

République Algérienne Démocratique et Populaire
Ministère de l'enseignement supérieur et de la recherche scientifique

UNIVERSITE Dr. TAHAR MOULAY SAIDA

FACULTE : TECHNOLOGIE

DEPARTEMENT : INFORMATIQUE



MEMOIRE DE MASTER

OPTION :

Réseaux Informatique et Systèmes Répartis

Thème

Une stratégie de réplication de données pour gérer
la tolérance aux pannes et l'équilibrage de charge
dans le cloud computing

Présenté par :

Deghbouch Hicham
Nedjadi Lahcene

Encadré par :

Mr LIMAM Saïd

Promotion : juin 2018

REMERCIEMENTS

Tout d'abord, nous remercions Allah le tout-puissant qui nous a donné le courage, la force et la volonté pour mener ce travail.

Nous tenons à remercier notre encadreur Mr. LIMAM SAID, pour l'orientation, la confiance, la patience qui ont constitué un apport considérable sans lequel ce travail n'aurait pas pu être mené au bon port. Qu'il trouve dans ce travail un hommage vivant à sa haute personnalité.

Un grand merci pour nos familles, surtout nos parents qui nous ont épaulés, soutenus et suivis tout au long de ce projet.

Enfin on remercie toutes les personnes qui ont participé De près ou de loin à la réalisation de ce travail.

DÉDICACE

À mes chers parents, mes frères et ma sœur

Deghbouch Hicham

DÉDICACE

À mes chers parents, mes frères et ma sœur,et mes amis.

Nedjadi Lahcene

TABLE DES MATIÈRES

Introduction Générale	1
1 Cloud Computing	3
1.1 Introduction	5
1.2 Description du cloud computing	5
1.2.1 Historique	7
1.3 Caractéristiques	8
1.4 Virtualisation	9
1.5 Les différents services du Cloud Computing	9
1.5.1 Le logiciel en tant que service (SaaS) :	9
1.5.2 Plate forme en tant que service (PaaS) :	10
1.5.3 Infrastructure en tant que service (IaaS) :	11
1.5.4 Avantages et Inconvénients des services	11
1.6 Types de Cloud Computing	12
1.6.1 Cloud privé	12
1.6.2 Cloud public	12
1.6.3 Cloud hybride	12
1.6.4 Le Cloud communautaire	13
1.7 Architecture du cloud computing	14
1.8 Avantages du Cloud Computing	15
1.9 Inconvénients du Cloud Computing	15
1.10 Les grands défis relatifs à l'adoption du Cloud Computing	16
1.10.1 Sécurité	16

TABLE DES MATIÈRES

1.10.2	Tolérance aux fautes et disponibilité	16
1.10.3	L'équilibrage de charge	17
1.11	Conclusion	17
2	Tolérance aux Panne	18
2.1	Introduction	20
2.2	Notion de sûreté de fonctionnement	20
2.2.1	Attributs de la sûreté de fonctionnement	21
2.2.2	Entravs	22
2.2.3	Moyens d'assurer la sûreté de fonctionnement	24
2.2.4	Classification des pannes	25
2.2.5	Détection des pannes	27
2.3	Méthodes de tolérance aux pannes	29
2.4	Techniques de tolérance aux pannes dans les systèmes répartis . .	30
2.4.1	Tolérance aux pannes par sauvegarde (checkpointing) . . .	31
2.4.2	Tolérance aux pannes par journalisation	34
2.4.3	Comparaison des protocoles	37
2.4.4	Migration	38
2.4.5	Tolérance aux pannes par duplication (Approche masquante)	39
2.5	Protocole de réplication	43
2.5.1	Protocole de réplication passive	43
2.5.2	Protocole de réplication active	44
2.5.3	Protocole de réplication semi-active	44
2.6	La tolérance aux pannes dans les cloud	45
2.7	L'équilibrage de charge	46
2.7.1	Problème de l'équilibrage de charge	46
2.7.2	Le système d'équilibrage de charge	47
2.7.3	Les algorithmes d'équilibrage de charge dans le Cloud Com- puting	49
2.8	Conclusion	50
3	Notre Contribution	52
3.1	Introduction	53
3.2	Objectif du travail	53
3.3	Description de l'approche proposée	54
3.3.1	Topologie de cloud	54

TABLE DES MATIÈRES

3.3.2	Modèle proposé	55
3.4	Décision de réplication (quand répliquer)	55
3.5	Degré de réplication (combien de répliques)	59
3.6	Placement des répliques	60
3.7	L'équilibrage de charge	60
3.8	Détection des pannes	61
3.9	Conclusion	62
4	Implémentation	63
4.1	Introduction	64
4.2	Langage et environnement de développement	64
4.3	Langage de programmation Java	64
4.4	Environnements de développement	65
4.4.1	Eclipse :	65
4.4.2	CloudSim :	65
4.5	Interface principale	69
4.5.1	Configuration des paramètres de simulation	69
4.6	Lancement de la simulation	72
4.7	Résultats expérimentaux	72
4.7.1	Expérience 1 : Temps de réponse moyen	73
4.7.2	Expérience 2 : nombre de répliques créés	73
4.7.3	Expérience 3 : Impact de la fréquence des pannes sur le comportement de notre approche	74
4.7.4	Expérience 4 : l'équilibrage de charge	76
4.7.5	Expérience 5 : Dépenses monétaires du fournisseur (Cost)	77
4.8	Conclusion	78
	Conclusion Générale et Perspectives	79

TABLE DES FIGURES

1.1	L'environnement de Cloud computing	6
1.2	Les trois modèles de service du Cloud Computing et la répartition des responsabilités entre le fournisseur et le client.	10
1.3	Avantages et Inconvénients des services de cloud computing[11]. .	11
1.4	Les modèles de déploiement dans le Cloud computing	13
1.5	Architecture globale et les composants de base du cloud computing [13].	14
2.1	L'arbre de la sûreté de fonctionnement	21
2.2	Modes de défaillance.	24
2.3	La relation entre défaillance, erreur et faute	24
2.4	Les classes des pannes	27
2.5	Messages "Ping/Pong"	29
2.6	messages de vie (heartbeats)	29
2.7	Comparaison des différentes méthodes de tolérance aux fautes par reprise	38
2.8	Migration des processus	39
2.9	protocole de réplication passive	44
2.10	Protocole de réplication active	44
2.11	Protocole de réplication semi-active	45
2.12	Composants d'un système d'équilibrage de charge	48
3.1	Topologie de cloud	55

TABLE DES FIGURES

4.1	Architecture générale de CloudSim	66
4.2	Les classes de CloudSim	67
4.3	Impact du nombre de Cloudlets sur le temps de réponse moyen . .	73
4.4	Impact du nombre de cloudlet sur le nombre répliques.	74
4.5	Impact de nombre des pannes sur le temps de réponse.	75
4.6	L'impact des pannes sur le Nombre de cloudlets exécutées avec succès	76
4.7	L'équilibrage de charge	76
4.8	Dépenses monétaires du fournisseur	77

INTRODUCTION GÉNÉRALE

La demande croissante pour de nouveaux services informatiques plus économiques a permis l'émergence d'une nouvelle architecture qu'est le Cloud Computing.

Le Cloud Computing est un paradigme distribué à grande échelle, dans lequel les ressources informatiques sont abstraits, virtualisées, dynamiques, évolutives, hautement disponibles et configurable. Les services sont offerts sur demande à des clients externes via une connexion internet haute débit.

La qualité de service (QoS) est une question cruciale dont la fiabilité et la performance sont deux points importants des services offerts par le Cloud Computing. La fiabilité d'un service cloud se préoccupe par la façon probable que le cloud peut réussir à fournir le service demandé par l'utilisateur, tandis que la performance d'un service Cloud concerne la rapidité avec laquelle le cloud peut fournir le service demandé.

Comme tous services informatiques, les services cloud sont vulnérables aux défaillances. En plus, dans des tels systèmes le taux de pannes croît avec la taille du système lui-même. Ce qui impose l'utilisation d'un mécanisme de tolérance aux fautes pour assurer la fiabilité et la qualité (QoS) des services.

La tolérance aux pannes est un domaine qui a été largement étudié. Dans la littérature, il existe plusieurs mécanismes et manières d'envisager une panne dans un système réparti à grande échelle telles que le cloud computing. On peut diviser les mécanismes de tolérance aux pannes en trois catégories : les protocoles par points de reprise, les protocoles par journalisation et La tolérance aux pannes par duplication, 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.

La tolérance aux pannes par duplication consiste en la création de copies multiples des composants sur des nœuds différents. Ces copies doivent être placées de façon intelligente pour assurer une utilisation optimale des ressources. Cette approche par duplication rend possible le traitement des pannes en les masquant.

L'utilisation optimale des ressources de calcul des systèmes cloud est un aspect très important qui mobilise beaucoup de chercheurs à travers le monde. Cette optimalité nécessite une répartition de la charge de travail sur les différents nœuds de calcul. Il faut en effet éviter, dans la mesure du possible, les situations où certains nœuds sont surchargés alors que d'autres sont sous-chargés ou complètement libres. L'équilibrage de la charge de travail sur les diverses ressources disponibles s'avère être un véritable défi.

Notre contribution consiste à proposer une approche pour gérer l'équilibrage de charge et la tolérance aux pannes dans le cloud computing basée sur la réplication. Pour cela, nous avons organisé notre travail en quatre chapitres :

- Le premier chapitre présente les notions et concepts de base du Cloud computing, ses services, ses types, et ses objectifs.
- Le deuxième chapitre est dédié aux concepts de sûreté de fonctionnement ainsi que les différentes techniques de tolérance aux pannes sont évoquées.
- Le troisième chapitre est destiné à l'étape de conception de notre contribution qui prendra en compte la gestion de tolérance aux pannes.
- Le quatrième chapitre est destiné à l'implémentation de la stratégie proposée. L'étude d'évaluation et le positionnement de notre solution sont décrites à partir des résultats d'expérimentation et leurs interprétations.

CHAPITRE 1

CLOUD COMPUTING

Contents

1.1	Introduction	5
1.2	Description du cloud computing	5
1.2.1	Historique	7
1.3	Caractéristiques	8
1.4	Virtualisation	9
1.5	Les différents services du Cloud Computing	9
1.5.1	Le logiciel en tant que service (SaaS) :	9
1.5.2	Plate forme en tant que service (PaaS) :	10
1.5.3	Infrastructure en tant que service (IaaS) :	11
1.5.4	Avantages et Inconvénients des services	11
1.6	Types de Cloud Computing	12
1.6.1	Cloud privé	12
1.6.2	Cloud public	12
1.6.3	Cloud hybride	12
1.6.4	Le Cloud communautaire	13
1.7	Architecture du cloud computing	14
1.8	Avantages du Cloud Computing	15
1.9	Inconvénients du Cloud Computing	15
1.10	Les grands défis relatifs à l'adoption du Cloud Computing	16

1.10.1 Sécurité	16
1.10.2 Tolérance aux fautes et disponibilité	16
1.10.3 L'équilibrage de charge	17
1.11 Conclusion	17

1.1 Introduction

L'évolution des technologies de l'information et de la communication (TIC) a donné naissance à un nouveau paradigme : le Cloud Computing. Ce paradigme, malgré sa relative jeunesse, ne cesse de prendre de l'ampleur et de susciter l'intérêt des entreprises et de la communauté scientifique. En effet, les promesses de cette nouvelle vision de l'informatique sont nombreuses et les éventuels freins à son adoption, bien que non négligeables, sont souvent occultés par ses avantages[2].

L'informatique dans le nuage est plus connue sous sa forme anglo-saxonne : « Cloud Computing », mais il existe de nombreux synonymes francophones tels que : « informatique dans les nuages », « infonuagique » (Québec) ou encore « informatique dématérialisée ». C'est un domaine qui regroupe les technologies de distribution, à la demande et via Internet, de services informatiques logiciels et matériels. L'idée principale de ces technologies est de distribuer des ressources informatiques comme un service d'utilité publique, conformément à ce qui avait été imaginé par les pionniers de l'informatique moderne, il y a plus de 40 ans[1].

Dans ce chapitre, nous allons présenter globalement l'historique du Cloud computing et l'origine de ce terme, suivi d'une définition explicite de ce dernier qui sera basée sur une analyse des définitions proposées par le monde académique. Nous décrivons aussi la virtualisation qui est une partie essentielle dans l'informatique en nuages, sans oublier les services de Cloud, les types de Cloud et ses acteurs ainsi que les avantages, les inconvénients, les objectifs principaux du Cloud computing.

1.2 Description du cloud computing

Durant la dernière décennie, il y eut plusieurs tentatives de définitions pour le Cloud Computing. Nous donnons dans ce qui suit quelques-unes de ces définitions. Le « cloud computing » est un néologisme utilisé pour décrire l'association d'Internet (« cloud », le nuage) et l'utilisation de l'informatique (« computing »). C'est une manière d'utiliser l'informatique dans laquelle tout est dynamiquement couplé et évolutif et dans laquelle les ressources sont fournies sous la forme de services au travers d'Internet. Les utilisateurs n'ont ainsi besoin d'aucune connaissance ni expérience en rapport avec la technologie derrière les services proposés[6].

Cette nouvelle technologie permet la mise à disposition dynamique des tech-

nologies d'information sur Internet et les présente comme services selon le modèle « pay-as-you-go » [4].

