MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE
LA RECHERCHE SCIENTIFIQUE
UNIVERSITE ABDELHAMID IBN BADIS - MOSTAGANEM

Faculté des Sciences Exactes et de l'Informatique
Département de Mathématiques et d'Informatique
Filière : Informatique

MEMOIRE DE FIN D'ETUDES
Pour l'Obtention du Diplôme de Master en Informatique
Option : **Ingénierie des Systèmes d'Information**

THEME :

Le Checkpointing Pour Les Bots Dans Le Cloud Computing

Etudiant(e) : « **Zerdani Sara** »

Encadrant(e) : « **B. Meroufel** »

Année Universitaire 2016/2017

Résumé:

Le Cloud Computing fait référence à l'utilisation des capacités de calcul des ordinateurs distants, où l'utilisateur dispose d'une puissance informatique considérable sans avoir à posséder des unités puissantes. La probabilité de voir une panne intervenir durant l'exécution devient forte lorsque le nombre de nœuds augmente ; puisqu'il est impossible d'empêcher totalement les pannes, une solution consiste à mettre en place des mécanismes de tolérance aux pannes. La tolérance aux pannes est devenue une tâche majeure pour les ingénieurs informatiques et les développeurs de logiciels, car l'apparition de pannes augmente le coût de l'utilisation des ressources. Dans ce travaille nous devons développer une architecture de tolérance aux pannes dans les Cloud Computing. Nous avons proposé l'approche qui se base essentiellement sur le mécanisme de checkpoint dans le but de minimiser le temps perdu par les pannes et qui permet de garantir la continuité des services du Cloud Computing d'une façon transparent et efficace en présence des pannes.

Mots clés: Tolérance aux pannes; Cloud computing; Datacenter; Virtualisation; Checkpointing; Pannes.

REMERCIEMENTS

Je remercie **Allah Sobhano** de m'avoir donné la force, le courage et la patience pour terminer cette mémoire.

Quelques lignes sont trop courtes pour exprimer ma profonde reconnaissance pour ma directrice de thèse, M.me B.Meroufel, pour m'avoir appris à être moins « bonne élève » et plus autonome tout au long de ce travail de recherche.

Je n'oublierai pas de remercier tous mes amies et collègues chercheurs du

Département d'Informatique, avec lesquelles ce fut toujours très agréable de travailler et surtout pour la bonne ambiance tout au long de la thèse, en particulier ;<3 mon Hamtaro<3, ma sœur ikram, Fatiha, Hafsa, Imen,.....

Je voudrai aussi exprimer ma grande affection à mes parents. Je les remercie pour leur amour, leur soutien et leur confiance. Mon père a toujours placé l'éducation comme la première priorité dans ma vie et m'a incité à fixer des très hauts objectifs pour moi-même. Il m'a appris à mettre l'honnêteté et le courage au-dessus de toutes les autres vertus.

Je ne saurais terminer sans adresser mes remerciements à toutes les personnes, ont m'a donné le soutien moral, physique ou autres de près ou de loin, ont contribué à la réalisation de ce travail.

Dédicaces

A mes chers parents, pour tous leurs sacrifices, leur amour, leur tendresse, leur soutien et leurs prières tout au long de mes études.

A ma chère sœur Ikram pour leurs encouragements permanents, et leur soutien moral.

A mes chers amies pour leur appui et leur encouragement.

A toute ma famille pour leur soutien tout au long de mon parcours universitaire.

Que ce travail soit l'accomplissement de vos vœux tant allégués, et le fuit de votre soutien infaillible.

Merci d'être toujours là pour moi.

Table des Matières

Liste des Figures

Liste des Tableaux

Liste des Abréviations

SLA Service Level Agreement
BoTs Bags of Tasks
SdF Sûreté de Fonctionnement
NIST National Institute of Standards and Technology
FIFO First In First Out
CIC Communication Induced Checkpointing
PWD Piecewise Deterministic Assumption
ACP Adaptatif Checkpointing
IaaS Infrastructure as a Service
PaaS Platform as a Service
SaaS Software as a Service
CRM Customer Relationship Management
BI Business Intelligence
ERP Entreprise Resource Planning
CPU Central Processing Unit
VM VirtualMachine

Introduction Générale

L'année 2008 a vu l'émergence du terme Cloud Computing (ou l'informatique dans les nuages) dans les journaux spécialisés et les annonces des nouvelles solutions chez tous les grands acteurs de l'informatique: Microsoft, Google, Amazon, IBM, Dell, Oracle,...

Dans l'effervescence qui accompagne toute grande nouveauté dans le monde de l'informatique, le Cloud Computing est apparu pour certains comme une révolution et pour d'autres comme un simple terme de Marketing qui ne fait que rassembler des services et des technologies qui existent depuis longtemps.

L'approche du cloud computing s'appuie principalement sur le concept de virtualisation. Ce concept est un ensemble de techniques permettant de faire fonctionner sur une seule machine plusieurs systèmes d'exploitation et/ou plusieurs applications, isolés les uns des autres. Un cloud est constitué d'un ensemble de machines virtuelles qui utilisent la même infrastructure physique.

Les techniques de virtualisation permettent notamment la migration de machines virtuelles d'un nœud physique á un autre. En fait, les objectifs recherchés á travers la migration de machines virtuelles sont multiples: La répartition de charge, l'accès aux ressources, la tolérance aux pannes ou encore pour accomplir certaines activités de maintenance.

L'instar des autres services informatiques, les services de Cloud ne sont pas à l'abri d'une défaillance. En plus dans des tels systèmes le taux de pannes croit avec la taille du système lui-même. Dans un tel contexte, la sureté de fonctionnement des applications est un élément de première importance, c'est pourquoi un mécanisme de tolérance aux pannes devient nécessaire pour en assurer certains aspects.

La tolérance aux pannes dans les systèmes répartis est un domaine qui a été très largement étudié depuis une trentaine d'années. La littérature propose aujourd'hui un grand nombre de protocoles de tolérance aux pannes par recouvrement arrière, que l'on peut diviser en deux catégories: les protocoles par points de reprise et les protocoles par journalisation. Chacune de ces catégories présente des propriétés différentes en termes de performance, et n'est parfois pas applicable selon le système ou selon l'application considérée.

Notre contribution consiste à proposer une approche de tolérance aux pannes permettant de masquer les défaillances avec le minimum de surcharge dans un système spécialisé dans les BoTs (Bags of Tasks). Notre approche utilise le checkpointing pour assurer la continuité des services sans violer les contraintes SLA (service level agrements). [1]

Dans la première partie de ce manuscrit nous définissons l'environnement du cloud computing. Dans la seconde partie nous étudions les techniques de tolérances aux pannes et nous focalisons sur les points de reprises (Checkpointing). Nous terminons avec une conclusion.

CHAPITRE 1: Cloud Computing

Le Cloud Computing est un nuage de services et de données. Plus précisément, c'est un paradigme, et à ce titre, il est difficile de lui donner une définition exacte et de dire avec certitude s'il s'agit ou non de Cloud. Ce chapitre propose un ensemble des définissions du cloud et donne une description générale sur ses différentes composants et architectures.

1.1 Définition du cloud computing

Le Cloud Computing est l'association de différents concepts architecturaux, techniques et économiques.

- **Selon I. Foster**[2] : le Cloud Computing est un paradigme informatique distribuée qui est conduit par des échelles économiques, dans lequel un pool de ressources virtualisées, une évolutive dynamique, une puissance de calcul, un moyen de stockage, des plates-formes et des services sont fournis à la demande, à des clients externes sur Internet.
- **Pour le Syntec**[3] : le Cloud Computing consiste en « une interconnexion et une coopération de ressources informatiques, situées au sein d'une même entité ou dans diverses structures internes, externes ou mixtes, et dont le mode d'accès est basé sur les protocoles et standards Internet ». « [il permet de] disposer d'applications, de puissance de calcul, de moyens de stockage […] comme autant de « services ». Ceux-ci seront mutualisés, dématérialisés (donc indépendants de toutes contingences matérielles, logiciels et de communication), contractualisés (en terme de performances, niveau de sécurité, coûts), évolutifs (en volume, fonction, caractéristique) et en libre-service.».
- **Selon le Burton Group (racheté récemment par Gartner)**[3] : le Cloud Computing regroupe « l'ensemble des disciplines, technologies et modèles d'entreprise utilisé pour fournir des capacités informatiques (logiciels, plates-formes, matériels) à la manière d'un service à la demande, évolutif et élastique. ».

En octobre 2011 **le « National Institute of Standards and Technology » (NIST)**[4]a publié la version finale de sa définition du Cloud Computing. Selon le NIST donc, le Cloud Computing doit posséder 5 caractéristiques essentielles :

- Le service doit être en libre-service à la demande.
- Le service doit être accessible sur l'ensemble du réseau.
- Le service doit avoir une mutualisation des ressources.
- Le service doit être rapidement élastique.
- Le service doit être mesurable.

C'est cette définition que nous allons détailler. Excepté que nous ajouterons la notion de « paiement à la consommation », qui est finalement le pendant commercial de la caractéristique d'élasticité du Cloud, mais qui n'apparaît pas dans la définition du NIST (Voir Figure 1.1).

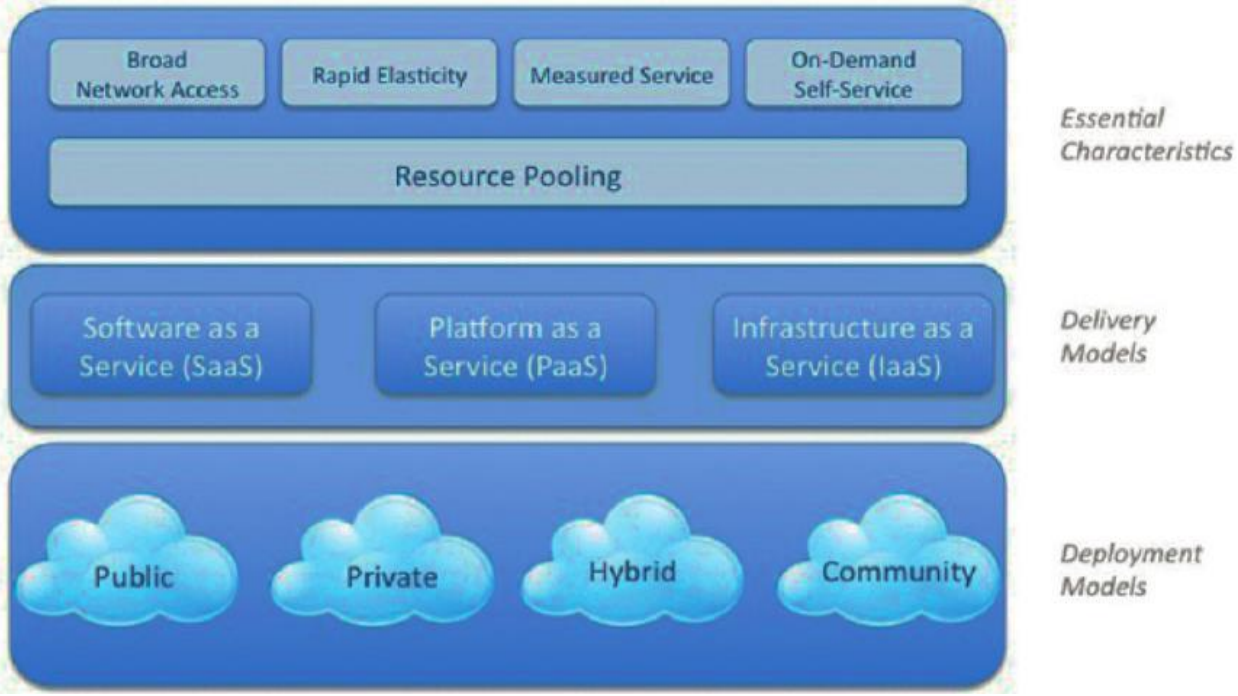


Figure 1.1 - Cloud Computing selon le NIST. [4]

1.2 Virtualisation dans les clouds

La virtualisation a été la première pierre vers l'ère du Cloud Computing. En effet, cette notion permet une gestion optimisée des ressources matérielles dans le but de pouvoir y exécuter plusieurs systèmes virtuels a sur une seule ressource physique et fournir une couche supplémentaire d'abstraction du matériel.

Le principe de virtualisation permet aussi d'être plus flexible dans l'allocation des ressources, par exemple si une machine virtuelle manque de ressources car le serveur physique sur lequel elle repose n'est pas ou plus assez puissant il est dès lors très simple de virtualiser cette machine virtuelle sur un autre serveur plus puissant. [5]

1.3 Caractéristiques du cloud computing

Les caractéristiques du cloud computing sont [5] :

- **Auto-guérison :** Tout système Cloud doit contenir une ou plusieurs copies de chaque application déployée en lui, de telle façon qu'en cas de disfonctionnement de l'application en cours, l'application en copie vient la remplacer en prenant l'état actuel de l'application en échec. Les applications en copies doivent être maintenues et mise à jours à chaque fois que l'application en cours est modifiée.
- **Multi location (Multi-Tenancy) :** Sur le Cloud une même application peut être utilisée par plusieurs clients en même temps, en préservant la sécurité et les données privées de chaque client. Cela est possible en utilisant des outils de virtualisation qui permettent de partager un serveur sur plusieurs utilisateurs.
- **Evolutivité linéaire :** Un système basé Cloud computing technologie a la faculté de découper les principales taches du système en un ensemble de petits morceaux, et de les distribuer sur l'infrastructure virtuelle du Cloud.

- **Service orienté :** Tout système Cloud est basé sur un ensemble de services indépendants les uns des autres. Ce qui donne de la puissance à la technologie Cloud c'est le rassemblement de ces services en une seule unité afin de présenter un service Cloud qui couvre tous les niveaux sur lesquels une application est basée (Niveau virtuel, niveau logiciel, une interface facile à utiliser …. Etc.)
- **SLA (Service Level Agreement) :** OU « accord de service »Avec les services Cloud, un client peut négocier le niveau de service qu'il lui convient et il doit payer pour cela. Dans le cas où les ressources Cloud sont en surcharge, le système crée d'autres entités d'applications Cloud en utilisant les outils de virtualisation disponible afin de respecter les termes du contrat SLA.
- **Virtualisation :** Un système basé sur le cloud computing est un système complètement virtuel et indépendant de la couche physique sur laquelle les ressources et applications sont déployées.
- **Flexibilité :** Les services Cloud sont destinés à supporter les lourdes comme les petites charges de travail en augmentant ou réduisant automatiquement les ressources utilisées selon les tailles des taches à traiter.

1.4 Modèles de services du cloud

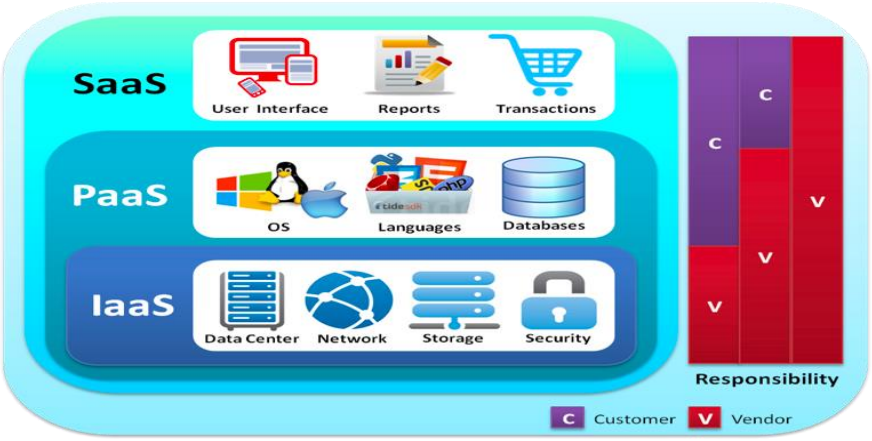


Figure 1.2- Les Modèles de Service du Cloud Computing. [3]

Le Cloud Computing est représenté en 3 composantes principales (Voir Figure 1.2). Chacune d'elle externalise avec une part variable les briques applicatives. Par ordre croissant du niveau d'externalisation : [4]

- **IaaS** (Infrastructure as a Service): Concerne les serveurs, moyens de stockage, réseau … Le modèle IaaS consiste à pouvoir disposer d'une infrastructure informatique hébergée. Ainsi une entreprise pourra par exemple louer des serveurs Linux, Windows ou autres systèmes, qui tourneront en fait dans une machine virtuelle chez le fournisseur de l'IaaS.
- **PaaS** (Platform as a Service): Concerne les environnements middleware, de développement, de test,… Le modèle PaaS consiste à mettre à disposition un environnement prêt à l'emploi, fonctionnel et performant (logiciel serveur, base de données, stockage etc…).
- **SaaS** (Software as a Service): Concerne les applications d'entreprise: CRM, outils collaboratifs, messagerie, BI, ERP,… Le modèle SaaS permet de déporter une application chez un tiers. Le

terme SaaS évoque bien un service dans le sens où le fournisseur vend une fonction opérationnelle, et non des composants techniques requérant des compétences informatiques.

Les avantages et les inconvénients de chaque service sont résumés dans le tableau 1.1 :

	avantage	inconvénient
Saas	-pas d'installation -plus de licence -migration	-logiciel limité -sécurité -dépendances des prestataires
PaaS	-pas d'infrastructure nécessaire -pas d'installation -environnement hétérogène	-limitation des langages -pas de personnalisation dans la configuration des machines virtuelles
IaaS	-administration -personnalisation -flexibilité d'utilisation	-sécurité -besoin d'un administrateur système

Tableau 1.1- Avantages et Inconvénients des services cloud. [4]

1.5 Modèle de déploiement du cloud

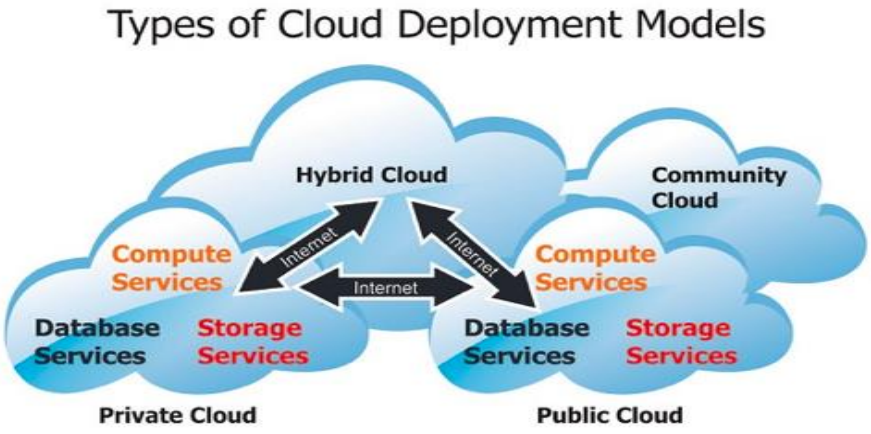


Figure 1.3- Les Modèle de déploiement Cloud Computing. [4]

Un cloud correspond à une infrastructure distante, dont on ne connaît pas les détails architecturaux, et qui est connue pour les services informatiques qu'elle offre. Aussi, il est courant d'utiliser le terme un nuage pour désigner l'infrastructure gérée par un prestataire donné. On pourra alors parler du nuage d'Amazon,

de celui de Google, et ainsi de suite. On peut distinguer quatre types principaux de modèles de déploiement pour ces nuages : le nuage privé, le nuage communautaire, le nuage public et le nuage hybride (Voir la figure 1.3). [6]

- **Le nuage privé** : L'infrastructure d'un nuage privé n'est utilisée que par un unique client. Elle peut être gérée par ce client ou par un prestataire de service et peut être située dans les locaux de l'entreprise cliente ou bien chez le prestataire, le cas échéant. L'utilisation d'un nuage privé permet de garantir, par exemple, que les ressources matérielles allouées ne seront jamais partagées par deux clients différents.
- **Le nuage communautaire** : L'infrastructure d'un nuage communautaire est partagée par plusieurs organisations indépendantes et est utilisée par une communauté qui est organisée autour des mêmes besoins, vis-à-vis de son utilisation. Par exemple, dans le projet Open Cirrus, le nuage communautaire est partagé par plusieurs universités dans le cadre d'un projet scientifique commun. Son infrastructure peut être gérée par les organisations de la communauté qui l'utilise ou par un tiers et peut être située, soit au sein des dites organisations, soit chez un prestataire de service.
- **Le nuage public** : L'infrastructure d'un nuage public est accessible publiquement ou pour un large groupe industriel. Son propriétaire est une entreprise qui vend de l'informatique en tant que service.
- **Le nuage hybride** : L'infrastructure d'un nuage hybride est une composition de deux ou trois des types de nuages précédemment cités. Les différents nuages qui la composent restent des entités indépendantes à part entière, mais sont reliés par des standards ou par des technologies propriétaires qui permettent la portabilité des applications déployées sur les différents nuages. Une utilisation type de nuage hybride est la répartition de charge entre plusieurs nuages pendant les pics du taux d'utilisation.

