Université Abou Bekr Belkaid
Tlemcen Algeria
جامعة أبي بكر بلقايد
تلمسان الجزائر

République Algérienne Démocratique et Populaire

Université Abou Bakr Belkaid– Tlemcen

Faculté des Sciences

Département d'Informatique

Mémoire de fin d'études

Pour l'obtention du diplôme de Master en Informatique

Option : Réseaux et Systèmes Distribués (R.S.D)

Thème

Tolérance aux fautes des applications Cloud basées sur la technologie des conteneurs

Réalisé par :

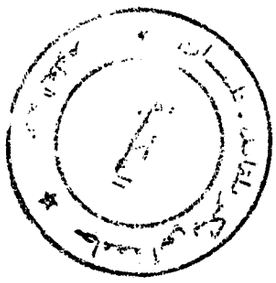
- **BELHAMRA Sarah**

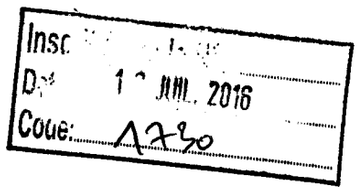
Présenté le 22 Juin 2016 devant le jury composé de :

- *Mme F.DIDI* *(Président)*
- *Mme D.DIB-MALTI* *(Encadreur)*
- *Mr M.MAATALAH* *(Examinateur)*
- *Mme S.HAMZA CHERIF* *(Examinateur)*

Année universitaire: 2015-2016

Dédicaces





A ma chère famille & mes amis …

A mon grand-père

Que dieu lui procure santé et longue vie.

Remercîments

En préambule à ce modeste mémoire je remercie ALLAH de m'avoir aidé et donner la patience et le courage durant ces langues années d'étude.

Je souhaite particulièrement adresser mes remerciements les plus sincères aux personnes qui m'ont apporté leur aide et qui ont contribué à l'élaboration de ce mémoire ainsi qu'à la réussite de cette formidable année universitaire. Ces remerciements vont tout d'abord au corps professoral et administratif de la Faculté d'Abou Bakr Belkaid des Sciences, département d'informatique, pour la richesse et la qualité de leur enseignements et qui déploient de grands efforts pour assurer à leurs étudiants une formation actualisée.

***Je** tiens à remercier sincèrement et respectivement mon encadreurs Madame Malti Djawda, qui s'est toujours montrée à l'écoute et très disponible tout au long de la réalisation de ce mémoire, ainsi pour l'aide et le temps qu'elle m'a consacré.*

***Je** remercie également tous mes ami(e)s qui m'ont accompagné et soutenu tout au long de ces années d'études.*

***Enfin,** j'adresse mes plus sincères remerciements ma famille surtout mes chers parents (M.Salah et H.Nacera), mes sœurs (Hanane et Fedwa), mes frères (Karim et Lotfi) et ma nièce (Manel) pour leur contribution, leur soutien et leur patience, que dieu vous garde pour moi.*

Merci à tous…

Sarah BELHAMRA

TABLE DES MATIERES

INTRODUCTION

Avec l'apparition d'internet dans les années 1990 jusqu'aux installations informatiques ubiquitaires actuelles, l'Internet a changé le monde de l'informatique de manière radicale, ce qui confirme la vision de Leonard Kleinrock l'un des principaux scientifiques du projet ARPANET (tout premier nom d'internet en 1969), qui prédisait l'évolution majeure de l'industrie informatique. Le parcours d'internet est passé par le concept de l'informatique parallèle puis l'informatique distribué jusqu'aux grilles et réseaux informatiques et récemment par le Cloud Computing.

Le Cloud Computing (l'informatique en nuages) est défini comme un environnement informatique qui offre des ressources informatiques sous forme de services distants sur internet. L'accès aux ressources Cloud se fait selon une architecture en couche composée de trois principaux niveaux : IaaS, PaaS et SaaS. Au niveau IaaS (Infrastructure as a Service) le consommateur dispose d'un accès complet à tous types de ressources : processeur, stockage, mémoire, périphériques réseaux. Par exemple : Amazon EC2. Au niveau PaaS (Platform as a Service) est un environnement d'hébergement est mis à la disposition des consommateurs pour ses applications. Le consommateur a la capacité de contrôler uniquement ses applications dans l'environnement hôte. Par exemple : la plateforme de développement Google Apps Engine. Au niveau SaaS (Software as a Service) le consommateur utilise une application Cloud hébergée, par exemple : Google drive.

Le Cloud Computing repose directement sur le concept de virtualisation. La virtualisation permet d'apporter une couche d'abstraction entre le matériel physique utilisé et les différents systèmes d'exploitation et applications que l'on souhaite utiliser.

Récemment la technologie de conteneurisation est de plus en plus utilisée à la place de la virtualisation étant plus avantageuse que la virtualisation. La conteneurisation consiste à encapsuler une application dans un conteneur. La conteneurisation est généralement connue par sa facilité et sa rapidité de lancement et pour son économie coté utilisation de ressources.

Comme tous services informatiques, les services Cloud sont vulnérables aux défaillances. Ce qui impose la disposition de mécanismes de tolérance aux fautes dans les environnements de Cloud Computing. La tolérance aux pannes dans ces systèmes est un domaine qui a été largement étudié. Il existe un grand nombre de protocoles et de

mécanismes de tolérance aux pannes précis pour chaque type de fautes. Cependant il y a un manque d'étude de tolérance aux fautes pour les Clouds utilisant des conteneurs.

L'objectif de ce projet est d'étudier la tolérance aux fautes dans un environnement Cloud basé sur la technologie des conteneurs. Nous nous intéressons plus particulièrement au mécanisme de sauvegarde/reprise (checkpoint/restore) et nous proposons une approche utilisant ce mécanisme pour rétablir un environnement cloud conteneurisé en cas de défaillance.

Le reste de ce mémoire est structuré comme suit. L'état de l'art sur les principaux domaines de ce mémoire est décrit en deux chapitres, le Chapitre I présente les concepts et terminologies liés au Cloud Computing et le Chapitre II présente les mécanismes de la tolérance aux fautes dans les différents systèmes informatiques. Le troisième chapitre présente l'objectif de ce projet. Au quatrième et dernier chapitre, nous décrivons notre approche de tolérance aux fautes et sa mise en œuvre. Enfin nous concluons notre mémoire et nous présentons quelques perspectives.

Chapitre I:

Cloud Computing

I. Introduction

Au milieu du 20ieme siècle, Internet a commencé à prendre forme, et sur le papier, dans les diagrammes et les présentations, il a généralement été représenté sous la forme d'un nuage. Bien sûr, le temps a passé et la technologie s'est rattrapé avec les nouvelles idées, parmi elles l'émergence de l'informatique en nuage « Cloud Computing » chez les grands acteurs de l'informatique : Salesforce.com, Amazon, Google, Oracle, Microsoft...

Dans ce premier chapitre nous présentons les concepts et la terminologie du Cloud Computing. Nous présentons également les technologies de base utilisées dans le Cloud : la *virtualisation* et la *conteneurisation*.

II. Cloud Computing

Le Cloud Computing ou informatique en nuage représente une infrastructure où la puissance de calcul et le stockage sont gérés sur des serveurs distants auxquels les usagers se connectent via une liaison Internet. N'importe quel terminal que ce soit ordinateur de bureau ou portable, téléphone mobile, tablette tactile ou autres objets connectés devient un point d'accès pour exécuter de divers applications ou consulter des données qui sont hébergées sur les serveurs cloud (voir Figure I.1). Le Cloud se caractérise également par son élasticité qui permet aux fournisseurs d'adapter automatiquement la capacité de stockage et la puissance de calcul aux besoins des utilisateurs [1] [4]. Ci-dessous les définitions du Cloud Computing présentées par CISCO et NIST :

« *Le Cloud Computing est une plateforme de mutualisation informatique fournissant aux entreprises des services à la demande avec l'illusion d'une infinité de ressources* [2] ». Définition de CISCO.

« Le *Cloud Computing est un modèle pour permettre, un accès pratique omniprésente au réseau sur demande à un pool partagé de ressources informatiques configurables (réseaux, serveurs, stockage, applications et services) qui peuvent être provisionnés rapidement et libérés avec un effort de gestion minimale ou interaction de fournisseur de service* [3] ». Définition de NIST.

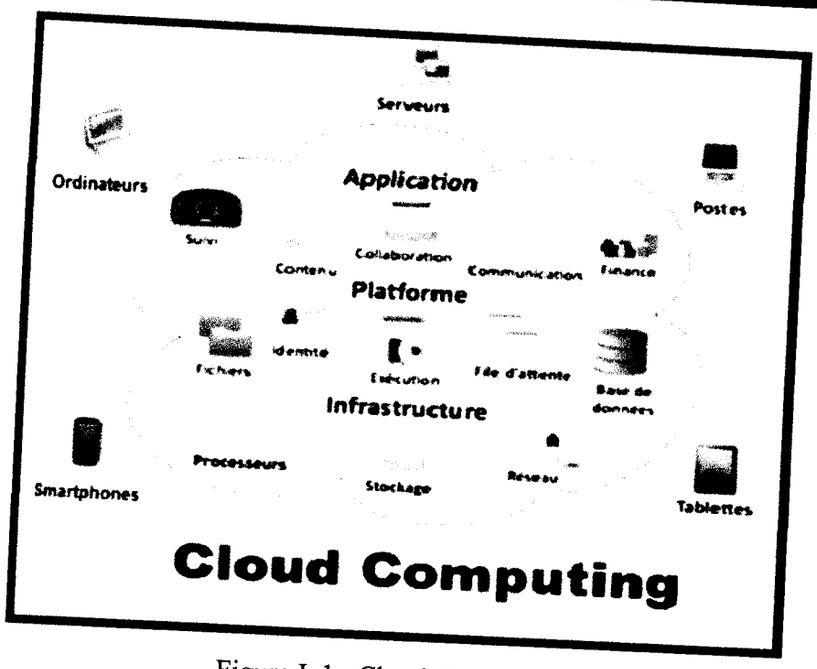


Figure I. 1 : Cloud Computing.

Parmi les différents services offerts par le Cloud Computing, il y a [37] :

- Services de courrier électronique tel que : Gmail, Yahoo, Hotmail…
- Services de stockage Cloud tel que : Google Drive, Dropbox, SkyDrive…
- Services de musique en nuage tel que : Google Musique, Amazon Cloud Player, iTunes / iCloud…
- Cloud Apps tel que : Google Docs, Photoshop Express…
- Les systèmes d'exploitation Cloud tel que : Google Chrome OS, Jolicloud…

II.1 Les caractéristiques du Cloud Computing

Le Cloud Computing se distingue des environnements informatiques classiques par les caractéristiques suivantes [7] :

- Réservoir de ressources (non localisées) qui signifie un usage simplifié grâce à l'indépendance d'outils informatiques traditionnels.
- élasticité (redimensionnement rapide) et d'agilité car il permet d'accéder plus rapidement à des ressources IT (serveur, stockage ou bande passante).
- Facturation à l'usage ce qui implique une réduction des coûts (pay-as-you-go).
- Accès réseau large bande.

- Accès à la demande et la disponibilité des services garantit.
- La sécurité grâce aux dispositifs de sécurité (réplication des données, plan de reprise d'activité, cyberdéfense,…).

II.2 Les niveaux de services du Cloud Computing

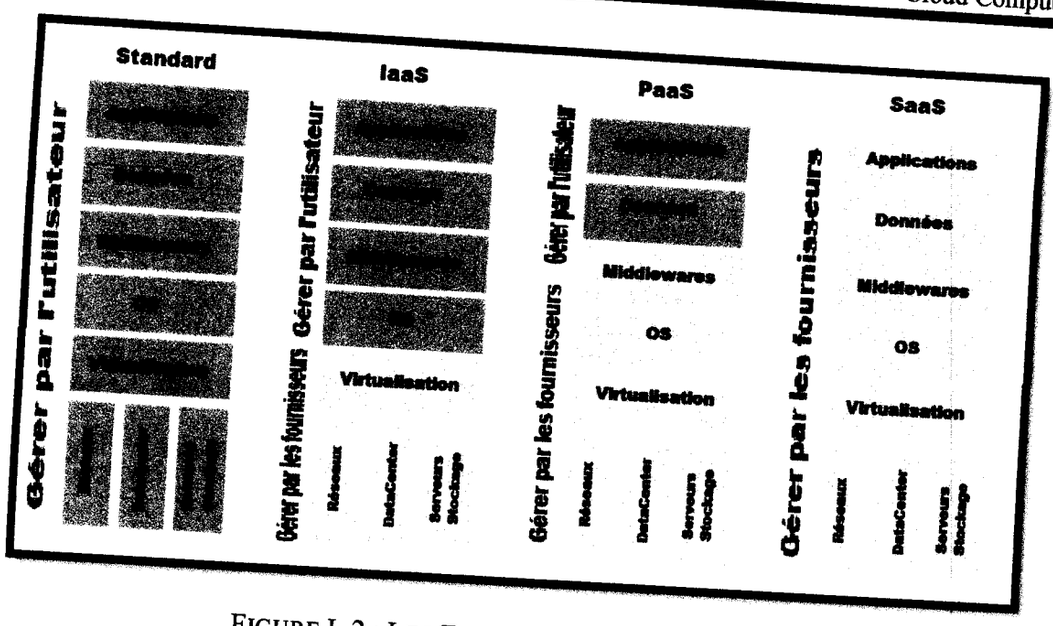


FIGURE I. 2 : LES TYPES DE SERVICE DU CLOUD.

II.3 Les modèles de déploiement

Il existe plusieurs modèles de déploiement des services Cloud Computing, mais les plus connus sont trois : public, privé et hybride [5] [6] [8] (voir Figure I.3).