Wang et al. [2] Définissent le Cloud comme « un ensemble de services mis en réseau, offrant sur demande des plates-formes informatiques extensibles et peu chères, garantissant une certaine qualité de service, généralement personnalisée. Ces plates-formes doivent être accessibles de façon simple et continue » .

Dans une autre définition, les auteurs présentent le cloud computing comme un type de système parallèle et distribué, constitué d'une collection d'ordinateurs interconnectés et virtualisés et ils sont dynamiquement fournis et présentés comme une seule ou plusieurs ressources de calcul basés sur le contrat de service à niveau établi par la négociation entre le fournisseur de services et les consommateurs[3].

Selon l'Institut national des normes et de la technologie, Cloud computing est un modèle pour permettre un accès pratique à la demande du réseau à un ensemble partagé de ressources informatiques configurables (par exemple, les réseaux, les serveurs, le stockage, les applications et les services) qui peuvent être provisionnés rapidement et libérés avec un effort de gestion minimale ou par l'interaction de fournisseur de services[5].

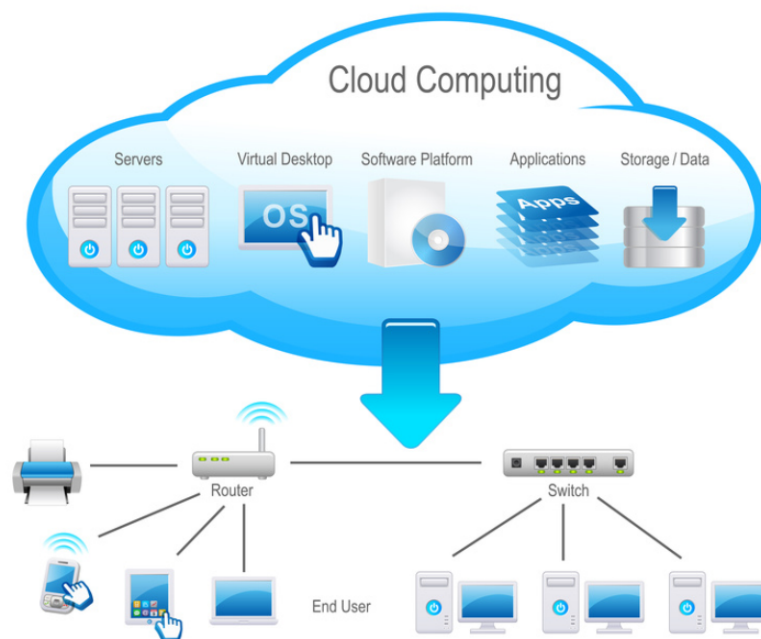


FIGURE 1.1 – L'environnement de Cloud computing

1.2.1 Historique

Les fondations de cloud computing peuvent être retracées jusqu’aux années soixante où John McCarthy, pionnier de l’intelligence artificielle, a pour la première fois formulé l’idée d’une « informatique utilitaire », en anglais utility computing. [3] L’idée consiste à pouvoir fournir à l’utilisateur de la puissance de calcul, des capacités de stockage et des capacités de communication, de la même façon que l’on lui fournit l’électricité ou l’eau dans les réseaux publics.

Bien avant la naissance du terme de Cloud computing, les informaticiens utilisaient déjà des services de Cloud computing comme le webmail2, le stockage de données en ligne (photos, vidéos...) ou encore le partage d’informations sur les réseaux sociaux. Dans les années 90, un autre concept avait déjà préparé le terrain au Cloud computing. Il s’agit de L’ASP (Application Service Provider) qui permettait au client de louer l’accès à un logiciel installé sur les serveurs distants d’un prestataire, sans installer le logiciel sur ses propres machines. Le Cloud computing ajoute à cette offre la notion d’élasticité avec la possibilité d’ajouter de nouveaux utilisateurs et de nouveaux services d’un simple clic de souris[8].

Il est communément admis que le concept de Cloud Computing a été initié par le géant Amazon en 2002. Le cybermarchand avait alors investi dans un parc informatique afin de pallier les surcharges des serveurs dédiés au commerce en ligne constatées durant les fêtes de fin d’année. A ce moment-là, Internet comptait moins de 600 millions d’utilisateurs mais la fréquentation de la toile et les achats en ligne étaient en pleine augmentation. En dépit de cette augmentation, les ressources informatiques d’Amazon restaient peu utilisées une fois que les fêtes de fin d’année étaient passées. Ce dernier a alors eu l’idée de louer ses capacités informatiques le reste de l’année à des clients pour qu’ils stockent les données et qu’ils utilisent les serveurs. Ces services étaient accessibles via Internet et avec une adaptation en temps réel de la capacité de traitement, le tout facturé à la consommation. Cependant, ce n’est qu’en 2006 qu’Amazon comprit qu’un nouveau mode de consommation de l’informatique et d’internet faisait son apparition. Réalisant ce qu’ils pourraient faire de toute cette puissance, de nombreuses compagnies ont ensuite commencé à montrer un certain intérêt à échanger leurs anciennes infrastructures et applications internes contre ce que l’on appelle les ”pay per-use service” (services payés à l’utilisation)[7].

Actuellement, que ce soit pour les petites, moyennes ou grandes entreprises,

le Cloud Computing est devenu la solution de prédilection pour le déploiement de leurs services informatiques. Ainsi, on estime qu'actuellement 70% du trafic réseau global est imputable au Cloud, et que celui-ci va doubler et atteindre un taux de 86% en 2019. De même, l'International Data Corporation, un cabinet d'étude de marché américain, prévoit que les services Cloud vont générer plus de 95 milliards de dollars de recettes en 2016, et près de 200 milliards de dollars en 2020 [2].

1.3 Caractéristiques

Le Cloud computing se distingue des solutions traditionnelles par les caractéristiques suivantes [9] :

- **Large accessibilité via le réseau** : Les services sont accessibles en ligne et sur tout type de support (ordinateur de bureau, portable, smartphone, tablette). Tout se passe dans le navigateur Internet.
- **Mesurabilité du service** : L'utilisation du service par le client est supervisée et mesurée afin de pouvoir suivre le niveau de performance et facturer le client en fonction de sa consommation réelle.
- **Solution multi-client** : Une même instance d'un logiciel est partagée par l'ensemble des clients de façon transparente et indépendante. Tous les clients utilisent la même version du logiciel et bénéficient instantanément des dernières mises à jour. Chaque client dispose d'un paramétrage utilisateur qui lui est propre.
- **Disponibilité à la demande** : Le service peut être souscrit rapidement et rendu opérationnel automatiquement avec un minimum d'interaction avec le fournisseur.
- **Élasticité immédiate des ressources** : Des ressources supplémentaires peuvent être allouées au service pour assurer la continuité du service en cas de pic de charge, ou bien être réallouées à un autre service dans le cas inverse.
- **Mutualisation des ressources** : Des ressources utilisées pour exécuter le service sont mutualisées pour servir à de multiples clients. Les multiples serveurs sollicités, totalement interconnectés, ne forment plus qu'une seule ressource virtuelle puissante et performante.

1.4 Virtualisation

Avec l'avènement récent du Web 2.0 et la disponibilité accrue de la bande passante sur Internet, les technologies de virtualisation représentent un facteur clé du cloud computing. La caractéristique la plus importante est la possibilité d'installer sur la même machine physique (serveur) plusieurs systèmes d'exploitation sur différentes machines virtuelles. À son tour, cette technologie a l'avantage supplémentaire d'une réduction globale des coûts grâce à l'utilisation minimale de matériel et par conséquent une consommation d'énergie réduite.

Le concept machine virtuelle (VM) remonte aux années 1960 ; il a été introduit par IBM comme un moyen pour fournir un accès interactif et simultané à leurs ordinateurs centraux. Une VM est une instance de la machine physique et elle donne l'illusion aux utilisateurs d'avoir un accès direct à une machine physique (PM). Les VMs sont utilisées pour permettre le partage des ressources d'un matériel très coûteux. Chaque VM est une copie entièrement protégée et isolée du système. La virtualisation est donc utilisée pour réduire les coûts du matériel et d'améliorer la productivité globale en permettant à plusieurs utilisateurs de travailler simultanément sur la même PM. Cela permet à la virtualisation d'augmenter l'utilisation de la machine.

L'objectif principal de la virtualisation est de cacher les caractéristiques physiques des ressources informatiques afin que les autres systèmes, les applications ou les utilisateurs finaux interagissent avec ces ressources[10].

1.5 Les différents services du Cloud Computing

Le Cloud Computing fournit une infrastructure, plate-forme et application comme des services, qui sont rendus disponibles comme des services payants dans un modèle " pay-as-you-go " aux consommateurs [13]. Ces services dans l'industrie sont respectivement référencés comme Infrastructure as a Service (IaaS), Plate-forme as a Service (PaaS) et le Software as a Service (SaaS)[5] Voir figure 1.2.

1.5.1 Le logiciel en tant que service (SaaS) :

Est comme son nom l'indique, un modèle de fourniture de logiciels hébergés à distance. L'utilisateur ne gère ni l'infrastructure du cloud ni la plate-forme où l'application s'exécute. Plus besoin d'installer l'application sur ses propres

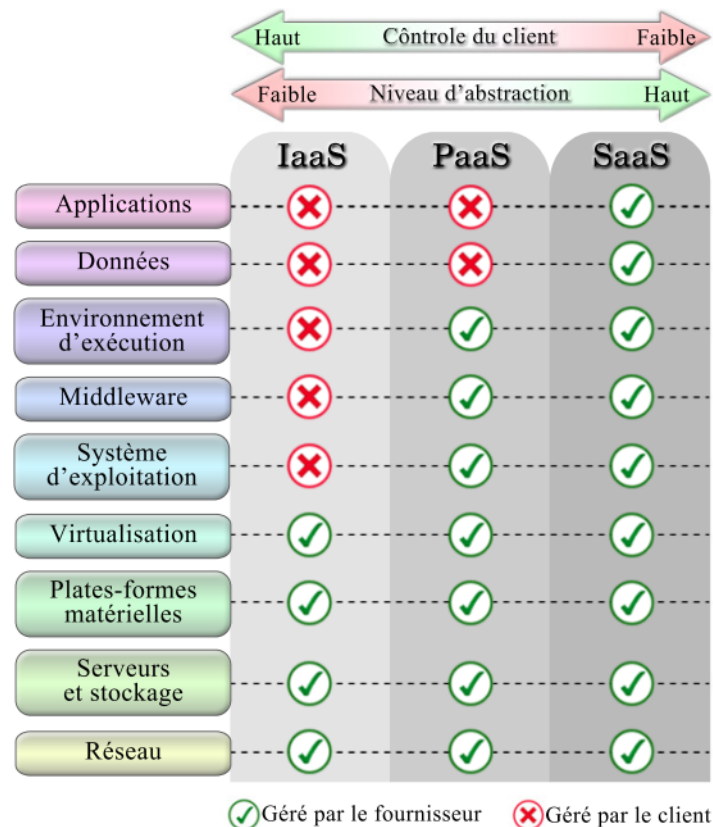


FIGURE 1.2 – Les trois modèles de service du Cloud Computing et la répartition des responsabilités entre le fournisseur et le client.

ordinateurs, le client y accède via sa connexion Internet et n'a donc pas à mettre à jour ou à gérer le fonctionnement et la sécurité du logiciel, toutes ces tâches sont effectuées par l'éditeur (le fournisseur). Ce qui simplifie la maintenance et le support. Ces applications sont accessibles à partir de différents périphériques clients par le biais d'une interface client légère, comme un navigateur Web (par exemple : le courrier électronique basé sur le Web), ou une interface spéciale. Parmi les exemples les plus connus, on retrouve : Google Apps, Microsoft Office et OnLive.

1.5.2 Plate forme en tant que service (PaaS) :

Les PaaS sont des services Cloud destinés aux développeurs d'applications qui leur facilitent le déploiement de leurs applications dans le cloud à l'aide d'outils (langages de programmation, bibliothèques, ...) pris en charge généralement par le fournisseur. Les développeurs n'ont donc pas accès à l'infrastructure,

mais ont le contrôle sur les paramètres de configuration de leur environnement d'hébergement (serveur, base de données, ...), leur permettant ainsi de se concentrer uniquement sur le développement de leurs applications et de ne pas perdre de temps sur leur déploiement. Exemples de PaaS : Google App engine ou AppFog.

1.5.3 Infrastructure en tant que service (IaaS) :

C'est la couche la plus basse des niveaux de services Cloud. Sur une IaaS l'utilisateur gère librement son infrastructure et peut définir et contrôler précisément les serveurs qu'il utilise, le système d'exploitation, le stockage, etc. Par rapport à d'autres modèles de service, ce modèle offre un niveau de contrôle et une flexibilité élevée aux clients, mais exige un effort d'administration important. De ce fait, c'est un modèle qui est plus destiné aux architectes informatiques. Dans ce modèle de service, les fournisseurs mettent à disposition du client une ou plusieurs machines physiques ou, plus généralement, virtuelles (c.-à-d. des VMs) avec différentes capacités en calcul, en mémoire, en stockage ou en transfert réseau. Le client peut alors librement choisir les systèmes d'exploitation et les applications qu'il souhaite installer sur ces machines, et il s'occupe de leur administration. Exemple d'IaaS : Amazon et son EC2[14].