1.6 Bénéfices et Limites du Cloud Computing

Chaque technologie possède des avantages et des limites : [6]

1.6.1 Bénéfices du Cloud Computing

L'adaptabilité des infrastructures informatiques au besoin des entreprises avec une souplesse dans les 2 sens (plus ou moins de puissances, de stockage etc..) est le principal avantage du Cloud Computing.

Les bénéfices sont multiples et différents selon la population concernée.

a) **Pour les responsables informatiques, les atouts sont essentiellement financiers et humains**

- Investissement initial faible: pas de serveur, pas de logiciel à installer, pas de réseau informatique à étendre etc…
- Le Cloud est une charge d'exploitation et non une immobilisation comptable qui ampute les capacités d'investissement.
- Maîtrise des coûts: ne payant qu'à l'usage, le coût est proportionnel et prévisible dans le temps. La mise à jour des versions est transparente (inclus dans l'abonnement) et le support optimisé.
- Flexibilité : allocation dynamique des ressources pour faire face aux pics de charges.

b) **Pour les entreprises clientes**

Il s'agit pour elles de rester concentrer sur leur métier et de disposer d'un système d'information toujours disponible grâce aux engagements que prendra le prestataire en termes de qualité de service (Service Level Agreement ou SLA).

c) Pour les utilisateurs finaux

- Adaptation des services à chaque profil.
- Accès aux services de n'importe quel poste de travail de l'entreprise voire de l'extérieur de l'entreprise et, pour certaines applications nomades, depuis un Smartphone.

1.6.2 Limites du Cloud computing

- Confidentialité et Sécurité limitées.
- Perte de cohérence.
- Perte de performances (accès disque, gestion des pannes possibles, latence réseau entre serveurs).
- Risque juridiques liés à la localisation (d'où la proximité avec l'hébergeur).

1.7 Conclusion

Le Cloud Computing est promis à un bel avenir, comme le montre les chiffres de ce mémoire. Il est le résultat de l'évolution de nos usages, de l'apparition de la bulle internet et de la virtualisation. Il n'a cessé de s'enrichir depuis les années 60 et arrivera certainement à maturité dans les années à venir, imitant le modèle de ses années, les mainframes et le client/serveur.

Puisque les contraintes SLA contrôlent la relation entre les clients et les fournisseurs des clouds [2], la gestion de tolérance aux pannes devienne nécessaires pour assurer la continuité des services et la satisfaction des clients. Dans le prochain chapitre, nous présenteront un ensemble des concepts et des techniques de tolérances aux pannes.

CHAPITRE 2 : Tolérance aux pannes

La tolérance aux fautes dans les systèmes distribués est un thème de recherche récurrent à cause de sa dépendance avec les différents services de ces systèmes, les nouvelles technologies et les exigences des utilisateurs en matière de disponibilité, de sécurité et de fiabilité. Avec l'émergence de la technologie des cloud computing, la tolérance aux fautes est devenue une de ses importantes propriétés car la fiabilité de ses ressources ne peut pas être totalement garantie.

Dans ce chapitre nous présentons les différentes concepts liées aux tolérances aux pannes ainsi les différentes techniques et stratégies utilisées pour la gestion de fiabilité.

2.1 Notion de sûreté de fonctionnement

La sûreté de fonctionnement d'un système peut se définir selon *Jean-Claude Laprie* dans son guide de la sûreté de fonctionnement, « la propriété qui permet aux utilisateurs du système de placer une confiance justifiée dans le service qu'il leur délivre ». Un système sûr de fonctionnement doit assurer la continuité du service quel que soit les défaillances survenues au cours de son exécution. Cela implique la réparation des fautes ou pannes, qui sont les conséquences de la non-fiabilité du système. Cette fiabilité doit garantir la sécurité. Toutes ces conditions vont assurer la disponibilité des ressources du système réparti, propriété de base des environnements distribués (Voir Figure 2.1).

Les fautes étant les causes des défaillances, la sûreté de fonctionnement cherche à les combattre, si possible en évitant qu'elles se produisent (prévention), ou en les éliminant (élimination).

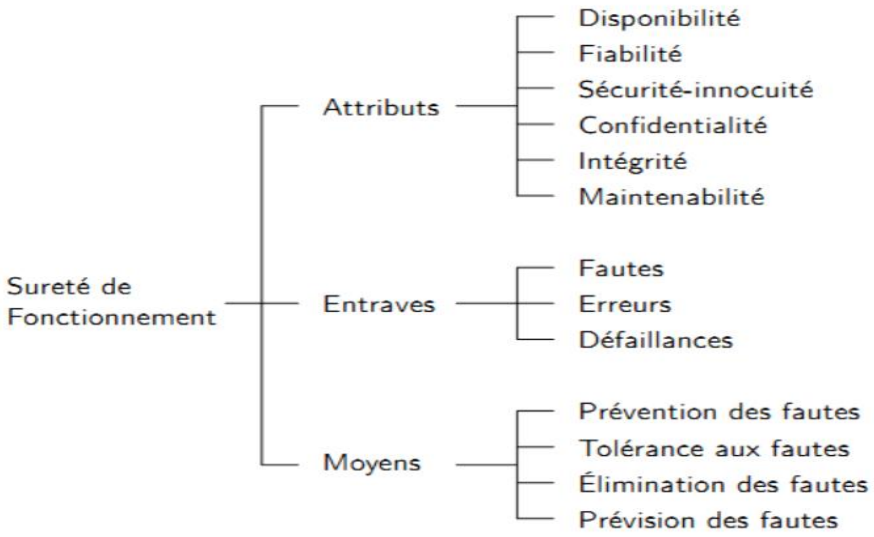


Figure 2.1- Arbre de la sûreté de fonctionnement. [7]

2.1.1 Entraves à la sûreté de fonctionnement

Les fautes, **les erreurs** et **les défaillances** représentent les entraves à la sûreté de fonctionnement. Une erreur est un état susceptible d'entraîner une défaillance.

La faute est la cause supposée d'une erreur.

La défaillance est la manifestation d'une erreur.

La Figure 2.2 montre la relation de causalité qui existe entre les fautes, les erreurs et les défaillances. Par propagation, plusieurs erreurs peuvent être générées avant qu'une défaillance ne se produise.

Lorsque le système est défaillant, il n'est plus apte à assurer le service défini par l'utilisateur. En empêchant ainsi une erreur de se propager, on améliore la sûreté de fonctionnement d'un système tout en tolérant les fautes. [7]

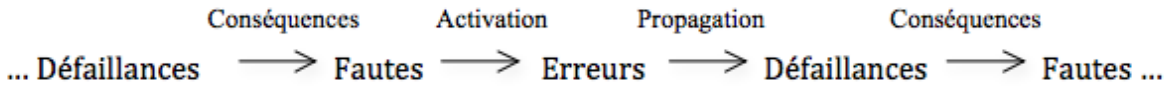


Figure 2.2 - Entraves à la sûreté de fonctionnement.

2.1.2 Attributs de la sûreté de fonctionnement

Pour évaluer la sûreté de fonctionnement d'un système (voir Figure 2.1), les six propriétés de base doivent être vérifiées pour garantir la sûreté : [7] [8]

- **Fiabilité:** Elle évalue la continuité du service délivré par le système et défini par l'utilisateur.
- **Disponibilité:** Pour qu'un système soit sûr de fonctionnement, il faut que les ressources soient disponibles à tout moment.
- **Maintenabilité**: C'est la capacité du système à subir des réparations et des modifications.
- **Sécurité-innocuité:** Elle doit permettre la résistance aux pannes catastrophiques sur l'utilisateur et l'environnement.
- **Confidentialité**: La notion de partage dans les systèmes impose la confidentialité sur toutes les informations en mettant en place des mécanismes d'authentification
- **Intégrité :** C'est un attribut qui permet d'évaluer l'aptitude du système à éviter les altérations de son état.

2.1.3 Moyens d'assurer la sûreté de fonctionnement

Pour que le système vérifie les attributs nécessaires à la sûreté de fonctionnement, des moyens ont été définis dans [8] comme le montre l'arbre de la sûreté de fonctionnement:

- **L'élimination des fautes**: Méthodes de recherche et de suppression des fautes de conception et développement
- **La prévision des fautes**: Analyse de la fréquence ou du nombre de fautes et de la gravité de leurs conséquences
- **L'évitement des fautes**: Méthodes destinées à empêcher l'occurrence même des fautes
- **La tolérance aux fautes ou tolérance aux pannes**: Mécanismes au sein du système destinés à assurer les propriétés de SdF (sûreté de fonctionnement) voulues en présence des fautes.

Dans le cas de notre étude, nous nous intéressons à la tolérance aux fautes. L'objectif de la tolérance aux fautes est la préservation de la continuité du service malgré l'apparition de fautes.

2.1.4 Classification des pannes

Dans les environnements répartis, il existe plusieurs types de fautes classés selon le degré de défaillances:

Les fautes peuvent être:

- **Fautes franches ou par arrêt total** : C'est le modèle de panne le plus simple ; le système arrête subitement de fonctionner et se retrouve dans un état « incorrect ».
- **Fautes par omission** : Ce type de faute survient lorsque des messages sont perdus.
- **Fautes par valeur :** Elles concernent essentiellement les résultats produits par les composants qui peuvent avoir des valeurs incorrectes.
- **Fautes byzantines** : Elles sont très complexes parce que le système continue en présence de messages qui ne suivent pas sa spécification.
- **Fautes par temporisation** : Le comportement erroné du système est dû au temps.

Le classement de ces fautes selon le degré de complexité est illustré dans la Figure 2.3.

Figure 2.3 - Les types de fautes classés selon le degré de défaillances.

[8]

2.2 Stratégies de détection de pannes

Il existe deux classes de détection de pannes : [8]

- **Tolérance aux pannes réactive**

  Les politiques de tolérance de panne réactifs est de réduire l'effet des défaillances sur l'exécution de l'application lorsque la panne survient efficacement. Il existe différentes techniques qui sont basés sur ces politiques comme points de reprise et le redémarrage (*Checkpointing/Restart*), Réplication, Re-soumission (*Task Resubmission)*…

- **Tolérance aux pannes proactive**

  Le principe de politiques de tolérance de panne proactives est d'éviter la reprise de défauts, les erreurs et les échecs en prédisant et de remplacer de manière proactive les composants suspects avec d'autres composants qui fonction. Certaines des techniques qui sont basées sur les politiques de tolérance aux pannes proactive sont Fault Tolerance proactive en utilisant la migration préemptive (*Preemeptive Migration*), rajeunissement de logiciel (*Software Rejuvenation*), l'équilibrage de charge, etc.

2.2.1 Modèles de détection des pannes réactive

Le problème de la détection des pannes est résolu différemment selon le modèle de communication du système. On distingue deux modèles différents :

- **Le modèle push**, dans lequel chaque processus du système doit régulièrement informer les détecteurs de pannes de son état.

Si ce détecteur n'a pas reçu de message de type " **je suis vivant** " de la part d'un processus depuis un temps donné, ce processus est suspecté d'être défaillant.

- **Le modèle pull**, dans lequel ce sont les détecteurs de pannes qui envoient régulièrement des requêtes de type "**es-tu vivant?**" aux processus du système. Un processus qui ne répond pas dans un temps donné est suspecté d'être défaillant.

2.3 Techniques de tolérance aux pannes dans les systèmes répartis

La tolérance aux fautes est l'aptitude d'un système informatique à accomplir sa fonction malgré la présence ou l'occurrence de fautes, qu'il s'agisse de dégradations physiques du matériel, de défauts logiciels, d'attaques malveillantes, d'erreurs d'interaction homme-machine.

La tolérance aux pannes dans les systèmes répartis et parallèles est le plus souvent réalise par l'emploi d'un mécanisme de redondance ou reprise. Les mécanismes de redondances mis en œuvre appartiennent à deux catégories: les techniques basées sur la duplication et les techniques basées sur une mémoire stable.

La figure 2.4 résume les techniques de tolérance aux pannes et leurs types :

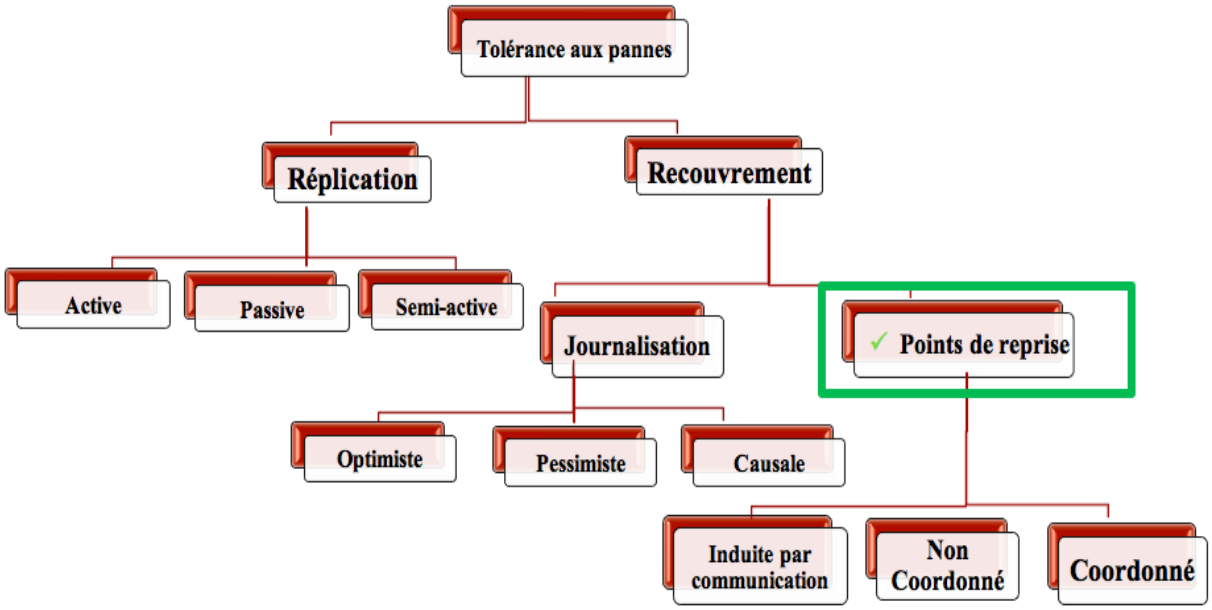


Figure 2.4 - Techniques de tolérance aux pannes dans les systèmes répartis.

2.3.1 Tolérance aux pannes par réplication (Duplication)

La tolérance aux fautes basée sur la réplication consiste à créer des copies multiples des processus sur des machines différentes.

Ils existent trois techniques de réplication proposées dans la littérature : la réplication active, passive et semi-active. [9]

- **La réplication active**

C'est une technique dans laquelle toutes les répliques traitent les mêmes messages d'entrée, mais en gardant leur état étroitement synchronisé pour assurer qu'elles reçoivent les messages dans le même ordre (Voir la Figure 2.5).

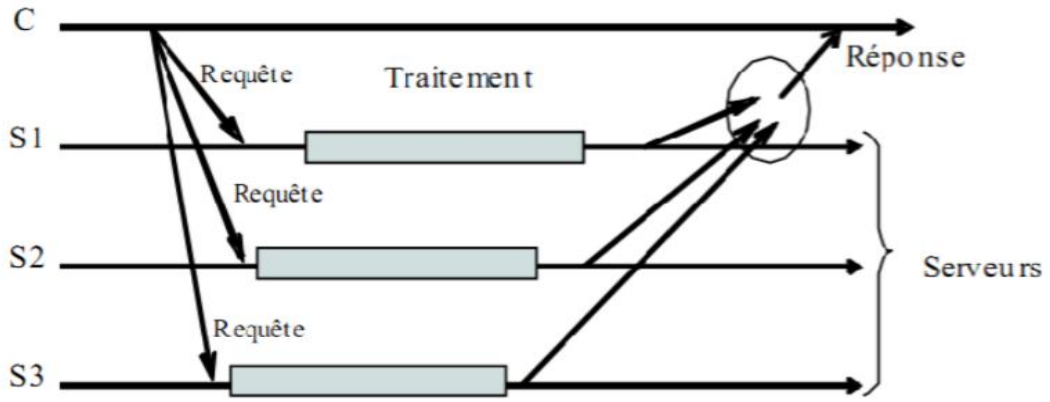


Figure 2.5 - Exemple de protocole de réplication active. [9]

En cas de panne, les erreurs sont masquées par les copies non défaillantes qui assurent le relais. La défaillance d'une copie est masquée par les autres répliques.

- La réplication passive

On distingue deux types de copies de processus dans cette technique : une copie primaire et des copies secondaires. Seule la copie primaire traite les messages d'entrée et fournit les messages de sortie. Pour maintenir la cohérence, la copie primaire transmet par intervalle de temps son nouvel état aux copies secondaires (Voir la Figure 2.6).

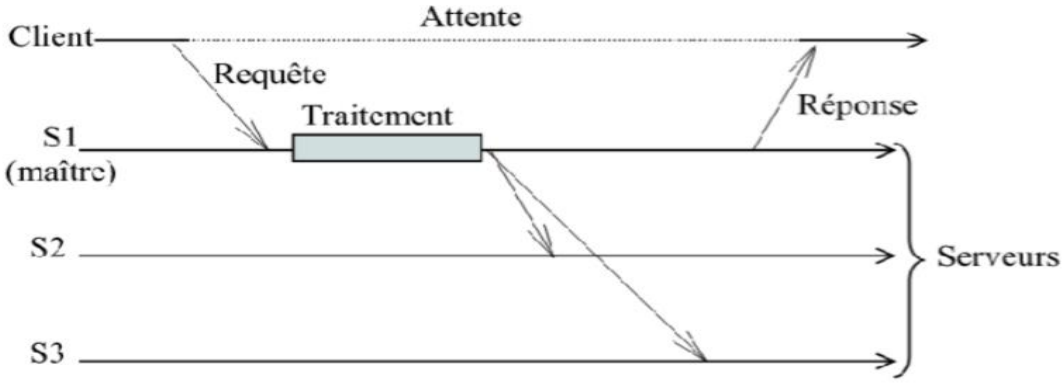


Figure 2.6 -Exemple de protocole de réplication passive. [9]

Lors d'une panne de la copie primaire, une des copies secondaires la remplace et le traitement des messages reçus depuis la dernière sauvegarde est perdu.

- **La réplication semi-active**

Cette technique permet un recouvrement de fautes plus rapide que la réplication passive. Les messages d'entrée sont traités par toutes les copies, mais une seule, le leader, fournit les messages de sortie. Pour mettre à jour leur état interne en l'absence de fautes, les autres répliques traitent directement les messages d'entrée, ou utilisent les « notifications » du leader (Voir la Figure 2.7).

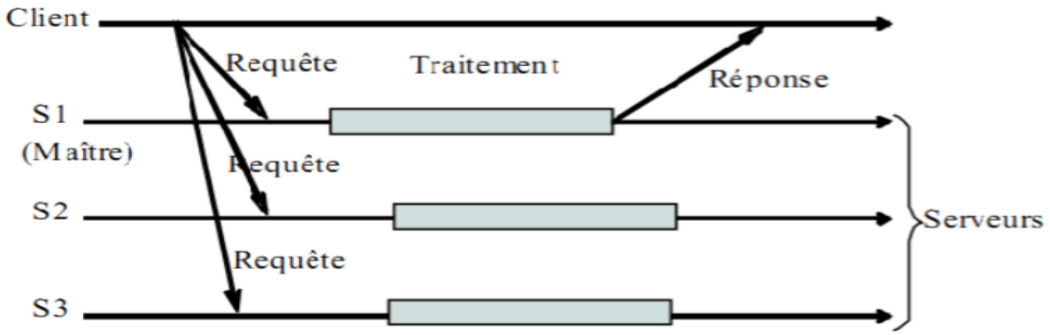


Figure 2.7 - Exemple de protocole de réplication semi-active. [9]

En cas de panne franche du leader une des copies peut ainsi prendre le relais sans perte.

2.3.2 Recouvrement d'erreur

Le recouvrement d'erreur consiste à remplacer un état erroné par un état stable garantissant un système fonctionnel. Il existe deux techniques de recouvrement d'erreur (Voir Figure 2.8).

- **Le recouvrement par reprise** : C'est une technique générale qui ramène le système à un « état correct sans erreur » qu'il occupait (Retour arrière). Elle nécessite la sauvegarde périodique de l'état du système sur support stable.
- **Le recouvrement par poursuite** : Le système est ramené à un nouvel état « reconstitué », sans effectuer de retour arrière. La reconstitution n'est souvent que partielle. C'est une technique spécifique parce que la reconstitution d'état correct dépend de l'application et exige une analyse préalable des différents types d'erreurs possibles.