II.3.1 Le Cloud Public

Dans ce modèle l'infrastructure est gérée par un fournisseur de Cloud qui a des objectifs commerciaux et les utilisateurs n'ont aucun contrôle sur cette dernière. L'infrastructure du Cloud public offre des services, elle est mise à la disposition des utilisateurs Cloud grand public ou d'un grand groupe d'entreprises via Internet. Les ressources informatiques peuvent être partagées entre plusieurs utilisateurs.

II.3.2 Le Cloud Privé

Dans un Cloud privé, l'utilisateur des ressources contrôle et possède l'infrastructure. Cette dernière est gérée avec des solutions de gestion de Cloud pour offrir les ressources et services par le biais d'interface Cloud. Les Ressources disponibles sont destinées seulement pour une utilisation privée et non pas au grand public.

II.3.3 Le Cloud Hybride

Le Cloud Hybride est un environnement qui combine les deux modèles Public et Privé. Les ressources peuvent être allouées à partir d'un Cloud Privé et d'un Cloud Public. Il est possible de stocker et gérer les données confidentielles dans l'environnement privé et celles qui sont

moins confidentielles dans un Cloud Public. Ils sont toutefois liés par l'entremise de technologies standards ou propriétaires.

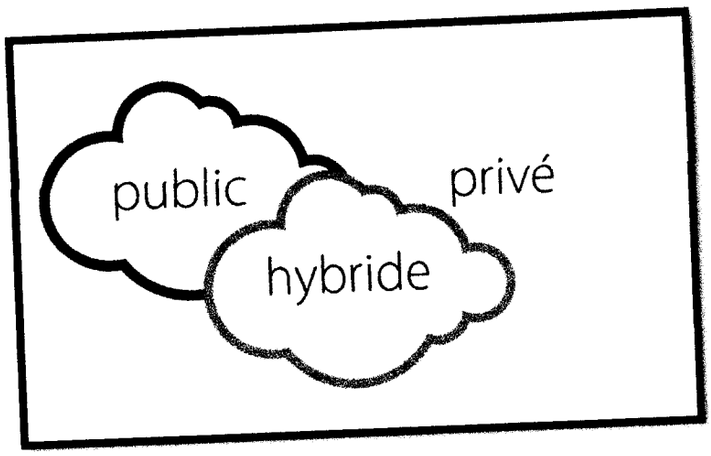


Figure I. 3 : Les modelés du Cloud Computing.

III. Virtualisation

« La virtualisation consiste à faire fonctionner un ou plusieurs systèmes d'exploitations/ applications comme un simple logiciel, sur un ou plusieurs ordinateurs - serveurs / système d'exploitation, au lieu de ne pouvoir en installer qu'un seul par machine. »

Dans la virtualisation, on peut distinguer trois catégories [11] [19] :

- **La virtualisation de stockage :** elle est obtenue en isolant les données de leur localisation physique. Elle permet de masquer les spécificités physiques des unités de stockage. Côté utilisateur, les unités de stockage sont vues comme un unique volume.
- **La virtualisation de serveur :** elle consiste à faire fonctionner plusieurs serveurs virtuels sur un seul serveur physique. Le but de ce type de virtualisation est d'augmenter la capacité de serveurs et en même temps réduire les couts.
- **La virtualisation de réseau :** consiste à combiner des ressources réseau matérielles et logicielles dans une seule unité administrative. L'objectif est de fournir aux systèmes et utilisateurs un partage efficace, contrôlé et sécurisé des ressources réseau.

Dans ce projet on s'intéresse à la virtualisation de serveur. Dans ce qui suit on présente ses différents types.

III.1 Virtualisation de serveurs

La virtualisation est un mécanisme informatique qui consiste à utiliser des moyens techniques (matériels et/ou logiciels) afin de faire fonctionner plusieurs systèmes, serveurs ou applications, sur une seule machine physique en donnant allusion qu'ils fonctionnent sur différentes machines. La virtualisation est un composant technique de base dans le Cloud Computing [9] [10].

La virtualisation repose sur le mécanisme suivant :

- Un système d'exploitation principal « système hôte » est installé sur un serveur physique unique. Ce système sert d'accueil à d'autres systèmes d'exploitation.
- Un logiciel de virtualisation « hyperviseur» est installé sur le système d'exploitation principal. Il permet la création d'environnements clos et indépendants sur lesquels seront installés d'autres systèmes d'exploitation « systèmes invités » ou machines virtuelles. Ces dernières ont un accès aux ressources du serveur physique (mémoire, espace disque…) [9].

Dans ce qui suit on présente deux technologies de la virtualisation de serveurs : virtualisation système et virtualisation applicative.

III.1.1 Virtualisation Système

La virtualisation système fait référence à l'utilisation de logiciel pour permettre au matériel du système d'exécuter plusieurs instances de différents systèmes d'exploitation en même temps, permettant ainsi d'exécuter différentes applications nécessitant différents systèmes d'exploitation sur même serveur physique [39].

Les technologies et les logiciels qui offrent la virtualisation système sont : VM et VMM.

-Machine Virtuelle (VM) : Une VM est un environnement d'application ou de système d'exploitation installé sur un logiciel totalement isolé, qui imite un matériel dédié (ordinateur). Côté utilisateur, l'interaction avec une machine virtuelle est la même qu'avec un matériel dédié. Une machine virtuelle (VM) se comporte exactement comme un ordinateur physique. Elle contient son propre matériel virtuel : CPU, mémoire RAM, disque dur et carte d'interface réseau (NIC) basés sur du logiciel.

-Hyperviseur : Un hyperviseur aussi connu comme un gestionnaire de machine virtuelle (VMM) est une entité logicielle chargée de l'attribution des ressources d'une ou plusieurs machines physiques aux machines virtuelles, ainsi que de la gestion de ces dernières. On distingue deux types d'hyperviseurs [12] [13] [14] :

- **Hyperviseur de type 1 (natif)**

Il s'installe directement sur la couche physique matérielle du serveur. Au démarrage d'une machine physique, cet hyperviseur prend directement le contrôle du matériel (voir Figure I.4). Comme exemple d'hyperviseur de type 1 nous citons : Hyper-V, Xen, VMware, ESXi, KVM,...

- **Hyperviseur de type 2 (hébergé)**

Il est assez comparable à un émulateur il s'installe sur un système d'exploitation déjà en place (OS chargé sur la couche matérielle) (voir Figure I.5). Il se démarre la plupart du temps comme une simple application. Par exemple nous avons : Workstation, Oracle VM (Virtual Box), Parallels, Fusion,...

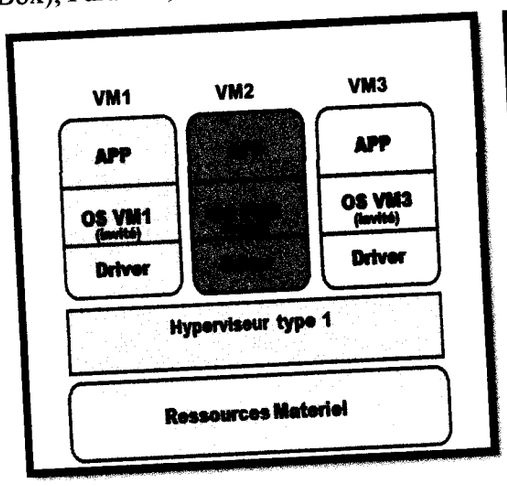


Figure I. 4 : Hyperviseur de type 1.

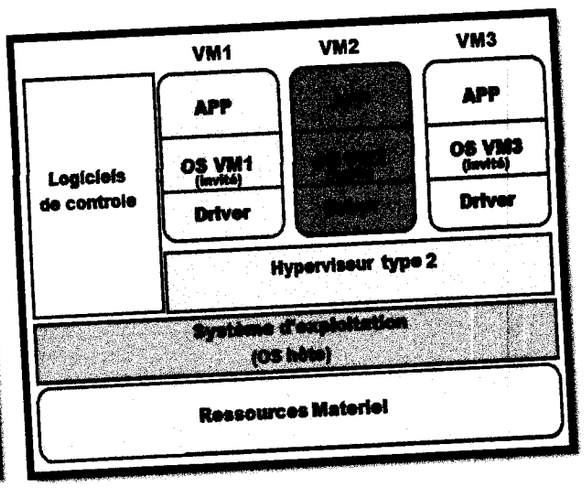


Figure I. 5 : Hyperviseur de type 2.

III.1.2 Virtualisation applicative

Contrairement à la virtualisation système, le principe de cette virtualisation est de mettre en œuvre une couche de virtualisation sous forme d'une application qui elle-même crée des machines virtuelles qui peuvent être soit des simples interpréteurs de langage ou aussi complexe que la JVM. Elle permet d'exécuter partout (sous différents environnements et OS) une application et garantit la portabilité des applications. Exemples de machines virtuelles applicatives : JVM, conteneurs

- JVM (Java Virtual Machine) : est l'exemple le plus connu de la virtualisation applicative. La JVM est un environnement d'exécution pour applications Java mais pleins d'autre langages aussi tel que : Python(Jython) ou Ruby(JRuby).

- **Conteneurs :** est une approche de la virtualisation dans laquelle la couche de virtualisation fonctionne comme une application au sein du système d'exploitation (OS).

IV. Conteneurisation

IV.1 Définition:

La conteneurisation est une alternative légère à la virtualisation de machine qui consiste à encapsuler une application dans un conteneur avec son propre environnement d'exploitation.

Un conteneur fournit les ressources nécessaires pour exécuter des applications comme si elles étaient les seules applications en cours d'exécution dans le système d'exploitation et ceci avec une garantie d'un fonctionnement sans conflits avec d'autres conteneurs d'application en cours d'exécution sur la même machine. Les conteneurs s'installent par-dessus le noyau du même système d'exploitation (voir figure I.6). Comme les conteneurs de chaque application sont libérés de la charge d'un OS, ils sont nettement plus petits, plus légers qu'un OS d'une VM et plus faciles à migrer ou à télécharger, plus rapides à sauvegarder ou à restaurer, c'est pour cela qu'un système d'exploitation peut avoir plusieurs conteneurs [15].

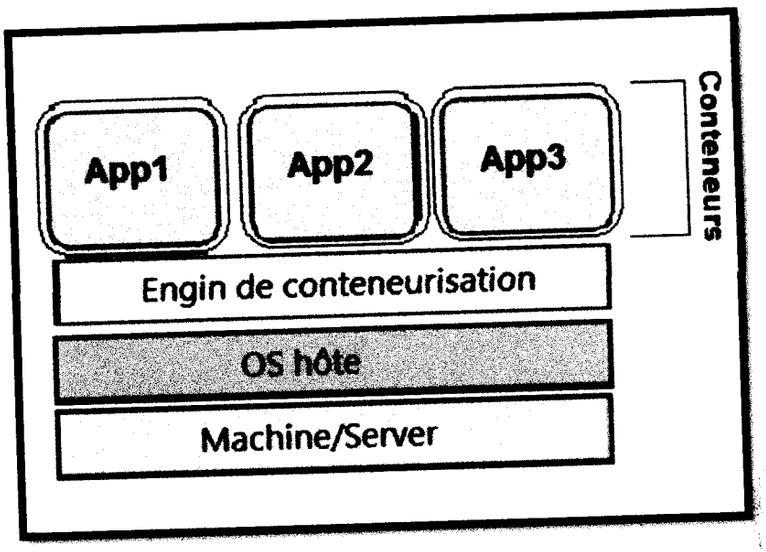


Figure I. 6 : La containerisation.

IV.2 Types de conteneurs Linux

Il existe de nombreuses technologies de conteneurs Linux qui fonctionnent tous en utilisant les mêmes principes d'isolement d'espace d'application au sein d'un système d'exploitation. Voici les conteneurs les plus connus :

IV.2.1 LXC :

LXC (LinuX Containers) est une technologie de virtualisation au niveau du système d'exploitation linux, qui permet la création et l'exécution de plusieurs Environnements Virtuels isolés (EV) sur une même machine (hôte de contrôle).

LXC utilise la fonctionnalité des groupes de contrôle de Linux, qui a été introduit dans la version 2.6.24 afin de permettre au processeur de l'hôte une meilleure partition et allocation de la mémoire dans des niveaux d'isolement appelés espaces de noms. Ces fonctionnalités offrent aussi aux applications une isolation totale des ressources (notamment processeur, accès E/S et mémoire), et ce sans recourir à des machines virtuelles système à part entière [16] [17].

IV.2.2 Docker :

Docker anciennement appelé dotCloud est définit comme étant un environnement, plateforme de conteneurisation et même un gestionnaire de conteneurs open source pour les systèmes d'exploitation basés sur UNIX. En réalité c'est une extension des capacités de LXC. Donc, il permet de créer de nombreux environnements isolés sur un seul noyau.

Docker est développé dans le langage Go et il utilise LXC, les groupes de contrôle, et le noyau Linux lui-même. Comme il est basé sur LXC, un container Docker ne comprend pas un système d'exploitation distinct.

Docker, ne remplace pas les conteneurs Linux. L'idée consiste à utiliser LXC comme base, puis à ajouter des capacités de niveau supérieur. Par example:

- Docker autorise la portabilité entre machines (qui exécutent aussi Docker) et permet ainsi à une application et à ses composants d'exister en tant qu'objet mobile unique.
- Docker propose des outils de génération automatisée de build. Ces outils aident les développeurs à passer plus facilement d'un code source à des applications conteneurisées, ou à utiliser des outils d'accompagnement, tels que Chef, Maven, Puppet et autres, afin d'automatiser ou de rationaliser le processus de build.

Docker est l'une des dernières technologies de virtualisation proposé au grand public [16] [17].