1.5.4 Avantages et Inconvénients des services

Types.	Avantages.	Inconvénients .
SaaS.	<ul style="list-style-type: none"> - Pas d'installation . - Plus de licence . - Migration . 	<ul style="list-style-type: none"> - Logiciel limité. - Besoin de sécurité . - Dépendance des prestataire. - Services dédiés.
PaaS.	<ul style="list-style-type: none"> - Ne nécessite pas d'infrastructure. - Pas d'installation . - Environnement hétérogène . 	<ul style="list-style-type: none"> - Limitation des langages . - Pas de personnalisation dans la configuration des machines virtuelles.
IaaS.	<ul style="list-style-type: none"> - Administration . - Personnalisation . - Flexibilité d'utilisation . 	<ul style="list-style-type: none"> - Besoin de sécurité. - Besoin d'un administrateur Système .

FIGURE 1.3 – Avantages et Inconvénients des services de cloud computing[11].

1.6 Types de Cloud Computing

Le concept de Cloud Computing est encore en évolution. On peut, toutefois, dénombrer trois types de Cloud Computing [3] :

1.6.1 Cloud privé

Cloud privé (également appelé Cloud interne) est un terme marketing pour une architecture informatique propriétaire qui fournit des services hébergés à un nombre limité de personnes derrière un pare-feu. Typiquement, les Clouds privés sont mis en application au centre de traitement des données de l'entreprise et contrôlés par les ressources internes. Un Cloud privé maintient les données de corporation dans les ressources sous la commande du tutelle légale et contractuelle de l'organisation.

1.6.2 Cloud public

Cloud public (ou Cloud externe) est un modèle standard du Cloud Computing, dans lequel un prestataire de services met des ressources, telles que les applications et le stockage, à la disposition du grand public sur Internet. Les services de ce Cloud peuvent être gratuits ou offerts sur un modèle de payer-par-utilisation. L'un des principaux avantages de ce type de Cloud est que la mise en place est facile et peu coûteuse parce que le matériel, l'application et les coûts de bande passante sont couverts par le fournisseur. Les Clouds externes sont connus aussi pour leur évolutivité pour répondre aux besoins.

1.6.3 Cloud hybride

Pour rencontrer les avantages des deux approches, de nouveaux modèles d'exécution ont été développés pour combiner les Clouds publics et privés dans une solution unifiée, c'est les Clouds hybrides. Des applications avec des préoccupations d'ordre juridique, réglementaire ou d'un service important pour les renseignements peuvent être dirigées vers un Cloud privé. D'autres applications avec des conditions moins rigoureuses de normalisation ou d'un service moins strict peuvent s'appuyer sur une infrastructure de Cloud public. La mise en œuvre d'un modèle hybride nécessite une coordination supplémentaire entre les secteurs privé et public du système de gestion des services. Ceci implique un outil de gestion

fédéré de politique, une sécurité fédérée, une gestion des biens d'information, un contrôle coordonné d'approvisionnement et un système unifié de surveillance. Par exemple, une organisation peut utiliser un service de Cloud public, comme le Cloud d'Amazon Elastic Compute (EC2) pour l'informatique générale, mais stocke les données des clients dans son propre centre de données.

1.6.4 Le Cloud communautaire

Dans un Cloud communautaire, l'infrastructure est utilisée et partagée par plusieurs organisations qui appartiennent à une certaine communauté qui ont les mêmes préoccupations et intérêts (p. ex. : mission, besoins en sécurité, politique de gestion, et conformité à différentes considérations éthiques). Tout comme le Cloud privé, le Cloud communautaire est donc un système fermé qui n'est accessible que par un nombre limité d'organisations. Cependant, les coûts associés à son utilisation sont moins importants car distribués entre plusieurs organisations. L'infrastructure d'un Cloud communautaire est, dans l'absolu, gérée et détenue par les organisations membres du Cloud. Cependant, tout comme pour les Clouds privés, elle peut être gérée, voire détenue, par une entreprise tierce[2].

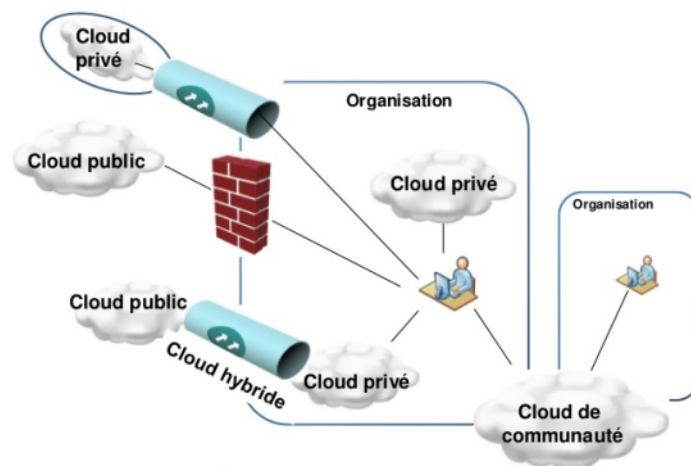


FIGURE 1.4 – Les modèles de déploiement dans le Cloud computing

1.7 Architecture du cloud computing

En faisant abstraction de détails de plus haut niveau, l'architecture globale du cloud computing comporte essentiellement[13] :

- **Des clients** : personne, entreprise, groupe qui accèdent aux différents services offerts par le cloud.
- **Des services** : Différents niveaux de services et données sont gérés par des fournisseurs afin de les offrir à la demande des clients du cloud.
- **Un réseau** : Intermédiaire entre le client et le fournisseur, qui permet de transiter les services (chemin que les services entreprennent) c'est le réseau Internet.
- **Des fournisseurs** : Ce sont des entités chez lesquelles on alloue les services cloud.
- **Des serveurs** : Où sont stockés les services cloud, ils sont éparpillés partout dans le monde qui constituent le cœur hardware de cloud computing. Les entités citées ci-dessus sont connectées selon le modèle de déploiement du cloud privé et public.

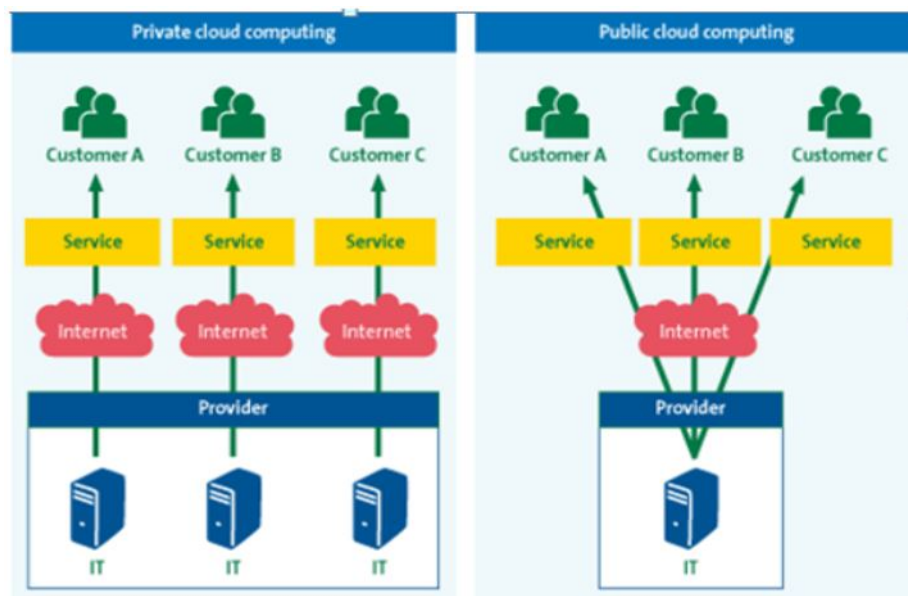


FIGURE 1.5 – Architecture globale et les composants de base du cloud computing [13].

1.8 Avantages du Cloud Computing

Avantages du Cloud Computing L'intérêt des Cloud Computing est évident dans le sens où au lieu d'acheter cher des serveurs et des logiciels, qui ne sont pas utilisés à 100%, les entreprises les louent et ne paient que pour l'usage qu'elles en font. Elles peuvent aussi, en quelques minutes, accéder à des capacités de stockage et de calcul supplémentaires, auxquelles elles n'auraient, dans le cas de PME, jamais pu prétendre pouvoir les payer seules. En plus de cela, Cloud offre plusieurs avantages aux utilisateurs[11].

- Flexibilité pour choisir des fournisseurs qui offrent des services fiables et à grande échelle aux entreprises.
- La virtualisation, c'est à dire pas besoin d'investir pour concevoir une plateforme de Cloud Computing.
- Réduction du coût dû à l'efficacité opérationnelle et déploiement plus rapide de nouveaux services aux entreprises.
- Flexibilité de la répartition des coûts pour les clients.
- Un démarrage rapide : Le cloud computing permet de tester le plan économique rapidement, à coûts réduits et avec facilité.
- L'agilité pour l'entreprise : Résolution des problèmes de gestion informatique simplement sans avoir à vous engager à long terme.
- Un développement plus rapide des produits : Réduisant le temps de recherche pour les développeurs sur le paramétrage des applications.
- Pas de dépenses de capital : Plus besoin de locaux pour élargir vos infrastructures informatiques.

1.9 Inconvénients du Cloud Computing

Hormis les avantages cités ci-dessus, Cloud Computing possède quelques inconvénients, parmi lesquels[11] :

- La bande passante peut faire exploser le budget : La bande passante qui serait nécessaire pour mettre cela dans le Cloud est gigantesque, et les coûts seraient tellement importants qu'il est plus avantageux d'acheter le stockage soi-même plutôt que de payer quelqu'un d'autre pour s'en charger.

- Les performances des applications peuvent être amoindries : Un Cloud public n'améliorera définitivement pas les performances des applications.
- La fiabilité du Cloud : Un grand risque lorsqu'on met une application qui donne des avantages compétitifs ou qui contient des informations clients dans le Cloud.
- Taille de l'entreprise : Si votre entreprise est grande alors vos ressources sont grandes, ce qui inclut une grande consommation du cloud. Vous trouverez peut-être plus d'intérêt à mettre au point votre propre Cloud plutôt que d'en utiliser un externalisé. Les gains sont bien plus importants quand on passe d'une petite consommation de ressources à une consommation plus importante.

1.10 Les grands défis relatifs à l'adoption du Cloud Computing

Le Cloud Computing a connu un succès rapide notamment du fait qu'il permette aux clients de réaliser une économie d'échelle importante en payant uniquement pour les ressources utilisées tout en déléguant la gestion de l'infrastructure au fournisseur. Cependant, il existe encore un nombre important de défis inhérents à l'adoption du Cloud Computing[12].

1.10.1 Sécurité

Lors de l'adoption du Cloud, le client s'attend à recevoir une sécurité identique, voire meilleure, que celle qu'il aurait eu dans une installation privée. En effet, le fournisseur dispose souvent de plus de moyens pour gérer la sécurité physique de l'infrastructure Cloud, mais aussi celle des données et des applications qui s'y exécutent. Cependant, le fait de déléguer cette sécurité à un tiers comporte un certain nombre de nuances qui doivent être prises en compte par le client

1.10.2 Tolérance aux fautes et disponibilité

Le fournisseur Cloud est censé offrir à ses clients un environnement tolérant aux fautes où le client ne risque ni de perdre ses données, ni de voir l'exécution de ses applications perturbée. De même, une certaine disponibilité doit être garantie

pour les services Cloud. Ainsi, l'exécution des applications déployées dans le Cloud doit être continue, et le client doit avoir à tout moment la possibilité de déployer une nouvelle application, ou de tout simplement consulter sa consommation en ressources.

Ces deux besoins sont souvent bien définis dans un contrat de niveau de service SLA entre le client et le fournisseur. Toute violation à ce contrat peut alors être sanctionnée financièrement ou juridiquement.

1.10.3 L'équilibrage de charge

L'utilisation efficace des ressources et l'équilibrage de charge sont des objectifs ultimes pour les fournisseurs de services, afin de maximiser leurs profits, et assurer une qualité de services optimale[4].

1.11 Conclusion

Au cours de cette première partie, nous avons fourni une base théorique sur le Cloud Computing, en présentant ses types, ses services (IaaS, PaaS, SaaS), ainsi que ses avantages et inconvénients. Enfin, nous avons présenté les grands défis auquel la communauté scientifique doit faire face pour augmenter l'attrait du Cloud Computing et finir de convaincre certaines entreprises encore hésitantes pour adopter ce nouveau paradigme.

Parmi ces défis, nous avons évoqué la Tolérance aux Fautes et l'équilibrage de charge. Nos contributions s'articulent autour de ces deux points, Nous enchaînons donc, dans le chapitre suivant, en dressant un état de l'art sur la Tolérance aux Fautes dans le Cloud Computing.