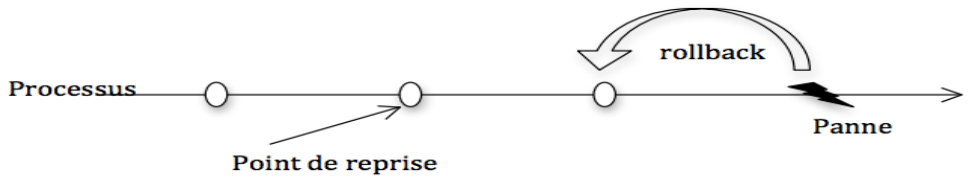


Figure 2.8 - Recouvrement arrière après panne (rollback-recovery)

2.3.2.1 Concepts généraux

Il existe plusieurs concepts à savoir :

- Mémoire stable : Elle peut être définit comme un support persistant de stockage dont le rôle principal est d'assurer une accessibilité et une protection aux données contre les pannes pouvant affecter le système. Ainsi, suite à une panne, un état correct ayant était stocké antérieurement à cette panne sur la mémoire stable reste accessible, cela permet au système le retour à un état antérieur.

- Etat globale : C'est une collection des états locaux de tous les processus participant au calcul et les états des canaux de communication entre les processus. Dans la Figure 2.9 l'état global C1= {x3, y3, z3} et C2= {x2, y2, z2}.

- Recouverabilité : Un état recouvrable est un état stable, c'est-à-dire sauvegarde sur une mémoire stable, depuis lequel une exécution peut repartir correctement.

- Message Orphelin : Est un message qui apparait comme reçu dans l'état d'un processus alors qu'il n'apparait pas dans l'état du processus qui l'a émis. Dans la Figure 2.9, le message *m2* est orphelin par rapport à l'état C1.

- Message en-transite : Est un message qui peut avoir été envoyé mais ne pas avoir été reçu, Un message en transit par rapport à un état global est un message qui a été envoyé avant un point de reprise appartenant à cet état global et reçu après un point de reprise appartenant à cet état global. Dans la Figure 2.9, le message *m1* est en-transite par rapport à l'état C1.

- Effet de domino : Est caractérisé par une cascade de retours arrière lors de la reprise du système après une panne.

- Etat global cohérent: Un état global cohérent est un état qui peut se produire durant une exécution correcte de l'application. Informellement, un état cohérent est un état qui ne contient pas des messages orphelins, car ces derniers correspondent à un état qui ne peut pas se produire lors une exécution correcte. Le rôle du protocole de tolérance aux fautes est de reconstruire un état global cohérent à partir de l'état potentiellement incohérent du système après une panne. Dans la Figure 2.9, C2 est un état global cohérent.

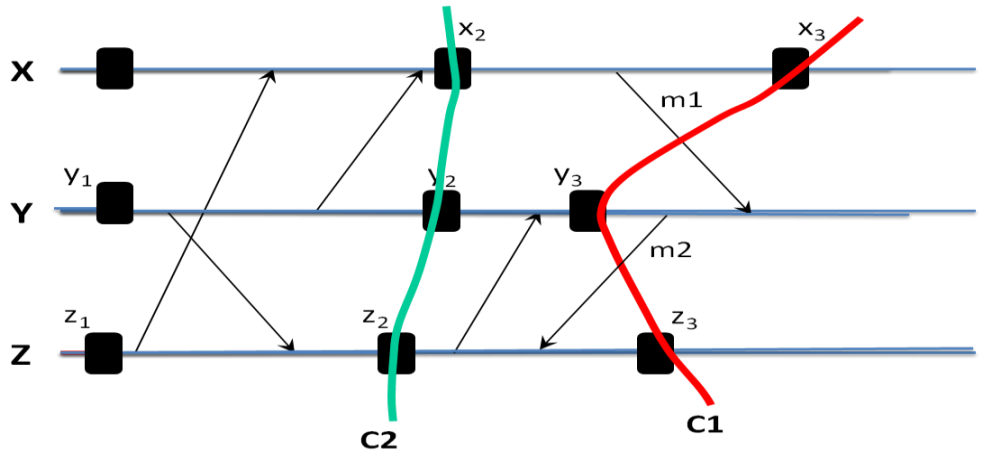


Figure 2.9 - État global cohérent

2.3.2.2 Tolérance aux pannes par sauvegarde (checkpointing)

Par définition, un point de reprise (checkpoint) est la sauvegarde de l'état d'un processus à un instant donné, sur un support de stockage stable. Un support stable est un espace de stockage accessible par toutes les applications utilisées dans les clouds disposant des droits suffisants, et protégé contre toute sorte de pannes. Il est souvent représenté par un serveur de stockage.

La tolérance aux fautes par recouvrement arrière se base sur les protocoles de points de reprise pour éviter la perte de temps de calcul dans les environnements de calcul haute performance comme les clouds. Il existe différentes catégories de protocoles de points de reprise classés selon la méthode utilisée pour effectuer la sauvegarde: les protocoles de points de reprise coordonnés, non-coordonnés et induits par les communications. [10]

i. Points de reprise coordonnés

Ce protocole exige aux processus de coordonner l'établissement de leurs points de reprise pour former un état global cohérent. L'algorithme bloquant se déroule ainsi (Figure 2.10):

- Les communications sont bloquées pendant l'établissement des points de reprise.
- Un coordinateur fait un point de reprise et diffuse un message 'checkpoint-request' à tous les processus pour initier une phase de sauvegarde.
- Quand un processus reçoit ce message, il arrête son exécution et vide ses canaux de communication, fait une tentative de point de reprise et envoie un accusé de réception au coordinateur.
- Avoir reçu l'accusé de réception de tous les processus, le coordinateur diffuse un message 'commit' à tous les processus pour leur ordonner de faire un point de reprise.
- Chaque processus supprime ainsi l'ancien point de reprise, et sauvegarde son état. Le processus est maintenant libre de terminer son exécution et d'échanger des messages avec les autres processus du système.

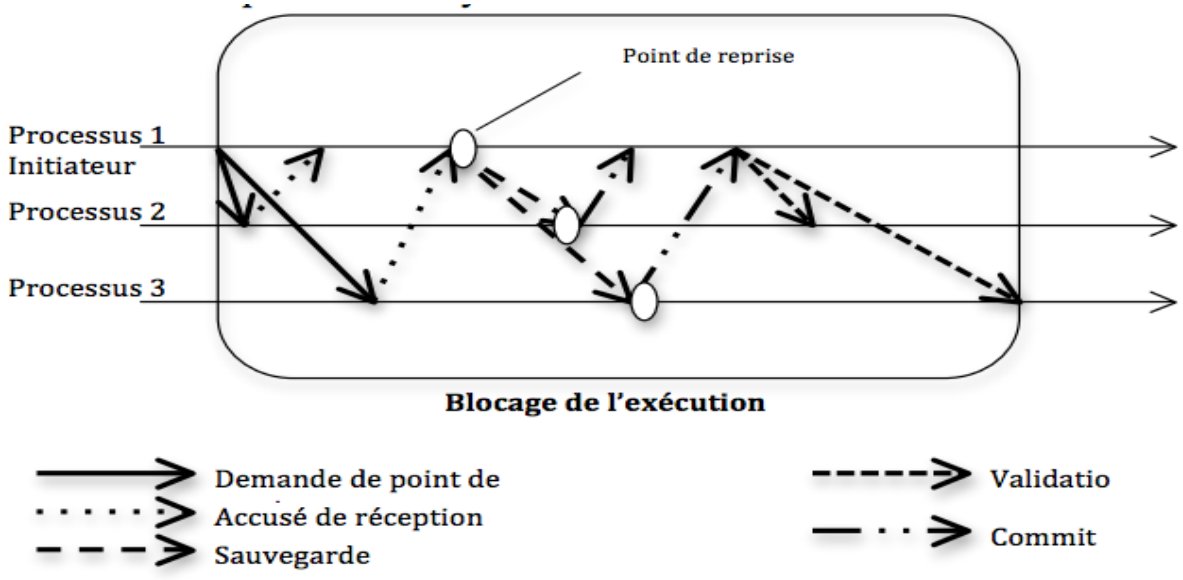


Figure 2.10 - Point de reprise coordonné. [10]

Cette approche simplifie le recouvrement et évite l'effet domino, puisque chaque processus redémarre toujours au point de reprise le plus récent. Aussi, le protocole exige à chaque processus de maintenir un seul point de reprise permanent en mémoire stable, ce qui réduit le surcoût dû au stockage ainsi que la nécessité d'avoir un algorithme de récupération des points de reprise (garbage collection). En pratique, le protocole coordonné est facile à implémenter. Son principal inconvénient est cependant la large latence introduite par la coordination des processus participants. Pour améliorer les performances de la sauvegarde coordonnée, plusieurs techniques ont été proposées :

- **Point de reprise coordonné non-bloquant**: Il fonctionne généralement sous l'hypothèse des canaux FIFO (First In First Out; premier arrivé, premier servi), en utilisant des marqueurs. En effet, lors de la sauvegarde des points de reprise, chaque processus enregistre son état local et envoie sur tous ses canaux de sortie un marqueur pour informer les autres processus voisins qu'il a effectué un point de reprise. À la réception de ce marqueur, un processus recevant ce message pour la première fois sauvegarde à son tour son état et diffuse le marqueur, et ainsi de suite.
- **Point de reprise coordonné minimale**: Ce protocole ne fait intervenir que les processus ayant participé à la dernière sauvegarde. Seuls les processus susceptibles de créer une incohérence en cas de reprise sauvegardent leur état. L'algorithme se déroule en deux phases. Durant la première phase, l'initiateur identifie tous les processus dont il a reçu un message depuis le dernier point de reprise, et leur envoie une demande, le message étant un orphelin potentiel. Sur réception de la demande, chaque processus identifie à son tour tous les processus pouvant créer des messages orphelins et leur envoie aussi une demande, jusqu'à ce que tous les processus soient identifiés. Après l'identification des processus, la deuxième phase correspondant à la sauvegarde des états locaux proprement dite. [3]
- **Point de reprise coordonné avec horloge de synchronisation**: Le protocole utilise des horloges de synchronisation pour effectuer des points de reprise locaux au niveau de tous les processus participants, et approximativement au même moment sans un initiateur. Le protocole assure qu'aucun message n'est échangé durant la sauvegarde.

ii. Points de reprise indépendants

Ce protocole évite la phase de synchronisation en laissant à chaque processus l'autonomie de sauvegarder son état local (Figure 2.11) L'avantage de cette autonomie est que chaque processus pourra sauvegarder son état au moment opportun, par exemple si la quantité d'information est minimale. Ce qui réduit le surcoût du stockage en termes de quantité d'information à sauvegarder. [10]

Les étapes du recouvrement sont:

- Le processus de récupération entame un retour arrière par la diffusion d'un message de 'dependancy-request' pour rassembler les 'informations de dépendance' maintenues par chaque processus.
- Quand un processus reçoit ce message, il arrête son exécution et envoie les informations de dépendances enregistrées sur la mémoire stable.
- L'initiateur calcule la ligne de recouvrement basée sur les 'informations de dépendance' globales et diffuse un message de 'roll back request' contenant la ligne de recouvrement.
- Sur réception de ce message, chaque processus appartenant à la ligne de recouvrement reprend son exécution, sinon il revient au point de reprise précédent, celui indiqué par la ligne de recouvrement (Figure 2.11). La dernière étape de l'algorithme peut causer l'effet domino, qui peut engendrer une perte du travail réalisé avant la panne et la sauvegarde inutile de points de reprise

qui ne feront jamais partie d'un état global cohérent. Le protocole non coordonné fait revenir en arrière le processus concerné par la panne, ainsi que ceux qui en dépendent causalement. Il oblige aussi chaque processus à maintenir plusieurs points de reprise.

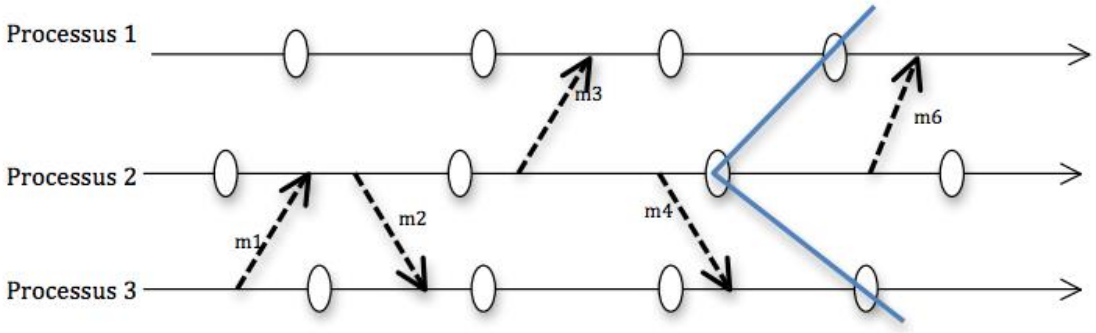


Figure 2.11- Point de reprise indépendant

iii. Points de reprise induits par les communications

Ce protocole (CIC: Communication Induced Checkpointing) définit deux types de points de reprise : [11]

- Des points de reprise locaux pris par les processus de manière indépendante, pour éviter la synchronisation de la sauvegarde coordonnée.
- Des points de reprise forcés en fonction des messages reçus et envoyés et des informations de dépendance estampillées sur ces messages, pour ainsi éviter l'effet domino de la sauvegarde non coordonnée, garantir l'avancement de la ligne de recouvrement.

À l'opposé des protocoles de point de reprise coordonné, le surcoût dû à l'accès au support stable disparaît parce que le protocole CIC ne nécessite aucun échange de message pour forcer un point de reprise. En effet, il utilise la technique utilisée par la journalisation causale en insérant la signalisation sur les messages échangés.

Les techniques de sauvegarde de point de reprise induit par les communications peuvent être classées en deux catégories :

- **Model-based coordination**: Elle repose sur la prévention des modes de communication et de points de reprise qui pourraient entrainer des Z-cycles.
- **Index-based coordination**: Cette catégorie utilise les horloges logiques. En effet, ces horloges sont ajoutées aux messages afin de savoir à quel moment il faut sauvegarder un point de reprise.

2.3.2.3 Analyse comparative des différentes méthodes de point de reprise

Nous allons comparer ici les approches présentés dans la partie précédente. Nous ne considérons que les approches synchronisés et induites par messages, en effet :

L'approche indépendante n'implique pas réellement l'utilisation d'un protocole, la recherche d'un état global cohérent est faite après une panne au moment de la reprise. Nous comparons sur les points suivants : [11]

i. Surcoût pendant l'exécution

Les protocoles synchronisés bloquant ont en général un surcoût élevé durant l'exécution à cause de la phase de synchronisation qui stoppe l'exécution.

ii. Nombre moyen de retours arrière

Les protocoles synchronisés ont ici un avantage certain : tous les points de reprise sont utiles, et lorsqu'n processus prend un point de reprise, il est assuré de ne pas devoir retourner plus loin que ce point en cas de panne.

Dans le cas des protocoles induits par messages, on trouve deux types de solution.

- La première est d'assurer que tous les points de reprise sont utiles, c'est à dire qu'ils font partie d'un état global cohérent. Ces protocoles sont basés sur l'approche model-based.
- La deuxième solution est de maximiser la probabilité tous les points soient utiles, sans pour autant l'assurer. C'est le cas des protocoles index-based.

iii. Surcoût de stockage et ramassage

Sur ce point aussi, les protocoles synchronisés sont avantageux : puisque tout état global nouvellement créé est forcément un état cohérent, alors l'état global précédent peut être effacé de la mémoire. Le surcoût est donc minimal puisqu'il n'est nécessaire de conserver qu'un seul état global (plus bien sûr celui en construction), et le ramassage des états devenus inutiles devient trivial.

2.3.2.4 Tolérance aux pannes par journalisation

Depuis plusieurs décennies, la journalisation des messages a été une technique très utilisée pour assurer la tolérance aux pannes dans les systèmes répartis. Les protocoles de journalisation nécessitent la sauvegarde périodique des états locaux des processus, et l'enregistrement sur support stable de tous les messages reçus après l'établissement des points de reprise locaux.

En cas de panne d'un processus, il est relancé à partir du dernier point de reprise, et tous les messages reçus après ce dernier lui sont renvoyés dans l'ordre où ils ont été initialement reçus. [11]

i. Journalisation optimiste

Ce protocole utilise l'hypothèse selon laquelle la journalisation d'un message sur support fiable sera complète avant qu'une panne ne se produise. En effet, au cours de l'exécution des processus, les déterminants des messages, à savoir leur contenu et identifiants, sont conservés en mémoire volatile avant d'être périodiquement vidés sur support stable.

ii. Journalisation pessimiste

Ce protocole a été conçu dans l'hypothèse qu'une panne peut se produire après n'importe quel événement non déterministe. Il enregistre en mémoire stable le déterminant de chaque message avant que ce dernier ne soit autorisé à interagir avec le système.

iii. Journalisation causale

Ce protocole combine les avantages des deux précédents mécanismes de journalisation. Comme la journalisation optimiste, elle réduit le surcoût à l'exécution dû à l'accès synchronisé au support stable. Comme la journalisation pessimiste, il ne crée jamais de processus orphelin, l'état de chaque processus défaillant est recouvrable à partir de son dernier point de reprise, et il n'y a pas de risque d'effet domino. La journalisation causale permet aux processus d'effectuer des interactions avec l'extérieur de manière indépendante.

2.4 Comparaison de différents protocoles de tolérance aux pannes Par reprise

Le tableau 2.1 propose une comparaison des avantages et des inconvénients de différentes techniques de tolérance aux pannes par reprise:

Protocole	**Hypothèse PWD**	**Processus orphelin**	**Effet Domino**	**Nombre de sauvegardes**
Journalisation optimiste	Oui	Non	Non	Une
Journalisation pessimiste	Oui	Possible	Non	Possible
Journalisation causale	Oui	Non	Non	Une
Sauvegarde coordonnée	Non	Non	Non	Une
Sauvegarde indépendante	Non	Possible	Possible	Toutes
Sauvegarde induite par les communications	Non	Possible	Non	Plusieurs

Tableau 2.1- Comparaison des différentes méthodes de tolérance aux pannes par reprise.

2.5 Conclusion

Dans ce chapitre, nous avons fait un survol sur les différentes techniques de sureté de fonctionnement et de tolérance aux fautes qui datent depuis l'émergence des systèmes distribués. Les politiques de tolérance aux fautes comme la réplication et le recouvrement donnent des résultats très satisfaisants dans les systèmes distribués ordinaires. Le passage à l'échelle de ces systèmes, qui s'est accentué par l'apparition des clusters et plus tard par les cloud de calcul et les cloud computing, a créé des contraintes nouvelles, comme la dynamicité et les contraintes SLA (dans le cas des clouds). Ceci a conduit les chercheurs améliorer les techniques de tolérances aux pannes pour assurer le maximum de fiabilité avec le minimum du coût et de surcharge.

CHAPITRE 3 : Conception

Introduction

Ce chapitre sera consacré à notre approche de tolérance aux pannes basée sur le checkponting dans un environnement Cloud. Nous présenterons l'architecture du protocole proposé ainsi que sa spécification et sa conception. Nous décrirons les algorithmes des différentes procédures de notre approche de tolérance aux fautes basée sur les points de reprise. Le langage de modélisation que nous avons choisi pour la conception de notre solution est UML [14].

3.1. Objectifs du travail

Le Cloud Computing fait référence à l'utilisation des capacités de calcul des ordinateurs

Distants, où l'utilisateur dispose d'une puissance informatique considérable sans avoir à posséder des unités puissantes. L'approche du Cloud Computing s'appuie principalement sur le concept de virtualisation. Où ce concept est un ensemble de techniques permettant de faire fonctionner sur une seule machine plusieurs systèmes d'exploitation et/ou plusieurs applications, isolés les uns des autres. Un Cloud est constitué d'un ensemble de machines virtuelles qui utilisent la même infrastructure physique.

La probabilité d'avoir une panne durant l'exécution devient un phénomène naturel lorsque le nombre de nœuds augmente. Supposons, par exemple, une application distribuée

S'exécutant sur plusieurs nœuds et qui nécessite un temps d'exécution de 3 jours. Si un nœud participant à l'exécution de cette application se bloque à cinq minutes avant la fin de l'exécution, en absence des mécanismes de sauvegarde/reprise (Checkpoint/restart) presque 3 jours d'exécution seront gaspillés [15]. Puis qu'il est impossible d'empêcher totalement les pannes, une solution consiste à mettre en place une stratégie de tolérance aux fautes basée sur le checkpointing.