V. Comparaison entre VMs et Conteneurs

Le tableau suivant montre les principales différences entre les machines virtuelles et les conteneurs [14] [18] :

VMs	Conteneurs
Les hyperviseurs doivent créer une copie complète du système d'exploitation qui fonctionne sur un espace matériel virtuel.	Libéré de la charge d'un système d'exploitation
Chaque VM « invitée » contient un système d'exploitation complet, avec ses pilotes, fichiers binaires ou bibliothèques, ainsi que l'application elle-même.	Partage un système d'exploitation hôte unique, avec ses fichiers binaires, bibliothèques ou pilotes.
Lourd, compliqué et lent à déployer.	Léger, simple et rapide à déployer.
Gaspille la mémoire du serveur et limite forcément le nombre de VM prises en charge par chaque serveur.	Réduit le gaspillage des ressources.

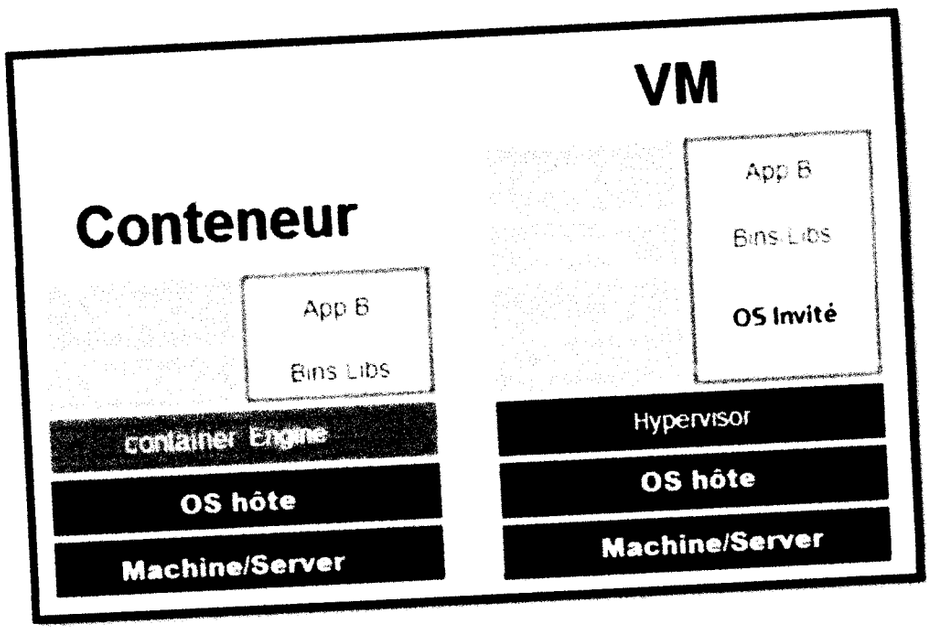


Figure I. 7 : Conteneur vs VM.

VI. Conclusion

Dans ce premier chapitre, nous avons présentés les notions de base du Cloud Computing. Précisément, nous avons définis ses différents services et modèles.

Le Cloud Computing est une solution de la dématérialisation de l'informatique. La virtualisation est un composant technique clé dans le Cloud Computing.

Dans le chapitre suivant nous discuterons en premier lieu les principes de la sureté de fonctionnement des systèmes puis nous nous approfondissons dans le principal sujet de ce mémoire : la tolérance aux fautes et ses mécanismes.

Chapitre II :

La Tolérance aux Fautes

I. Introduction

Le Cloud Computing comme tout autre système n'est pas à l'abri des fautes et des pannes. C'est pour cela qu'il existe plusieurs mécanismes et méthodes pour tolérer tout type de fautes. La tolérance aux fautes a pour but d'éviter les défaillances, c'est l'un des moyens permettant d'assurer la sureté de fonctionnement d'un système.

Ce chapitre présente les principes liés à la conception de systèmes informatiques tolérants aux fautes. Nous abordons, tout d'abord les notions de base, méthodes et techniques de la sureté de fonctionnement puis nous décrivons quelques techniques de la tolérance aux fautes dans les systèmes et notamment les systèmes répartis et les Clouds.

II. Principes de la sureté de fonctionnement

La sûreté de fonctionnement d'un système est sa capacité à fournir un service de confiance justifiée. Cette propriété regroupe trois notions : ses attributs, ses entraves et ses moyens (voir Figure II.1) [21] [22].

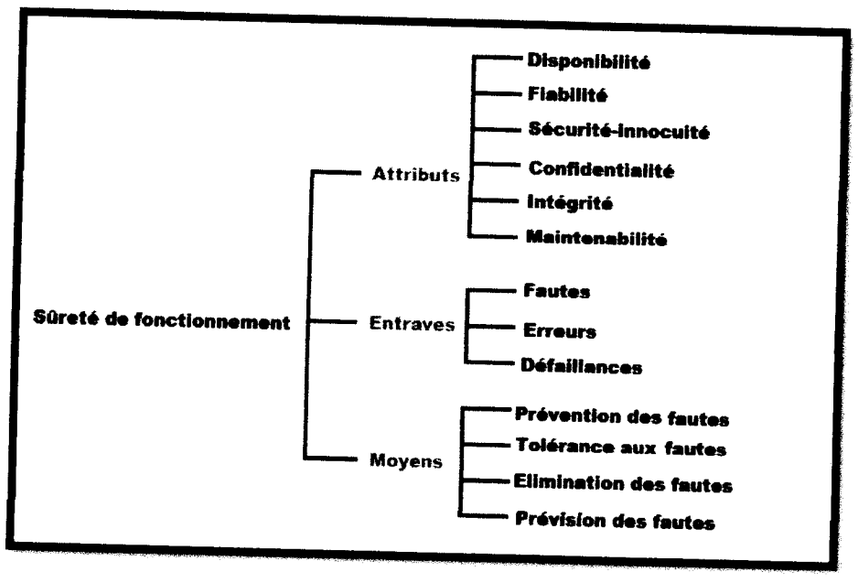


Figure II. 1 : L'arbre de la sûreté de fonctionnement.

Dans ce qui suit nous décrivons chaque notion de cet arbre.

II.1 Attributs de la sureté de fonctionnement

Les attributs de la sûreté de fonctionnement sont définis par diverses propriétés que doit vérifier la sûreté de fonctionnement du système. Parmi ses attributs il y a [21] [22] [23] [24]

-Les Erreurs : C'est la partie fausse de l'état interne du système. C'est aussi la conséquence d'une faute. Les erreurs peuvent être latentes ou détectées. Une erreur est latente tant qu'elle n'a pas été reconnue en tant que telle. Une erreur est détectée par un algorithme ou un mécanisme de détection qui permet de la reconnaître comme telle.
Une erreur activée est appelée erreur effective, elle crée une défaillance. Une erreur peut disparaître avant d'être détectée. La propagation d'une erreur produit de nouvelles erreurs (voir Figure II.2).

-Les Défaillances : une défaillance est la déviation de l'accomplissement de la fonction du système. C'est aussi l'incapacité d'un système à assurer un service spécifié par l'utilisateur. Une défaillance survient lorsqu'une erreur traverse l'interface système-utilisateur et affecte le service délivré par le système.

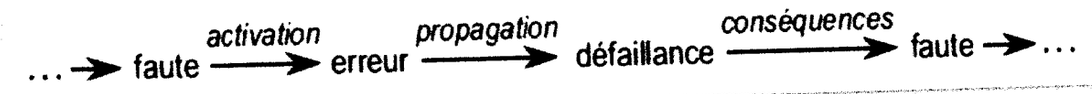


Figure II. 2 : Chaine des entraves de la sureté de fonctionnement.

Pour assurer la sûreté de fonctionnement, il faut soit éviter que la source de la chaîne fondamentale des entraves - la panne - ne se produise, soit éviter que la panne ne se transforme en erreur, puis en défaillance.

II.3 Moyens d'assurer la sûreté de fonctionnement

Les moyens utilisés pour assurer la sureté de fonctionnement d'un système lors de son déploiement, constituent un ensemble de méthodes, outils et de solutions.
Parmi les moyens les plus connues il y a [21] [22] [23] [25] [26] :

-La Prévention des fautes : le but est d'empêcher l'introduction ou l'occurrence de fautes dans le système, par construction.
-La Tolérance aux fautes : le principe est de continuer à fournir des services spécifiés malgré la présence de fautes.
-L'élimination des fautes : permet de réduire le nombre, la gravité ou l'impact des fautes dans le système, par vérification, par maintenance.

-La prévision des fautes : c'est une technique pour estimer la présence, la création et les conséquences des fautes dans le système, par évaluation.

III. Taxonomie des fautes

La détection des fautes revient à la capacité d'identifier leurs types ou modèles. Les fautes peuvent être classées selon plusieurs critères, comme décrit dans l'arbre suivant [21] [22] (Figure II.3).

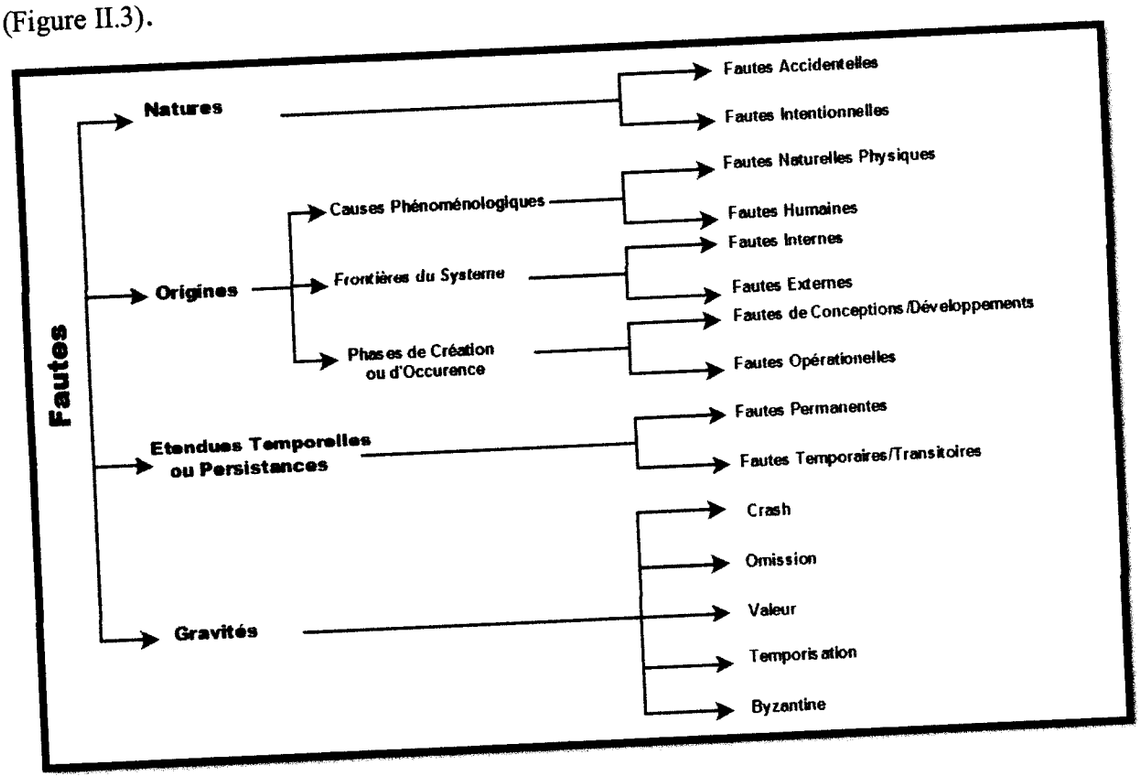


Figure II. 3 : Arbre des classes des fautes.

IV. Mécanismes de tolérance aux fautes

La capacité d'un système à fonctionner malgré une défaillance d'une de ces composantes est appelée tolérance aux fautes ou tolérance aux pannes.

La tolérance aux fautes est mise en œuvre par deux techniques : la détection des erreurs et le rétablissement du système [1].

Dans ce qui suit on va voir les techniques de détection de panne puis les méthodes de tolérance aux fautes.

IV.1 Détection des pannes

La détection des pannes ou erreurs est la première étape de la tolérance aux fautes. Son objectif est de prévenir (si possible) l'occurrence d'une défaillance provoquée par l'erreur, et éviter la propagation de l'erreur à d'autres composants. Ceci facilite l'identification ultérieure de la faute (en vue de son élimination ou de sa prévention). Il y a deux formes de détections :

- Détection concomitante (simultanée) est réalisée lors de l'exécution normale du service.
- Détection préemptive est effectuée lors de suspensions du service destiné à révéler l'éventuelle présence d'erreurs latentes ou de fautes dormantes.

La détection d'erreur s'appuie sur les paramètres suivants :

- **La latence :** délai entre production et détection de l'erreur.
- **Le taux de couverture :** pourcentage d'erreurs détectées.

La détection d'erreurs comprend deux techniques [21] [28] :

- **Comparaison des résultats de composants dupliqués** : ici le cout est élevé, sa latence est faible et son taux de couverture est élevé.
- **Contrôle de vraisemblance :** son coût est modéré, sa latence est variable selon la technique et son taux de couverture est souvent faible.Le contrôle de vraisemblance peut être mis en œuvre par : du matériels/logiciels spécifiques ou des programmes de tests périodiques ou apériodiques.

IV.2 Rétablissement du système :

C'est la transformation de l'état erroné du système en un état exempt d'erreur détectée et de faute qui puisse être activée à nouveau. Le traitement d'erreur, invoqué à la demande, est suivi du traitement de fautes constituant l'étape de rétablissement du système.

Voici les étapes de rétablissement du système chacune avec ses techniques :

IV.2.1 Traitement d'erreur : consiste à éliminer les erreurs de l'état du système, et ceci avec les techniques suivantes :

Reprise : c'est la technique la plus utilisée. Ici le système est ramené dans un état sauvegardé survenu avant l'occurrence d'erreur (état sauvegardé : point de reprise).

Poursuite : cette technique recherche un nouvel état exempt d'erreur détecté et trouvé.

Compensation : état erroné comporte suffisamment de redondance pour permettre de masquer l'erreur et de délivrer un service approprié.