CHAPITRE 2

TOLÉRANCE AUX PANNE

Contents

2.1	Introduction	20
2.2	Notion de sûreté de fonctionnement	20
2.2.1	Attributs de la sûreté de fonctionnement	21
2.2.2	Entravs	22
2.2.3	Moyens d'assurer la sûreté de fonctionnement	24
2.2.4	Classification des pannes	25
2.2.5	Détection des pannes	27
2.3	Méthodes de tolérance aux pannes	29
2.4	Techniques de tolérance aux pannes dans les systèmes répartis	30
2.4.1	Tolérance aux pannes par sauvegarde (checkpointing)	31
2.4.2	Tolérance aux pannes par journalisation	34
2.4.3	Comparaison des protocoles	37
2.4.4	Migration	38
2.4.5	Tolérance aux pannes par duplication (Approche mas- quante)	39
2.5	Protocole de réplication	43
2.5.1	Protocole de réplication passive	43
2.5.2	Protocole de réplication active	44
2.5.3	Protocole de réplication semi-active	44

2.6	La tolérance aux pannes dans les cloud	45
2.7	L'équilibrage de charge	46
2.7.1	Problème de l'équilibrage de charge	46
2.7.2	Le système d'équilibrage de charge	47
2.7.3	Les algorithmes d'équilibrage de charge dans le Cloud Computing	49
2.8	Conclusion	50

2.1 Introduction

Lors du déploiement d'une application, l'utilisateur s'attend à ce que celle-ci puisse continuellement répondre à son besoin tout en respectant son cahier des charges. En particulier, il s'attend à ce que l'application soit fiable, hautement disponible, maintenable et sécurisée. L'ensemble de ces attributs définit le niveau de sûreté de fonctionnement de cette application[2].

Pour maximiser cette propriété, il faut en particulier, que les applications puissent s'affranchir des pannes pouvant survenir dans le système hôte. Cet objectif, peut être atteint en mettant en œuvre différentes techniques de Tolérance aux Fautes.

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

Puisqu'il est impossible d'empêcher totalement les pannes, une solution consiste à mettre en place des mécanismes de redondance, en dupliquant les ressources critiques, ou des techniques de sauvegarde et de retour en arrière. Lorsqu'une des ressources tombe en panne, les autres ressources prennent le relais afin de laisser le temps aux administrateurs du système de remédier à l'avarie. Idéalement, dans le cas d'une panne matérielle, les éléments matériels fautifs devront pouvoir être « extractibles à chaud » (en anglais « hot swappable »), c'est-à-dire pouvoir être extraits puis remplacés, sans interruption de service[15].

Dans ce chapitre, nous présentons les différents concepts liés à la tolérance aux pannes ainsi que les diverses techniques de gestion des pannes utilisées.

2.2 Notion de sûreté de fonctionnement

La tolérance aux pannes s'inscrit dans le contexte plus large de la sûreté de fonctionnement.

La sûreté de fonctionnement (dependability) est définie comme la capacité de fournir un service dans lequel un utilisateur peut raisonnablement placer sa confiance. De façon plus quantitative, la sûreté de fonctionnement permet de décider si un système est capable d'assurer que la fréquence de défaillance du service et la gravité de ces défaillances restent inférieures à un minimum considéré comme acceptable[16].

Pour bien comprendre les tenants et les aboutissants de la sûreté de fonction-

nement, nous présentons dans ce qui suit ses attributs, ses entraves et ses moyens voir la figure 2.1.

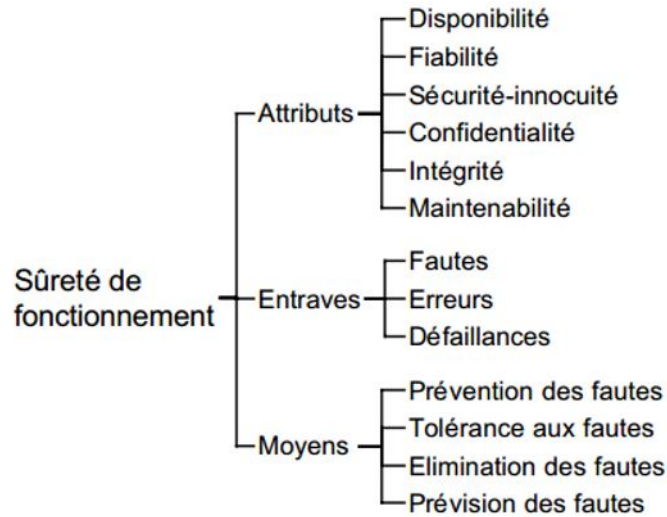


FIGURE 2.1 – L'arbre de la sûreté de fonctionnement

2.2.1 Attributs de la sûreté de fonctionnement

Les attributs de la sûreté de fonctionnement d'un système mettent plus ou moins l'accent sur les propriétés que doivent vérifier la sûreté de fonctionnement du système.

Ces attributs permettent d'évaluer la qualité du service fourni par un système. Dans [17], six attributs de la sûreté de fonctionnement sont définis :

1. **Disponibilité** : c'est la propriété requise par la plupart des systèmes sûrs de fonctionnement. Il s'agit de la fraction de temps durant laquelle le système est disponible pour des fins utiles (en excluant les temps de défaillance et de réparation). C'est la probabilité que le système soit opérationnel à l'instant t .
2. **Fiabilité** : cet attribut évalue la continuité du service, c'est à dire. Le taux en temps de fonctionnement pendant lequel le système ne subit aucune faute. C'est la probabilité conditionnelle qu'un système fonctionne sans défaillance pendant l'intervalle de temps $[0, t]$ sachant qu'il était opérationnel à l'instant 0.
3. **Sécurité-innocuité** : Évalue la continuité du service avant l'arrivée d'une défaillance catastrophique. Elle utilise donc la même expression mathématique

que la fiabilité appliquée uniquement aux défaillances catastrophiques.

4. **Confidentialité** : cet attribut évalue la capacité du système à fonctionner en dépit de fautes intentionnelles et d'intrusions illégales.
5. **Intégrité** : l'intégrité d'un système définit son aptitude à assurer des altérations approuvées des données.
6. **Maintenabilité** : cette propriété décrit la souplesse du système vis-à-vis des modifications apportées en vue de sa maintenance. C'est le temps d'interruption de fonctionnement due à des opérations de maintenance préventive (avant qu'une défaillance ne survienne), corrective (à la suite d'une défaillance), et perfective (pour faire évoluer le système).

L'importance des attributs de la sûreté de fonctionnement présentés ci-dessus est principalement liée aux applications et à leurs besoins. Par exemple, pour les applications critiques comme le pilotage de fusées, une grande importance doit être donnée à tous les attributs de la sûreté de fonctionnement. Les applications parallèles à longue durée d'exécution font prévaloir la maintenabilité et la fiabilité.

2.2.2 Entraves

Les entraves à la sûreté de fonctionnement sont un ensemble d'états dans un système qui peuvent influencer négativement sur l'un des attributs précédemment décrits, une combinaison de ces attributs, ou même sur l'ensemble de ces attributs. Ces entraves sont de trois types [16] : les fautes, les erreurs et les défaillances.

1. **Fautes** : Une faute ou panne est caractérisée par sa nature, son origine et son étendue temporelle [17].
 - La nature d'une faute donne le caractère intentionnel ou accidentel d'une faute, les fautes intentionnelles sont créées délibérément dans le dessein de nuire tandis que les fautes accidentelles apparaissent de manière fortuite.
 - La durée d'une faute exprime la dépendance vis-à-vis de conditions ponctuelles, Une faute temporaire est présentée pendant une durée limitée et disparaît si telle ou telle condition interne ou externe ne peut l'éliminer.
 - L'origine d'une faute est la source d'une faute. Une faute est due à des phénomènes soit physiques soit humains. Elle est provoquée soit par une partie de l'état du système soit par l'environnement. Elle appartient soit à la phase de conception soit à la phase d'exploitation du système.

- L'étendue d'une faute précise la portion du système affectée. Une faute locale affecte une partie du système alors qu'une faute globale en affecte plusieurs.
2. **L'erreur** : correspond à un état incorrect dans un système. Elle se produit après l'activation d'une faute ou à cause d'une autre erreur. Dans [18] Une classification des erreurs selon leurs types a été proposée. Premièrement, le service ne correspond pas en valeur à celui spécifié. Deuxièmement, le service n'est pas délivré dans l'intervalle de temps spécifié. Plusieurs catégories sont définies dans [18], le service rendu est toujours en avance ou toujours en retard ou encore arbitrairement en avance ou en retard. Le fait que le système omette de rendre le service est considéré également comme un type d'erreur. Un arrêt (crash) est défini par le fait que le système omet définitivement de délivrer des services.
 3. **Défaillances** : une défaillance dénote l'incapacité d'un élément du système à assurer le service spécifié par l'utilisateur. La défaillance est caractérisée par son domaine, sa perception par les utilisateurs et ses conséquences sur l'environnement [17].
 - Le domaine de défaillance est le domaine des erreurs actives.
 - Une défaillance est cohérente si tous les utilisateurs en ont la même perception. Sinon, c'est une défaillance incohérente ou byzantine.
 - Les conséquences sur l'environnement sont appréciées de mineures ou bénignes à majeures ou catastrophiques suivant les dommages subis
 4. **Mode de défaillance (Failure mode)** : Un système ne défaille généralement pas toujours de la même façon, ce qui conduit à la notion de mode de défaillance.

Un mode de défaillance spécifie l'effet par lequel une défaillance est observée. Un mode de défaillance peut être caractérisé selon quatre points de vue [25], comme indiqué dans le Tableau (2.2), ce qui définit la classification des défaillances du service. La gravité des défaillances instaure un classement des conséquences des défaillances sur l'environnement du système. Lorsque la fonction du système comporte un ensemble de fonctions élémentaires, la défaillance d'un ou de plusieurs services remplissant ces fonctions peut laisser le système dans un mode dégradé, qui offre encore un sous-ensemble de services à l'utilisateur. Plusieurs de ces modes dégradés peuvent être identifiés, tels que service ralenti, service restreint, service d'urgence, etc.

Défaillance	Domaine	Défaillance en valeur : l'information délivrée ne correspond pas avec la fonction du système.
		Défaillance temporelles : le moment de la délivrance de l'information ne permet pas l'accomplissement de la fonction du système.
	Détectabilité	Défaillance signalées : la possibilité de détecter et de signaler que le service délivré est incorrect.
		Défaillance non signalées : La délivrance d'un service incorrect n'est pas détectée.
	Cohérence	Défaillances cohérentes : tous les utilisateurs aperçoivent le service incorrect identiquement.
		Défaillances incohérentes ou défaillances byzantines : certains ou tous les utilisateurs n'aperçoivent pas le service incorrect de la même manière.
	Conséquences	Défaillances bénignes : le dommage causé au système ou à son environnement est négligeable et sans présenter de risque pour l'homme.
		Défaillance critique (hasardeuse) : elle entraîne la perte d'une ou des fonctions importantes du système et cause des dommages importants au système.
		Défaillances catastrophiques

FIGURE 2.2 – Modes de défaillance.

Dans ce cas, le système est dit avoir souffert des défaillances partielles. Nous pouvons ainsi schématiser la relation entre ces entraves par la Figure 2.3.



FIGURE 2.3 – La relation entre défaillance, erreur et faute

2.2.3 Moyens d'assurer la sûreté de fonctionnement

Il s'agit des méthodes et techniques permettant de fournir au système l'aptitude à délivrer un service conforme à l'accomplissement de sa fonction, et de donner confiance dans cette aptitude. [19] Le développement d'un système sûr de fonctionnement passe par l'utilisation combinée de ces méthodes qui peuvent être classées en quatre catégories [20].

- **La prévention des fautes** : vise à empêcher l'apparition ou l'introduction des fautes dans le système. Elle repose sur des règles de développement (modularisation, utilisation de langage fortement typé, preuve formelle, etc.). Ce sont généralement les approches de vérification des modèles conceptuels ; Par exemple, si un composant matériel est particulièrement enclin à la surchauffe, un système de refroidissement adéquat peut être utilisé. De même, si un composant logiciel est potentiellement vulnérable à une intrusion réseau, alors un pare-feu peut être préventivement installé.
- **L'élimination des fautes** : qui se focalise sur les techniques permettant de réduire la présence de fautes ou leurs impacts. L'élimination de faute se fait pendant les phases de validation ou lors des premières utilisations du système. Cela est réalisé par des méthodes statiques de preuve de la validité du système (simulation, preuves analytiques, tests, ...).
- **La prévision des fautes** : anticipe ces dernières pour ensuite pouvoir appliquer des moyens de l'élimination ou de la tolérance aux fautes. Il s'agit d'une estimation et évaluation de la présence des fautes (temps, nombre, impact) et de leurs conséquences. Ceci est réalisé généralement par des méthodes d'injection de fautes afin de valider le système relativement à ces fautes ; Par exemple, il est possible de prévoir certaines pannes en surveillant la température des processeurs et la vitesse des ventilateurs de refroidissement. De même, la défaillance d'un disque dur peut être prédite en analysant différents paramètres tels que le délai de mise en marche du disque et le taux d'erreur lors des écritures et des lectures [21].
- **La tolérance aux pannes ou aux fautes** : qui essaye de fonctionner en dépit des fautes. Le degré de tolérance aux pannes se mesure par la capacité du système à continuer à délivrer son service en présence de fautes.

2.2.4 Classification des pannes

Les pannes peuvent être classées selon leur origine ou selon leur degré de gravité. Ces deux classifications sont indépendantes dans le sens où, deux mêmes pannes ayant la même origine peuvent avoir un degré de gravité différent et inversement. Nous présentons ces deux classifications dans ce qui suit [2].

Selon leur origine

Les pannes peuvent être d'origine matérielle, logicielle, humaine ou environnementale [2].

