119	0
900	200	1,41	99	0
1000	200	1,34	89	0
2000	200	0,98	45	0
3000	200	0,7	29	0
4000	200	0,62	20	0
5000	200	0,64	16	0
6000	200	0,43	12	0
7000	200	0,49	11	0
8000	200	0,41	10	0
9000	200	0,43	10	0
10000	200	0,36	8	0
20000	200	0,36	5	0
30000	200	0,26	4	0
40000	200	0,09	0	0
50000	200	0,07	0	0
60000	200	0,06	0	0
70000	200	0,06	0	0
80000	200	0,05	0	0
90000	200	0,05	0	0
100000	200	0.05	0	0

3 Résultats avec l'approche de Young sans connaissance du MTTF réel

MTTF Réel (heures)	MTTF à priori (heures)	longueur initiale moyenne du job	temps ajouté (%)	nb pannes	jobs restants
100	100000	1000	92,49	1588	0
200	100000	1000	41,9	615	0
300	100000	1000	26,86	383	0
400	100000	1000	18,91	290	0
500	100000	1000	14,29	231	0
600	100000	1000	12,9	175	0
700	100000	1000	10,7	144	0
800	100000	1000	9,34	121	0
900	100000	1000	8,18	104	0
1000	100000	1000	7,14	92	0
2000	100000	1000	3,99	45	0
3000	100000	1000	2,54	29	0
4000	100000	1000	1,75	22	0
5000	100000	1000	1,35	17	0
6000	100000	1000	1,1	12	0
7000	100000	1000	0,98	11	0
8000	100000	1000	0,93	10	0
9000	100000	1000	1,01	10	0
10000	100000	1000	0,7	8	0
20000	100000	1000	0,3	5	0
30000	100000	1000	0,24	4	0
40000	100000	1000	0,2	0	0
50000	100000	1000	0,22	0	0
60000	100000	1000	0,13	0	0
70000	100000	1000	0,16	0	0
80000	100000	1000	0,13	0	0
90000	100000	1000	0,13	0	0
100000	100000	1000	0,14	0	0
100	5000	1000	17,37	5539	0
200	5000	1000	9	2675	0
300	5000	1000	5,85	1674	0
400	5000	1000	4,27	1227	0
500	5000	1000	3,54	990	0
600	5000	1000	3,15	841	0
700	5000	1000	2,66	708	0
800	5000	1000	2,35	615	0
900	5000	1000	1,98	533	0
1000	5000	1000	1,86	475	0
2000	5000	1000	1,21	255	0
3000	5000	1000	0,89	156	0
4000	5000	1000	0,71	119	0
5000	5000	1000	0,63	88	0
6000	5000	1000	0,59	75	0
7000	5000	1000	0,57	70	0
8000	5000	1000	0,48	56	0
9000	5000	1000	0,52	51	0
10000	5000	1000	0,49	44	0
20000	5000	1000	0,39	20	0
30000	5000	1000	0,36	12	0
40000	5000	1000	0,36	10	0
50000	5000	1000	0,34	8	0
60000	5000	1000	0,34	7	0
70000	5000	1000	0,34	6	0
80000	5000	1000	0,34	6	0
90000	5000	1000	0,33	5	0
100000	5000	1000	0,34	5	0
100	500	1000	6,33	5010	0
200	500	1000	3,75	2526	0
300	500	1000	2,74	1619	0
400	500	1000	2,35	1202	0
500	500	1000	2,06	979	0
600	500	1000	1,92	827	0
700	500	1000	1,76	698	0
800	500	1000	1,65	614	0
900	500	1000	1,62	530	0
1000	500	1000	1,56	478	0
2000	500	1000	1,3	254	0
3000	500	1000	1,21	158	0
4000	500	1000	1,16	120	0
5000	500	1000	1,13	88	0
6000	500	1000	1,1	74	0
7000	500	1000	1,09	70	0
8000	500	1000	1,08	56	0
9000	500	1000	1,08	50	0
10000	500	1000	1,08	44	0
20000	500	1000	1,05	20	0
30000	500	1000	1,04	12	0
40000	500	1000	1,03	10	0
50000	500	1000	1,04	8	0
60000	500	1000	1,03	7	0
70000	500	1000	1,03	6	0
80000	500	1000	1,03	6	0
90000	500	1000	1,03	5	0
100000	500	1000	1,03	5	0
100	100000	200	73,63	1588	0
200	100000	200	34,81	615	0
300	100000	200	22,15	383	0
400	100000	200	17,49	290	0
500	100000	200	13,53	231	0
600	100000	200	9,97	175	0
700	100000	200	8,41	144	0
800	100000	200	7,63	121	0
900	100000	200	5,5	104	0