L'objectif consiste à proposer un mécanisme de tolérance aux pannes pour les clouds computing. Nous avons proposé une approche de tolérance aux pannes permettant de masquer les défaillances avec le minimum de surcharge dans un système spécialisé dans les BoTs (Bags of Tasks). Notre approche utilise le checkpointing pour assurer la continuité des services sans violer les contraintes SLA (service level agrements). Notre approche nommé ACP (Adaptatif Checkpointing) adapte l'intervalle de checkpointing pour améliorer les performances du checkpointing et de système. Les sections suivantes expliquent en détaille l'ACP.

3.2. Stratégie proposée

Cette section décrit l'architecture proposée et la mise en œuvre pour supporter la tolérance aux pannes dans le Cloud Computing.
L'approche proposée est une nouvelle stratégie qui utilise les points de reprise complètement indépendant pour les taches Bots mais avec un intervalle adaptatif qui minimise la surcharge.

3.3. BoTs

Les applications Bag-of-Tasks (BoT) sont:

- Complètement parallèles
- Applications dont les tâches sont indépendantes les unes des autres.

Utilisé dans des divers scénarios, y compris:

- l'exploration de données (data mining)

- La biologie
- Les recherches massives (key breaking),
- Simulations,
- calculs fractales,
- l'imagerie par ordinateur

A cause de l'indépendance des tâches Bots, les applications BoT peuvent être exécutés avec succès sur des systèmes distribués à grand échelle.

3.4. Architecture de système

La structure de notre architecture (voir **Figure 3.1)** utilise un groupe de modules qui travaillent en coopération afin de contrôler d'une manière distribuée les défaillances et de déclencher le processus de tolérance aux pannes.

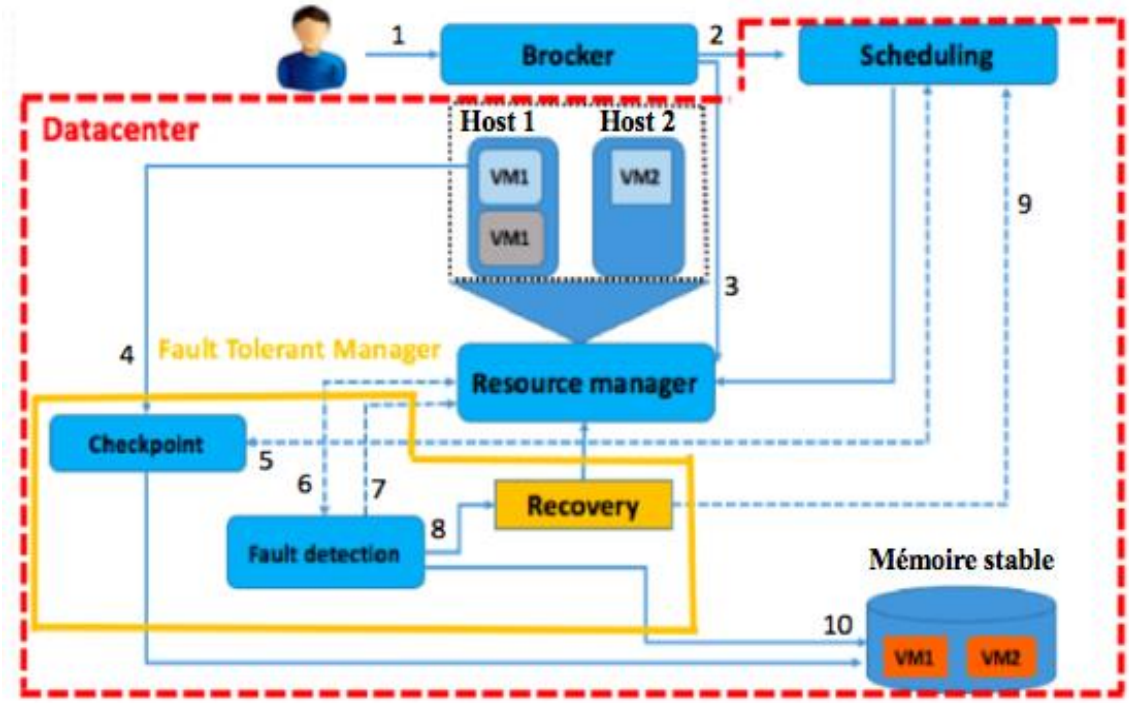


Figure 3.1 -Schéma du système

Les composants de notre système sont:

- **Data Center :** Un Data Center est une entité informatique composée d'un ensemble de machines physiques. Il est caractérisé par le nombre et la vitesse des processeurs, la capacité de la mémoire et de stockage, la bande passante ainsi que le coût de chaque ressource.
- **Broker :** Le Broker assure la gestion des VMs dans plusieurs Data Centers et le routage du trafic vers les Data Centers appropriés. Il choisit aussi le Data Center qui fournit le meilleur service pour les demandes envoyées par chaque utilisateur. Le Broker est responsable de la communication entre les utilisateurs et les Data Centers.
- **Physical node :** Une machine physique (hôte) est caractérisée par le nombre des VMs, la capacité de stockage, le nombre des requêtes arrivées, la vitesse du CPU et le nombre des cores.

- **VMs :** Une machine virtuelle consiste à créer plusieurs environnements d'exécution sur une seule machine physique. Elle fournit à chaque utilisateur un service selon la demande.
- **Resource manager:** Fournit les ressources nécessaires pour l'exécution des taches selon les contraintes SLA du broker et l'utilisateur.
- **Scheduler :** Responsable de faire l'ordonnancement des taches fournit par le broker. L'ordonnancement des taches est un processus qui "mappe" et gère l'exécution de tâches interdépendantes sur des ressources distribuées fournit par le ressource manager. Il alloue des ressources appropriées pour les tâches de sorte que l'exécution puisse être achevée, tout en satisfaisant les objectifs et contraintes imposés par l'utilisateur.
- **Fault tolérance manager: il contient 3 sous modules :**
- **Checkpointer :** c'est le gestionnaire de checkpointing, il est responsable de créer et stocker les checkpoints dans la mémoire stable. Il calcule aussi l'intervalle de checkpointing périodiquement pour minimiser la surcharge.
- **Recovery manager:** Il est responsable au lancement de la procédure de reprise qui correspond à redémarrer les tâches depuis leur dernier point de reprise (Checkpoint) stocké dans la mémoire stable.
- **Fault detector:** responsable de surveiller et détecter les pannes dans le système.

3.4.1. Gestionnaire de tolérance aux pannes (Fault tolérance manager):

Dans cette section nous détaillons chaque module de ce service pour bien comprendre le fonctionnement de notre approche.

3.4.2. Checkpointer

Le module de Checkpoint est chargé de sauvegarder périodiquement une copie d'état des machines virtuelles. Faire un Checkpoint de la tâche qui s'exécute dans une machine virtuelle doit inclure toutes les informations nécessaires pour reprendre l'exécution de la tâche dans un autre nœud. Ce qui nécessite la sauvegarde de l'état courant des tâches (essentiellement sa mémoire et le contenu du disque) (voir la **Figure 3.2**).

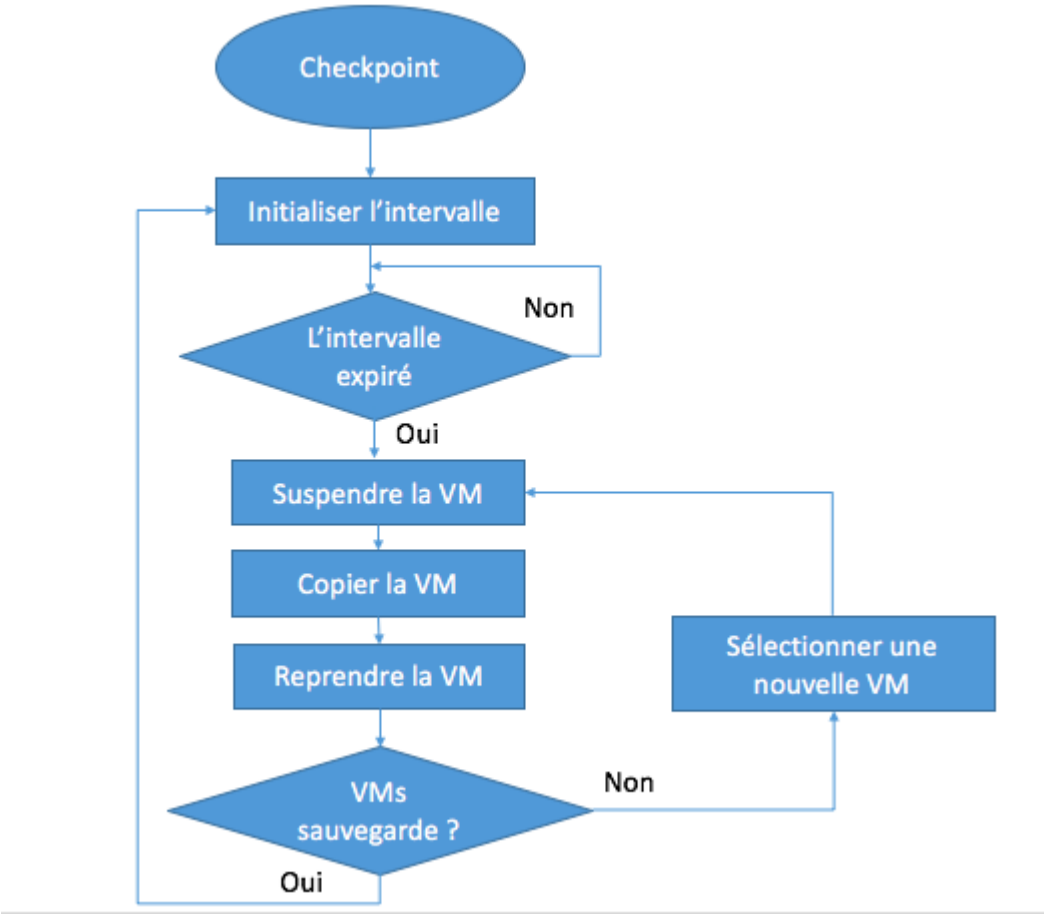


Figure 3.2 - Organigramme d'activité du checkpointing

L'une des questions les plus importantes lors de la mise en œuvre du Checkpointing concerne le choix de sa fréquence, en d'autres termes, le choix de l'intervalle entre les moments de création de deux Checkpoints successifs. Ce choix doit se faire de telle sorte à ce que le coût du Checkpointing sur l'exécution des VMs soit réduit tout en offrant un niveau de fiabilité suffisant.

L'intervalle de checkpointing peut être statique ou dynamique :

- Statique : la longueur d'intervalle est inchangeable dès le début.
- Dynamique : la longueur d'intervalle de checkpointing change durant l'exécution d'application.

L'intervalle de checkpointing peut être aussi fixe et calculable :

- Fixe : l'intervalle de checkpointing est donné donnée par une valeur fixe.
- Calculable : l'intervalle de checkpointing est calculé par l'utilisation de plusieurs paramètres.

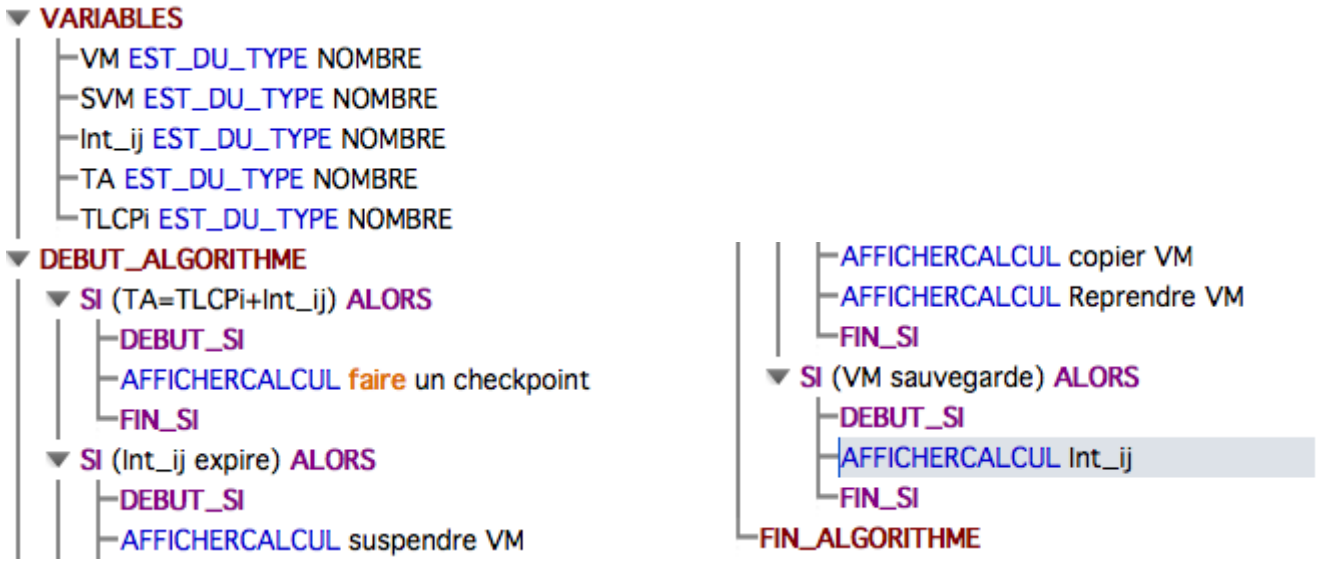
Notre intervalle de checkpointing est de type dynamique et calculable. Dans ce cas ; l'intervalle de checkpointing est recalculé périodiquement pour s'adapter avec le changement du système en terme de coût et de fiabilité et même des performances. Après chaque checkpointing, le gestionnaire de checkpointing calcule le paramètre Scp (Voir la Formule (3.1):

$$Scp(j, VM_i) = \frac{TP(VM_i)}{\alpha C(VM_i) + \beta E(VM_i) + \gamma F(VM_i)} \quad (3.1)$$

Où:

- α, β, γ : Facteurs de poids et $\alpha + \beta + \gamma = 1$
- J : le nombre des tours de checkpointing (Rounds)
- $TP(VM_i)$: le travaille effectué depuis le dernier checkpointing (Round j-1) jusqu'à le temps actuel
- $C(VM_i)$: le coût nécessaire pour la création checkpoint dans VM_i
- $E(VM_i)$: l'énergie nécessaire pour la création du checkpoint dans VM_i
- $F(VM_i)$: la fiabilité actuel du VM_i

Si Scp dépasse un certain seuil, le gestionnaire de checkpointing crée un checkpoint pour la VM_i. Cette stratégie favorise les VMs qui ont exécutées beaucoup de travaille qui risque d'être perdus en cas de panne et qui ont le minimum de fiabilité, du coût et d'énergie. Le seuil de checkpointing est défini par tâtonnement à partir d'historique.



Algorithme de checkpoint

Où :
TA : Temp Actuel.
TLCPi : Temp de dernier checkpoint pour VMi.
Int_ij : Intervalle de checkpointing de round j pour VMi

3.4.3. Fault detector

La détection des pannes par l'approche proposée est réalisée par l'envoi des messages de vie (heartbeat) et watchdog [17]. Le FD (Fault detector) attend les messages de vie de chaque VM j. Si le message de vie d'une VM i n'arrive pas après un certain temps, la VM i sera déclaré en panne et ajouter à la liste des VMs en pannes.

Une VM envoie régulièrement des messages de vie à FD. Le cycle de heartbeat/ watchdog détermine la rapidité de FD pour détecter une défaillance d'une VM, c'est à dire, le temps de réponse du système de détection de panne. Afin de ne pas surcharger le système par des messages de heartbeat inutiles ou retarder la détection des pannes, le cycle de heartbeat/ watchdog dépends du degré de fiabilité F_i du VM_i. La fréquence d'envoie des heartbeat augmente si F_i augmente. Dans ce cas, si la probabilité de panne d'une VM augmente, elle doit envoyer plus de heartbeat messages pour une détection rapide des pannes.

La détection des pannes se fait en deux parties (voir **Figure 3.3**). La première partie consiste à envoyer des messages de vie de VM et la deuxième partie consiste à attendre les messages de vie des VM (coté FD).

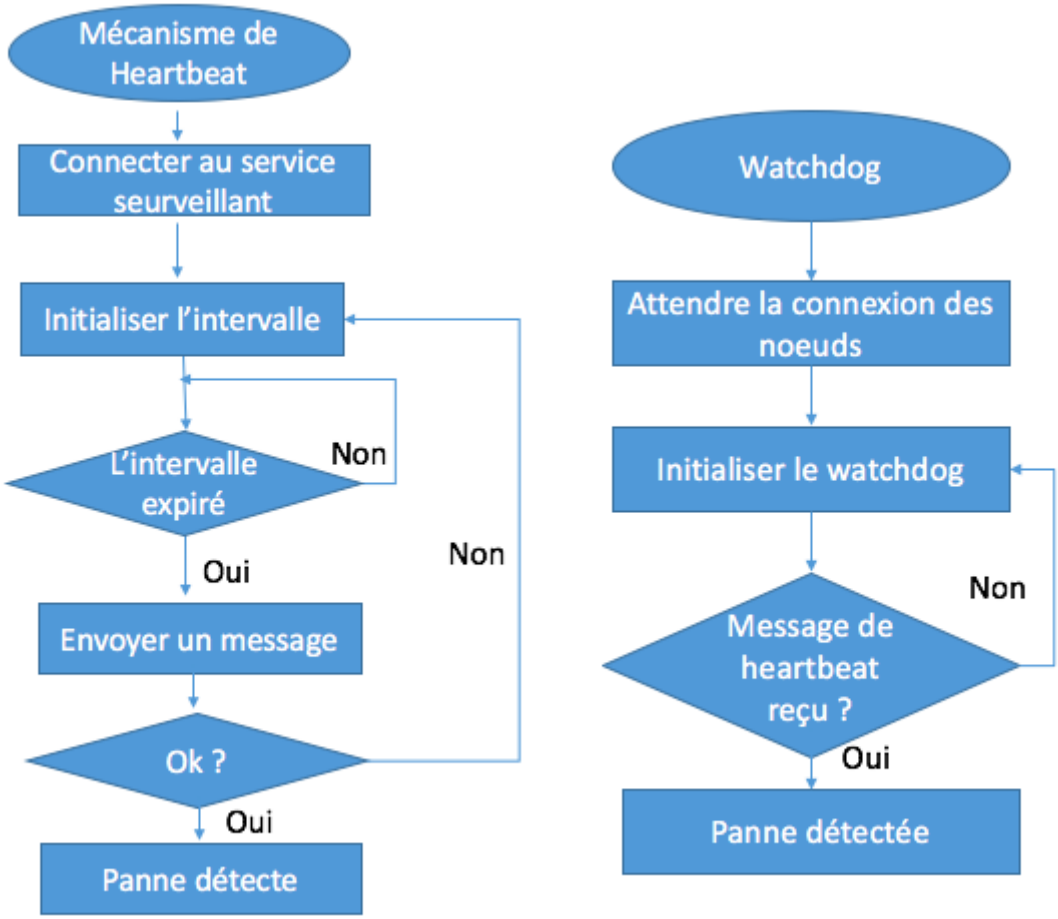


Figure 3.3 - Organigramme d'activité de heartbeat/ watchdog

3.4.4. Recovery manager

Quand le détecteur de pannes (FD) détecte une panne, il informe le gestionnaire de recouvrement (recovery manager RM). Le FD utilise les informations de gestionnaire de ressources pour sélectionner une VM disponible dans le même Data-center pour lancer la procédure de reprise qui correspond à redémarrer les tâches depuis leur dernier point de reprise (voir **Figure 3.4**). S'il a plusieurs VMs candidates, la VM la plus fiable et moins coûteuse sera sélectionnée.