- **Pannes d'origine logicielle :** Les pannes d'origine logicielle sont les plus fréquentes, et représentent plus de 50% des pannes enregistrées dans différents systèmes étudiés. Ces pannes sont dues à des fautes incorporées durant le stade de développement du logiciel à cause d'un manque de budget, d'une mauvaise planification ou tout simplement à cause de l'incompétence de certaines parties impliquées dans le projet. L'activation de ces fautes peut survenir à n'importe quel moment lors du fonctionnement du logiciel. En particulier, le manque de ressources, une configuration nouvelle de l'environnement d'exécution, l'interaction avec d'autres systèmes ou encore le vieillissement du logiciel peuvent constituer des facteurs d'activation. Bien souvent, le redémarrage du logiciel fautif peut suffire à éradiquer la panne et à restaurer un état sain.
- **Pannes d'origine matérielle :** Les pannes d'origine matérielle sont dues à un défaut dans un ou plusieurs composants matériels du système. Le plus souvent, ce sont des pannes franches qui causent l'arrêt du fonctionnement du système. Elles sont donc facilement détectables, mais requièrent le remplacement des composants défectueux. Dès lors, le rétablissement du système nécessite beaucoup plus de temps en comparaison avec les pannes d'origine logicielle. Les composants matériels les plus enclins aux défaillances sont les disques durs qui sont à l'origine de plus de 80% des pannes.
- **Pannes d'origine humaine :** Ces pannes sont dues à l'intervention d'opérateurs humains, notamment lors du remplacement des composants, lors d'une reconfiguration ou encore, lors d'une mise à jour logicielle. Ces pannes peuvent être intentionnelles, mais sont le plus souvent causées par manque de compétences. D'après une étude portant sur la sûreté de fonctionnement de services déployés sur l'Internet, les pannes d'origine humaine constituent 15 à 30% de l'ensemble des pannes.
- **Pannes d'origine naturelle :** Ces pannes sont dues à des phénomènes naturels tels que les ouragans, les séismes, les canicules ou tout simplement la corrosion. La manifestation de ces phénomènes est relativement rare, mais il est difficile de les prévoir et de s'en prémunir. Cependant, certaines

zones géographiques sont plus sensibles que d'autres et des précautions particulières doivent être prises pour protéger les systèmes qui y sont déployés.

Selon leur degré

Les pannes peuvent être classées selon leur degré de gravité. Nous identifions alors quatre groupes de pannes qui sont présentés ci-dessous [7].

- **Les pannes franches (crash faults)** : soit le système fonctionne normalement (les résultats sont corrects), soit-il ne fait rien. Il s'agit du modèle de panne le plus simple auquel on essaie de se ramener aussi souvent que possible.
- **Les pannes par omission** : le système perd des messages entrant ou sortant d'un processus. On retrouve généralement ce modèle de pannes dans les réseaux.
- **Les pannes de temporisation** : les déviations par rapport aux spécifications concernent uniquement le temps (typiquement le temps de réaction à un événement).
- **Les pannes par valeurs** : les résultats produits par un composant défaillant sont incorrects. Ce type de panne est typiquement détecté par les mécanismes de certification des résultats proposés.
- **Les pannes byzantines** : le système peut faire n'importe quoi, y compris avoir un comportement malveillant.

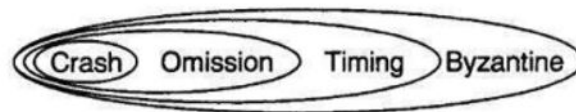


FIGURE 2.4 – Les classes des pannes

2.2.5 Détection des pannes

Les détecteurs de fautes sont un élément central dans les systèmes distribués tolérants aux fautes. La capacité d'un détecteur de fautes pour fonctionner de manière complète et efficace, en présence d'une messagerie non fiable ainsi que des composants sujetes à une forte occurrence de fautes, peut avoir un impact

majeur sur la performance de ces systèmes. "La complétude" est la garantie que la défaillance d'un membre du système soit finalement détectée par tous les autres membres. "L'efficacité" signifie que les défaillances sont détectées rapidement et avec une précision acceptable. Le premier travail pour répondre à ces deux propriétés était par Chandra et Toueg. [22] Les auteurs ont montré l'impossibilité pour tout algorithme de détection de fautes d'atteindre à la fois la complétude et l'efficacité dans un système non fiable et asynchrone. Cette impossibilité résulte de la difficulté inhérente de déterminer si un processus à distance s'est réellement défaillant ou si ses transmissions sont simplement retardées. Il est donc impossible de mettre en œuvre un service de détection de fautes fiables sans faire plus d'hypothèses sur le système. Ce résultat a lancé une vague de recherches théoriques pour la classification des détecteurs de fautes.

— **Cas des systèmes synchrones/asynchrone**

Dans un système synchrone, détecter une défaillance est une issue triviale. Puisque les délais sont liés et connus, les défaillances sont détectées à l'aide d'un délai de garde.

Un système asynchrone est un système pour lequel il n'y a aucune hypothèse temporelle sur les temps de transmission des messages ou sur les temps de calcul des processeurs. Comme dans le cas d'absence de communication en provenance d'un processus pendant une durée t , celui-ci est considéré potentiellement défaillant (suspect). Un temporisateur (délai de garde) est amorcé et un message spécial est envoyé à ce processus. En l'absence de réponse avant la fin du temporisateur, le processus est jugé défaillant et la tolérance à cette faute peut commencer. Cette technique simple n'est pas optimale, puisqu'elle a pour inconvénient de distinguer difficilement un processus lent d'un processus mort.

— **Messages "Ping/Pong"**

De manière périodique ou à la demande, les nœuds envoient un message 'Ping' à tous les autres nœuds ou à une partie d'entre eux. Cette technique peut permettre une détection plus ciblée : elle permet de ne surveiller qu'un sous-ensemble de nœuds. En revanche, pour obtenir autant d'informations qu'avec la technique d'échange de messages de vie, deux fois plus de messages sont nécessaires (chaque message de vie 'Pong' étant réclamé explicitement par un message 'Ping'). [23]

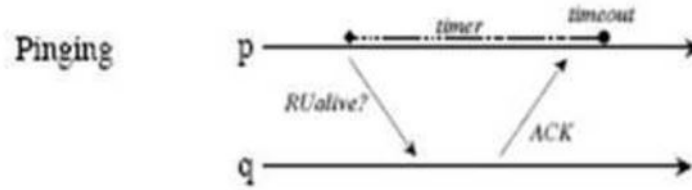


FIGURE 2.5 – Messages "Ping/Pong"

- **Echanges de messages de vie (heartbeats)** Chaque nœud envoie périodiquement un message de vie à tous les autres et attend donc, à chaque période, un message de vie de chacun d'entre eux. Lorsqu'un nœud ne reçoit pas de message de vie d'un autre nœud, il le considère comme défaillant. De nombreux projets de recherche se sont concentrés sur la fiabilité de la suspicion de défaillance, en prenant en compte la variation de latence dans l'arrivée des messages de vie d'un nœud particulier. [23]

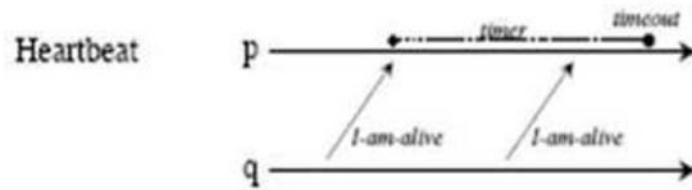


FIGURE 2.6 – messages de vie (heartbeats)

2.3 Méthodes de tolérance aux pannes

La tolérance aux fautes est une des méthodes permettant le développement de systèmes sûrs de fonctionnement. Elle vise à éviter les défaillances et est mise en œuvre par la détection des erreurs et le rétablissement du système. La détection permet d'identifier la présence d'une erreur. Elle peut être effectuée soit pendant l'exécution normale du service, soit pendant une suspension du service (ce qui permet de révéler l'éventuelle présence d'erreurs latentes ou de fautes dormantes). Le rétablissement du système vise à transformer l'état erroné en un état exempt d'erreur et de faute. Le traitement de fautes fait appel à des techniques de diagnostic, de passivation, de reconfiguration ou de réinitialisation. Le traitement de l'erreur peut se faire par trois techniques [7] : la reprise, la poursuite et la

compensation.

- **Reprise** : est la technique la plus couramment utilisée. L'état du système est sauvegardé régulièrement. Le système est ramené dans un état sauvegardé (point de reprise) survenu avant l'occurrence d'erreur.
- **Poursuite** : consiste à rechercher un nouvel état exempt d'erreur. Ceci peut par exemple être réalisé en associant un traitement exceptionnel lorsqu'une erreur est détectée. Le système est amené dans un nouvel état exempt d'erreur.
- **Compensation** : nécessite que l'état du système comporte suffisamment de redondance pour permettre sa transformation en un état exempt d'erreur. Elle est transparente vis-à-vis de l'application car elle ne nécessite pas de ré-exécuter une partie de l'application (reprise), ni d'exécuter une procédure dédiée (poursuite). Reprise et poursuite sont invoquées à la demande, après qu'une ou plusieurs erreurs aient été détectées. Contrairement à la reprise, la poursuite ne garantit pas que le service soit fournie (exemple : saut d'un pas de calcul sans résultat). La compensation peut, quant à elle, être appliquée à la demande ou systématiquement, indépendamment de la présence ou de l'absence d'erreur. Le traitement d'erreur est, si besoin est, suivi du traitement de faute. Ils constituent le rétablissement du système, d'où le qualificatif de la stratégie de tolérance aux fautes correspondante : détection et rétablissement. Le masquage de fautes résulte de l'application systématique de la compensation. Un tel masquage pouvant entraîner une diminution non perçue des redondances disponibles, sa mise en œuvre pratique comporte généralement une détection d'erreur, conduisant au masquage et détection[24].

2.4 Techniques de tolérance aux pannes dans les systèmes répartis

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

2.4.1 Tolérance aux pannes par sauvegarde (checkpointing)

Le Checkpointing (point de reprise) est le processus d'enregistrement périodique des états du système pendant l'exécution sans défaillance, habituellement au serveur de stockage stable. Le checkpointing peut fournir la tolérance aux pannes en cas de transit et les échecs permanents puisqu'il consiste à un retour en arrière, rechargement des derniers points de reprise cohérents, et reprendre l'exécution à partir de ces points. Pour assurer que le système recouvre à partir d'un état cohérent (correct), le système doit stocker toutes les données nécessaires sur l'état actuel de l'application en cours d'exécution et son environnement tels que la communication, l'image de la mémoire, etc. Une bonne récupération ne peut être garantie que si le protocole checkpoint enregistre un état global cohérent. Un état global est un ensemble contenant un point de reprise par processus de l'application. Un état global (nommé aussi ligne de recouvrement) forme une coupe de l'exécution, c'est-à-dire l'ensemble des événements qui ont servi à la réduction depuis les différents états initiaux vers chacun des états représentés par les points de reprise. L'état cohérent doit satisfaire les deux propriétés suivantes

- **Cohérence** : Un état global est cohérent s'il ne contient pas un message orphelin. Un message orphelin dans un état global donné est un message qui a été envoyé après un point de reprise appartenant à cet état global et reçu avant un point de reprise appartenant à cet état global. Ce type de message peut être créé entre les nœuds dépendants. Un nœud No envoie un message à NI donc NI dépend du nœud No. [26]
- **Recouvrement** : Un état global assure le recouvrement s'il ne contient pas un message en transit. 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.

La propriété de cohérence assure que les nœuds (éléments de calcul) commencent leur réexécution à partir d'un état correct qui peut être produit dans le cas d'une exécution sans panne.

La propriété de recouvrement assure que tous les messages en transit, au moment de checkpointing, sont inclus dans l'état global. Sinon, ces messages se perdent lors de la récupération, parce qu'ils ne sont pas renvoyés par les nœuds sources. Un état qui assure la cohérence et le recouvrement est un état de forte

cohérence.

Protocoles de checkpointing

Les protocoles de reprise par sauvegarde peuvent être classés en trois catégories selon le mode de construction de l'état global cohérent de la reprise : la sauvegarde coordonnée, la sauvegarde non coordonnée et la sauvegarde induite par les communications.

1. Sauvegarde coordonnée

La sauvegarde coordonnée est réalisée en coordonnant les processus de manière à assurer que l'ensemble des états des processus forme un état global cohérent. Il existe différentes manières de coordonner les processus. On distingue en particulier la sauvegarde coordonnée bloquante ou la sauvegarde coordonnée non bloquante.

- **La sauvegarde coordonnée bloquante** [27] est réalisée en plusieurs étapes. Tout d'abord, tous les processus de calcul sont arrêtés et les canaux de communication sont vidés. Puis chaque processus sauvegarde son état local. Enfin, les calculs peuvent reprendre. Les canaux de communication étant vides au moment de la sauvegarde, l'état global de l'application correspond à l'état local de tous ses processus et il est cohérent.
- **La sauvegarde coordonnée non bloquante** [28] permet d'identifier l'état des canaux de communication grâce à des «messages marqueurs». Lors d'une étape de sauvegarde, chaque processus sauvegarde son état local puis diffuse immédiatement un marqueur sur tous ses canaux de communication. Ensuite, il sauvegarde tous les messages reçus sur chaque canal de communication jusqu'à la réception d'un marqueur.

Cet ensemble de messages correspond à l'état du canal de communication qui sera alors sauvegardé sur la mémoire stable.