1000	100000	200	5,23	92	0
2000	100000	200	2,31	45	0
3000	100000	200	1,55	29	0
4000	100000	200	1,35	22	0
5000	100000	200	1,06	17	0
6000	100000	200	0,82	12	0
7000	100000	200	0,67	11	0
8000	100000	200	0,85	10	0
9000	100000	200	1,04	10	0
10000	100000	200	0,7	8	0
20000	100000	200	0,35	5	0
30000	100000	200	0,48	4	0
40000	100000	200	0,04	0	0
50000	100000	200	0,04	0	0
60000	100000	200	0,04	0	0
70000	100000	200	0,04	0	0
80000	100000	200	0,04	0	0
90000	100000	200	0,04	0	0
100000	100000	200	0,04	0	0
100	5000	200	16,04	1044	0
200	5000	200	8,32	496	0
300	5000	200	5,53	341	0
400	5000	200	4,08	263	0
500	5000	200	3,1	199	0
600	5000	200	3,19	161	0
700	5000	200	2,71	141	0
800	5000	200	2,12	121	0
900	5000	200	1,85	100	0
1000	5000	200	1,49	87	0
2000	5000	200	1,23	45	0
3000	5000	200	0,85	30	0
4000	5000	200	0,55	20	0
5000	5000	200	0,64	16	0
6000	5000	200	0,5	12	0
7000	5000	200	0,48	11	0
8000	5000	200	0,46	10	0
9000	5000	200	0,45	10	0
10000	5000	200	0,45	8	0
20000	5000	200	0,44	5	0
30000	5000	200	0,39	4	0
40000	5000	200	0,29	0	0
50000	5000	200	0,29	0	0
60000	5000	200	0,29	0	0
70000	5000	200	0,29	0	0
80000	5000	200	0,29	0	0
90000	5000	200	0,29	0	0
100000	5000	200	0,29	0	0
100	500	200	6,24	959	0
200	500	200	3,65	477	0
300	500	200	2,85	337	0
400	500	200	2,4	258	0
500	500	200	2	196	0
600	500	200	1,85	156	0
700	500	200	1,75	141	0
800	500	200	1,63	120	0
900	500	200	1,59	99	0
1000	500	200	1,49	89	0
2000	500	200	1,26	44	0
3000	500	200	1,19	29	0
4000	500	200	1,12	20	0
5000	500	200	1,09	15	0
6000	500	200	1,06	12	0
7000	500	200	1,07	11	0
8000	500	200	1,06	10	0
9000	500	200	1,08	10	0
10000	500	200	1,04	8	0
20000	500	200	1,02	5	0
30000	500	200	1,03	4	0
40000	500	200	1	0	0
50000	500	200	1	0	0
60000	500	200	1	0	0
70000	500	200	1	0	0
80000	500	200	1	0	0
90000	500	200	1	0	0
100000	500	200	1	0	0

4 Résultats avec l'approche additive (sans connaissance du MTTF réel)

n	MTTF Réel (heures)	MTTF à priori (heures)	longueur initiale moyenne du job	temps ajouté (%)	nb pannes	jobs restants
0,8	100	1000	1000	4,93	947	0
0,8	200	1000	1000	3,52	481	0
0,8	300	1000	1000	2,84	334	0
0,8	400	1000	1000	2,6	258	0
0,8	500	1000	1000	2,34	194	0
0,8	600	1000	1000	1,99	159	0
0,8	700	1000	1000	1,9	142	0
0,8	800	1000	1000	1,72	119	0
0,8	900	1000	1000	1,58	100	0
0,8	1000	1000	1000	1,48	90	0
0,8	2000	1000	1000	1,05	44	0
0,8	3000	1000	1000	0,94	30	0
0,8	4000	1000	1000	0,73	20	0
0,8	5000	1000	1000	0,75	16	0
0,8	6000	1000	1000	0,59	12	0
0,8	7000	1000	1000	0,59	11	0
0,8	8000	1000	1000	0,57	10	0