IV.2.2 Traitement de faute : prévention d'une nouvelle activation de faute constituée des étapes suivantes :

Diagnostic : la première étape qui identifie et enregistre les causes de l'erreur détectée, en termes de localisation et de type ou nature.

Passivation : c'est assurer l'exclusion logique et physique des composants fautifs du processus, autrement dit cette étape consiste à empêcher la réactivation des fautes, elle rend la faute dormante.

Reconfiguration : réaffecte les tâches parmi les composants non défaillants ou intègres des composants en réserve.

Réinitialisation : fait le contrôle, mise à jour et l'enregistrement de la nouvelle configuration, et aussi la mise à jour des tables et des enregistrements du système.

V. Tolérance aux fautes dans les systèmes répartis

V.1 Systèmes répartis

"Un système réparti est un ensemble de machines autonomes connectées par un réseau, et équipées d'un logiciel dédié à la coordination des activités du système ainsi qu'au partage de ses ressources." [35].

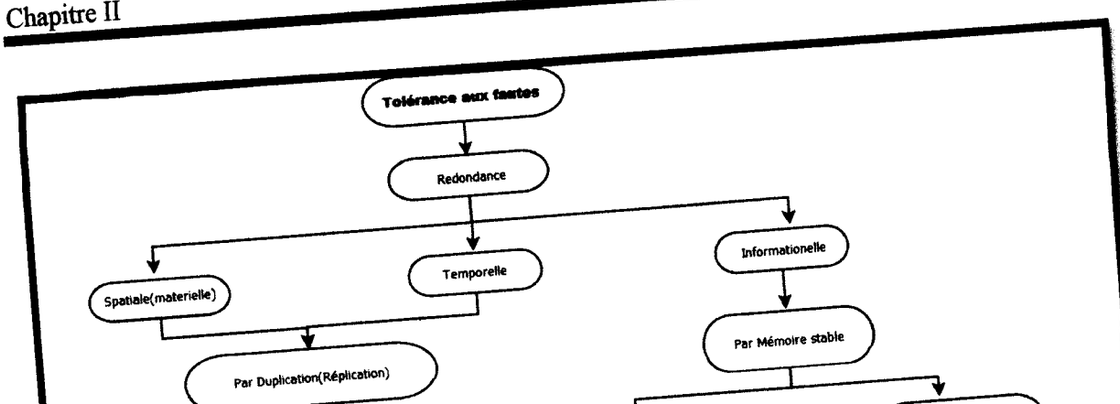
Un système réparti est un système informatique dans lequel les ressources ne sont pas centralisées. Quand on parle de ressources par exemple : stockage (disques, bases de données), puissance de calcul,... [32].

V.2 Techniques de tolérance aux pannes

La tolérance aux fautes dans les systèmes répartis est principalement basée sur le mécanisme de redondance. Il y a trois types redondance : d'informations (duplication de données/détection d'erreur), temporelle (traitements et calculs multiples) ou architecturale/matérielle (composants matériels/logiciels multiples) [22] [25] [28].

Comme nous pouvons le voir dans la figure II.4, les types de redondance se devisent en deux catégories :

- Redondance spatiale et redondance temporelle se font par les techniques de la duplication.
- Redondance d'information se fait par et les techniques basées sur une mémoire stable.



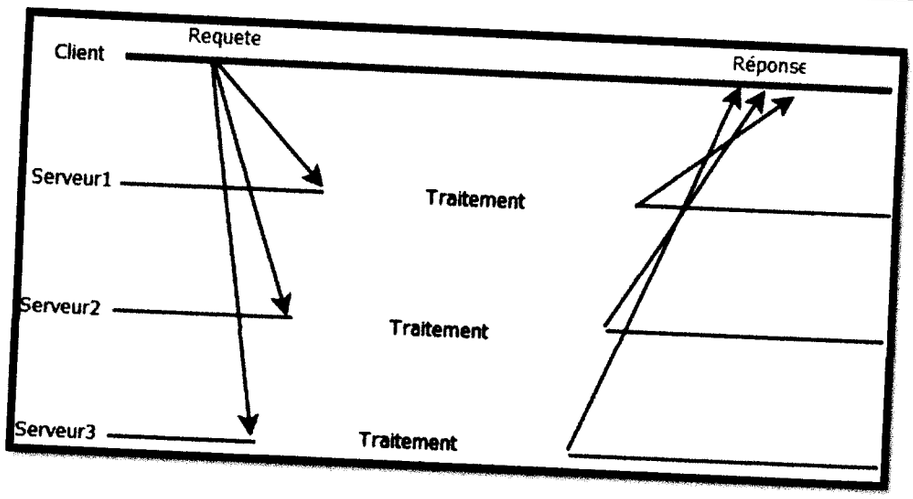


Figure II. 5 : Duplication active.

-Duplication passive (primary-backup) : contrairement à la duplication active, la duplication passive distingue deux groupes de serveurs. La requête du client est envoyée à une seule copie appelée « serveur primaire » qui l'exécute et la traite. Les autres répliques appelées « secondaires » (backups en anglais) ne reçoivent aucune requête du client (voir Figure II.6). Une fois le traitement terminé, le serveur primaire met à jour les serveurs secondaires puis il envoie la réponse au client. Dans le cas où le serveur primaire n'est plus fonctionnel, une des copies secondaires est élue pour devenir primaire.

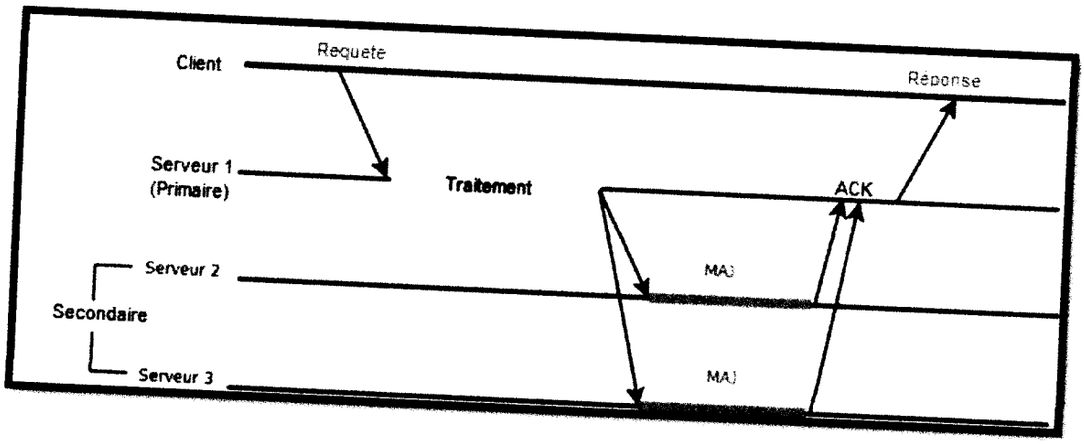


Figure II. 6 : Duplication passive.

-Duplication Semi-active : est une technique hybride entre la duplication active et la duplication passive, elle désigne une copie comme étant un meneur (maitre/leader) et les autres copies comme des suiveurs (followers en anglais) qui traitent les requêtes envoyées par le client comme dans la duplication active sauf que seul le leader renvoie la réponse comme dans la duplication passive (voir figure II.7).

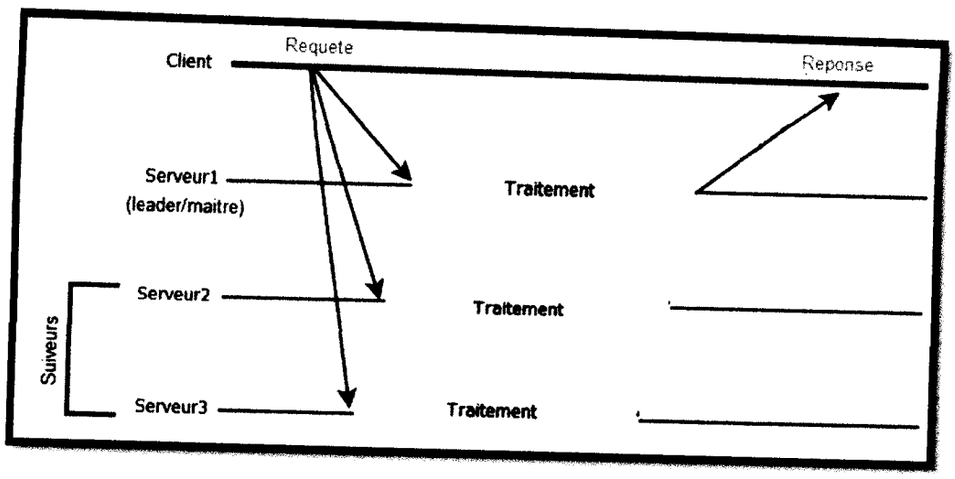


Figure II. 7 : Duplication semi-active.

V.2.2 Tolérance aux pannes par Mémoire stable :

La tolérance aux pannes par recouvrement arrière repose sur l'existence d'une mémoire stable.

Une mémoire stable représente un support persistant de stockage, son rôle principal est d'assurer l'accessibilité et la protection des données contre les pannes. Cette dernière doit avoir les propriétés suivantes [22] [30] [31] :

- **Accessibilité aux données :** la mémoire doit être accessible à tous les éléments de l'application.
- **Protection contre les pannes :** la mémoire doit survivre et résister aux pannes qui sont tolérées par le mécanisme de tolérance aux pannes.

Comme nous avons vu dans la figure II.4, la tolérance par mémoire stable se fait par deux techniques : la sauvegarde périodique de point de reprise et la journalisation.

A. Sauvegarde (CheckPointing):

Cette technique consiste à sauvegarder l'état des processus dans un support de stockage stable, pour pouvoir ensuite restaurer l'état du système à partir d'un point de reprise. Il y a trois catégories de sauvegarde [33] :

- **Sauvegarde synchronisée (coordonnée)**

La sauvegarde synchronisée se fait avec des algorithmes appelés synchrones, cohérents ou globaux. Leur principe est de définir l'état global cohérent au moment de la sauvegarde et plus précisément avant de procéder à l'écriture des fichiers de sauvegarde sur la mémoire stable. Ainsi, au moment de la sauvegarde, une phase de synchronisation durant laquelle les calculs sont interrompus est imposée à tous les processus de

l'application afin de déterminer l'état global cohérent.

L'avantage de cette approche est de ne pas produire d'effet domino en cas de reprise car les points de reprise sont garantis cohérents. De plus, un seul point de reprise par processus est nécessaire ce qui réduit le surcoût de stockage et de libération de points de reprise. Le principal inconvénient de cette technique est le surcoût introduit par la coordination de tous les processus participants.

Afin d'améliorer les performances de la sauvegarde coordonnée, les techniques suivantes ont été proposées :

- la sauvegarde coordonnée non bloquante évite de bloquer le processus durant la phase de sauvegarde en créant un processus clone chargé de la sauvegarde ;
- la sauvegarde avec horloge synchronisée évite la phase de synchronisation par envois de messages, en utilisant une horloge synchronisée ;
- la sauvegarde coordonnée minimale évite la phase de synchronisation globale entre tous les processus en se basant sur le fait qu'un processus n'a besoin de se coordonner qu'avec les processus avec lesquels il a des dépendances.

- **Sauvegarde indépendante (non coordonnée)**

Les techniques non coordonnées évitent la phase de synchronisation en laissant à chaque processus la décision de sauvegarder son état. Le principal avantage de la sauvegarde non coordonnée est qu'un processus peut procéder à la sauvegarde de son état lorsque celui-ci est minimal, réduisant ainsi le surcoût de la sauvegarde en termes de quantité d'informations à sauvegarder.

Mais l'inconvénient principal de cette approche est le risque d'effet domino qui peut causer une perte importante du travail réalisé avant la panne et la sauvegarde inutile de points de reprise, ceux-ci ne faisant jamais partie d'un état global cohérent. Ces points de reprise augmentent le coût de l'exécution normale et ne servent pas à la reprise après panne. De plus, la sauvegarde non coordonnée oblige chaque processus à maintenir plusieurs points de reprise. Afin de déterminer un état global cohérent, chaque processus dispose d'un journal en mémoire stable dans lequel il enregistre tout ou partie des messages échangés ainsi que leurs historiques. Lorsqu'une défaillance survient, l'algorithme de reprise utilise les points de reprise locaux et les journaux afin de déterminer une ligne de recouvrement.

- **Sauvegarde induite par communication**

La sauvegarde induite par les communications, est un compromis entre la sauvegarde coordonnée et la sauvegarde non coordonnée.

L'idée principale est d'une part, d'éviter la coordination des processus en permettant la sauvegarde non coordonnée des processus, appelée sauvegarde locale, et d'autre part, d'éviter l'effet domino en forçant un processus à sauvegarder son état en cas d'évaluation de la ligne de recouvrement. Cette sauvegarde est alors appelée sauvegarde forcée.
Le principe de la sauvegarde induite par les communications est d'étendre un ensemble de points de reprise locale de l'application à un ensemble de points de reprise constituant un état global cohérent.

B. Journalisation (Logging):

Le principe de cette technique est de sauvegarder l'histoire d'exécution des applications durant leur exécution. En cas de panne d'un processus, son exécution est relancée et est contrôlée jusqu'à ce que le processus atteigne l'état dans lequel il était juste avant la panne. Le système retrouve alors un état cohérent. Il y a trois types de journalisations [33] :

- **Journalisation pessimiste**

Le principe de cette approche est d'enregistrer tous les messages en transit dans le système sur une mémoire stable d'une façon synchrone. Quand un message arrive dans le contexte du récepteur il est forcément enregistré par la suite il pourra être rejoué en cas de panne. Ainsi, tous les points de reprise du processus sont utilisables pour une reprise, puisque tous les événements non déterministes ont été enregistrés depuis la prise de ce point. L'ordre de ces messages doit lui aussi être conservé, de manière à recréer lors de la reprise la même histoire de message.