L'avantage de la sauvegarde coordonnée est qu'elle n'est pas sensible à l'effet domino lors de la reprise. Seule la dernière sauvegarde est nécessaire pour un redémarrage qui réduit le surcout de stockage. Le principal inconvénient est le surcout induit par la synchronisation des processus. D'autres méthodes ont été proposées pour tenter d'améliorer les performances :

- **La sauvegarde avec horloges synchronisées [29]** synchronise les horloges des processus. Si chaque processus effectue sa sauvegarde et s'il attend un temps suffisant (dépendant des déviations entre les horloges et du temps de détection d'une défaillance), on est assuré de la cohérence des sauvegardes sans avoir échangé de messages.
- **La sauvegarde coordonnée minimale [30]** ne synchronise un processus qu'avec les processus dont il dépend réellement. Ceci est réalisé en deux étapes. Durant la première, un processus identifie les processus dont il dépend (il émet des messages qui sont diffusés de proche en proche selon les dépendances entre processus) ; durant la deuxième, les processus identifiés réalisent leur sauvegarde.

2. Sauvegarde non coordonnée

La sauvegarde non coordonnée [31] évite la coordination et laisse à chaque processus la décision de sauvegarder son état quand il le souhaite. Ainsi un processus peut décider de sauvegarder son état quand ça lui convient le mieux, par exemple quand la taille de son état est minimale. Lors de la reprise, un algorithme analyse les différences entre les points de sauvegarde des processus pour tenter de déterminer l'ensemble des sauvegardes les plus récentes formant un état global cohérent. Cependant cette approche comporte plusieurs inconvénients. Tout d'abord, lors de la reprise, il y a un risque d'effet domino [32] : lors de la construction de l'état global cohérent, les dépendances entre les messages peuvent entraîner un retour à l'état initial. On peut ainsi perdre une grande partie du travail déjà effectué. De plus, certaines sauvegardes peuvent être inutiles pour la construction d'un état global cohérent. Ces sauvegardes induisent un surcoût mais ne contribuent pas au redémarrage. Enfin, cette méthode oblige les processus à conserver à priori toutes les sauvegardes. Un ramasse-miette peut être utilisé pour supprimer les sauvegardes inutiles en détectant l'ensemble des sauvegardes le plus récent qui constitue un état global cohérent.

3. Sauvegarde induite par les communications (Induits par message)

La sauvegarde induite par les communications (Communication-Induced Checkpointing ou CIC) est un compromis entre la sauvegarde coordonnée et la sauvegarde non coordonnée. Ce protocole utilise deux types de points de sauvegarde : les sauvegardes locales et les sauvegardes forcées. Les sau-

vegardes locales sont des sauvegardes que le processus décide d'effectuer indépendamment. Les sauvegardes forcées sont des sauvegardes qui doivent être effectuées pour empêcher l'effet domino en cas de reprise[33]. On distingue deux familles de protocoles :

- **Les protocoles model-based** les communications et les prises de points de reprise doivent respecter un certain motif. Par exemple, si tout envoi et toute réception de message est précédé d'un point de reprise, alors tous les points appartiennent au moins à un état global cohérent, et le dernier point pris fait toujours partie de la ligne de recouvrement [34]. Ces protocoles sont généralement basés sur les notions de chemins et de cycles pour déterminer les états globaux cohérents.
- **Les protocoles index-based** les points de reprise sont indexés, chaque message porte l'index du dernier point de l'émetteur. Sur réception d'un message indiquant un index supérieur au sien, le récepteur doit prendre un point de reprise avant la prise en compte du message : l'incohérence est évitée "au derniermoment". Ce type de protocole utilise donc une méthode basée sur la suppression des dépendances causales entre les points de reprise des processus qui ont des communications directes.

2.4.2 Tolérance aux pannes par journalisation

Le principe de la tolérance aux fautes par journalisation est de sauvegarder l'histoire de l'application. Les protocoles par journalisation utilisent à la fois la sauvegarde locale de l'état des processus et la journalisation des événements déterministes pour permettre la reprise de l'application. Il est alors possible de reprendre l'exécution des processus défaillants (et uniquement des processus défaillants) à partir de leur dernière sauvegarde en rejouant les événements non déterministes sauvegardés. Pour cela, les protocoles basés sur la journalisation reposent sur l'hypothèse PWD :

Définition 1 : Hypothèse PWD (Piece Wise Deterministic assumption) [35] :

- Un processus est modélisé par une séquence d'intervalles d'état.
- Chaque intervalle débute par un événement non déterministe.
- L'exécution du processus durant chaque intervalle est déterministe.

Chaque processus crée un journal des événements non déterministes qu'il observe pendant l'exécution normale (sans panne). En cas de panne, le processus défaillant est capable de reconstruire son état d'avant la panne en partant de sa dernière sauvegarde et en rejouant les événements non déterministes inscrits dans son journal. L'hypothèse PWD garantit que cette reconstruction aboutira au même état.

Pour pouvoir bénéficier de cette hypothèse, il faut être capable de détecter et d'enregistrer ces événements non déterministes. Ces événements peuvent être des réceptions de messages ou des événements internes au processus (des décisions d'ordonnancement par exemple). Il faut également remarquer que ces informations (le journal et les sauvegardes périodiques) doivent être stockées sur une mémoire stable.

Lors de la reprise par journalisation, seuls les processus défaillants retournent en arrière. La reprise force l'exécution des processus redémarrés à être identique à celle qui s'est produite avant la panne. L'état obtenu après la reprise est donc exactement l'état de l'application d'avant la défaillance. Cet état global est nécessairement cohérent puisqu'il correspond à un état de l'application lors d'une exécution sans panne.

La reprise par journalisation n'est donc pas sensible à l'effet domino. C'est pourquoi la journalisation est souvent utilisée conjointement à la sauvegarde non coordonnée.

Sous l'hypothèse PWD, il est possible de définir la notion de processus orphelin et la condition de « non-orphelinité ».

Définition 2 : On appelle **processus orphelin** un processus dont l'état dépend d'un événement non déterministe qui ne peut être reproduit à la reprise [31].

L'existence de ces processus orphelins caractérise les états globaux « pathologiques » pour lesquels il n'est pas possible de reprendre une exécution. En effet, un processus orphelin est un processus dont l'état dépend d'un événement non déterministe qui ne pourra pas être reproduit à la reprise.

Pour pouvoir redémarrer l'application, les protocoles de journalisation doivent assurer la condition de « non-orphelinité ».

Proposition : Condition de « **non-orphelinité** » Lors de la reprise des processus défaillants, le système ne doit contenir aucun processus orphelin.

Une formalisation de cette condition de « non-orphelinité » peut être trouvée dans

[31]. Les protocoles de reprise par journalisation diffèrent par leur manière d’assurer et d’implanter cette condition de « non-orphelinité ». La suite présente la journalisation pessimiste, la journalisation optimiste et la journalisation causale.

1. Journalisation pessimiste

Le protocole de reprise par journalisation pessimiste [35] repose sur l’hypothèse (pessimiste) qu’une défaillance peut se produire immédiatement après un évènement non déterministe. Le principe de ce protocole est donc de ne pas permettre à un processus de dépendre d’un évènement non déterministe tant que celui-ci n’a pas été stocké sur un support stable.

Concrètement, si on considère que les évènements non déterministes sont uniquement les réceptions de messages, ce protocole impose aux processus de sauvegarder tous les messages reçus avant d’émettre un message à un autre processus. [37] Les sauvegardes effectuées doivent donc être réalisées de manière synchrone.

L’avantage de ce protocole est qu’il ne crée jamais de processus orphelin. Cependant, la sauvegarde synchrone induit un surcout conséquent lors d’une exécution sans panne, en particulier pour les applications effectuant beaucoup de communications.

2. Journalisation optimiste

La journalisation optimiste [35] veut améliorer les performances en faisant l’hypothèse (optimiste) qu’une panne ne se produira pas avant la sauvegarde de l’évènement non déterministe. Ainsi, la contrainte est relâchée et les sauvegardes peuvent être réalisées de manière asynchrone. Cependant, le désavantage de cette méthode est qu’elle ne garantit pas strictement la « condition de non-orphelinité ». Ainsi les processus qui n’ont pas encore sauvegardé leurs évènements non déterministes sont des processus orphelins. Pour obtenir un état global cohérent, ces processus devront revenir en arrière au dernier état où ils n’étaient pas orphelins. Il faut remarquer que ce calcul pour obtenir l’état global cohérent à la reprise peut avoir un cout important.

3. Journalisation causale

La journalisation causale [38, 31] combine les avantages de la journalisation pessimiste pour la reprise et les avantages de la journalisation optimiste en ce qui concerne le surcout à l’exécution. L’inconvénient est sa complexité. Le principe est de conserver en mémoire locale les informations permettant

de rejouer un évènement non déterministe mais également les informations de précédence (au sens de la relation de précédence causale de Lamport[40]) avec les autres évènements non déterministes [37]. Ces informations (appelées également le déterminant) sont aussi ajoutées à tous les messages émis vers les autres processus. Ces informations sont retirées de la mémoire locale des processus une fois qu'elles ont été enregistrées sur un support stable.

Ainsi, à tout moment, un processus connaît l'historique des évènements qui ont produit son état et celui des autres processus. Ces informations le protègent des défaillances des autres processus et permettent de garantir la condition de « non-orphelinité ». [38]

2.4.3 Comparaison des protocoles

La figure 2.7 propose une comparaison des avantages et des inconvénients de différentes techniques de tolérance aux fautes par reprise. Les critères utilisés sont les suivants.

- **Hypothèse PWD** : indique si cette technique repose sur l'hypothèse PWD. On remarque que seuls les protocoles par journalisation font cette hypothèse.
- **Processus orphelins** : indique si l'état correspondant à la dernière sauvegarde peut contenir des processus orphelins. Les processus orphelins peuvent être évités en utilisant plusieurs sauvegardes.
- **Effet domino** : indique s'il y a un risque d'effet domino au moment de la reprise. L'effet domino va obliger à conserver toutes les sauvegardes pour perdre le moins de calculs possibles. Les techniques par journalisation ne sont pas sensibles à l'effet domino puisqu'elles ont sauvegardé les messages pouvant en être à l'origine.
- **Nombre de sauvegardes** : donne le nombre de sauvegardes par processus à conserver pour redémarrer dans un état global cohérent. Ceci est la conséquence de la possibilité d'apparition de processus orphelins et de l'effet domino.

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

FIGURE 2.7 – Comparaison des différentes méthodes de tolérance aux fautes par reprise

2.4.4 Migration

La migration de processus est le déplacement d'un processus en cours d'exécution d'une machine source vers une machine destination, ces deux machines étant reliées par un réseau de communication et n'utilisant pas de mémoire partagée. La migration de processus d'une machine source vers une machine destination consiste à interrompre le processus qui s'exécute sur la machine source pour extraire son contexte d'exécution, à transférer ce contexte vers la machine destination, à créer, sur la machine destination, un nouveau processus auquel on affectera le contexte d'exécution transféré et à mettre à jour les liens de communication avec les autres processus. Une fois ceci fait, le processus sur la machine source doit être détruit tandis que le processus sur la machine destination est lancé et représente le processus déplacé. La Figure 2.8 illustre brièvement le mécanisme de migration de processus. Il existe différentes stratégies de migration de processus. Cette différence est due principalement au contexte d'exécution considéré et transféré lors de la migration. En effet, le contenu du contexte d'exécution transféré diffère d'un mécanisme à l'autre ou, plus exactement, d'un degré de migration à un autre.

Les techniques de virtualisation permettent notamment la migration de machines virtuelles d'un nœud physique à un autre.

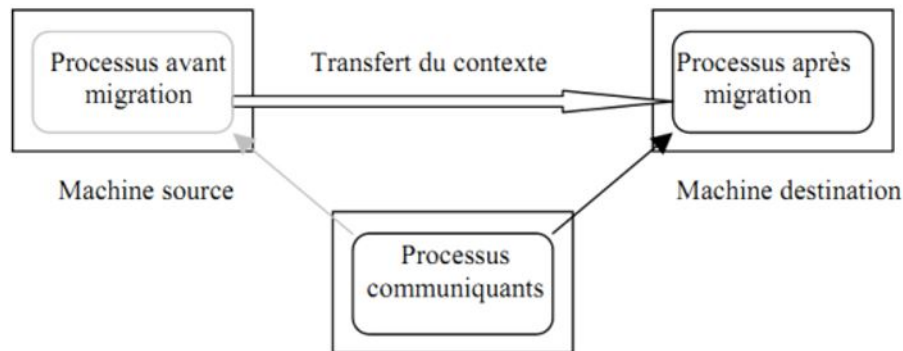


FIGURE 2.8 – Migration des processus

2.4.5 Tolérance aux pannes par duplication (Approche masquante)

Une approche est dite masquante si un processus qui ne subit pas de défaillance ne peut jamais s'apercevoir qu'une défaillance a eu lieu. De telles approches consistent habituellement à créer des répliques soit matérielles ou logicielles ou données.

Réplication

L'idée d'utiliser la redondance dans les systèmes informatiques pour masquer les défaillances des composants a été introduite par Von Neumann en 1956 [41]. Avec plusieurs répliques, une entité répliquée continue à fournir un service à un client même si une ou plusieurs répliques sont défaillantes.

La réplication met en œuvre un processus qui est chargé de la création, du placement et de la gestion de copies d'entités physiques et/ou logicielles. Les entités répliquées peuvent être des données, du code, des objets, des composants physiques ou une combinaison de tous ces éléments.