0,8	9000	1000	1000	0,63	10	0
0,8	10000	1000	1000	0,49	8	0
0,8	20000	1000	1000	0,47	5	0
0,8	30000	1000	1000	0,5	4	0
0,8	40000	1000	1000	0,42	0	0
0,8	50000	1000	1000	0,47	0	0
0,8	60000	1000	1000	0,51	0	0
0,8	70000	1000	1000	0,55	0	0
0,8	80000	1000	1000	0,57	0	0
0,8	90000	1000	1000	0,6	0	0
0,8	10^6	1000	1000	0,61	0	0
0,5	100	1000	1000	4,89	948	0
0,5	200	1000	1000	3,29	477	0
0,5	300	1000	1000	2,91	337	0
0,5	400	1000	1000	2,48	258	0
0,5	500	1000	1000	2,2	195	0
0,5	600	1000	1000	1,97	159	0
0,5	700	1000	1000	1,84	142	0
0,5	800	1000	1000	1,62	119	0
0,5	900	1000	1000	1,54	100	0
0,5	1000	1000	1000	1,51	90	0
0,5	2000	1000	1000	0,97	44	0
0,5	3000	1000	1000	0,86	30	0
0,5	4000	1000	1000	0,7	20	0
0,5	5000	1000	1000	0,68	16	0
0,5	6000	1000	1000	0,54	12	0
0,5	7000	1000	1000	0,49	11	0
0,5	8000	1000	1000	0,55	10	0
0,5	9000	1000	1000	0,55	10	0
0,5	10000	1000	1000	0,52	8	0
0,5	20000	1000	1000	0,39	5	0
0,5	30000	1000	1000	0,64	4	0
0,5	40000	1000	1000	0,43	0	0
0,5	50000	1000	1000	0,47	0	0
0,5	60000	1000	1000	0,52	0	0
0,5	70000	1000	1000	0,55	0	0
0,5	80000	1000	1000	0,58	0	0
0,5	90000	1000	1000	0,6	0	0
0,5	10^6	1000	1000	0,62	0	0
0,2	100	1000	1000	4,67	946	0
0,2	200	1000	1000	3,35	477	0
0,2	300	1000	1000	2,7	334	0
0,2	400	1000	1000	2,28	257	0
0,2	500	1000	1000	2,16	195	0
0,2	600	1000	1000	1,89	158	0
0,2	700	1000	1000	1,72	140	0
0,2	800	1000	1000	1,68	119	0
0,2	900	1000	1000	1,43	99	0
0,2	1000	1000	1000	1,38	89	0
0,2	2000	1000	1000	0,91	44	0
0,2	3000	1000	1000	0,81	29	0
0,2	4000	1000	1000	0,65	20	0
0,2	5000	1000	1000	0,64	16	0
0,2	6000	1000	1000	0,49	12	0
0,2	7000	1000	1000	0,53	11	0
0,2	8000	1000	1000	0,5	10	0

0,2	9000	1000	1000	0,5	10	0
0,2	10000	1000	1000	0,46	8	0
0,2	20000	1000	1000	0,49	5	0
0,2	30000	1000	1000	0,47	4	0
0,2	40000	1000	1000	0,45	0	0
0,2	50000	1000	1000	0,49	0	0
0,2	60000	1000	1000	0,53	0	0
0,2	70000	1000	1000	0,56	0	0
0,2	80000	1000	1000	0,58	0	0
0,2	90000	1000	1000	0,6	0	0
0,2	10^6	1000	1000	0,62	0	0
0,5	100	500	1000	4,91	4951	0
0,5	200	500	1000	3,48	2514	0
0,5	300	500	1000	2,8	1620	0
0,5	400	500	1000	2,39	1202	0
0,5	500	500	1000	2,17	979	0
0,5	600	500	1000	2,01	829	0
0,5	700	500	1000	1,78	698	0
0,5	800	500	1000	1,67	613	0
0,5	900	500	1000	1,61	531	0
0,5	1000	500	1000	1,54	477	0
0,5	2000	500	1000	1,13	255	0
0,5	3000	500	1000	0,88	156	0
0,5	4000	500	1000	0,74	119	0
0,5	5000	500	1000	0,65	88	0
0,5	6000	500	1000	0,61	74	0
0,5	7000	500	1000	0,6	70	0
0,5	8000	500	1000	0,57	56	0
0,5	9000	500	1000	0,5	51	0
0,5	10000	500	1000	0,48	45	0
0,5	20000	500	1000	0,38	20	0
0,5	30000	500	1000	0,34	12	0
0,5	40000	500	1000	0,37	10	0
0,5	50000	500	1000	0,4	8	0
0,5	60000	500	1000	0,39	7	0
0,5	70000	500	1000	0,4	6	0
0,5	80000	500	1000	0,39	6	0
0,5	90000	500	1000	0,39	5	0
0,5	10^6	500	1000	0,41	5	0
0,5	100	100000	1000	15,57	1038	0
0,5	200	100000	1000	11,67	522	0
0,5	300	100000	1000	8,54	351	0
0,5	400	100000	1000	7,03	268	0
0,5	500	100000	1000	5,85	206	0
0,5	600	100000	1000	5,04	170	0
0,5	700	100000	1000	4,37	143	0
0,5	800	100000	1000	3,91	119	0
0,5	900	100000	1000	2,73	101	0
0,5	1000	100000	1000	2,32	88	0
0,5	2000	100000	1000	1,7	45	0
0,5	3000	100000	1000	1,17	30	0
0,5	4000	100000	1000	1,1	22	0
0,5	5000	100000	1000	0,9	16	0
0,5	6000	100000	1000	0,81	12	0
0,5	7000	100000	1000	0,75	11	0
0,5	8000	100000	1000	0,76	10	0