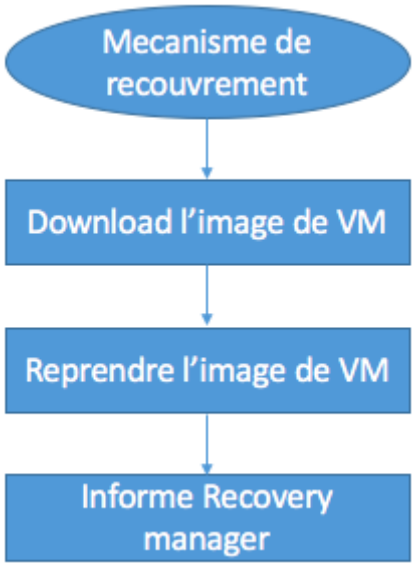


Figure 3.4 - Organigramme d'activité de recouvrement

```
VARIABLES
    VM EST_DU_TYPE NOMBRE
    FI EST_DU_TYPE NOMBRE
    C EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    SI ( VM>1 ) ALORS
        DEBUT_SI
        SI (VM = max Fi(VM) & VM = min C(VM)) ALORS
            DEBUT_SI
            LIRE VM
            AFFICHERCALCUL checkpoint
            FIN_SI
        FIN_SI
FIN_ALGORITHME
```

Algorithme de recouvrement

Où :
Fi(VM) : Fiabilité de VM
C(VM) : Coût de VM

3.5. Fonctionnement du système

Le Broker transforme l'application d'utilisateur en BoTs puis il donne le résultat et les contraintes SLA au ressource manager. Le ressource manager utilise ces informations pour allouer les ressources nécessaire pour l'exécution de cette application. La liste des ressources et Bots sont utilisés par le Scheduler qui fait l'allocation de chaque tache dans une ressource qui minimise le temps de réponse et améliore les performances. Après le lancement d'application, le checkpointer calcule périodiquement l'intervalle de checkpointing et exécute le checkpointing si l'intervalle expire. Si le Fault detector détecte une panne, il contacte le recovery manager pour assurer le recouvrement du VM en panne dans une autre VM fournit par le ressource manager.(voir Figure 3.5)

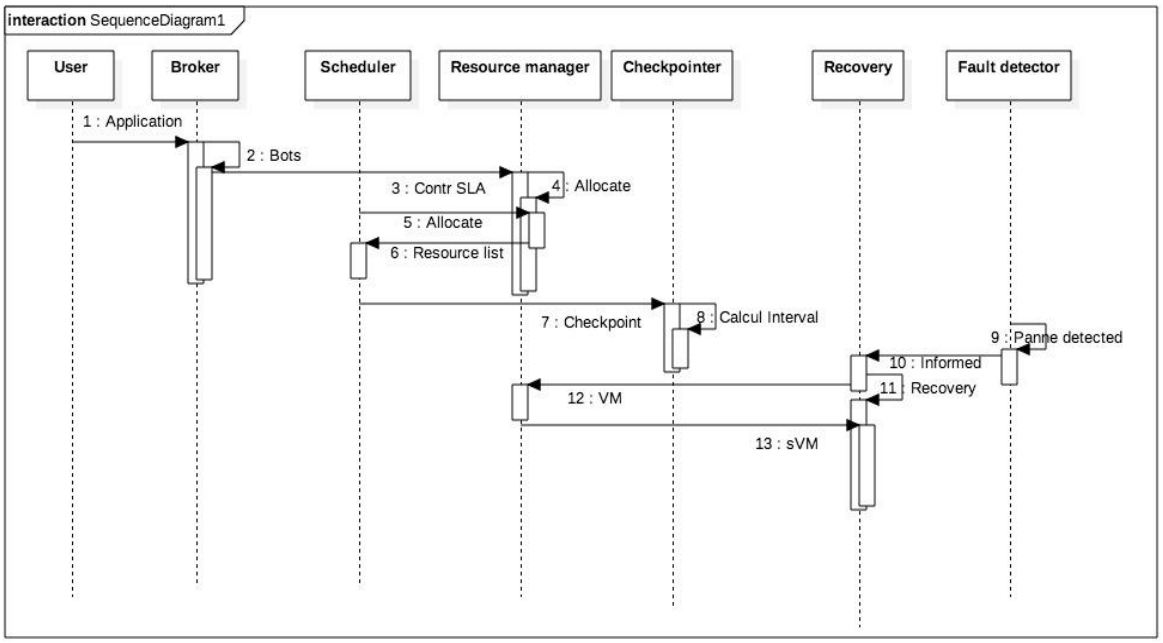


Figure 3.5 -Diagramme de séquence du système

3.6. Métriques de performance

Pour l'évaluation de la performance efficace d'une approche et la comparaison entre les algorithmes, il est essentiel de définir des métriques de performance qui déduisent les caractéristiques pertinentes des algorithmes. Pour prouver l'efficacité de nos approches proposées, nous avons utilisé 3 métriques qui sont

- la consommation d'énergie par les Data Center,
- La surcharge de checkpointing
- Violation SLA

3.6.1. Consommation d'énergie par les Data Center

Ils existent plusieurs méthodes pour calculer la consommation d'énergie, parmi lesquels nous avons utilisé la méthode "puissance par rapport à l'utilisation".
De nombreuses études [12] [13] ont montré que la consommation d'énergie par les serveurs peut être décrite par une relation linéaire entre la consommation d'énergie et l'utilisation du processeur. Ces études confirment qu'une puissance moyenne consommée par un serveur inactif est de 70% de l'énergie consommée par rapport à un serveur pleinement utilisé. D'après [12], si l'utilisation du processeur est supérieure à 30%, la valeur inférieure est toujours 0,3. Donc, ils ont défini **la consommation d'énergie** $P(u)$ par la Formule (3.2) :

$$P(u) = P\max * (0,7 + 0.3u) \quad (3.2)$$

Tel que : *Pmax* = 250w pour des serveurs modernes. La constante 0.7 est la puissance moyenne consommée par un serveur inactif.
u : est l'utilisation du processeur.
Donc la consommation d'énergie par rapport à un Data Center est calculée par la Formule (3.3) [2] :

$$E_D = \frac{\sum_{j=1}^{m} p(u)_j}{m} \quad (3.3)$$

- m : nombre de machines physiques dans un Data Center.

3.6.2. Nombre des checkpoints inutiles.

Le nombre des checkpoints inutiles $NI(VM_i)$ c'est le nombre des checkpoints crées sans que la VM i tombe en panne. Ce nombre représente les efforts perdus pour assurer la tolérance aux pannes. Le nombre des checkpoints inutiles totale est représenté par un pourcentage (voir la formule (3.4))

$$NI = \frac{\sum_{i=1}^{n} NI(VM_i)}{NT} \times 100 \quad (3.4)$$

Où:

- n : le nombre des VMs participants dans l'application
- NT : le nombre des checkpoints totales (inutile & utiles)

3.6.3. surcharge de checkpointing

La surcharge de checkpointing est le paramètre le plus important pour mesurer l'efficacité du checkpointing [18]. La surcharge peut être mesurée de plusieurs façons selon les besoins de simulation. Dans notre cas, la surcharge ST est calculée par la formule suivante (voir la formule (3.5)) :

$$ST = \frac{TE_{cp}}{TE_{\overline{cp}}} \quad (3.5)$$

Où :

- TE_{cp} : le temps d'exécution d'application avec l'utilisation du checkpointing,
- $TE_{\overline{cp}}$: le temps d'exécution d'application sans l'utilisation du checkpointing.

3.6.4. Violation des SLAs

Les exigences de la qualité de service sont extrêmement importantes pour le
Cloud computing. Elles sont généralement formalisées sous forme de SLA. Service Level Agreement (SLA) est un document qui définit la qualité de service requise entre un prestataire et un client [16]. Le contrat SLA peut être déterminé en termes de plusieurs contraintes telles que le débit minimum ou le temps de réponse maximum fourni par le système déployé. Etant donné que ces caractéristiques peuvent varier selon les applications, il est nécessaire de définir une métrique générique qui peut être utilisée dans notre simulation pour estimer le niveau du SLA qui est délivré par l'infrastructure. Pour ce qui nous concerne, nous définissons le niveau global de violation SLA causé par le système comme la somme de surcharge en termes de temps d'exécution et du cout et de budget (Voir la formule (3.6))

$$V = \alpha 1\,(ST) + \beta 1\,(SC) + \gamma 1\,(SM) \quad (3.6)$$

Où:

- $\alpha 1, \beta 1, \gamma 1$: Facteurs de poids et $\alpha 1 + \beta 1 + \gamma 1 = 1$
- SC : Surcharge en termes de coût
- SM : Surcharge en termes de budget

SC et SM sont calculés de la même façon que ST.

3.7. Conclusion

Ce chapitre a présenté la solution adoptée pour traiter la problématique rencontrée. Pour cela nous avons proposé une plate-forme de tolérance aux pannes basée sur les points de reprises dans un environnement de cloud computing.

Le prochain chapitre sera consacré à la partie l'implémentation et évaluation de notre proposition de tolérance aux pannes. Nous mettrons en évidence l'importance et l'efficacité de l'approche proposée par la comparée avec une autre approche de checkpointing classique.

CHAPITRE 4 : Implémentation

4.1. Langage et environnement de développent

Nous avons développé le simulateur sur une machine avec un processeur 1.7 GHz Intel Core i5, doté d'une capacité mémoire de 4GB 1333MHz DDR3, Graphics Intel HD Graphics 3000 384 MB sous MacBook Air. Nous avons utilisé l'environnement de développement Java NetBeans.

4.1.1. langage de programmation java :

Java est un langage de programmation informatique orienté objet et un environnement d'exécution informatique portable créé par James Gosling et Patrick Naughton employés de Sun Microsystems avec le soutien de Bill Joy (cofondateur de Sun Microsystems en 1982), présenté officiellement le 23 mai 1995 au SunWorld [18].

Java est à la fois un langage de programmation et un environnement d'exécution. Le langage Java a la particularité principale que les logiciels écrits avec ce dernier sont très facilement portables sur plusieurs systèmes d'exploitation tels qu'Unix, Microsoft Windows, Mac OS ou Linux avec peu ou pas de modifications... C'est la plate-forme qui garantit la portabilité des applications développées en Java.

Le langage reprend en grande partie la syntaxe du langage C++, très utilisé par les informaticiens. Néanmoins, Java a été épurée des concepts les plus subtils du C++ et à la fois les plus déroutants, tels que l'héritage multiple remplacé par l'implémentation des interfaces. Les concepteurs ont privilégié l'approche orientée objet de sorte qu'en Java, tout est objet à l'exception des types primitifs (nombres entiers, nombres à virgule flottante, etc.).

Java permet de développer des applications autonomes mais aussi, et surtout, des applications client-serveur. Côté client, les applets sont à l'origine de la notoriété du langage. C'est surtout côté serveur que Java s'est imposé dans le milieu de l'entreprise grâce aux servlets, le pendant serveur des applets, et plus récemment les JSP (JavaServer Pages) qui peuvent se substituer à PHP, ASP et ASP.NET.

Les applications Java peuvent être exécutées sur tous les systèmes d'exploitation pour lesquels a été développée une plate-forme Java, dont le nom technique est JRE (Java Runtime Environnement - Environnement d'exécution Java). Cette dernière est constituée d'une JVM (Java Virtual Machine - Machine Virtuelle Java), le programme qui interprète le code Java et le convertit en code natif. Mais le JRE est surtout constitué d'une bibliothèque standard à partir de laquelle doivent être développés tous les programmes en Java. C'est la garantie de portabilité qui a fait la réussite de Java dans les architectures client-serveur en facilitant la migration entre serveurs, très difficile pour les gros systèmes [18].

4.1.2. Netbeans environnement de développement.

NetBeans est un environnement de développement intégré (EDI), placé en open source par Sun en juin 2000 sous licence CDDL et GPLv2 (Common Development and Distribution License). En plus de Java, NetBeans permet également de supporter différents autres langages, comme Python, C, C++, JavaScript, XML, Ruby, PHP et HTML. Il comprend toutes les caractéristiques d'un IDE moderne (éditeur en couleur, projets multi-langage, refactoring, éditeur graphique d'interfaces et de pages Web).

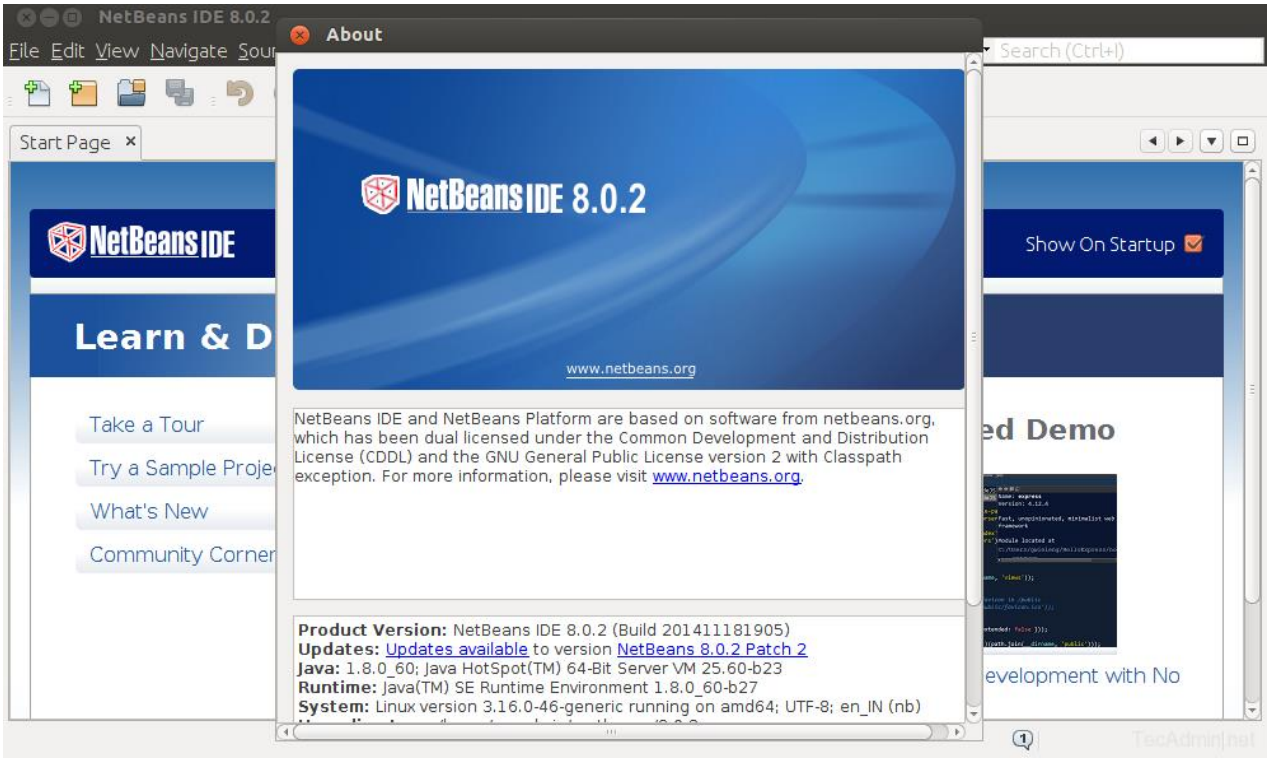


Figure 4.1 - l'environnement de développement NetBeans

Conçu en Java, NetBeans est disponible sous Windows, Linux, Solaris (sur x86 et SPARC), Mac OS X ou sous une version indépendante des systèmes d'exploitation (requérant une machine virtuelle Java). Un environnement Java Développent Kit JDK est requis pour les développements en Java.
NetBeans constitue par ailleurs une plateforme qui permet le développement d'applications spécifiques (bibliothèque Swing (Java)). L'IDE NetBeans s'appuie sur cette plateforme. L'IDE Netbeans s'enrichit à l'aide de plugins, Il comprend toutes les caractéristiques d'un IDE moderne (coloration syntaxique, projets multi-langage, refactoring, éditeur graphique d'interfaces et de pages web, etc.) [19].

4.2. Le simulateur CloudSim

Projet développé par le CLOUDS Laboratory de Melbourne en Australie, est un outil de simulation extensible permettant la modélisation et la simulation d'environnement de systèmes de type Cloud Computing de niveau IaaS. Cette section d'introduction aux fonctionnalités de CloudSim est largement inspirée de l'article présentant en d´détails ce simulateur [20].
CloudSim est un nouveau général, et extensible cadre de simulation qui permet la modélisation, la simulation et l'expérimentation de nouvelles cloud computing infrastructures et de services d'application.
Dans le cas Cloud Computing, les outils de Simulations comme CloudSim donne un offre d'importants avantages aux clients et fournisseurs. Pour les clients, il permettre de tester leurs services dans un environnement contrôlable avec exempt du coût et de vérifier la performance avant de publier les vrais nuages. Pendant ce temps pour les fournisseurs, de leur permettre de vérifier les types de location en fonction de divers prix et la charge ;
En outre, cela permettra d'optimiser le coût de l'accès aux ressources à l'amélioration des bénéfices. Sans ces outils, à la fois des clients et des fournisseurs doit se appuyer sur des évaluations imprécises, ou sur des approches essai-erreur, ces approches peuvent conduire à l'inefficacité performance des services et de réduire la génération de revenus. En outre, CloudSim aide les chercheurs et développeurs basés sur l'industrie pour tester la performance d'un service d'application développée dans une convenable et facile

à installer environnement. Il y a de nombreux avantages de l'utilisation CloudSim pour tester les performances de départ, comme:
(i) l'efficacité de temps: il prend très moins de temps et d'efforts pour mettre en œuvre des applications de cloud computing.
(ii) la flexibilité: les développeurs peuvent facilement modéliser et tester les performances de leurs applications et ses services dans des environnements hétérogènes (Microsoft Azure, Amazon EC2).

4.2.1. Architecture générale

La structure logiciel de cloudsim et ses composant est représenter par une architecture en couche comme il montré par la figure 4.2, Au niveau le plus bas est le moteur de simulation aux événements discrets Sim Java, qui implémente les fonctionnalités de base requises pour les cadres de simulation au niveau supérieur, telles que les files d'attente, le traitement des événements, création de composants du système (services, hôte, Datacenter, Broker, les machines virtuelles), la communication entre les composants et la gestion de l'horloge de simulation.

La gestion de toutes ces ressources est gérée par la couche Cloud Services, gérant l'approvisionnement des machines virtuelles, l'utilisation des CPUs, de la mémoire RAM, de la capacité de stockage et de la bande passante des machines physiques. Dans CloudSim, les tâches à exécuter sont assimilées à des cloudlets qui sont allouées aux machines virtuelles (couches VM Services et User Interface Structure). La configuration de chacun des composants doit être gérée par l'utilisateur (couche Simulation Spécification) par l'intermédiaire d'un fichier de configuration de simulation. De nombreux paramètres peuvent y être spécifiés : • Nombre de Data Center, machines physiques, machines virtuelles, tâches (cloudlets) • Caractéristiques des Data Center : architecture, système d'exploitation, hyperviseur, coûts de fonctionnement • Caractéristiques des machines physiques : nombre de CPUs, capacité des CPUs, capacité mémoire, capacité de stockage, bande passante • Caractéristiques des machines virtuelles : capacité CPUs, nombre de CPUs, capacité mémoire, capacité de stockage • Caractéristiques des cloudlets : nombre d'instructions à exécuter, taille des fichiers d'entrée et de sortie

CloudSim supporte la modélisation et la simulation de l'environnement de Datacenter basé sur Cloud, tel que des interfaces de gestion dédiées aux VMs, la mémoire, le stockage et la bande passante. La couche CloudSim gère l'instanciation et l'exécution des entités de base (VM, hôtes, Datacenters, applications) au cours de la période de simulation. Dans la couche plus haute de la pile de simulation, on trouve le code de l'utilisateur qui expose la configuration des fonctionnalités liées aux hôtes (ex: nombre de machines, leurs spécifications), les politiques d'ordonnancement de Broker, applications (ex: nombre de tâches et leurs besoins), VM, nombre d'utilisateurs.

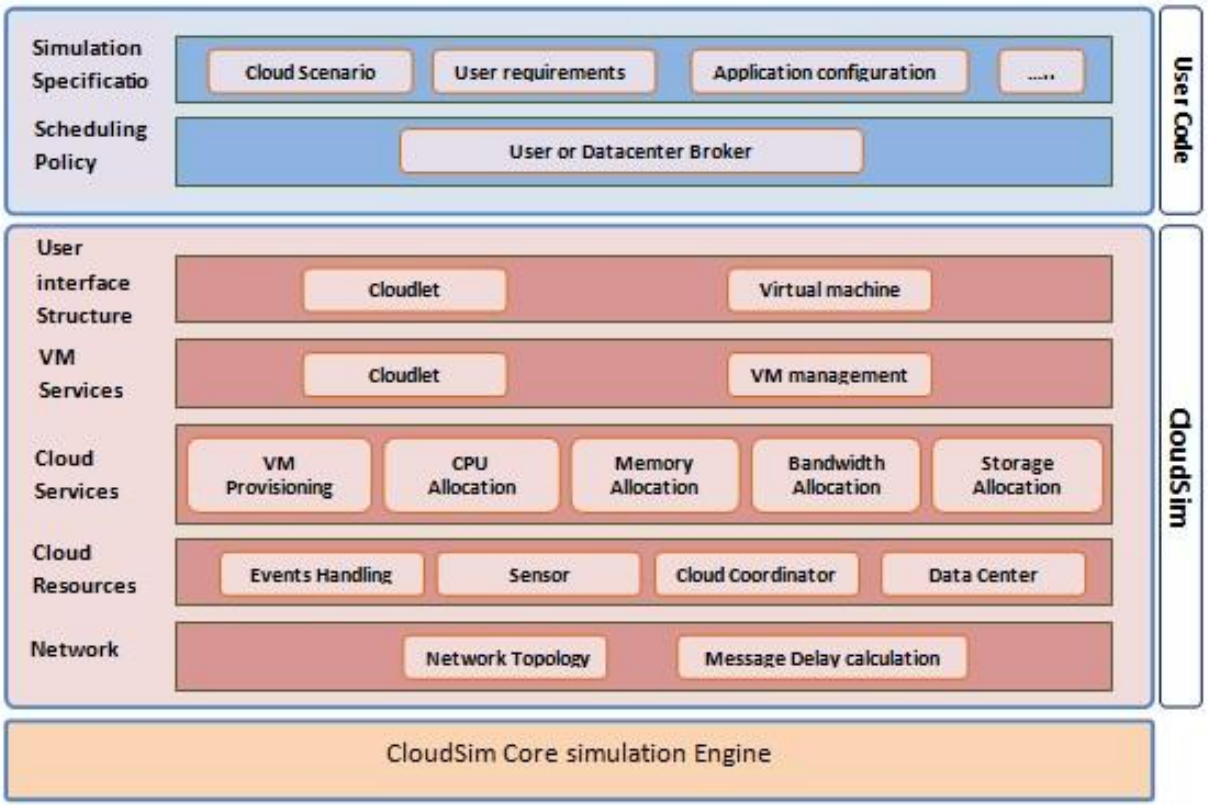


Figure 4.2 -Les couches de l'architecture du CloudSim.