La création des copies ou répliques d'une entité consiste à reproduire la structure et l'état des entités répliquées. La copie d'un fichier est un autre fichier de même contenu. La copie d'un programme est un autre programme qui exécute le même code et dont l'état d'exécution est celui du programme initial[42].

L'intérêt premier de cette réplication est que, si une donnée n'est plus disponible, le système peut continuer à assurer ses fonctionnalités en utilisant une donnée répliquée, ce qui permet d'augmenter la disponibilité des données et la tolérance aux pannes.

Granularité de réplication

Les protocoles de réplication considèrent principalement quatre types d'entités : les fichiers, les zones mémoires, les bases de données et les objets [43].

- **Fichiers** : dans le cas des fichiers, les techniques de copie se basent sur la structure séquentielle des fichiers et sur leur organisation hiérarchique en répertoires. La réplication garantit la tolérance aux pannes dans des systèmes et elle permet l'accès rapide aux fichiers. Plusieurs travaux se sont intéressés 'a cette granularité dans les système sà grande échelle.
- **Zones mémoire** : la réplication de la mémoire est appliquée dans les travaux sur les mémoires partagées réparties avec l'objectif de fournir l'abstraction d'un environnement d'exécution centralisée. Comme dans le cas des fichiers, la cohérence est gérée par rapport aux accès en lecture et en écriture et la copie est définie en fonction des structures hiérarchiques de blocs et de pages.
- **Bases de données** : comme les mémoires et les fichiers, les bases de données représentent un support essentiel pour le stockage de données et utilisent la duplication pour les tolérances aux pannes, les performances d'accès, le travail des usagers mobiles en mode déconnecté,etc. Toutefois, la possibilité de définir différents types de données impose l'utilisation de techniques de copie plus complexes qui prennent en compte la spécificité des données.
- **Objets** : étant une abstraction qui permet la modélisation de tout type d'entité, les objets sont largement utilisés et répliqués pour les besoins de tolérance aux pannes, de performance d'accès et de mise à l'échelle.

Les techniques de réplication

Ranganathan et Foster définissent les quatre questions auxquelles une stratégie de création de répliques doit répondre[44].

- Quand créer les répliques ? moment de la réplication.
- Quels fichiers doivent être répliqués ? choix de l'entité à répliquer.
- Où les répliques doivent-elles être placées ? placement des répliques.
- Comment une copie est-elle créée ? manière de répliquer une entité.

1. Moment de la réplication

Pour répondre à la question quand ? deux solutions sont possibles [45] :

- (a) **Réplication statique** : les répliques persistent jusqu'à ce qu'elles soient effacées par l'utilisateur du nœud sur lequel elles sont hébergées ou que leurs durées de vie respective expirent. L'avantage de ce schéma est sa simplicité, son inconvénient est sa non-adaptabilité aux changements de comportement des participants.
- (b) **Réplication dynamique** : contrairement à la réplication statique, la réplication dynamique crée et supprime automatiquement les copies selon l'évolution des demandes des utilisateurs. L'avantage est la réduction des points d'engorgements et l'équilibrage de la charge. L'inconvénient observé est l'induction de coûts supplémentaires causés par l'évaluation en temps-réel du trafic réseau pour prendre les décisions de réplication. Selon le moment de la réplication, on distingue :
 - I. **Réplication à la demande** : la réplique est créée suite à la demande d'un client.
 - II. **Réplication périodique** : elle est indépendante des requêtes des clients. Son but est de permettre la gestion automatique de répliques avec des stratégies adaptées aux comportements des clients. Le processus de réplication est déclenché à chaque intervalle de temps (période).

2. Choix de l'entité à répliquer

Pour répondre à la question quoi : les données répliquées sont généralement de deux types : des fichiers ou des objets. Les objets peuvent être composés d'un ensemble de fichiers distribués (on les appelle aussi collection). Selon les stratégies de réplication, les données à répliquer, peuvent être les plus populaires ou encore les plus fréquemment accédées.

3. Placement des répliques

Pour répondre à la question où : les stratégies de placement de répliques doivent tenir compte du fait que les sites potentiels : a) ne possèdent pas déjà de réplique de la donnée ; b) possèdent l'espace de stockage suffisant ; c) sont à une distance raisonnable en termes de temps de transfert.

4. Manière de répliquer une entité

Pour répondre à la question comment : Le processus de création de copie

dépend de la structure et de l'état de l'entité à répliquer.

La structure de l'entité peut être indivisible ou composée, alors que l'état peut être constitué de données, de code et éventuellement d'un état d'exécution.

Les problèmes de coûts sont au centre des stratégies de réplication. Un enjeu majeur de la réplication est la réduction de la latence d'accès ainsi que la consommation de bande passante.

Avantages et inconvénients de la réplication

Avantages

L'utilisation de la technique de réplication procure un certain nombre d'avantages que nous pouvons résumer comme suit :

- Permettre un parallélisme dans la consultation de la même donnée ;
- Améliorer la tolérance aux pannes : la réplication permet les accès aux données même en cas de défaillance d'un support puisque la donnée se trouve sur plusieurs endroits ;
- Améliorer les performances : La réplication permet d'améliorer le temps de réponse des requêtes et l'accès aux données pour deux raisons essentielles :
 - (i) Les requêtes sont traitées sur un serveur local sans accès à un réseau étendu qui nécessite de la communication.
 - (ii) Le traitement local allège la charge globale des serveurs.

Inconvénients

- **Placement des répliques** : Ce problème consiste à choisir, en fonction des objectifs des applications et de la réplication, des localisations physiques pour les répliques, qui réduisent les coûts de stockage et d'accès aux données.
- **Choix d'une réplique** : Il s'agit ici de sélectionner, parmi toutes les répliques d'une donnée, celle qui est la meilleure du point de vue de la consistance.
- **Cohérence des répliques** : les techniques de réplication n'assurent pas une cohérence des données de l'ensemble des répliques. Ainsi, il est possible

d'avoir, à un instant donné, des copies différentes d'un même ensemble de données sur différents nœuds.

Importance de la haute disponibilité dans le Cloud Computing

La disponibilité d'un service est donnée par la quantité de temps pendant laquelle le service est utilisé par les clients dans des conditions normales et anormales. La haute disponibilité découle du fait que ses clients ont besoin d'un accès permanent au service [46].

L'indisponibilité de certains services a un impact négatif pour leurs clients, c'est le cas des institutions bancaires, des entreprises de télécommunications, des applications militaires ou des hôpitaux. Les infrastructures Cloud doivent offrir une disponibilité supérieure à 99,9% ; Donc la dégradation des performances est une préoccupation plus grave que les défaillances de ressources dans de tels environnements [47].

L'augmentation de la demande pour la disponibilité continue des systèmes de calcul haute performance (HPC : High-performance computing system) est évidente. C'est une étape majeure vers l'informatique de capacité, où les applications scientifiques demandent des quantités considérables de temps (semaines et mois) sans interruption sur les machines HPC disponibles les plus rapides. [48].

2.5 Protocole de réplication

Trois principaux protocoles sont utilisés pour la gestion des répliques dans les systèmes distribués :

2.5.1 Protocole de réplication passive

Dans ce protocole, une seule copie reçoit une requête d'un client et l'exécute. Cette copie est désignée sous le nom de copie primaire (primary copy). Elle a la tâche d'effectuer tous les traitements, alors que les copies secondaires ne font aucune action (voir Figure). En cas de défaillance de la copie primaire, une copie secondaire devient (par un protocole d'élection) la nouvelle copie primaire. Pour assurer la cohérence, la copie primaire diffuse régulièrement son nouvel état à toutes les copies secondaires [44].

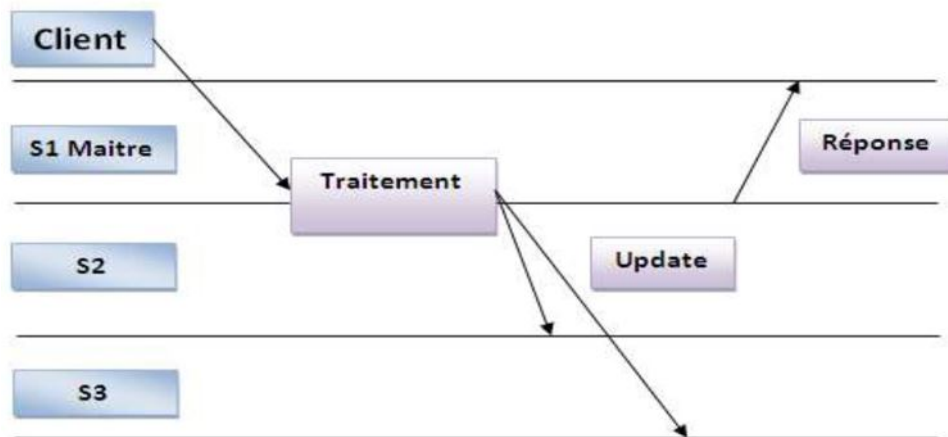


FIGURE 2.9 – protocole de réplication passive

2.5.2 Protocole de réplication active

Dans un protocole de réplication active, chaque copie joue un rôle identique à celui des autres copies. Toutes les copies reçoivent la même séquence, totalement ordonnée, des requêtes des clients, les exécutent puis renvoient la même séquence, totalement ordonnée, des réponses [33].

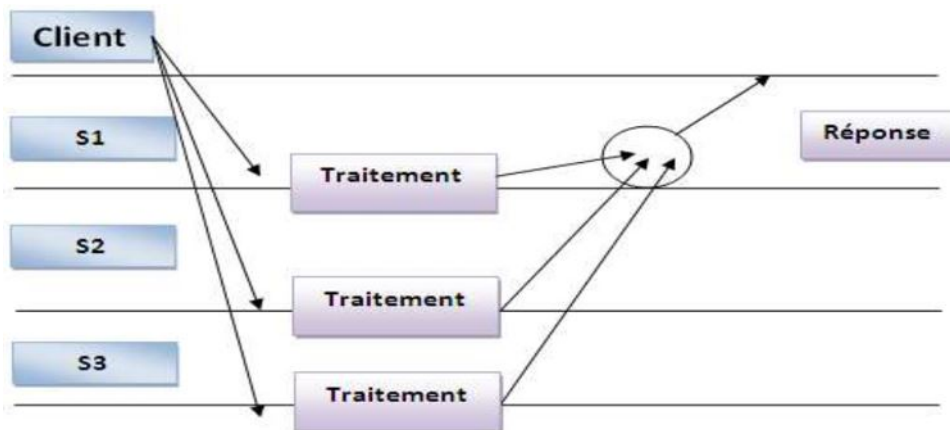


FIGURE 2.10 – Protocole de réplication active

2.5.3 Protocole de réplication semi-active

C'est un protocole hybride qui se situe entre les deux protocoles précédents, où toutes les copies exécutent en même temps la requête du client, mais une seule copie (leader) d'entre elles émet la réponse, les autres copies (suiveurs) mettent à jour leur état interne et sont donc étroitement synchronisées avec le leader.

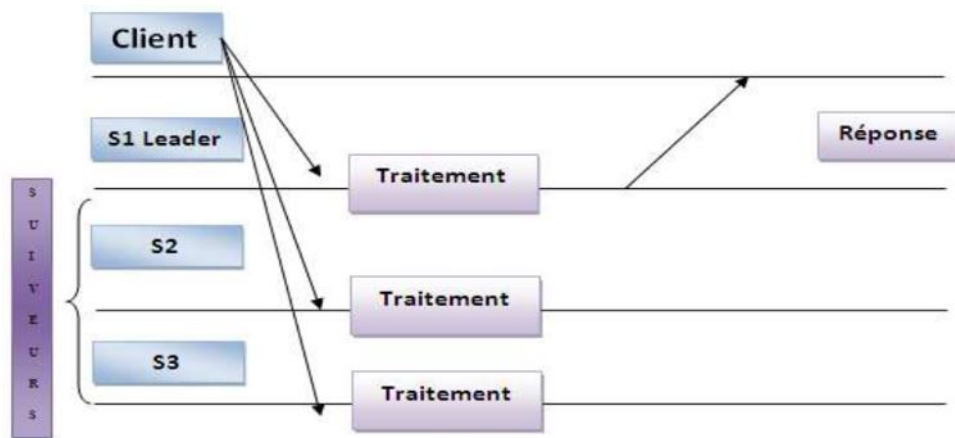


FIGURE 2.11 – Protocole de réplication semi-active

2.6 La tolérance aux pannes dans les cloud

Le Cloud Computing et la Virtualisation ont ouvert une nouvelle fenêtre dans la gestion des pannes. La mise en pause, la reprise, et la migration des machines virtuelles (VMs) sont des mécanismes puissants pour gérer les défaillances dans un tel environnement. Une machine virtuelle peut être facilement migrée vers un autre nœud lorsqu'une panne est inspectée ou détectée. Bien que la migration soit une bonne solution afin de traiter les pannes, elle a deux problèmes principaux. Premièrement, elle a des surcoûts considérables, particulièrement si les images de VMs entières doivent être émigrées. Deuxièmement, la prévision et la détection à l'avance des pannes est un problème majeur. Nous ne pouvons pas commencer la migration d'une VM si le nœud dont lequel elle s'exécute est défaillant. Jing Deng et al [50]. Proposent des techniques pour améliorer la tolérance aux pannes et la fiabilité d'un calcul scientifique assez général : multiplication matricielle. La multiplication de matrices sert comme une base pour la résolution et l'optimisation de nombreux problèmes complexes.