- **Journalisation optimiste**

La journalisation optimiste, qui a pour but d'améliorer la performance d'une exécution normale, elle s'appuie sur l'hypothèse "optimiste" qu'un processus peut compléter la journalisation de son événement non déterministe avant qu'une panne ne survienne. Donc, le stockage sur la mémoire stable des informations correspondant à un événement non déterministe est asynchrone et le processus n'a pas besoin de se bloquer pendant le stockage sur la mémoire stable. La journalisation optimiste ne garantit pas toujours la "condition de non orphelinité", ce qui rend l'opération de reprise compliquée, car il faut éventuellement retourner en arrière plusieurs fois avant de trouver un état cohérent.
Les inconvénients de cette journalisation est dans le risque d'un retour à l'état initial de calcul. Ce phénomène est communément appelé l'effet domino, ainsi le surcoût important dû au calcul de l'état global cohérent est introduit durant la reprise.

- **Journalisation causale**

Le principe de la journalisation causale est d'assurer que le déterminant de chaque événement non déterministe qui précède causalement l'état d'un processus se trouve soit en mémoire stable, soit est disponible localement pour ce processus.

La journalisation causale a l'avantage des deux méthodes précédentes en termes de performances à l'exécution et en cas de reprise. Comme la journalisation optimiste, la journalisation causale évite d'une part, la synchronisation avec la mémoire stable à l'exception du moment de la publication de résultats. D'autre part, comme dans la journalisation pessimiste, la journalisation causale ne crée jamais de processus orphelin. Cela permet la reprise de n'importe quel processus défaillant à partir de son dernier état sauvegardé sans risquer l'effet domino. L'inconvénient de cette technique réside en la complexité du protocole de recouvrement arrière suite à une panne.

VI. Tolérance aux pannes dans les Cloud

Comme tout autres systèmes informatiques, le Cloud Computing a ses propres techniques de tolérances aux fautes qui se basent sur divers paramètres tel que : le débit, le temps de réponse, l'évolutivité, la performance, la disponibilité, la fiabilité, la facilité d'utilisation et la sécurité.

Le Cloud est exposé à de différents types de fautes (pannes). Diverses techniques de tolérance aux pannes peuvent être utilisées au niveau des taches ou du flux, comme le montre la figure II.8 suivante [34] :

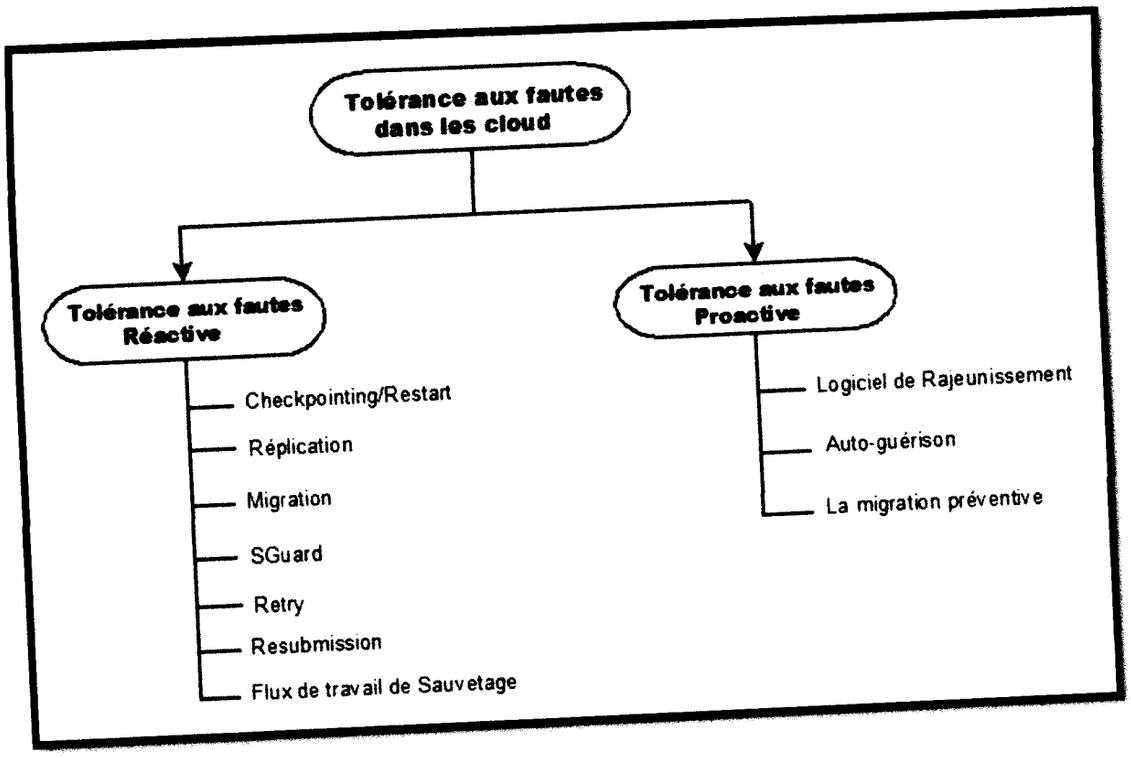


Figure II. 8 : Techniques de tolérance aux fautes dans les systèmes Cloud.

-Tolérance aux fautes réactive : les techniques de tolérance aux pannes réactives sont utilisées pour réduire l'impact des défaillances sur un système. Les techniques basées sur cette politique sont : sauvegarde/reprise, Réplication ...

-Tolérance aux fautes proactive : la tolérance aux pannes proactive prédit les fautes de manière proactive et remplace les composants suspects par d'autres composants fonctionnels.

VII. Conclusion

Dans ce chapitre, nous avons tout d'abord présenté les notions et le vocabulaire de la sureté de fonctionnement ainsi qu'un aperçu sur les différentes techniques de tolérance aux fautes dans les systèmes en générale, les systèmes distribués et le Cloud Coputing.

La tolérance aux fautes est donc l'un des moyens permettant d'assurer la sureté de fonctionnement d'un système.

Dans le prochain chapitre, nous allons présenter et expliquer l'objectif de notre projet puis nous présenterons notre contribution à ce dernier.

Chapitre III:

Objectif du projet

I. Objectif du projet

Les travaux effectués dans ce projet s'inscrivent dans le contexte du Cloud Computing utilisant les conteneurs. Nous nous sommes intéressés aux conteneurs qui sont la tendance actuelle dans le monde de la virtualisation. Les conteneurs sont de plus en plus utilisés dans les environnements Cloud, pour leurs légers impacts par rapport à la virtualisation système. Parmi les multiples avantages de l'utilisation des conteneurs on peut citer : leur légèreté, rapidité, isolement des applications, meilleures performances…

Plus précisément, nous nous intéresserons à la tolérance aux fautes dans les environnements de Cloud Computing utilisant des conteneurs. La tolérance aux fautes est l'une des préoccupations majeures de la communauté Cloud Computing.

D'après nos connaissances actuelles, la tolérance aux fautes dans les environnements de Cloud Computing utilisant des conteneurs n'a pas encore été étudiée. C'est pourquoi notre objectif est de concevoir, mettre en œuvre et tester une approche de tolérance aux fautes des applications Cloud hébergées dans ce type d'environnement. L'intérêt de tolérer les fautes dans les systèmes Cloud est d'assurer une bonne qualité et disponibilité des services, et par conséquent satisfaire les clients.

Parmi les types de pannes décrit en section III du chapitre II, nous nous sommes intéressés aux pannes franches (crash). Ce type de panne est les plus fréquents dans les environnements Cloud utilisant les conteneurs puisque les applications conteneurisées exécutées sur un serveur utilisent le même système d'exploitation ce qui rend le système d'exploitation vulnérable et il pourra planter (crasher) à tout moment.

Le mécanisme de tolérance aux fautes que nous avons choisi est basé sur la sauvegarde et reprise (checkpoint/restore). Ce mécanisme consiste à sauvegarder l'état des processus dans un support de stockage stable, pour pouvoir ensuite restaurer l'état du système à partir d'un point de reprise.

La difficulté majeure pour atteindre l'objectif de ce projet est le choix et la prise en main des outils nécessaires. Nous avons besoin d'un outil de conteneurisation pour la manipulation des conteneurs. Nous avons opté pour Docker car c'est la plateforme de conteneurisation la plus utilisé aujourd'hui dans les environnements cloud tel que : Amazon Web Services, Microsoft Azure et Google. Nous avons également besoin d'un outil qui nous permet d'effectuer des

sauvegarde/reprise des conteneurs afin de réaliser notre approche basée sur cette opération. Nous avons choisi CRIU puisque cet outil permet de faire le checkpoint et le restore des processus en cours d'exécution.

Dans le chapitre suivant, nous allons décrire avec plus de détails les outils utilisés et présenter notre approche ainsi que sa mise en œuvre et les expériences réalisées.

Chapitre IV:

Mécanisme de tolérance aux fautes des conteneurs proposé

I. Introduction

Ce dernier chapitre présente notre contribution à la tolérance aux fautes des conteneurs. En premier nous allons présenter notre approche de tolérance aux *fautes puis nous discuterons* les outils utilisés pour la mise en œuvre ensuite nous présenterons l'environnement de notre travail et nous terminerons avec les résultats de nos expériences.

II. Approche proposée

Avant de proposer une approche de tolérance aux pannes, il faut choisir un type de panne à tolérer. Nous avons choisi de simuler une panne franche, un crash au niveau des conteneurs. Notre approche de tolérance aux fautes est basée sur le mécanisme du checkpoint, plus précisément elle s'appuie sur trois phases : sauvegarde périodique, détection d'erreur puis le traitement d'erreur avec la méthode de la reprise (voir figure IV.1).

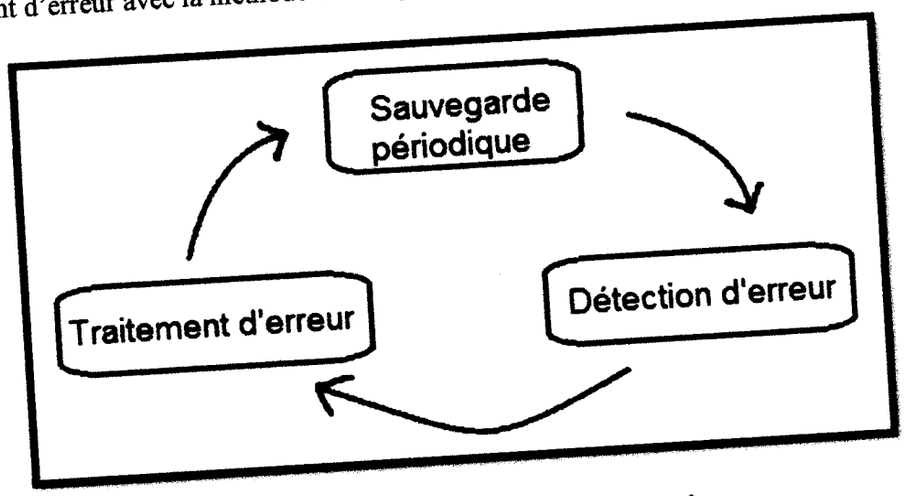


Figure IV. 1 : Les phases de notre approche.

Phase 1 : la sauvegarde périodique, consiste à faire un checkpoint pour sauvegarder l'état d'un conteneur sur une mémoire stable pour ne pas redémarrer une application du conteneur depuis le début en cas de panne.

Phase 2 : détection d'erreur, consiste à identifier les conteneurs défaillants.

Phase 3 : traitement d'erreur, consiste à redémarrer les conteneurs depuis leur dernier point de sauvegarde.

Voici une architecture détaillée de notre approche (voir Figure IV.2) :

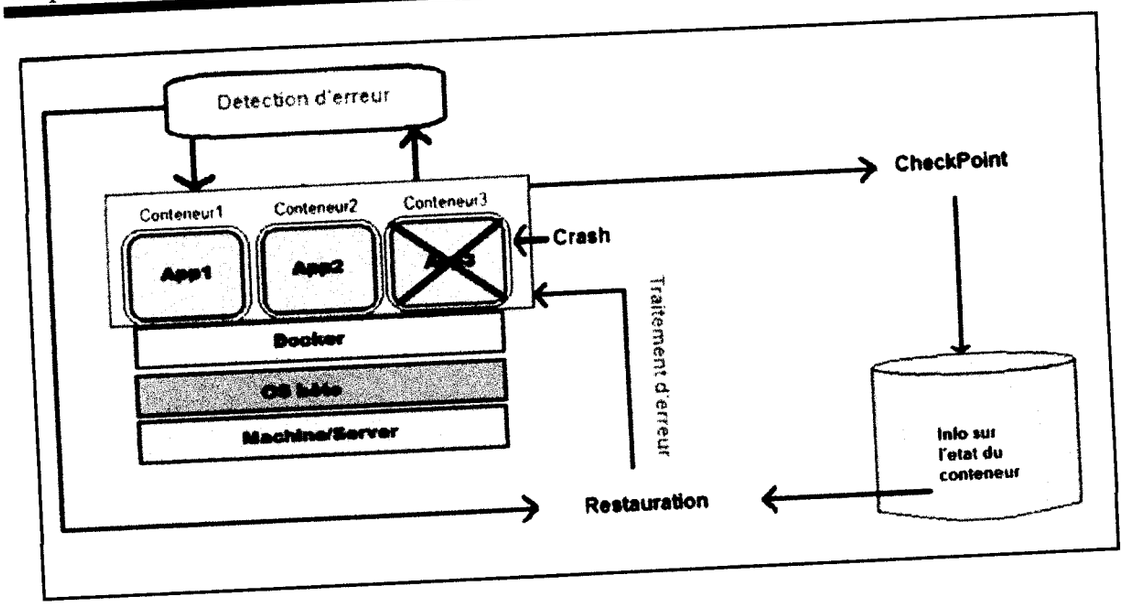
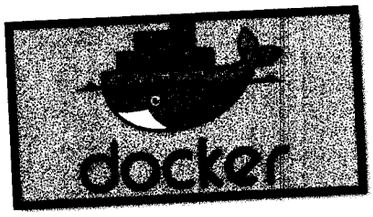


Figure IV. 2 : Architecture globale de l'approche proposée.