0,5	9000	100000	1000	0,8	10	0
0,5	10000	100000	1000	0,72	8	0
0,5	20000	100000	1000	0,3	5	0
0,5	30000	100000	1000	0,29	4	0
0,5	40000	100000	1000	0,05	0	0
0,5	50000	100000	1000	0,05	0	0
0,5	60000	100000	1000	0,04	0	0
0,5	70000	100000	1000	0,04	0	0
0,5	80000	100000	1000	0,04	0	0
0,5	90000	100000	1000	0,04	0	0
0,5	10^6	100000	1000	0,04	0	0
0,5	100	5000	1000	5,13	4954	0
0,5	200	5000	1000	3,56	2515	0
0,5	300	5000	1000	2,8	1617	0
0,5	400	5000	1000	2,42	1203	0
0,5	500	5000	1000	2,24	979	0
0,5	600	5000	1000	2	827	0
0,5	700	5000	1000	1,84	699	0
0,5	800	5000	1000	1,74	613	0
0,5	900	5000	1000	1,59	530	0
0,5	1000	5000	1000	1,56	478	0
0,5	2000	5000	1000	1,14	254	0
0,5	3000	5000	1000	0,84	156	0
0,5	4000	5000	1000	0,7	119	0
0,5	5000	5000	1000	0,64	88	0
0,5	6000	5000	1000	0,61	74	0
0,5	7000	5000	1000	0,55	70	0
0,5	8000	5000	1000	0,51	56	0
0,5	9000	5000	1000	0,48	51	0
0,5	10000	5000	1000	0,46	44	0
0,5	20000	5000	1000	0,32	20	0
0,5	30000	5000	1000	0,31	12	0
0,5	40000	5000	1000	0,26	10	0
0,5	50000	5000	1000	0,22	8	0
0,5	60000	5000	1000	0,3	7	0
0,5	70000	5000	1000	0,23	6	0
0,5	80000	5000	1000	0,23	6	0
0,5	90000	5000	1000	0,22	5	0
0,5	10^6	5000	1000	0,23	5	0
0,5	100	100000	200	15,57	1038	0
0,5	200	100000	200	11,67	522	0
0,5	300	100000	200	8,54	351	0
0,5	400	100000	200	7,03	268	0
0,5	500	100000	200	5,85	206	0
0,5	600	100000	200	5,04	170	0
0,5	700	100000	200	4,37	143	0
0,5	800	100000	200	3,91	119	0
0,5	900	100000	200	2,73	101	0
0,5	1000	100000	200	2,32	88	0
0,5	2000	100000	200	1,7	45	0
0,5	3000	100000	200	1,17	30	0
0,5	4000	100000	200	1,1	22	0
0,5	5000	100000	200	0,9	16	0
0,5	6000	100000	200	0,81	12	0
0,5	7000	100000	200	0,75	11	0
0,5	8000	100000	200	0,76	10	0