4.2.2. Description de l'application :

La version de Cloudsim 3.0.3 n'a pas d'interface graphique, il utiliser le modes console pendant l'exécution, qui rend difficile à l'utilisateur de profiter pleinement du cloudsim, comme la modification les paramètre de simulation l'affichage graphique des output (résultat de simulation) .Nous avons créé une interface graphique qui facilite a l'accès et la manipulation du simulateur, nous allons présenter les différents étapes effectuées pour réaliser notre simulation.

4.2.3. Modélisation du Cloud CloudSim

Est dédié à la simulation de services Cloud au niveau Infrastructure (IaaS). Pour ce faire, les simulations sont basées sur la gestion des machines physiques qui composent le ou les Data Center. A ces machines physiques sont allouées des machines virtuelles selon des politiques de placement qui peuvent être aisément modifiées. Aussi, le cycle de vie des machines virtuelles (création, allocation, migration éventuelle et destruction) détermine le démarrage, le déroulement et la terminaison des simulations.

4.2.4. Modélisation des machines virtuelles et des Cloudlets

La couche de virtualisation gère l'exécution des machines virtuelles sur les machines physiques et la façon dont elles exécutent les cloudlets. En effet, CloudSim propose deux modes d'exécution : temps partage et espace partage pour l'exécution des machines virtuelles sur les machines physiques et également pour les cloudlets au sein des machines virtuelles. Ce qui permet donc quatre configurations d'exécution différentes. L'exécution des machines virtuelles est également gérée par l'instanciation de brokers. Un broker peut contenir une ou plusieurs machines virtuelles. La fin de son cycle de vie est détermine par la durée de vie de chacune des machines virtuelles qui le composent. Ainsi, un broker termine son exécution seulement lorsque toutes les machines virtuelles qu'il contient ont également fini leur exécution.

4.3. Diagramme d'utilisation

Nous présentons maintenant une vue globale de notre simulateur qui permet de détailler les déférentes étapes effectuées pour réaliser une simulation. La Figure 4.3 représente les fonctionnalités d'utilisation nécessaires à l'application.

Le diagramme des cas d'utilisation, est une modélisation d'une fonctionnalité du système, il se détermine en observant et en précisant acteur par acteur les séquencés d'interactions avec le système. Les cas d'utilisation décrivent les fonctions du système selon le point de vue des utilisateurs.

Selon ce diagramme, l'utilisateur commence la simulation par la configuration du Cloud puis la configuration des machines virtuelles. Dès que les Cloudlets sont créés et personnalisés, après la disposer de l'application l'utilisateur peut injecter les pannes et sélectionner le mécanisme de tolérance aux pannes. Ensuite l'utilisateur peut lancer la simulation, les résultats de cette simulation sont présentés sous forme d'un tableau et un digramme.

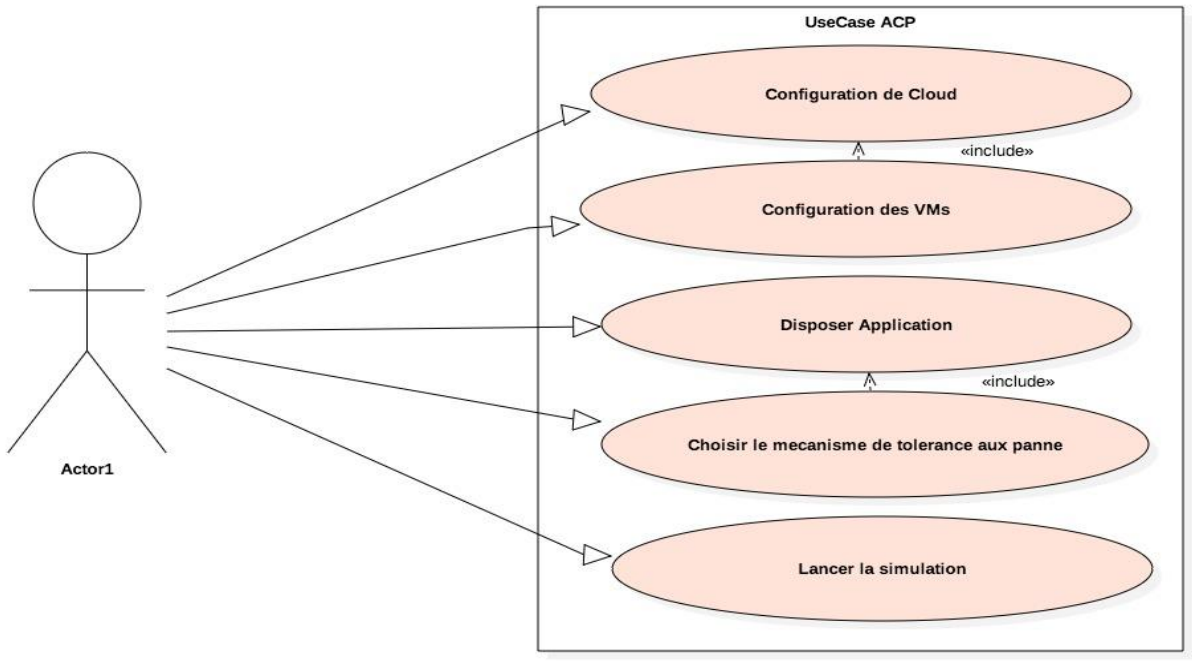


Figure 4.3 -Diagramme d'utilisation du simulateur

4.4. Diagramme de classes

- Les diagrammes de classes sont les diagrammes les plus courants dans la modélisation des systèmes orientés objet. Ils représentent un ensemble de classes, d'interfaces et de collaborations ainsi que leurs relations. On utilise les diagrammes de classes pour modéliser la vue de conception statique d'un système [22].
  L'ensemble des classes utilisées est présenté par la Figure 4.4, nous pouvons citer :
- La classe Datacenter : Elle a comme attributs :
  - IdDatacenter : L'identifiant du Datacenter.
  - Nb Host : Nombre de Host dans Datacenter.
  - Nb Vm : Nombre de Machines Virtuelles dans Datacenter.
- La classe Host : Elle a comme attributs :
  - IdHost : L'identifiant du Host.
  - Nb Vm : Nombre de machines virtuelles dans Host.

- La classe Broker : Elle a comme attributs : - IdBroker : L'identifiant du Broker.
  - VmList : La liste des machines virtuelles.
- La classe Gestionnaire de pannes : On peut citer comme attribut :
  -IdFault : L'identifiant des pannes.
- La classe VM (Machine Virtuelle) : Elle a comme attributs :
  - IdVm : L'identifiant de la machine virtuelle.
  - NbCloudlet : Nombre de cloudlet dans VM.
- La classe Cloudlet : Elle a comme attribut :
  -IdCloudlet : L'identifiant du Cloudlet.

Les associations : Pour les différentes associations utilisées dans ce diagramme de classes, nous pouvons citer :

- Détecter : Les pannes sont détectées par le Gestionnaire de pannes.
- Créer : Les VMs sont créées par le Broker.
- Affecter : Affectation des Cloudlets par le Broker.

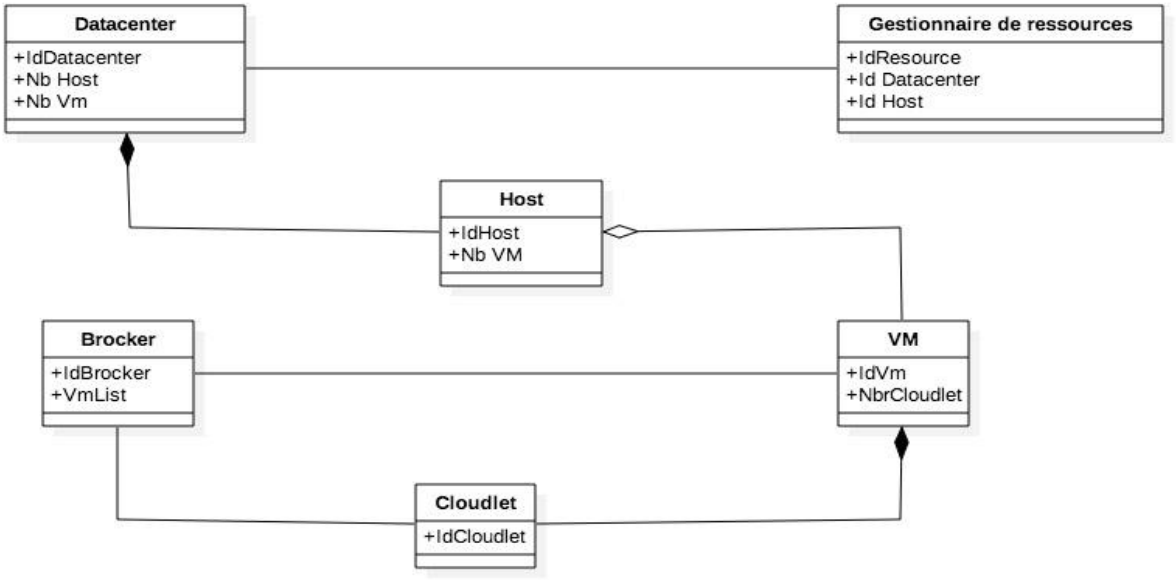


Figure 4.4 -Diagramme de classe de la conception de simulateur CloudSim

4.5. Interface principale

La version de Cloudsim n'a pas d'interface graphique, son exécution se fait sur console, donc nous avons créé une interface qui facilite l'accès au simulateur. Nous avons aussi Étendu la première version du CloudSim dans les travaux [10] et [12]. L'interface doit faire appel à Cloudsim ainsi qu'aux différentes approches qui se trouvent dans différents packages.
Dès le lancement de notre logiciel, la Figure 4.5 apparait en premier aux utilisateurs, elle est constituée d'une barre de menu contenant les menus suivants : Le menu Fichier, Simulation, Affichage et le menu Aide ; chacun de ces menus contient un en- semble d'items qui vont être détaillés par la suite. Nous allons nous intéresser dans cette partie à la démonstration de notre application à travers un exemple en faisant référence à quelques interfaces graphiques obtenues.

Figure 4.5 -Interface principale du CloudSim-FT

4.5.1. Configuration des paramètres de simulation

Les Figures 4.6, 4.7, 4.8 et 4.9 montrent les fenêtres de configuration des paramètres de simulation, faite par l'utilisateur, elle contient 4 parties principales :

Datacenter

Cette étape consiste à faire entrer les paramètres nécessaires propres à la topologie du réseau (Figure 4.6)comme : le nombre de Data Center, d'hôtes, de CPU, la vitesse de chaque CPU, le coût de traitement, la taille de la mémoire, le cout de la mémoire, la taille du disque dur, le coût de stockage et la bande passante ainsi que son coût

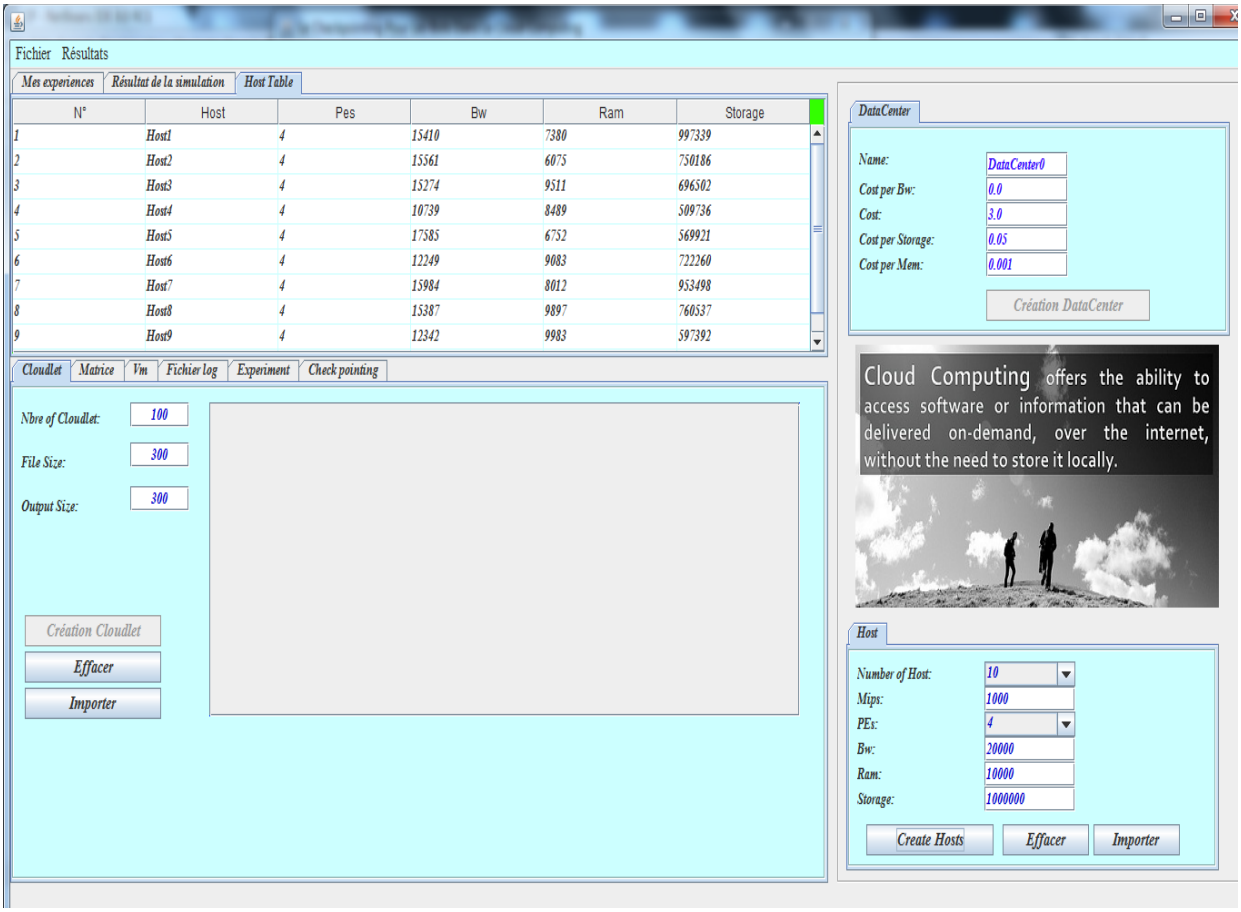


Figure 4.6 -Configuration du Datacenter

VirtualMachine (VM)

L'onglet VirtualMachine permet de configurer les machines virtuelles (Figures 4.7, 4.8, 4.9) par exemples : spécification du nombre de VM, de CPU dans une VM, la taille de la mémoire, la taille du disque dur et la bande passante.

- **Writers**

Pour améliorer le checkpoint, on a proposé d'utiliser Les writers. Ces derniers sont responsables du stockage des checkpoint, ils minimisent 70% de surcharge de checkpointing.

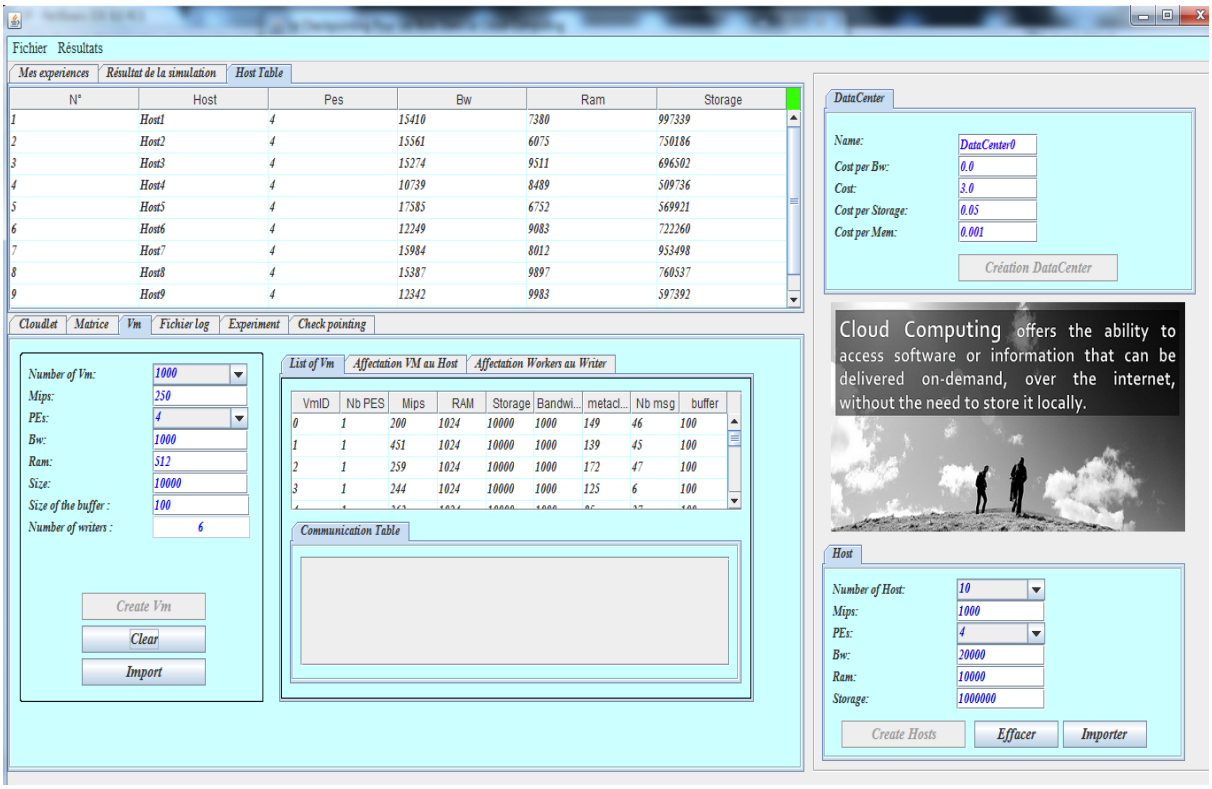


Figure 4.7 -Configuration des machines virtuelles (list of vm)

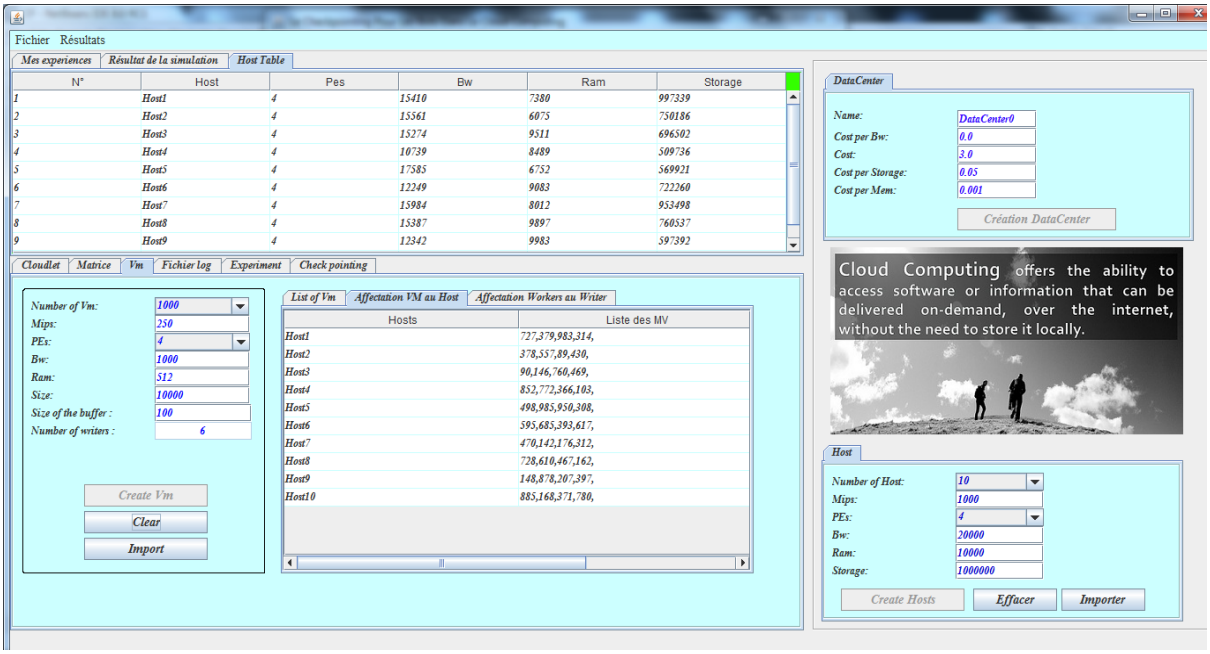


Figure 4.8 -Configuration des machines virtuelles(Affectation vm au host)