III. Outils utilisés

Pour mettre en œuvre notre approche nous avons utilisés les outils Docker et CRIU qui sont décrit ci-dessous.

III.1 Docker :



Docker est une plateforme open source, qui fournit un moyen d'exécuter des applications en toute sécurité isolées au niveau des conteneurs.

Docker utilise une architecture client-serveur. Il se compose des éléments de bases suivants :

- **Docker Daemon :** Le démon Docker est exécuté sur un ordinateur hôte. L'utilisateur n'a pas d'interaction directe avec le démon, mais plutôt en passant par le client Docker. Le daemon Docker fait tout le travail : la construction, l'exécution et la distribution des conteneurs Docker.

- **Client Docker :** Le client Docker est le CLI (command line interface) l'interface utilisateur principale de Docker. Il accepte les commandes de l'utilisateur et communique avec le daemon Docker. Les clients Docker interagissent avec serveur Docker ou daemon (démon).

- **Images Docker :** Les images sont des template qui contiennent notre environnement, notre système d'exploitation et notre application. Les images sont les blocs de construction du monde

Docker. Les images sont utilisées pour créer les conteneurs. Par exemple, une image peut contenir un système d' exploitation Ubuntu avec Apache et l'application web installée de l'utilisateur. Les images sont portables et peuvent être partagées, stockées et mises à jour.

- **Registres Docker :** Les registres Docker servent à stocker les images que l'utilisateur construit. Il existe deux types de registres : publiques et privés dont l'utilisateur charge ou télécharge des images. Le registre Docker public est fourni avec le Docker Hub, il contient les images que l'utilisateur crée lui-même ainsi les images que d'autres personnes ont construites et partagées. Exemple : NGINX, Ubuntu…

- **Conteneurs Docker :** Les conteneurs Docker sont similaires à un répertoire. Chaque conteneur est créé à partir d'une image Docker. Il contient tous ce qui est nécessaire pour exécuter une ou plusieurs applications (processus). Chaque conteneur est une plateforme d'application isolée et sécurisée.

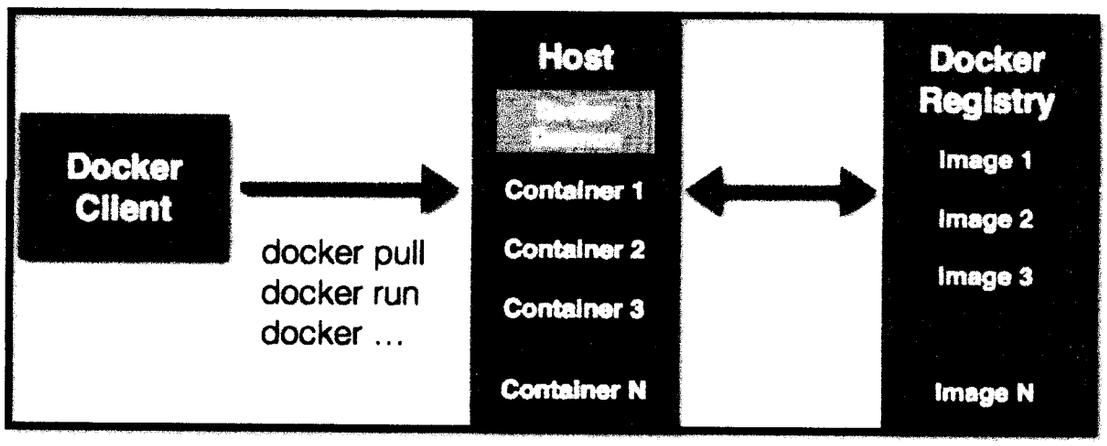


Figure IV. 3 : L'architecture de Docker.

III.2 CRIU (Checkpoint/Restore In Userspace) : est un outil logiciel pour le système d'exploitation Linux. Grâce à cet outil, on peut geler une application en cours d'exécution (ou juste une partie d'elle) et faire un checkpoint et le sauvegarder sous forme de collection de fichiers, ensuite on peut utiliser ces fichiers pour restaurer et par suite exécuter l'application à partir du point où elle a été gelée. La particularité du projet CRIU est qu'il est principalement mis en œuvre dans l'espace utilisateur.

Il y a deux façons pour faire le checkpoint et le restore d'un conteneur Docker :

a.Externe C/R : en utilisant ''criu'' directement sur la ligne de commande comme il est généralement fait pour tout arbre de processus. Les commandes utilisées ici sont :

criu dump, criu restore

Cette approche est externe par rapport au démon Docker. Après le checkpoint, le démon Docker pense que le conteneur a quitté. Après restauration, le démon Docker ne sait pas que le conteneur est à nouveau en cours d'exécution. Par conséquent, les commandes telles que : docker ps qui affiche la liste des conteneurs en cours d'execution, stop, kill et logs qui rapporte le journal d'un conteneur, ne fonctionneront pas correctement.

b.Native C/R : en utilisant les commandes : docker checkpoint et docker restore.

Cette approche est appelée native parce que le démon Docker est impliqué dans les deux opérations : checkpoint et restore. Par conséquent, sa notion de l'état du conteneur sera connue. Toutes les commandes telles que : docker ps qui affiche la liste des conteneurs en cours d'execution, stop, kill et logs qui rapporte le journal d'un conteneur, vont fonctionnées. Ceci est évidemment la méthode préférée de checkpointing et la restauration des conteneurs Docker.

IV. Implémentation

Pour pouvoir mettre en œuvre notre approche, nous avons implémentés 3 scripts Shell, le premier est responsable de la sauvegarde périodique, le second fait la détection d'erreurs et la restauration des conteneurs et le troisième est responsable d'injection d'erreur. Ci-dessous une description détaillée de chaque script.

Script 1 : CheckPoint.sh

Ce script consiste à faire un checkpoint de tous les conteneurs en cours d'exécution. Le checkpoint ce fait périodiquement, selon une fréquence que l'utilisateur doit préciser avant de lancer le script.

Le statut d'un conteneur après la sauvegarde de son état est : checkpointed.

Figure IV. 4 Statut d'un conteneur après le checkpoint.

Script 2 :Restore.sh

Ce script met tous les conteneurs dont leur statut est ''checkpointed'' dans une liste, pour pouvoir ensuite commencer leur restaurer un par un. Il contient une fréquence de restore en entrée.

Script 3 : injection_erreur.sh

Ce script a pour but de tuer les conteneurs en cours d'exécution aléatoirement en utilisant la commande de Docker : kill.

Ce script prend trois paramètres.Le premier paramètre '' frequence_MAJ '' représente la fréquence de mise à jour pour la mise à jour de la liste des conteneurs en cours d'exécution. Le deuxième paramètre ''multiple_container'' est un entier utilisé comme facteur de sélection des conteneurs à tuer. Le troisième paramètre '' frequence_test ''.

Au premier lieu le script rassemble tous les identifiants des conteneurs en cours d'exécution dans une liste. Cette liste est mise à jour périodiquement selon le premier paramètre.

Ensuite, en utilisant le paramètre '' multiple_container '', le script va calculer ses multiples. À chaque fois où il trouve un multiple de cette fréquence il exécute la commande suivante : Docker kill id_conteneur.

V. Configuration et Experiences

Pour débuter notre travail il nous avons utilisé la version Wily 15.10 de Ubuntu. Ensuite concernant l'installation de de Docker et CRIU, après plusieurs tests et expériences nous nous sommes confronter aux limites de CRIU (Externe C/R) qui nous empêchaient d'avancer dans notre objectif.

Nous avons utilisé le native C/R qui n'est pas implémenté dans version officielle mais dans une version expérimentale qui est proposée sur la plateforme GitHub [36].
Notre prochaine étape était la manipulation et la création des conteneurs Docker. Nous avons pu concevoir et manipuler un nombre considérable de conteneurs, où nous avons fait tourner des programmes et applications pour tester les différentes commandes de Docker.

Pour les tests, nous avons mesuré la durée de sauvegarde et de reprise de 10 conteneurs selon deux méthodes en séquentiel et en parallèle.

Ce premier diagramme (voir Figure IV.5) donne la durée de l'opération de la sauvegarde séquentielle de l'état des conteneurs :

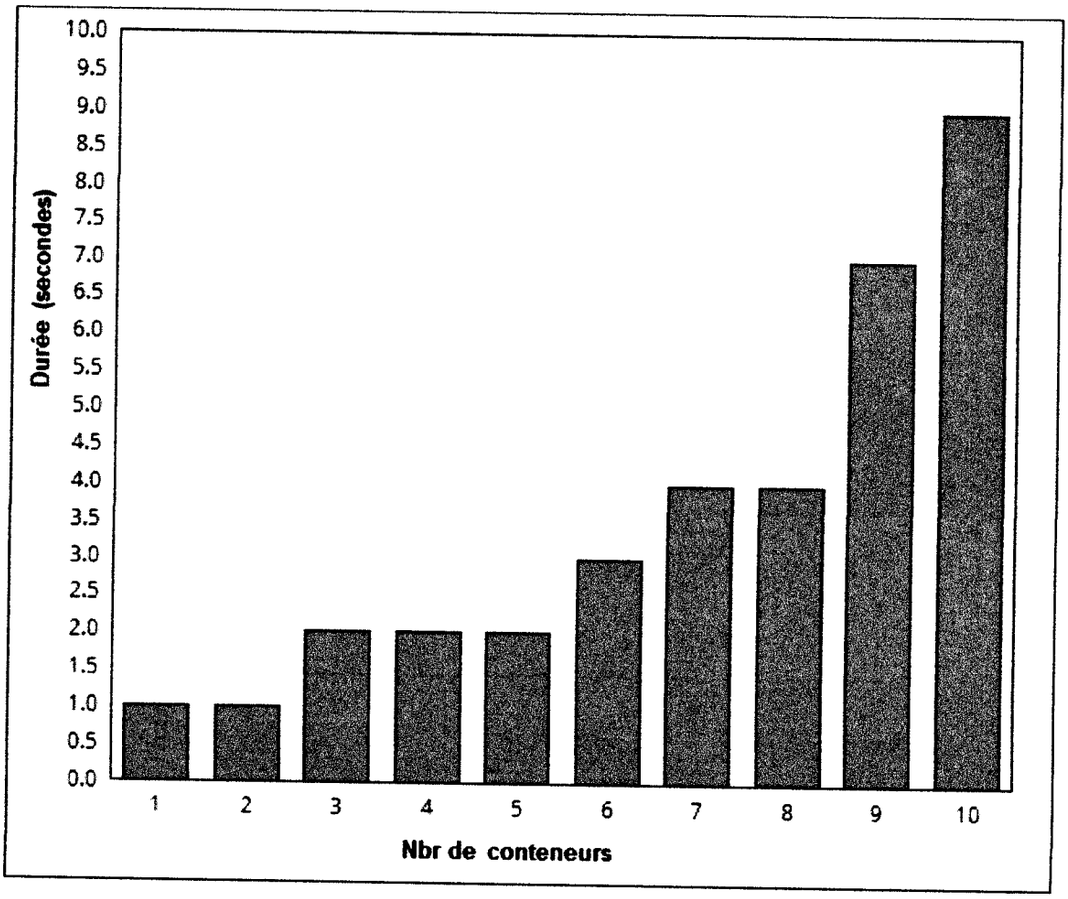


Figure IV. 5 : Diagramme de Sauvegarde séquentielle.

D'après les résultats obtenus, nous remarquons que le temps nécessaire pour la sauvegarde (checkpoint) séquentielle des conteneurs varie selon le nombre de conteneurs. La sauvegarde de 5 conteneurs dure 2 secondes ce qui donne 400 ms/conteneur. Ensuite pour sauvegarder l'état de 6 jusqu'à 8 conteneurs ça prendra entre 4 et 5 seconde ce qui donne 625 ms/conteneur. Par contre, pour la sauvegarde de 10 conteneurs nous remarquons qu'il faudra environ 1s/conteneur.

Le deuxième diagramme (voir Figure IV.6) reflète la variation de la durée de sauvegarde parallèle selon le nombre de conteneurs :

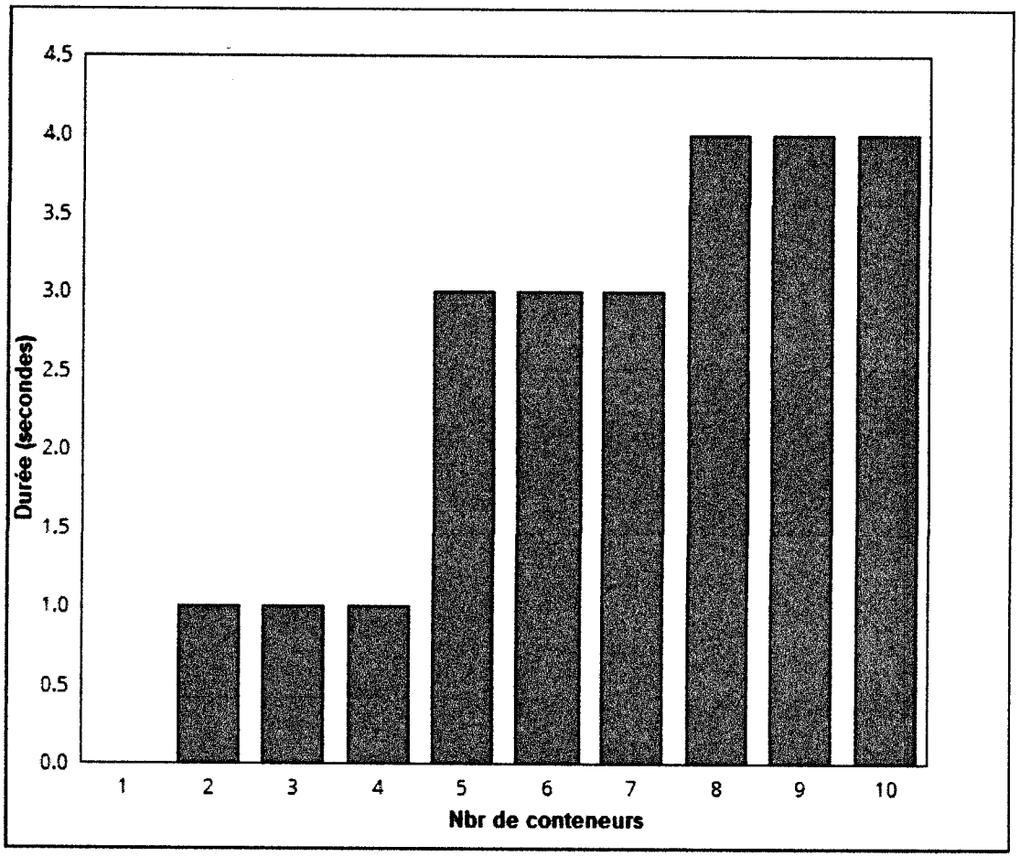


Figure IV. 6 : Diagramme de Sauvegarde parallèle.