0,5	9000	100000	200	0,8	10	0
0,5	10000	100000	200	0,72	8	0
0,5	20000	100000	200	0,3	5	0
0,5	30000	100000	200	0,29	4	0
0,5	40000	100000	200	0,05	0	0
0,5	50000	100000	200	0,05	0	0
0,5	60000	100000	200	0,04	0	0
0,5	70000	100000	200	0,04	0	0
0,5	80000	100000	200	0,04	0	0
0,5	90000	100000	200	0,04	0	0
0,5	10^6	100000	200	0,04	0	0
0,5	100	500	200	4.75	944	0
0,5	200	500	200	3.37	476	0
0,5	300	500	200	2.92	337	0
0,5	400	500	200	2.58	259	0
0,5	500	500	200	2.12	195	0
0,5	600	500	200	1.95	158	0
0,5	700	500	200	1.78	141	0
0,5	800	500	200	1.6	120	0
0,5	900	500	200	1.53	100	0
0,5	1000	500	200	1.52	90	0
0,5	2000	500	200	0.98	45	0
0,5	3000	500	200	0.79	29	0
0,5	4000	500	200	0.74	20	0
0,5	5000	500	200	0.68	16	0
0,5	6000	500	200	0.55	12	0
0,5	7000	500	200	0.59	11	0
0,5	8000	500	200	0.57	10	0
0,5	9000	500	200	0.61	10	0
0,5	10000	500	200	0.63	8	0
0,5	20000	500	200	0.58	5	0
0,5	30000	500	200	0.61	4	0
0,5	40000	500	200	0.59	0	0
0,5	50000	500	200	0.66	0	0
0,5	60000	500	200	0.73	0	0
0,5	70000	500	200	0.78	0	0
0,5	80000	500	200	0.82	0	0
0,5	90000	500	200	0.86	0	0
0,5	10^6	500	200	0.88	0	0

5 Résultats avec l'approche multiplicative (sans connaissance du MTTF réel)

a	MTTF Réel (heures)	MTTF à priori (heures)	longueur initiale moyenne du job	temps ajouté (%)	nb pannes	jobs restants
1,5	100	1000	1000	4,91	951	0
1,5	200	1000	1000	3,57	481	0
1,5	300	1000	1000	3,22	341	0
1,5	400	1000	1000	2,94	261	0
1,5	500	1000	1000	2,45	198	0
1,5	600	1000	1000	2,31	158	0
1,5	700	1000	1000	2,17	143	0
1,5	800	1000	1000	2,08	119	0
1,5	900	1000	1000	1,69	101	0
1,5	1000	1000	1000	1,5	89	0
1,5	2000	1000	1000	0,92	44	0
1,5	3000	1000	1000	0,76	30	0
1,5	4000	1000	1000	0,7	20	0
1,5	5000	1000	1000	0,71	16	0
1,5	6000	1000	1000	0,54	12	0
1,5	7000	1000	1000	0,79	11	0
1,5	8000	1000	1000	0,86	10	0
1,5	9000	1000	1000	0,66	10	0
1,5	10000	1000	1000	0,99	8	0
1,5	20000	1000	1000	0,95	5	0
1,5	30000	1000	1000	1,18	4	0
1,5	40000	1000	1000	0,39	0	0
1,5	50000	1000	1000	0,45	0	0
1,5	60000	1000	1000	0,5	0	0
1,5	70000	1000	1000	0,54	0	0
1,5	80000	1000	1000	0,57	0	0
1,5	90000	1000	1000	0,59	0	0
1,5	10^6	1000	1000	0,61	0	0
2	100	1000	1000	9,82	992	0
2	200	1000	1000	6,73	492	0
2	300	1000	1000	5,73	342	0
2	400	1000	1000	6,08	270	0
2	500	1000	1000	6,82	208	0
2	600	1000	1000	5,03	162	0
2	700	1000	1000	5,49	144	0
2	800	1000	1000	4,89	122	0
2	900	1000	1000	3,3	103	0
2	1000	1000	1000	3,4	92	0
2	2000	1000	1000	2,27	46	0
2	3000	1000	1000	1,57	30	0
2	4000	1000	1000	1,28	20	0
2	5000	1000	1000	1,39	16	0
2	6000	1000	1000	0,89	12	0
2	7000	1000	1000	1,35	11	0
2	8000	1000	1000	1,3	10	0
2	9000	1000	1000	1,44	10	0
2	10000	1000	1000	1,04	8	0
2	20000	1000	1000	0,95	5	0
2	30000	1000	1000	1,18	4	0
2	40000	1000	1000	0,39	0	0
2	50000	1000	1000	0,45	0	0
2	60000	1000	1000	0,5	0	0
2	70000	1000	1000	0,54	0	0
2	80000	1000	1000	0,57	0	0
2	90000	1000	1000	0,59	0	0
2	10^6	1000	1000	0,61	0	0
3	100	1000	1000	39,01	1246	0
3	200	1000	1000	29,23	605	0
3	300	1000	1000	29,68	407	0
3	400	1000	1000	26,4	311	0
3	500	1000	1000	20,5	244	0
3	600	1000	1000	15,72	179	0
3	700	1000	1000	11,67	151	0
3	800	1000	1000	10,82	128	0
3	900	1000	1000	7,7	105	0
3	1000	1000	1000	6,77	92	0
3	2000	1000	1000	2,41	45	0
3	3000	1000	1000	2,31	31	0
3	4000	1000	1000	1,85	21	0
3	5000	1000	1000	1,84	16	0
3	6000	1000	1000	1,79	13	0
3	7000	1000	1000	1,34	11	0
3	8000	1000	1000	1,3	10	0
3	9000	1000	1000	1,44	10	0
3	10000	1000	1000	1,04	8	0
3	20000	1000	1000	0,95	5	0
3	30000	1000	1000	1,18	4	0
3	40000	1000	1000	0,39	0	0
3	50000	1000	1000	0,45	0	0
3	60000	1000	1000	0,5	0	0
3	70000	1000	1000	0,54	0	0
3	80000	1000	1000	0,57	0	0
3	90000	1000	1000	0,59	0	0
3	10^6	1000	1000	0,61	0	0
1,5	100	500	1000	5,42	4972	0
1,5	200	500	1000	4,02	2535	0
1,5	300	500	1000	3,18	1626	0
1,5	400	500	1000	2,73	1209	0
1,5	500	500	1000	2,42	980	0
1,5	600	500	1000	2,21	827	0
1,5	700	500	1000	2,11	702	0
1,5	800	500	1000	2,02	615	0
1,5	900	500	1000	1,82	531	0
1,5	1000	500	1000	1,67	478	0
1,5	2000	500	1000	1,28	257	0
1,5	3000	500	1000	0,94	157	0
1,5	4000	500	1000	1,05	119	0
1,5	5000	500	1000	0,97	88	0
1,5	6000	500	1000	2,11	75	0