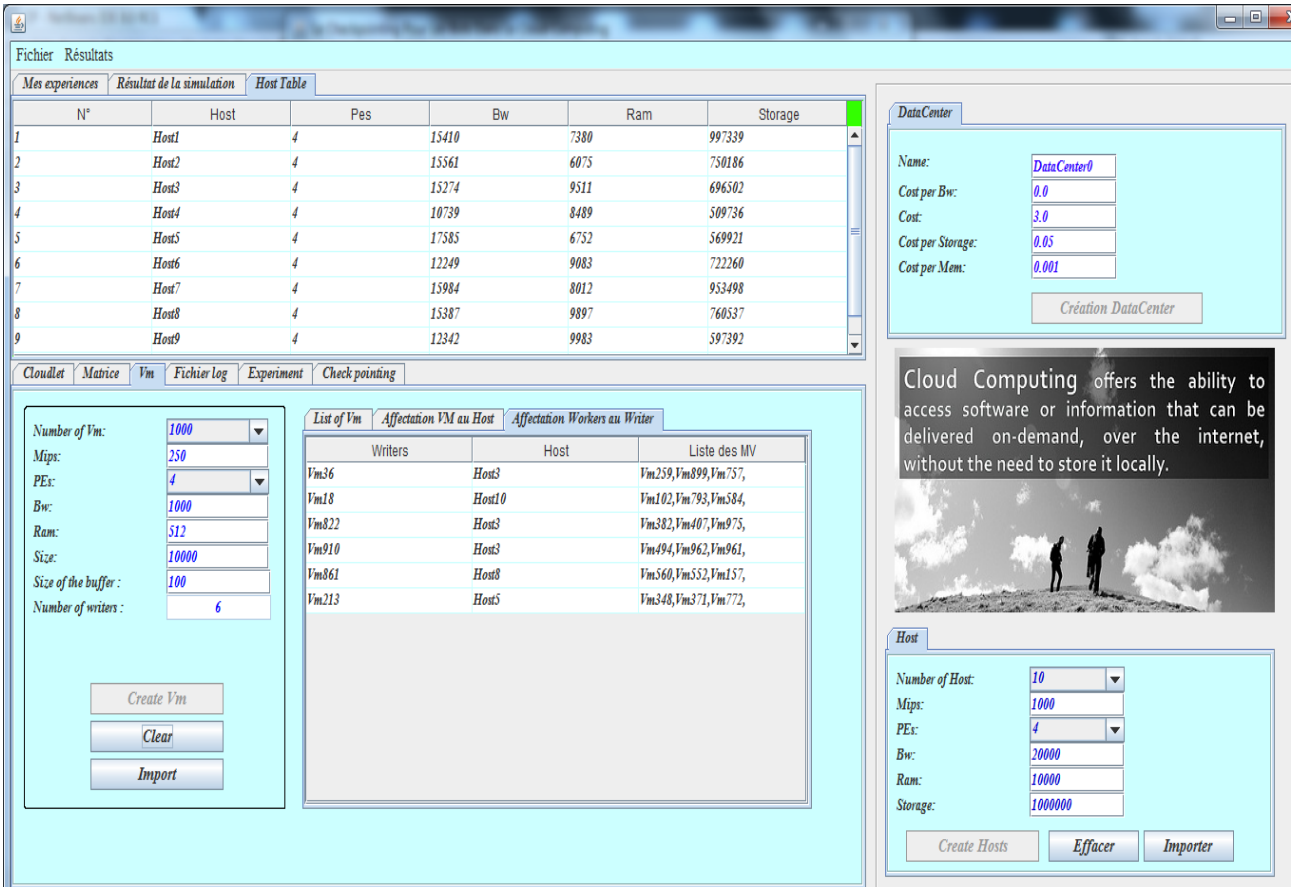


Figure 4.9 -Configuration des machines virtuelles (Affectation workers au writer)

Cloudlet

Permet de configurer les propriétés de la Cloudltet par les Figure 4.10

The cloudlet length : la taille de la Cloudlet à exécuter dans un CloudResource.

The cloudlet file size : la taille du fichier d'entrée de la Cloudlet avant l'exécution (la taille du programme + les données en entrée).

The cloudlet output size : la taille du fichier de sortie de la Cloudlet après l'exécution.

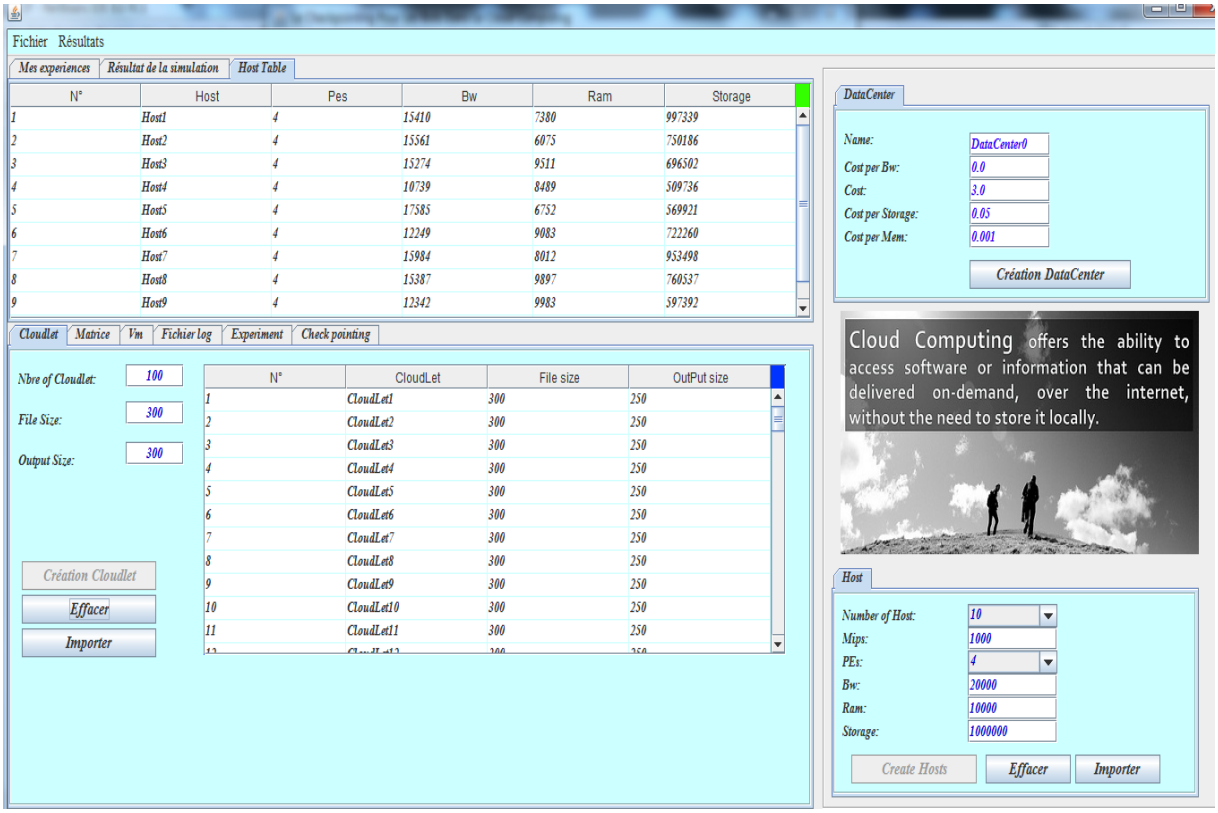


Figure 4.10 -Configuration des Cloudlets

Fault Tolerant

L'onglet Fault Tolerant permet à l'utilisateur d'injecter les pannes et de sélectionner le mécanisme de tolérance aux pannes. Without Fault-tolerant représente l'exécution sans aucune approche de tolérance aux pannes, Checkpoint est l'approche basée sur le mécanisme de sauvegarde, nous pouvons sélectionner le mécanisme de checkpoint Adaptatif ou Static.

Après avoir personnaliser les paramètres de simulation, l'utilisateur peut lancer la simulation en cliquant sur le bouton Start Simulation (voir Figures 4.11, 4.12, 4.13 et 4.14).

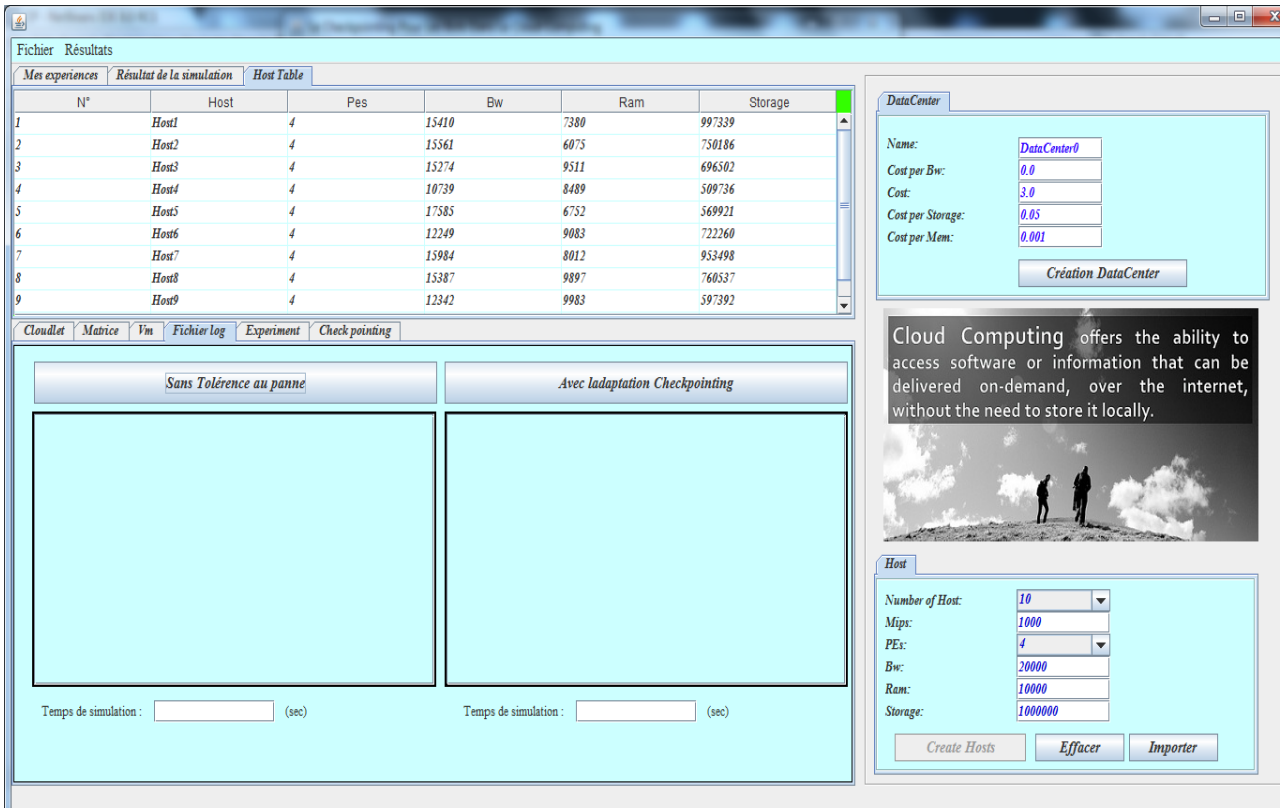


Figure 4.11 - configuration de mécanisme de tolérance aux pannes

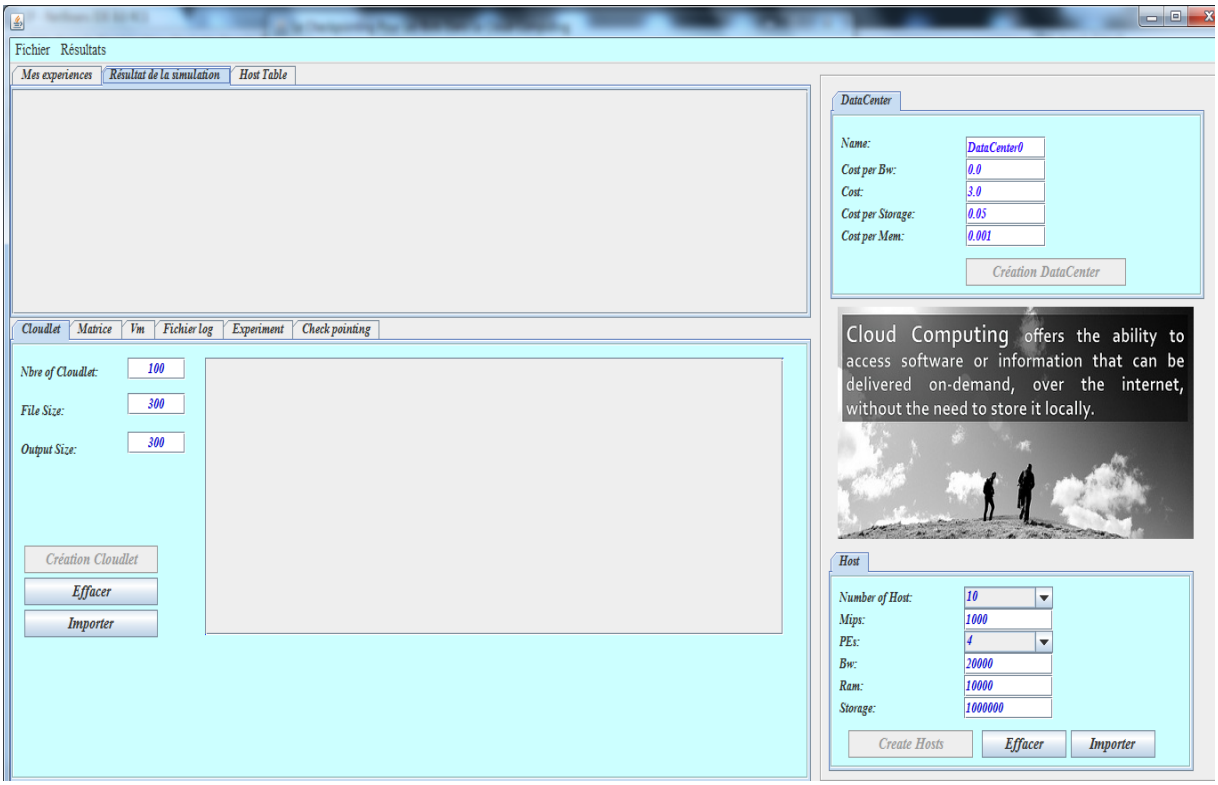


Figure 4.12 - configuration de mécanisme de tolérance aux pannes (Résultats de la simulation)

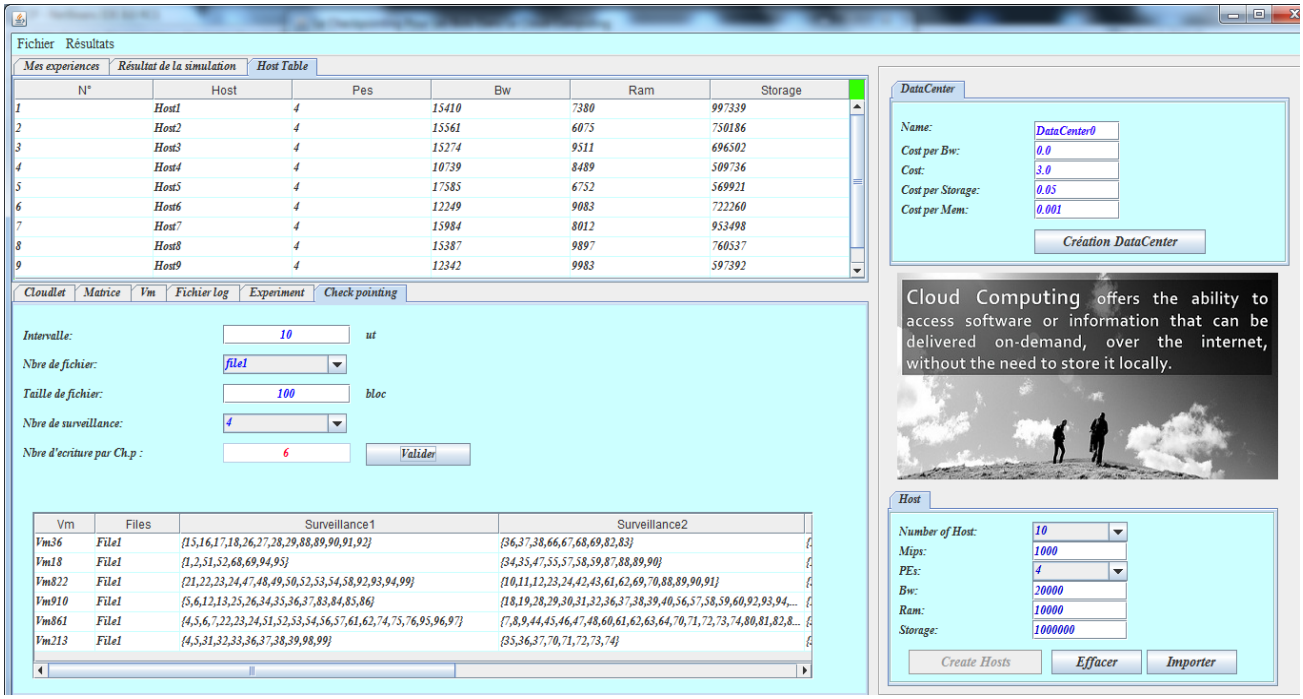


Figure 4.13 - configuration de mécanisme de tolérance aux pannes (Checkpointing)

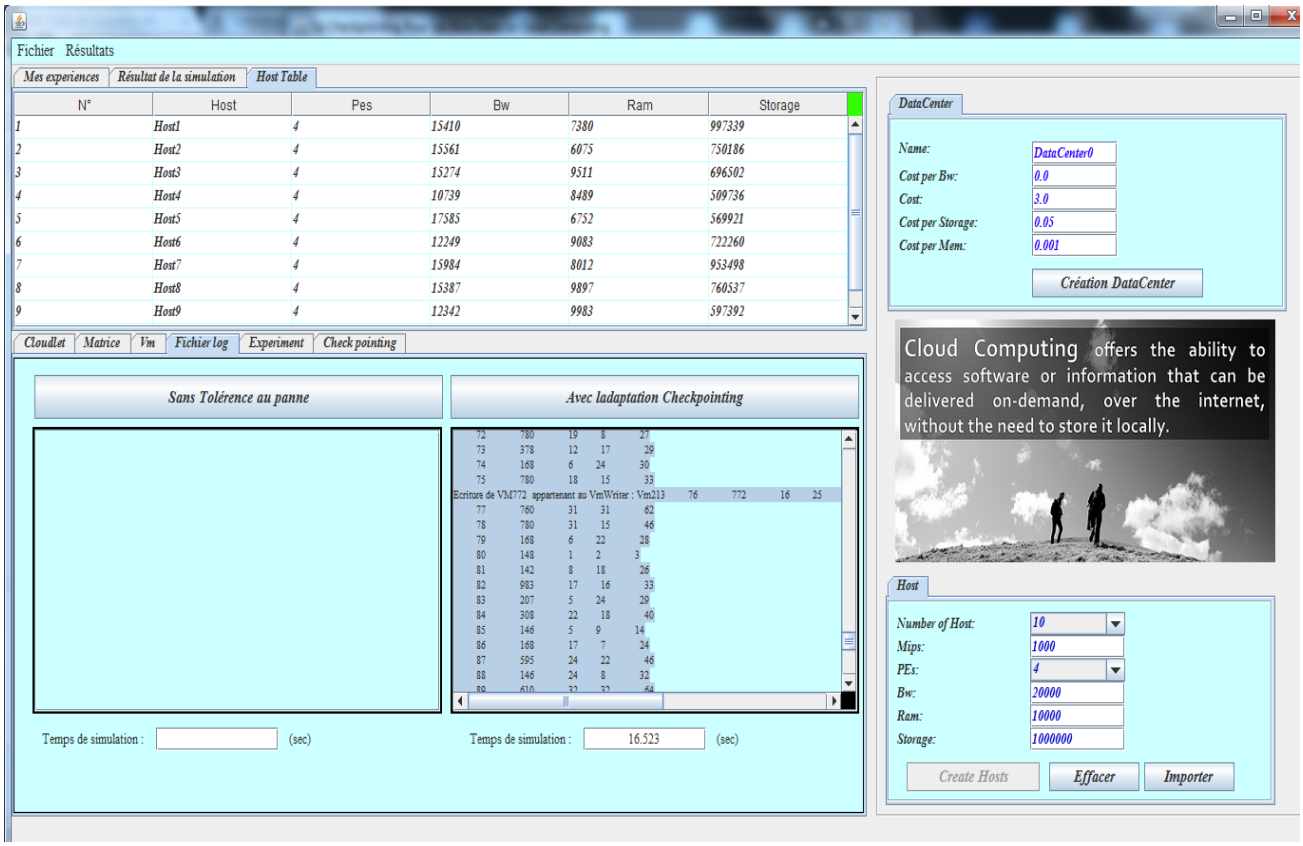


Figure 4.14 - configuration de mécanisme de tolérance aux pannes (avec l'adaptation checkpoint)

4.6 Résultats expérimentaux

Dans cette premier simulation, nous avons mesurer le nombre de cloudlet exécuter avec succès et le nombre de cloudlet échouées avec tolérance aux pannes et sans tolérance aux panne (voir figure 4.15 et 4.16).