En comparant les résultats des deux graphes des Figures IV.5 et IV.6, nous remarquons que la durée de sauvegarde parallèle est réduite par rapport à la sauvegarde séquentielle. La durée de la sauvegarde de 10 conteneurs est passée de 9 secondes à 4 secondes. Nous déduisons que la sauvegarde (checkpoint) parallèle est bien avantageuse en termes de réduction du temps.

Le diagramme suivant (voir Figure IV.7) présente la variation de la durée de l'opération séquentielle de la reprise :

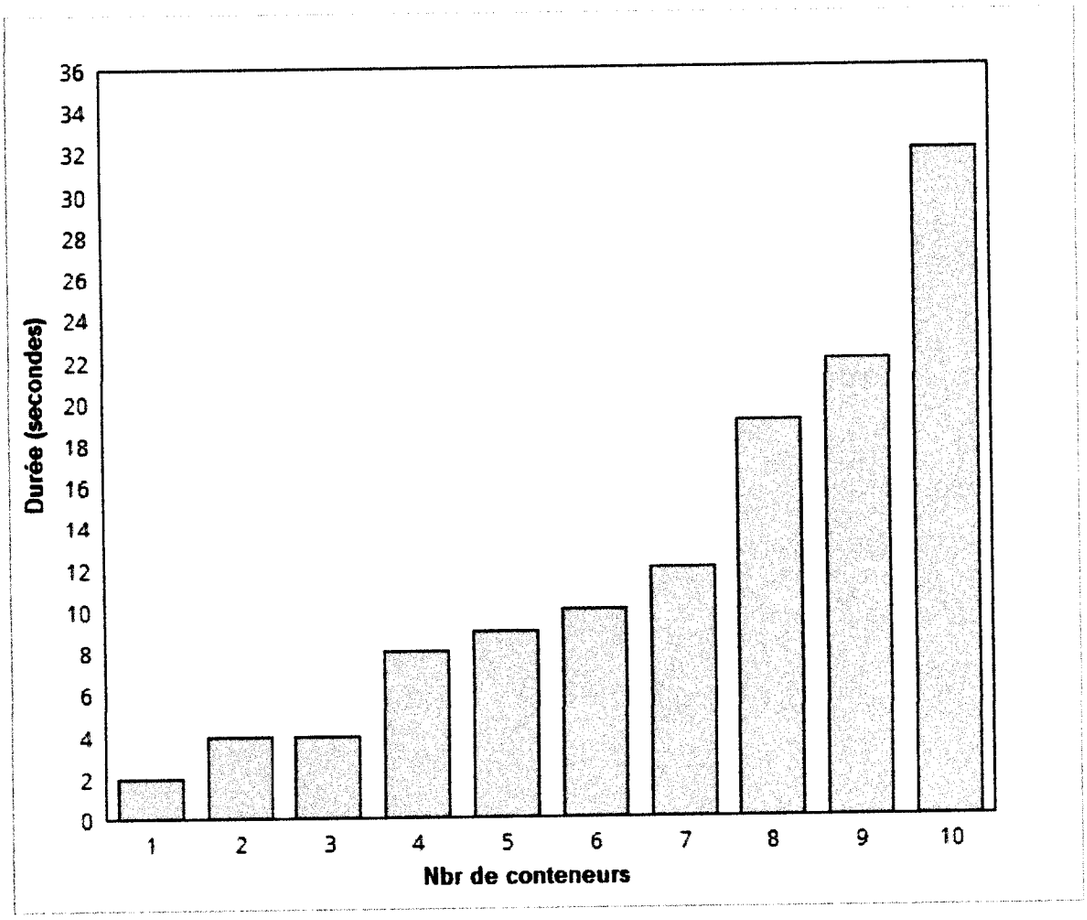
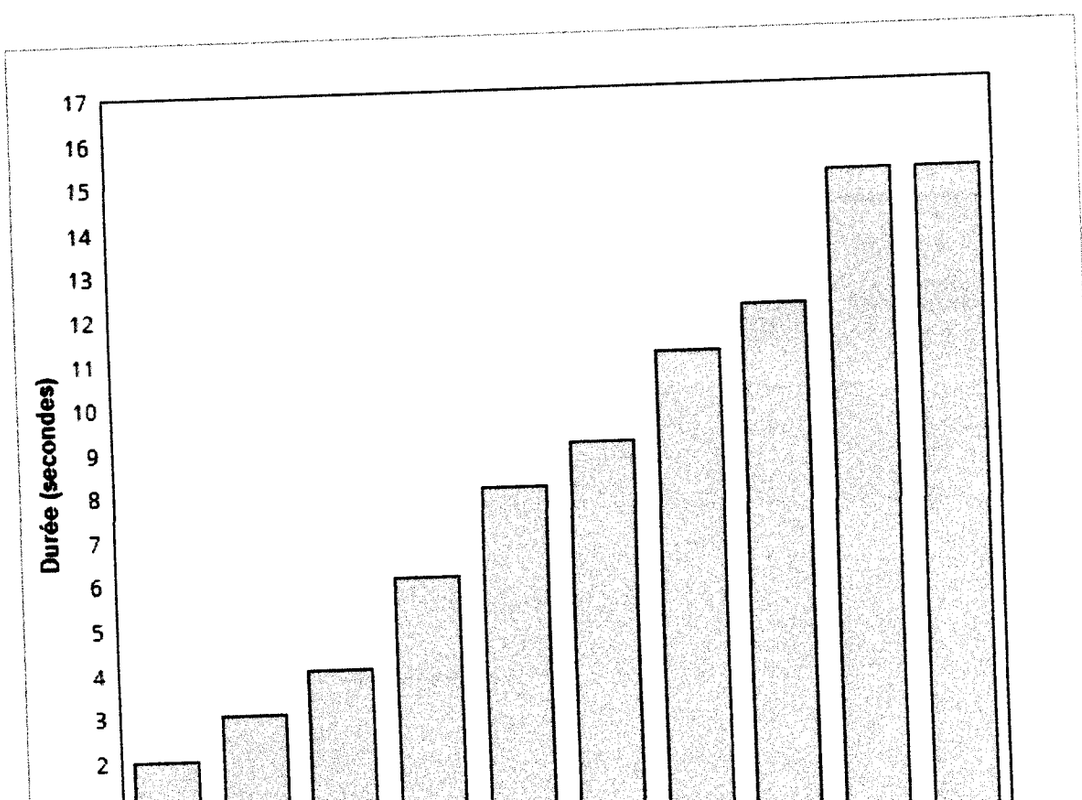


Figure IV. 7 : Diagramme de Reprise séquentielle.

Ce diagramme (voir Figure IV.8) présente la variation de la durée de reprise (restauration) parallèle des 10 conteneurs :



Nous avons également décidé de mesurer la durée du redémarrage (restart) d'un conteneur (voir Figure IV.9) puis de comparer les résultats avec celles de la reprise des conteneurs. Le redémarrage d'un conteneur consiste à relancer dès le début par exemple une application qui tourne sur ce conteneur.

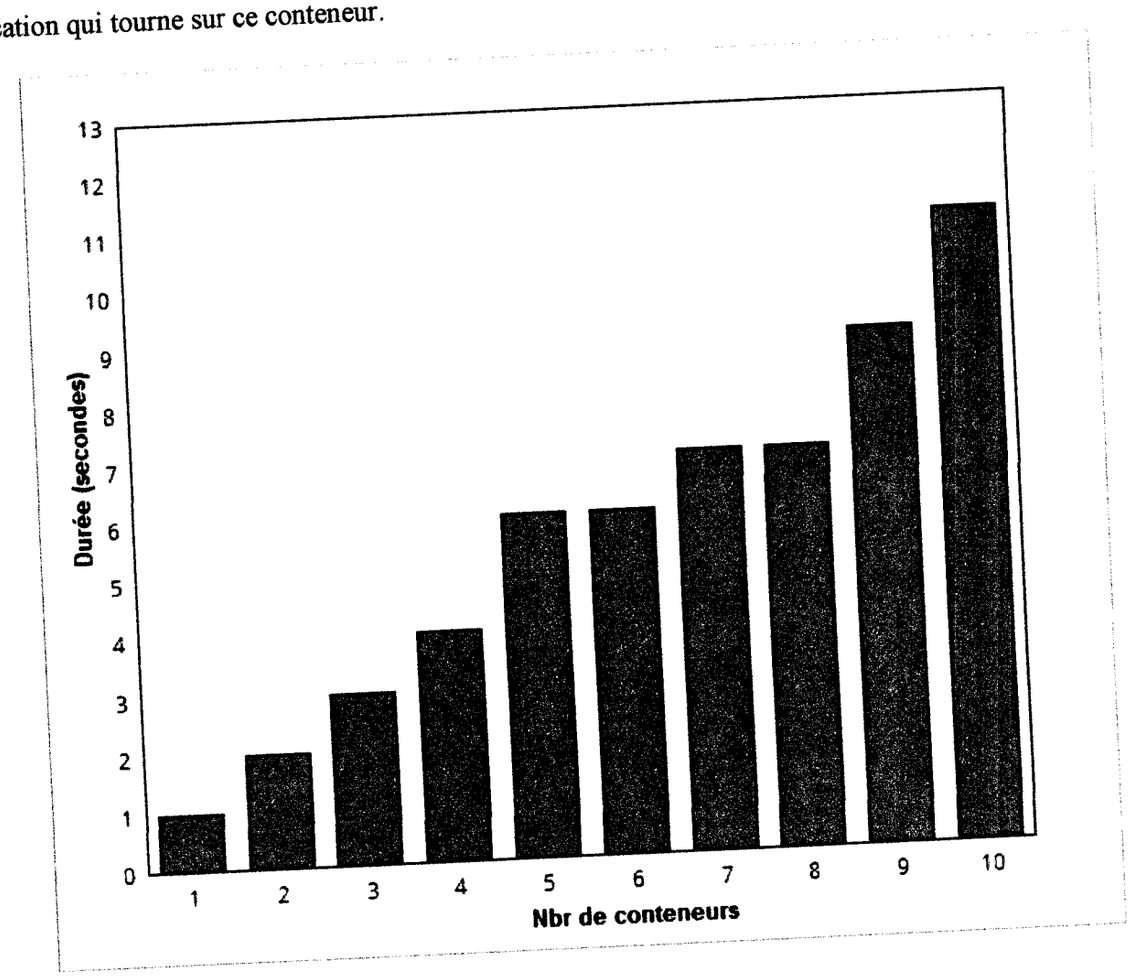


Figure IV. 9 : Diagramme du Redémarrage.

Les résultats obtenus dans la mesure du temps de redémarrage (restart) des conteneurs, nous montre que la durée de cette opération ressemble à celle de la sauvegarde (checkpoint).

En comparant les résultats de l'opération du redémarrage et de la reprise après la sauvegarde on constate qu'en moyenne la durée de redémarrage d'un conteneur est 1 seconde contrairement à la reprise qui dure 2 secondes. Nous pouvons justifier cette différence en précisant que l'opération de la restauration par la complexité de l'opération de reprise (restauration) par rapport à celles de sauvegarde et redémarrage. L'opération de reprise cherche d'abord les informations sur l'état du conteneur (le point de reprise) puis restaure le conteneur depuis ces

informations. La durée de la restauration reste négligeable par rapport au temps de calcul qu'on peut perdre si on n'utilise pas les sauvegardes.