1,5	7000	500	1000	3,3	71	0
1,5	8000	500	1000	4,24	61	0
1,5	9000	500	1000	5,84	54	0
1,5	10000	500	1000	4,11	47	0
1,5	20000	500	1000	2,65	22	0
1,5	30000	500	1000	1,84	13	0
1,5	40000	500	1000	1,36	10	0
1,5	50000	500	1000	1,14	8	0
1,5	60000	500	1000	1,03	7	0
1,5	70000	500	1000	0,8	6	0
1,5	80000	500	1000	0,92	6	0
1,5	90000	500	1000	0,97	5	0
1,5	10^6	500	1000	1,07	5	0
1,5	100	100000	1000	15,78	1039	0
1,5	200	100000	1000	11,72	525	0
1,5	300	100000	1000	8,71	352	0
1,5	400	100000	1000	7,16	268	0
1,5	500	100000	1000	6,37	211	0
1,5	600	100000	1000	5,35	171	0
1,5	700	100000	1000	4,53	143	0
1,5	800	100000	1000	4,05	120	0
1,5	900	100000	1000	2,9	101	0
1,5	1000	100000	1000	2,36	88	0
1,5	2000	100000	1000	1,69	45	0
1,5	3000	100000	1000	1,18	30	0
1,5	4000	100000	1000	1,19	22	0
1,5	5000	100000	1000	0,97	17	0
1,5	6000	100000	1000	0,82	12	0
1,5	7000	100000	1000	0,67	11	0
1,5	8000	100000	1000	0,7	10	0
1,5	9000	100000	1000	0,79	10	0
1,5	10000	100000	1000	0,72	8	0
1,5	20000	100000	1000	0,41	5	0
1,5	30000	100000	1000	0,39	4	0
1,5	40000	100000	1000	0,05	0	0
1,5	50000	100000	1000	0,05	0	0
1,5	60000	100000	1000	0,04	0	0
1,5	70000	100000	1000	0,04	0	0
1,5	80000	100000	1000	0,04	0	0
1,5	90000	100000	1000	0,04	0	0
1,5	10^6	100000	1000	0,04	0	0
1,5	100	5000	1000	5,6	4984	0
1,5	200	5000	1000	4,16	2529	0
1,5	300	5000	1000	3,28	1633	0
1,5	400	5000	1000	2,62	1207	0
1,5	500	5000	1000	2,44	980	0
1,5	600	5000	1000	2,27	833	0
1,5	700	5000	1000	2,09	700	0
1,5	800	5000	1000	2,01	615	0
1,5	900	5000	1000	1,8	531	0
1,5	1000	5000	1000	1,62	477	0
1,5	2000	5000	1000	1,25	257	0
1,5	3000	5000	1000	0,97	157	0
1,5	4000	5000	1000	0,86	119	0
1,5	5000	5000	1000	0,71	89	0
1,5	6000	5000	1000	0,63	75	0