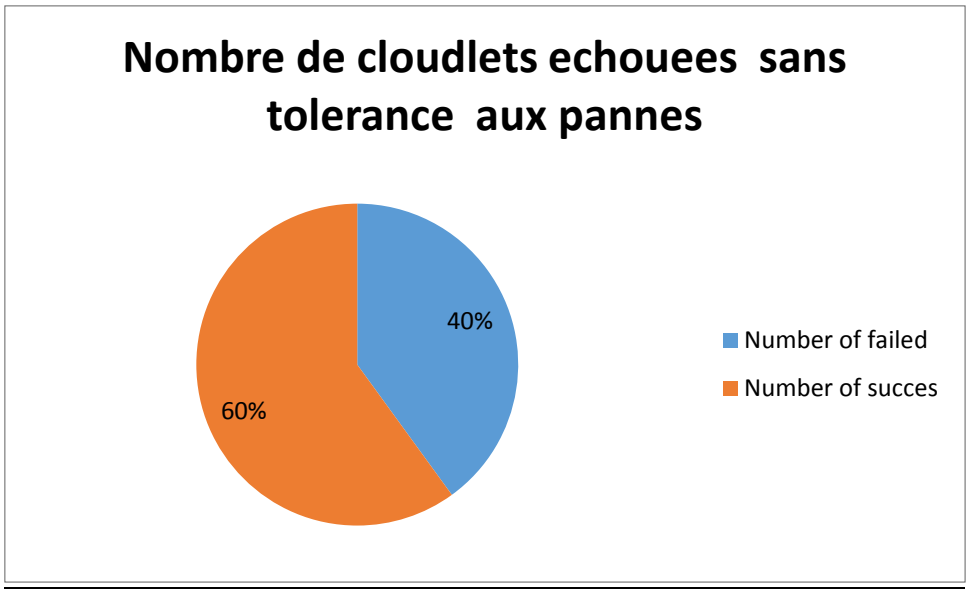


Figure 4.15 – nombre de cloudlets échouées sans tolérances aux pannes

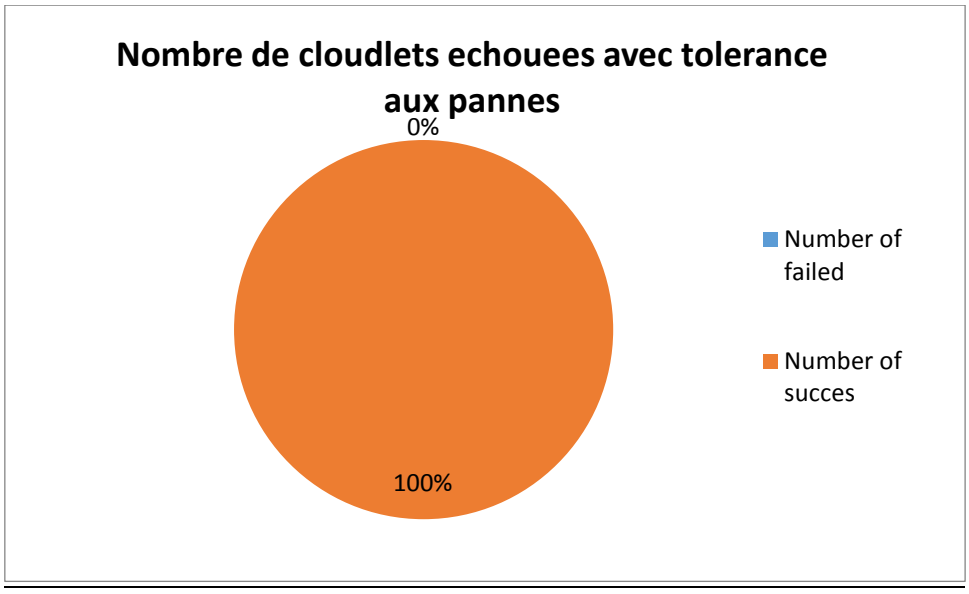


Figure 4.16 – nombre de cloudlets échouées avec tolérances aux pannes

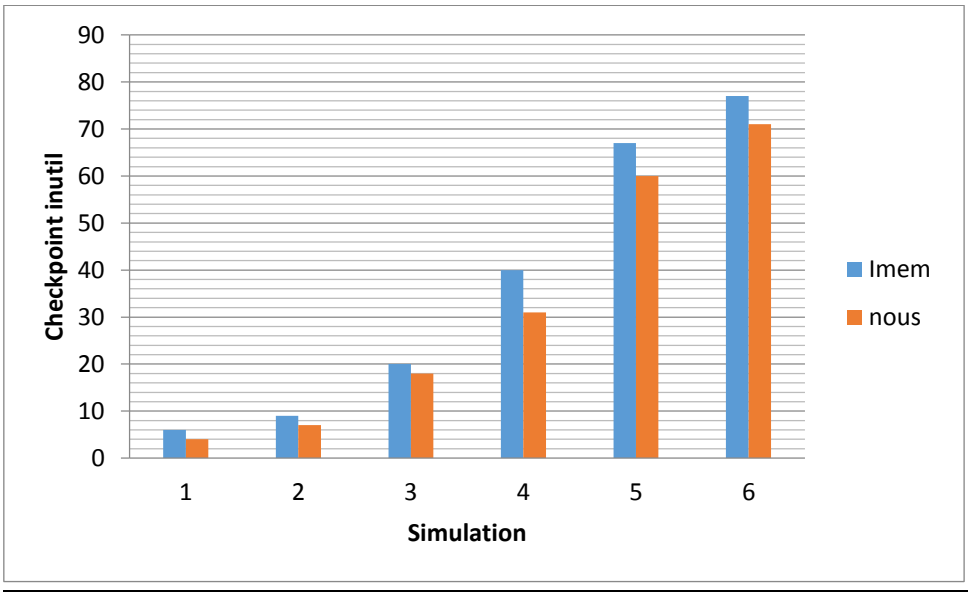
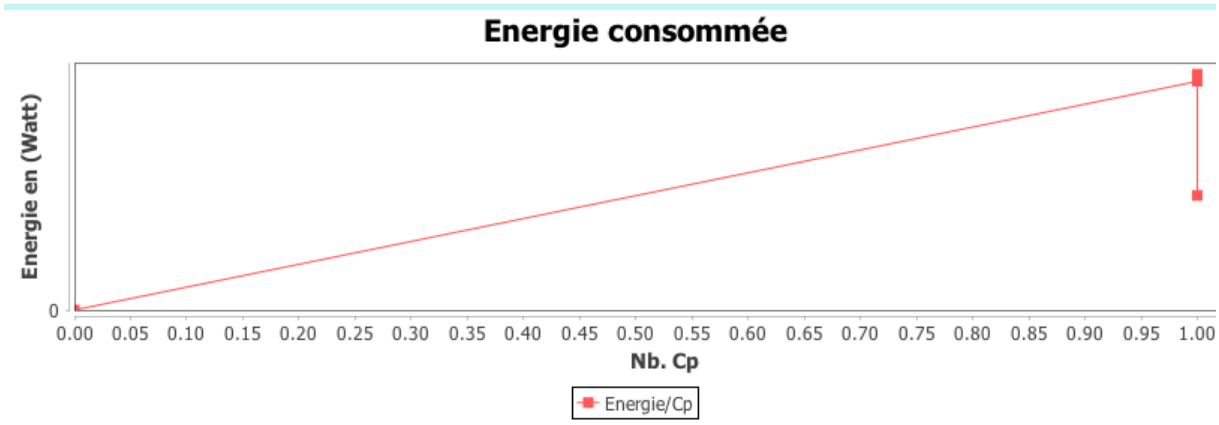


Figure 4.17 – Checkpoint inutil

Nous avons remarqué que le checkpoint inutile dans notre approche moins que le travail [1] car notre approche prendre en considération l'énergie et la charge du travail.



Nous avons remarqué que le nombre de checkpoint augmente quand l'énergie augmente.

Figure 4.18 – Energie consommée

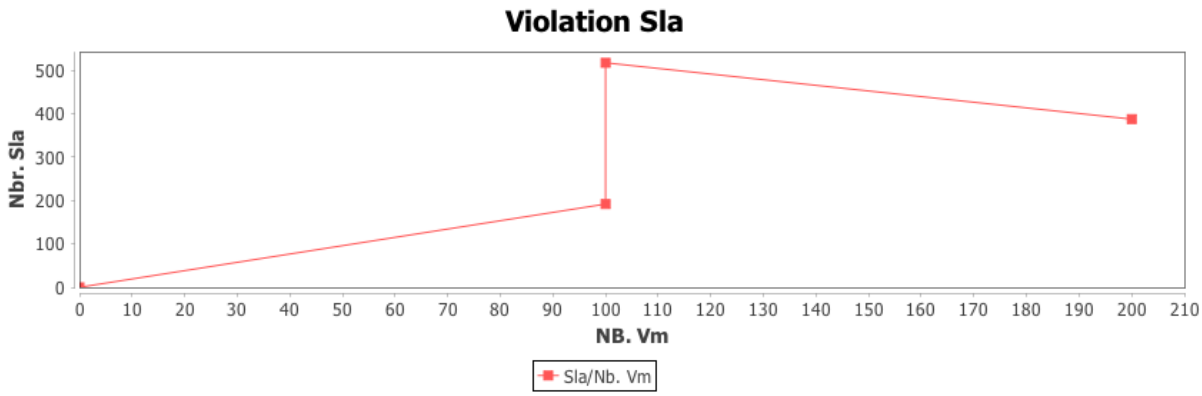


Figure 4.19 – Violation Sla

Nous avons remarqué que le nombre de violation sla se change celons le nombre de Vm.

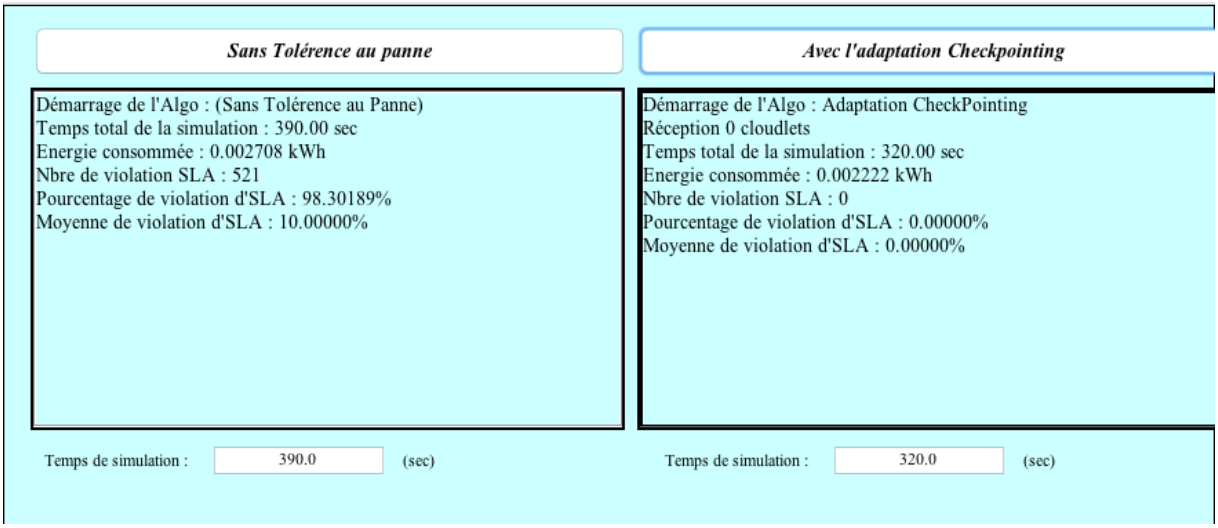


Figure 4.20 – Comparaison des approaches

Les travaux de recherches que nous avons menè dans le cadre de la problématique de la tolérance aux pannes dans les Cloud Computing, ainsi que les résultats que nous avons obtenu ont montrè que nos propositions ont permis d'obtenir effectivement des résultats meilleurs que l'approche sans tolérance aux pannes.

Le temps de simulation de notre approche est inférieur au 1er approche (sans tolérance au panne), la même chose concernant l'énergie consommée et la violation sla .

4.7 Conclusion

Nous avons effectué une série d'expérimentations pour évaluer le cout de notre approche de tolérance aux pannes proposées. Nous avons réalisé plusieurs simulations en jouant sur différents paramètres comme : le nombre de cloudlets, le nombre de pannes, l'intervalle de checkpoint. Les résultats montrent que l'utilisation de notre architecture réduit le temps perdu causé par les pannes et n'ont pas imposé un surcout élevé sur le temps d'exécution.

4.8 Conclusion général et perspectives

Un des points le plus important pour une utilisation plus efficace du système Cloud Computing est sans aucun doute l'étude de la fiabilité et la robustesse de ces systèmes.
Dans un tel contexte, la sureté de fonctionnement des applications est un élément de première importance où le taux de pannes dans de tels systèmes croit avec la taille du système lui-même, c'est pourquoi un mécanisme de tolérance aux pannes devient nécessaire pour en assurer certains aspects de son fonctionnement cohérent.
Le but de notre travail était la proposition d'une stratégie de tolérance aux pannes à base de checkpointing dans un environnement du cloud computing. Au cours de ce mémoire, Nous avons proposé une approche nommée ACP (Adaptatif Checkpointing) assure la continuité d'exécution des taches de types BoTs (Bag of Tasks) avec le minimum de surcharge et de violation de SLA surtout dans un environnement de type Soft-Deadlines. ACP manipule l'intervalle de checkpointing pour s'adapter avec la situation actuelle de coût et d'énergie ce qui réduit la production de CO2 et rend notre stratégie une stratégie verte.
Pour mettre en évidence notre approche proposée nous avons étendu le simulateur CloudSim et nous avons lancé un ensemble des expérimentations en créant plusieurs scénarios de pannes. Les performances sont évaluées à l'aide d'un ensemble des métriques telles que :
• la consommation d'énergie par les Data Center,
• La surcharge de checkpointing
• Violation SLA
Les résultats expérimentaux montrent l'efficacité de notre approche.
Dans les futures travaux nous pensons a d'étendre notre approche par l'utilisation de réplication pour assurer l'accessibilité des fichiers de checkpoint.

Bibliographie

[1] L. Saïd, gestion de tolérance aux fautes dans les cloud computing avec cloudsim.2012

[2] **I.Foster, Y. Zhao, I. Raicu, and S. Lu.**Cloud computing and grid computing 360-degree compared. s.l. : To appear at IEEE Grid Computing Environments, 2008. GCE08 co-located with IEEE/ACM Supercomputing.

[3] **Godefroy de Bentzmann.**Livre blanc « Tout ce que vous devez savoir sur l'informatique dans le nuage ». 148, Boulevard Haussmann - 75008 Paris : Syntec numérique, 2010.

[4] **NIST.** Nist definition of cloud computing v15. États-Unis : http://csrc.nist.gov/groups/ SNS/cloudcomputing/, 2010.

[5] **R. Buyya , S.K. Garg and R.N . Calheiros.** cloud computing: Challenges, architecture, and solutions. In Proceedings of the International Conference on Cloud and Service Computing (CSC), pages 1–10. Hong Kong, China : Sla-oriented resource provisioning, 2011.

[6] **R. Bloor , M. Kaufman and F. Halper.** Cloud Computing for Dummies. s.l. : Wiley Publishing, 2010. 9780470484708.

[7] **B. Randell ; A. Avizienis J; C. Laprie ; C. E.** "basic concepts and taxonomy of dependable and secure computing,". s.l. : IEEE Trans. Dependable Secure Computing, 2004. vol. 1.

[8] **P.David J; L.Dega; Y. Deswarte J; C. Laprie D.** "fault tolerant computing". s.l. : Willey Encyclopedia of Electrical and Electronics Engineering, 1999. ch. 7.

[9] **F.Pedone and Andr_e Schiper matthias Wiesmann.** "a systematic classification of replicated database protocols based on atomic broadcast.". s.l. : In ERSADS '99: 3rd Europeean Research Seminar on Advances in Distributed Systems, 1999. .

[10] **B. Meroufel and G.Belalem.** Lightweight coordinated checkpointing in cloud computing. s.l. : Journal of High Speed Networks, 2014.

[11] **R. Jhawar , V. Piuri and M. Santambrogio.** Fault tolerance management in cloudbcomputing: A system-level perspective. s.l. : IEEE Systems Journa, 2013.

[12] **X. Fan, W.-D. Weber, and L. A. Barroso,** "Power provisioning for a warehousesized computer," in Proceedings of the 34th Annual Intl. Symp. On Computer Architecture, (New York, NY, USA), pp. 13–23, ACM, June 2007.

[13] **D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang,** "Power and performance management of virtualized computing environments via lookahead control," in Proceedings of the 34th Annual Intl. Symp. On Computer Architecture, ICAC, pp. 3–12, September 2008.

[**14]** **P. Roques.** UML2 par la pratique. Eyrolles, 2008.

[15] **E.-N. Elnozahy. Manetho** : fault tolerance in distributed systems using rollback-recovery and process replication. PhD thesis, Rice University, Houston, TX, USA, 1993.

[16] **E. Arianyan, H. Taheri, and S. Sharifian**, "Novel energy and sla efficient resource management heuristics for consolidation of virtual machines in cloud data centers," Computers Electrical Engineering, vol. 47, pp. 222–240, 2015.

[17] **T. Chandra and S. Toueg**. Unreliable failure detectors for reliable distributed systems. ACM, 42(2) :225–267, 1996.

[18] **T. Zhijun, Y.-K. Richard, and W.-T. Tsai**. Alow overhead checkpointing and rollback recovery scheme for distributed systems. In Proceedings of the 8th Symposium on Reliable Distributed Systems, SRDS, pages 12–20, 1989.

[19] Jargon boster Qu'est-ce qu'une application composite : http://www.lansa.com/fr/resources/jargonbuster_compositeapp.htm , accès le 15.3.2015.

[20] langage java définition et historique [En ligne] Available at http://www.comoria.com/3516/Java_%28langage%29

[21] **Rodrigo N Calheiros, Rajiv Ranjan**, Anton Beloglazov, César AF De Rose et Rajk Umar Buyya. "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms". Dans: Software: Practice and Experience 41.1 (2011), p. 23–50 (cf. p. 48, 102, 103).

[22] **G. Booch, J. Rumbaugh, and I. Jacobson**. Le guide de l'utilisateur UML. Eyrolles, 2000. (Cité page 31.)

[23] **Belalem G. and Limam S.:** Towards Improving the Functioning of CloudSim Simulator. The International Conference on Digital Information Processing and Communications (ICDIPCí2011), V. Snasel, J. Platos, E. El Qawasmeh (Eds.), Proceedings Part II, CCIS Vol. 189, pp. 258-267, July 7-9, 2011, VSB- Technical University of Ostrava, Czech Republic, Springer, 2011. ISSN 1865-0929, ISBN 978-3-642-22409-6.

[24] Belalem G. and Limam S.: Fault Tolerant Architecture to Cloud Computing Using Adaptive Checkpoint. International Journal of Cloud Applications and Computing (IJCAC), Vol.1, No.4, pp. 60-69, 2011.