Depuis les résultats précédents de la reprise et de redémarrage de conteneurs nous pouvons donner l'exemple suivant qui montrer que notre approche de sauvegarde et reprise est efficace en termes de réduction de temps perdu (voir figure VI.10 et VI.11) :

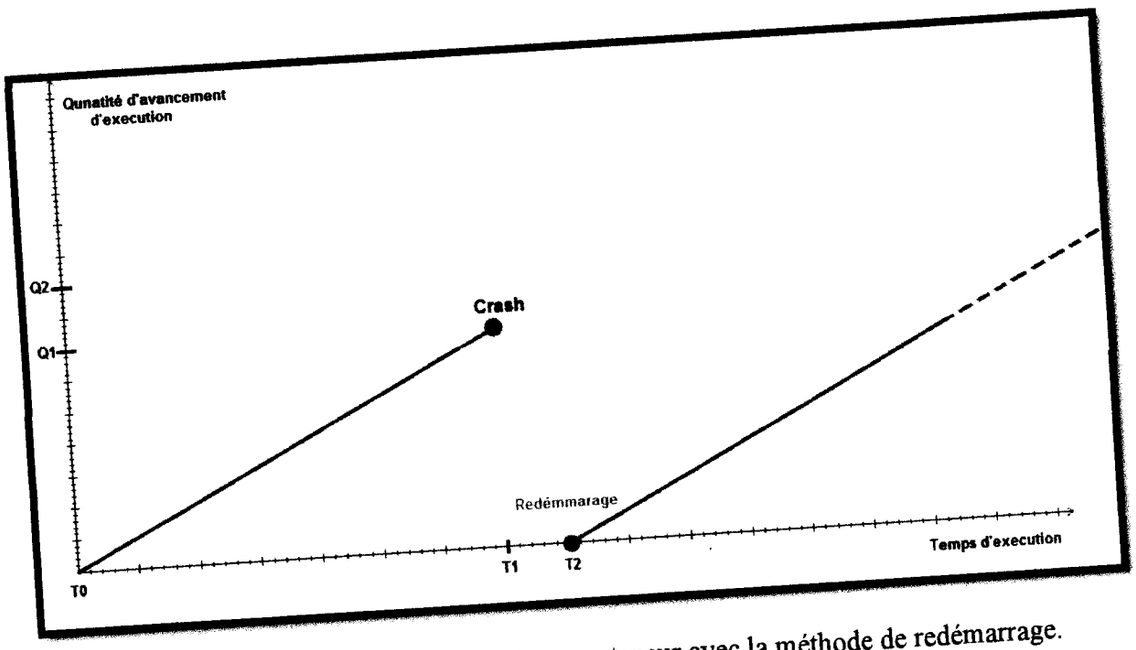


Figure IV. 10 : Exemple d'exécution d'un conteneur avec la méthode de redémarrage.

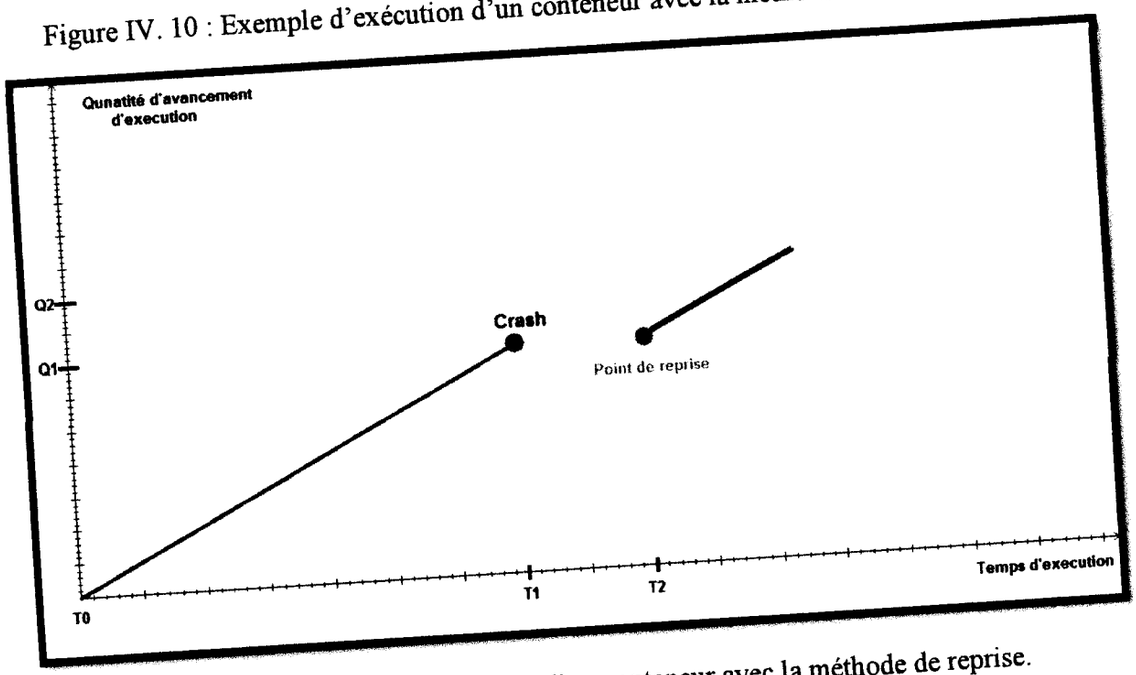


Figure IV. 11 : Exemple d'exécution d'un conteneur avec la méthode de reprise.

On suppose qu'un conteneur démarre à l'instant T0 et à l'instant T1 le conteneur crach avec une quantité d'exécution de Q1. On sait que l'opération de redémarrage dure en moyenne 1seconde et que l'opération de reprise dure en moyenne 2 secondes. Si on suppose que ce conteneur fini son exécution lorsque il atteint la quantité Q2, on trouve que la reprise est plus favorable bien que le temps d'un reprise est supérieur à celui d'un redémarrage.

VI. Conclusion

Dans ce chapitre nous avons présenté notre approche de tolérance aux fautes qui se base sur le mécanisme de sauvegarde et reprise (Checkpoint/Restore) tout en décrivant les outils utilisés. Pour évaluer notre approche de tolérance aux fautes nous avons effectué des tests sur la durée que va prendre cette approche. Les résultats que nous avons obtenus en comparant entre la méthode de sauvegarde/reprise avec un simple redémarrage montrent que l'utilisation de notre approche réduit le temps perdu à cause des pannes.

CONCLUSION

Le Cloud Computing est un modèle d'informatique dématérialisée permettant l'accès via un réseau internet, à un ensemble partagé de ressources configurables : réseaux, serveurs, stockage, applications,...

La fiabilité et la robustesse est l'une des critères importants pour l'utilisation efficace des Clouds. Le taux de pannes dans les systèmes Clouds augmente selon leurs tailles, pour cela un mécanisme de tolérance aux pannes est important.

Dans ce mémoire nous avons présenté une approche de tolérance aux fautes pour un environnement Cloud utilisant les conteneurs. Notre approche se base sur le mécanisme de sauvegarde périodique et reprise (Checkpoint/Restore), elle est constituée de trois phases : sauvegarde périodique, détection d'erreur et traitement d'erreur. Le but de cette approche est de minimiser le temps de calcul perdu suite à une panne.

Nous avons mis en œuvre notre approche en utilisant une version expérimentale de Docker. Nous avons ensuite effectué une série de tests et d'expérience tout en créant des scénarios de panne au niveau des conteneurs Docker. Nous avons évalué le temps de la sauvegarde et celui de la reprise après une panne. Les résultats obtenus ont montré que notre approche de sauvegarde/reprise à une bonne performance, elle minimise le temps perdu par le crash.

Le Checkpoint est limité par le surcout d'exécution qui représente le délai de l'interruption de l'exécution des conteneurs afin d'effectuer la sauvegarde, et le temps de reprise qui est le temps nécessaire pour le redémarrage des conteneurs depuis leur dernier point de reprise. Pour implémenter notre approche, nous avons rencontré des limites de la version expérimentale de Docker puisque ça nous a empêché de développer encore plus notre approche.

Pour poursuivre dans le concept de ce travail il sera nécessaire de proposer des pistes de recherche tel que :

La validation de notre approche sur d'autre plateforme de Cloud Computing par exemple : Cloud28+, Cloud Foundry «Diego». Des expériences complémentaires pour valider notre approche telle que l'évaluation de sa consommation en ressources. Enfin notre perspective principale consiste à étendre notre approche et la basé sur un checkpoint adaptatif qui optimise le temps d'exécution.

REFERENCES BIBLIOGRAPHIQUES

[1] George Thomas. Cloud computing. Disponible sur : http://www.futura-sciences.com/magazines/high-tech/infos/dico/d/informatique-cloud-computing-11573/

[2] CISCO: http://www.cisco.com/

[3] NIST: http://www.nist.org/

[4] ANNA M. Bien comprendre le concept du Cloud Computing. Disponible sur : http://www.yeswecloud.fr/cloud/bien-comprendre-le-concept-du-cloud-computing-642.html

[5] Le Cloud Computing. Disponible sur : http://www.systancia.com/fr/modeles-du-cloud-computing

[6] Houssem MEDHIOUB : Architectures et mécanismes de fédération dans les environnements cloud computing et cloud networking. Thèse de doctorat conjoint TELECOM SUDPARIS et l'université PIERRE ET MARIE CURIE, 19 Oct 2015.

[7] Qu'est-ce que le Cloud Computing. Disponible sur : https://www.numergy.com/centre-de-ressources/article/quest-ce-que-le-cloud-computing

[8] Judith HURWITZ, Robin BLOOR, Marcia KAUFMAN et Fern HALPER . Comparing Public, Private and Hybrid Cloud Computing Options. Disponible sur: http://www.dummies.com/how-to/content/comparing-public-private-and-hybrid-cloud-computin.html

[9] http://reseau-informatique.prestataires.com/conseils/virtualisation

[10] Djawida DIB : Migration dynamique d'applications réparties virtualisées dans les fédérations d'infrastructures distribuées. Rapport de Stage de Master Recherche, Université de Rennes 1, Juin 2010.

[11] La virtualisation, qu'est-ce que c'est. Disponible sur : http://www.tuto-it.fr/virtualisation.php

[12] Jean FEUR. Qu'est-ce qu'un hyperviseur ? .Disponible sur : http://blog.compufirst.com/serveur/quest-ce-qu-un-hyperviseur

[13] Flavien QUESNEL : Vers une gestion coopérative des infrastructures virtualisées à large échelle : le cas de l'ordonnancement. Thèse de doctorat, École Nationale Supérieure des Mines de Nantes, l'Université de Nantes Angers Le Mans, 20 février 2013.

[14] BELFOUZ Annas, VANDAMME Yohan : Projet n°10 : Hyperviseur vs Docker, le choc des virtualisations. Université d'AVIGNON.

[15] Michael DACONTA. Containers Add New Efficiency To Cloud Computing. Disponible sur :http://www.informationweek.com/cloud/containers-add-new-efficiency-to-cloud-computing/d/d-id/1112037

[16] Mike BAUKES. Docker vs LXC. Disponible sur : https://www.upguard.com/articles/docker-vs-lxc

[17] Tobby BANERJEE. Understanding the key differences between LXC and Docker. Disponible sur: https://www.flockport.com/lxc-vs-docker/

[18] Steven J Vaughan-Nichols. Why Containers Instead of Hypervisors. Disponible sur : http://blog.smartbear.com/web-monitoring/why-containers-instead-of-hypervisors/

[19] Alain FERNANDEZ. Qu'est-ce que la virtualisation. Disponible sur : http://www.piloter.org/techno/support/virtualisation.htm

[20] WebSphere Virtual Enterprise : La virtualisation de l'infrastructure applicative, une solution pour étendre la virtualisation des systèmes d'information. IBM,Avril 2008

[21] Jean ARLAT, Yves CROUZET, Yves DESWARTE, Jean-Charles FABRE, Jean-Claude LAPRIE, David POWELL : ''Tolérance aux fautes''. Encyclopédie de l'informatique et des systèmes d'information, Vuibert, Paris, France, 2006.

[22] Samir JAFAR : ''Programmation des systèmes parallèles distribués : tolérance aux pannes, résilience et adaptabilité ''.Thèse pour obtenir le grade de docteur de L'INPG.INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, 30 Juin 2006.

[23] Benjamin LUSSIER : ''Tolérance aux fautes dans les systèmes autonomes''. Thèse pour l'obtention le titre de docteur de l'Institut National Polytechnique de Toulouse, 24 avril 2007.

[24] Khaled BARBARIA : ''Architectures intergicielles pour la tolérance aux fautes et le consensus''. Thèse pour l'obtention du grade de Docteur de l'' Ecole Nationale Supérieure des

Télécommunications. Ecole doctorale d'Informatique, Télécommunications et Electronique de

Paris, janvier 2008.

algorithmes répartis Tolérance aux fautes''.

[25] Philippe QUEINNEC : '' Systèmes

Département Informatique et Mathématiques Appliquées

ENSEEIHT, 19 octobre 2015.

[26] Thomas ROBERT : ''Tolérance aux Fautes des Systèmes Informatiques''. Télécom ParisTech.

[27] Phuong-Quynh DUONG, Elizabeth PEREZ CORTES et Christine COLLET: '' La tolérance aux fautes adaptable pour les systèmes à composants : application à un gestionnaire de données''. Laboratoire LSR/IMAG.

[28] Sacha KRAKOWIAK: ''Tolérance aux fautes – 1 Introduction, techniques de base''. Université Joseph Fourier Projet Sardes (INRIA et IMAG-LSR), 2003-2004.

[29] Mohamed Taha BENNANI : '' Tolérance aux Fautes dans les Systèmes

Répartis à base d'Intergiciels Réflexifs Standards''. Thèse pour l'obtention le titre de docteur de l'Institut National des Sciences Appliquées, Toulouse, 20 juin 2005.

[30] Gérard LE LANN, Pascale MINET, David POWELL : '' Tolérance au fautes et systèmes répartis : Concepts et mécanismes''. Rapport de recherche, INRIA, novembre 1993.

[31] Christian DELBÉ : ''Tolérance aux pannes pour objets actifs asynchrones : protocole, modèle et expérimentations''. Thèse de doctorat, Université de Nice-Sophia Antipolis Faculté des sciences, 24 Janvier 2007.

[32] Fabrice ROSSI : '' Systèmes Répartis ''. Université Paris-IX Dauphine.

[36] Version expérimentale de Docker avec Checkpoint / *restauration utilisant CRIU*, disponible sur : https://github.com/boucher/docker/releases

[37] Abdelaziz Salem ALKHALAF. Cloud Computing. Disponible sur : http://www.paaet.edu.kw/mysite/Default.aspx?tabid=7807&language=en-US

[38] Frédéric DESPREZ : '' Cloud Computing ''. LIP ENS Lyon/INRIA Grenoble Rhône-Alpes EPI GRAAL.2010.

[39] Vangie BEAL. Operating system virtualization. Disponible sur : http://www.webopedia.com/TERM/O/operating_system_virtualization.html

LISTE DES FIGURES