1,5	7000	5000	1000	0,6	70	0
1,5	8000	5000	1000	0,51	56	0
1,5	9000	5000	1000	0,47	51	0
1,5	10000	5000	1000	0,45	44	0
1,5	20000	5000	1000	0,34	20	0
1,5	30000	5000	1000	0,27	12	0
1,5	40000	5000	1000	0,52	10	0
1,5	50000	5000	1000	0,53	8	0
1,5	60000	5000	1000	0,88	7	0
1,5	70000	5000	1000	0,63	6	0
1,5	80000	5000	1000	0,73	6	0
1,5	90000	5000	1000	0,76	5	0
1,5	10^6	5000	1000	0,84	5	0
1,5	100	100000	200	15,78	1039	0
1,5	200	100000	200	11,72	525	0
1,5	300	100000	200	8,71	352	0
1,5	400	100000	200	7,16	268	0
1,5	500	100000	200	6,37	211	0
1,5	600	100000	200	5,35	171	0
1,5	700	100000	200	4,53	143	0
1,5	800	100000	200	4,05	120	0
1,5	900	100000	200	2,9	101	0
1,5	1000	100000	200	2,36	88	0
1,5	2000	100000	200	1,69	45	0
1,5	3000	100000	200	1,18	30	0
1,5	4000	100000	200	1,19	22	0
1,5	5000	100000	200	0,97	17	0
1,5	6000	100000	200	0,82	12	0
1,5	7000	100000	200	0,67	11	0
1,5	8000	100000	200	0,7	10	0
1,5	9000	100000	200	0,79	10	0
1,5	10000	100000	200	0,72	8	0
1,5	20000	100000	200	0,41	5	0
1,5	30000	100000	200	0,39	4	0
1,5	40000	100000	200	0,05	0	0
1,5	50000	100000	200	0,05	0	0
1,5	60000	100000	200	0,04	0	0
1,5	70000	100000	200	0,04	0	0
1,5	80000	100000	200	0,04	0	0
1,5	90000	100000	200	0,04	0	0
1,5	10^6	100000	200	0,04	0	0
1,5	100	5000	200	5,92	959	0
1,5	200	5000	200	3,82	480	0
1,5	300	5000	200	2,98	338	0
1,5	400	5000	200	2,53	258	0
1,5	500	5000	200	2,29	195	0
1,5	600	5000	200	2,06	159	0
1,5	700	5000	200	1,75	141	0
1,5	800	5000	200	1,77	119	0
1,5	900	5000	200	1,57	99	0
1,5	1000	5000	200	1,56	89	0
1,5	2000	5000	200	1,1	45	0
1,5	3000	5000	200	0,83	30	0
1,5	4000	5000	200	0,59	20	0
1,5	5000	5000	200	0,63	16	0
1,5	6000	5000	200	0,53	12	0

1,5	7000	5000	200	0,52	11	0
1,5	8000	5000	200	0,5	10	0
1,5	9000	5000	200	0,49	10	0
1,5	10000	5000	200	0,45	8	0
1,5	20000	5000	200	0,36	5	0
1,5	30000	5000	200	0,39	4	0
1,5	40000	5000	200	0,21	0	0
1,5	50000	5000	200	0,22	0	0
1,5	60000	5000	200	0,23	0	0
1,5	70000	5000	200	0,24	0	0
1,5	80000	5000	200	0,25	0	0
1,5	90000	5000	200	0,26	0	0
1,5	10^6	5000	200	0,26	0	0
1,5	100	500	200	4,8	944	0
1,5	200	500	200	3,35	478	0
1,5	300	500	200	3,35	338	0
1,5	400	500	200	2,83	260	0
1,5	500	500	200	2,37	199	0
1,5	600	500	200	2,33	156	0
1,5	700	500	200	2,31	142	0
1,5	800	500	200	2,14	119	0
1,5	900	500	200	1,71	101	0
1,5	1000	500	200	1,7	90	0
1,5	2000	500	200	1,01	45	0
1,5	3000	500	200	0,82	30	0
1,5	4000	500	200	1,07	20	0
1,5	5000	500	200	1,41	16	0
1,5	6000	500	200	1,89	13	0
1,5	7000	500	200	1,38	11	0
1,5	8000	500	200	1,34	10	0
1,5	9000	500	200	1,52	10	0
1,5	10000	500	200	1,14	8	0
1,5	20000	500	200	1,07	5	0
1,5	30000	500	200	1,3	4	0
1,5	40000	500	200	0,55	0	0
1,5	50000	500	200	0,64	0	0
1,5	60000	500	200	0,71	0	0
1,5	70000	500	200	0,76	0	0
1,5	80000	500	200	0,81	0	0
1,5	90000	500	200	0,85	0	0
1,5	10^6	500	200	0,88	0	0

Bibliographie

1. **J.-C.Laprie.** *"Dependable Computing and Fault Tolerance : concepts and terminology", in proceedings of the 15th IEEE International Symposium on Fault Tolerant Computing (FTCS).* 1985. pp. 2-11.

2. **J.Gray.** "A census of tandem system availability between 1985 and 1990", IEEE Transactions on reliability. 1990. pp. 409-418.

3. **Gibson, B. Schroeder and G.A.** *"A large-scale study of failures in high-performance computing systems". IEEE Transactions on Dependable and Secure Computing.* 2010. pp. 337–351.

4. **N.Nagappan, K.V.Vishwanath.** *"Characterizing cloud computing hardware reliability", in Proceedings of the 1st ACM Symposium on Cloud Computing.* Juin 2010. pp. 193-204.

5. **Gibson, B. Schroeder and G.A.** *"A large-scale study of failures in high-performance computing systems".IEEE Transactions on Dependable and Secure Computing.* 2010. pp. 337–351.

6. **Stearley, A. Oliner.** *"What supercomputers say : A study of five system logs".In 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks.* 2007. pp. 575–584.

7. **B.cully, al.** *"Remus: High availability via asynchronous virtual machine replication", in Proceedings of the 5th USENIXS Symposium on Networked Systems Design and Implementation (NSDI).* 2008. p. 161.

8. **Cappello, B. Nicolae.** *"Blob CR:Efficient checkpoint-restart for HPC applications on IaaS clouds, using virtual disk image snapshots", in Proceedings of the International Conference for High Performance Computing, Networking,Storage and Analysis, seattle.* 2011. p. p34.

9. **J. S. Plank, K. Li, and M. A. Puening.** *"Diskless checkpointing". IEEE Transactions on Parallel and Distributed Systems.* 1998. pp. 972–986.

10. **J. S. Plank, M. Beck, G. Kingsley, and K. Li.** *"Libckpt : Transparent checkpointing under Unix". In Proceedings of the 1995 Winter USENIX Technical Conference.* 1995. pp. 213–223.

11. **M.Litzkow, M.Livny,and M.W.Mutka.** *"Condor - A Hunter of Idle Workstations". In Proceedings of the 8th International Conference of Distributed Computing Systems.* 1988. pp. 104-111.

12. **Samy, SADI.** *"Techniques de Checkpointing pour la tolérance aux fautes dans le Cloud Computing".* 2017.

13. **J.W.Young.** *"A first order approximation to the optimum checkpoint interval",Communications of the ACM 17(9).* 1974. pp. 530-531.

14. **J.T.Daly.** *“A higher order estimate model of the optimum checkpoint interval restart dumps”, Future Generation Computer Systems.* 2006. pp. 303-312.

15. **X.Lin, Y.Ling, J.Mi.** *"A variational calculus approach to optimal checkpoint placement", IEEE Transactions on Computers.* July2001. pp. 699-708.

16. **L. Zhu, al.** *"Research on Optimum Checkpoint Interval for Hybrid Fault Tolerance", in Proceedings of the 10th International Symposium on Advanced Parallel Processing Technologies (AAPT).* Août 2013. pp. 367-380.

17. **Wang, C. Mueller, F. Engelmann, C. Scott, S.L.** *" Proactive process-level live migration and back migration in HPC environments.* Journal of Parallel and Distributed Computing 72. 2012. pp. 254–267 .

18. **Dohi. H, Okamura.** *"Comprehensive evaluation of aperiodic checkpointing and rejuvenation schemes in operational software system".* Journal of Systems and Software . 2010. pp. 1591–1604.

19. **T. Ozaki, T. Dohi, and N. Kaio.** *"Numerical computation algorithms for sequential checkpoint placement. Performance Evaluation".* 2009. pp. 311–326.

20. **R.E. Barlow, L.C. Hunter, and F. Proschan.** *"Optimum checking procedures".Journal of the society for industrial and applied mathematics.* 1963. pp. 1078–1095.

21. **M.-S. Bouguerra, T. Gautier, D. Trystram, and J.-M. Vincent.** *“A flexible checkpoint/restart model in distributed systems” Proc. Eighth Int’l Conf. Parallel Processing and Applied Math.* 2010. pp. 206-215.

22. **Samy Sadi, Belabbas Yagoubi.** *ACS-Advanced Cloud Simulator:A Discrete Event Based Simulator for Cloud Computing Environments. On Networking and Advanced Systems,.* 2015. p. 11.