Ils étudient une stratégie de sélection de Cloud pour décomposer le problème de multiplication de matrice en plusieurs tâches qui seront soumises à différents Cloud et ils montrent que la tolérance aux pannes et la fiabilité par rapport à la défaillance dans le Cloud Computing peuvent être réalisés. Dans les Cloud de stockage, Bonvin et al. [51] Proposent une Clé-Valeur autogérée (self-managed Key-value) qui alloue dynamiquement les ressources d'un Cloud de données pour plusieurs applications dans une manière efficace et équitable. L'approche proposée offre et maintient dynamiquement des garanties différenciées multiples de

disponibilité pour chaque application différente en présence des pannes. Les auteurs utilisent une économie virtuelle, ou chaque partition de données (c'est à dire une gamme clé dans un espace cohérent de hachage) agit comme un optimiseur individuel et choisit s'il faut migrer, répliquer ou de se supprimer, basée sur la maximisation des bénéfices nets concernant l'utilité offerte par la partition et son stockage et les couts de maintenance. Zibin Zheng et al .[51] Proposent FTCloud qui est un Framework à base de classement de composants pour construire des applications de Cloud tolérantes aux pannes. FTCloud utilise les structures d'invocation de composants et les fréquences d'invocation pour identifier les éléments importants dans une application de Cloud. Un algorithme est proposé afin de déterminer automatiquement la stratégie optimale de tolérance aux pannes pour ces composants significatifs.

2.7 L'équilibrage de charge

L'équilibrage de charge est un élément important lors de la mise en place de services amenés à croître. Il faut s'assurer que la capacité à monter en charge soit la plus optimale possible afin d'éviter toute dégradation que cessoit en termes de performances ou de fiabilité lors d'affluences importantes.

Le principe de base de l'équilibrage de charge (*Load Balancing*) consiste à effectuer une distribution des tâches à des machines de façon intelligente. Les objectifs du L'équilibrage de charge sont comme suit :

- Amélioration de temps de réponse des services.
- Capacité à pallier la défaillance d'une ou de plusieurs machines.

2.7.1 Problème de l'équilibrage de charge

Le problème de l'équilibrage étant un problème relativement ancien, beaucoup d'approches ont été proposées pour le résoudre. Casavant et Kuhlont défini une taxonomie largement adoptée par la communauté scientifique dont les principales classes sont [52].

1. Approche Statique Vs. Approche Dynamique :

- **Dans une approche statique** : Les tâches sont assignées aux machines avant l'exécution de l'application qui les contient. Les informations concernant le temps d'exécution des tâches et les caractéristiques

dynamiques des machines sont supposées connues a priori. Cette approche est efficace et simple à mettre en œuvre lorsque la charge de travail est au préalable suffisamment bien caractérisée.

- **Dans une approche dynamique :** L'assignation des tâches aux machines se décide durant la phase d'exécution, en fonction des informations qui sont collectées sur l'état de charge du système. Ceci permet d'améliorer les performances d'exécution des tâches mais au prix d'une complexité dans la mise en œuvre de cette stratégie, notamment en ce qui concerne la définition de l'état de charge du système, qui doit se faire de manière continue.

2. Approche Centralisée Vs. Approche Distribuée :

Dans une approche centralisée, un site du système est choisi comme coordinateur. Il reçoit les informations de charge de tous les autres sites qu'il assemble pour obtenir l'état de charge global du système.

Dans le cas d'une approche distribuée, chaque site du système est responsable de collecter les informations de charge sur les autres sites et de les rassembler pour obtenir l'état global du système. Les décisions de placement de tâches sont prises localement, étant donné que tous les sites ont la même perception de la charge globale du système.

3. Approche Source-Initiative Vs. Receveur-Initiative :

L'approche source- initiative est appliquée lorsqu'un site, appelé source, détecte qu'il a une surcharge de travail et qu'il cherche à transférer le surplus vers un site faiblement chargé. L'approche receveur initiative s'applique lorsqu'un site faiblement chargé, appelé receveur, demande à recevoir tout ou partie du surplus des sites surchargés.

2.7.2 Le système d'équilibrage de charge

Un système d'équilibrage de charge est composé de deux éléments essentiels politiques et mécanismes [52]. Les politiques considèrent l'ensemble des choix à effectuer pour distribuer une charge de travail alors que les mécanismes réalisent physiquement la répartition de la charge et fournissent les informations exigées par les politiques.

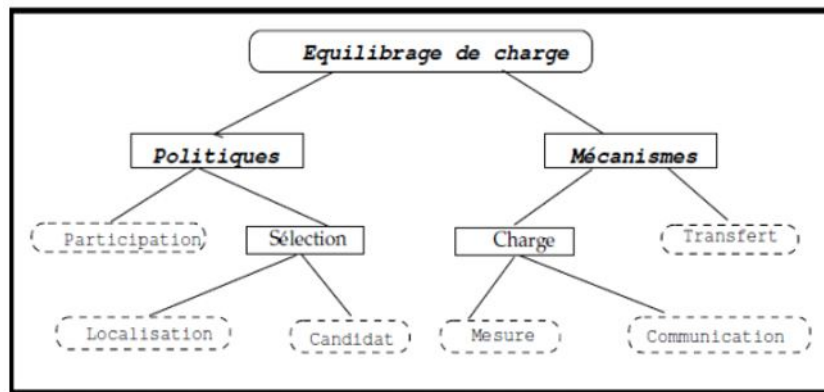


FIGURE 2.12 – Composants d'un système d'équilibrage de charge

Les Politiques

1. **Politique de participation** : Le but de cette politique consiste à déterminer si un site est dans un état approprié pour participer à un transfert de tâches comme source (site surchargé) ou comme receveur (site sous-chargé).
2. **Politique de sélection de la localisation** : Cette politique est responsable de trouver, pour un site donné, un partenaire (source ou receveur), une fois que la politique de participation a décidé que ce site était soit source, soit receveur.
3. **Politique de sélection des tâches à transférer** : Une fois que les politiques de participation et de localisation ont décidé qu'un site A est source et qu'un autre site B est receveur, cette politique est responsable du choix des tâches à transférer de A vers B.

Les Mécanismes

1. **Mécanisme de mesure de la charge** : Dans toute approche d'équilibrage de charge, une des difficultés majeures est celle qui consiste à évaluer la mesure de la charge d'un site. Dans la plupart des travaux existants, c'est la longueur de la file d'attente qui détermine la charge d'un site. Certains auteurs [52-36] préconisent comme indicateur de charge, une combinaison entre la longueur de la file d'attente CPU, celle des Entrées/Sorties et l'occupation mémoire. Dans le cas des grilles de calcul, il est nécessaire de tenir compte aussi de l'hétérogénéité des ressources et des réseaux de communication pour mesurer la charge d'un site.

2. **Mécanisme de définition de la charge** : Ce mécanisme essaie de définir la charge globale d'un système en collectant les informations de charge sur l'ensemble ou une partie des sites du système. Il faudra alors définir les méthodes selon lesquelles l'information de charge est collectée puis diffusée aux sites.

2.7.3 Les algorithmes d'équilibrage de charge dans le Cloud Computing

Les algorithmes d'équilibrage de charge servent à distribuer les requêtes des utilisateurs qu'arrivent à un centre de données, entre les machines virtuelles selon des techniques propres pour chaque algorithme. Généralement, on a deux types : statique et dynamique.

Algorithmes statiques :

1. **Round Robin Load Balancer** : Les requêtes dans cet algorithme sont distribuées entre les machines virtuelles à tour de rôle à l'aide d'un contrôleur de centre de données.
L'ordre d'allocation des tâches s'effectue dans chaque machine virtuelle localement et indépendamment de l'autre machine distante, sur la base du nombre des tâches et le nombre des machines virtuelles existants[53].
2. **Throttled Load** : Cet algorithme collecte l'état de la machine virtuelle ; si elle est valable ou non pour la nouvelle allocation. Ensuite, l'algorithme envoie l'identifiant(ID) de la machine virtuelle valable au contrôleur de centre de données pour une nouvelle allocation et exécution de la tâche. Si le traitement de la tâche est terminé, la machine virtuelle envoie le résultat au contrôleur de centre de données, et ce dernier notifie l'algorithme pour la désallocation [54].
3. **Central Manager Algorithme** : Le gestionnaire central des charges, affecte les tâches à La machine virtuelle avec une charge minimale par rapport aux autres machines dans chaque nouvelle allocation. Le gestionnaire, met à jour, de temps en temps, le statut de la charge du système lorsqu'il subit des changements. Ce statut permet de prendre la bonne décision d'équilibrage de charge, lors de la création des tâches par le centre de données [53].

Parmi Les inconvénients de ces algorithmes, c'est qu'ils ne prennent pas en considération la charge actuelle de la machine virtuelle en cours d'exécution.

Algorithmes dynamiques :

1. **Efficient Response Time Load Balancer** : Cet algorithme est basé sur le temps de réponse inférieur, pour l'allocation de la machine virtuelle par le centre de données. Premièrement « Efficient Response Time Load Balancer » détecte la machine avec un temps de réponse inférieur. Deuxièmement, il rend l'ID de la machine au contrôleur de centre de données pour l'allocation. Troisièmement, Lorsque la tâche est finie, le contrôleur notifie l'algorithme pour faire la mise à jour de la table d'allocation [49].
2. **Central queue algorithm** : Le contrôleur de centre de données à une file d'attente centrale dans laquelle les tâches sont classifiées par ordre FIFO (first in first out). Si une machine virtuelle passe dans le statut «underload», elle envoie une demande d'une nouvelle allocation de tâche au contrôleur de centre de données. Ce dernier supprime la tâche de la file d'attente, puis il l'envoie directement vers la machine concernée [49].
3. **Local queue algorithm** : À l'aide de cet algorithme, toutes les machines virtuelles auront des files d'attente locales. Lorsque ces machines passent en mode underload, le gestionnaire de charge local, cherche autres tâches depuis les autres machines virtuelles distantes. L'avantage de cet algorithme est la migration dynamique et l'allocation de toutes les tâches chargées dans le contrôleur de centre de données, vers les machines virtuelles [39].

2.8 Conclusion

Dans ce chapitre, nous avons dressé un état de l'art sur la sûreté de fonctionnement, et nous avons donné un aperçu général sur les différentes notions de tolérance aux pannes dans les systèmes distribués et le Cloud Computing.

Nous avons montré aussi les principaux concepts et caractéristiques liés à la tolérance aux pannes, ainsi que les techniques utilisées En particulier la réplication.

Enfin Nous avons présenté les différents algorithmes d'équilibrage de charges dans les environnements « Cloud Computing ».

Notre but dans ce travail est de garantir la tolérance aux pannes et en même

temps assurer l'équilibrage de charge dans le Cloud. Pour cela nous proposons dans le prochain chapitre une stratégie de tolérance aux pannes basée sur la réplication des données qui permet de garantir un bon fonctionnement du système en cas des pannes.

CHAPITRE 3

NOTRE CONTRIBUTION

Contents

3.1	Introduction	53
3.2	Objectif du travail	53
3.3	Description de l'approche proposée	54
3.3.1	Topologie de cloud	54
3.3.2	Modèle proposé	55
3.4	Décision de réplication (quand répliquer)	55
3.5	Degré de réplication (combien de répliques)	59
3.6	Placement des répliques	60
3.7	L'équilibrage de charge	60
3.8	Détection des pannes	61
3.9	Conclusion	62

3.1 Introduction

Ce chapitre est consacré à la conception détaillée de notre stratégie de tolérance aux fautes et l'équilibrage de charge. Nous commençons par décrire le modèle proposé. Ce modèle est basé sur le mécanisme de réplication réalisé dans notre approche. Nous détaillerons également les caractéristiques de notre approche et les algorithmes nécessaires pour son fonctionnement.

3.2 Objectif du travail

Le Cloud Computing exige la gestion d'un nombre massif de données afin de les rendre accessibles aux clients qui les réclament. Mais la panne d'un composant du system cloud peut causer une indisponibilité de ces données. Notre objectif est de garantir la fiabilité du system cloud même en présence des pannes.

La stratégie de réplication des données permet d'assurer une disponibilité élevée de données et une haute performance. Elle est utilisée aussi afin de gérer la tolérance en panne.

Les stratégies de réplication de données doivent répondre à des questions essentielles telles que :

1. comment assurer une certaine disponibilité de données pour permettent la continuation de service en cas des pannes. En effet l'augmentation du nombre de répliques conduit à accroître la disponibilité des données et la fiabilité, mais l'espace de stockage sera également augmenté et peut devenir insuffisant.
2. Où la réplique doit être placée? Placer les nouvelles répliques sur le site de l'emplacement approprié permet d'assurer un bon équilibrage de charge entre les sites de cloud, aussi peut favoriser la réduction de la consommation de bande passante du réseau et réduit donc le temps de réponse. L'idée principale est de conserver les données à proximité de l'utilisateur afin de rendre l'accès efficace et rapide. Mais le comportement dynamique des utilisateurs du réseau rend plus difficile la décision sur le placement des nouvelles répliques. La stratégie de réplication est guidée par des facteurs tels que la demande des données, les conditions du réseau, le coût de transfert et le coût de stockage. Une stratégie de placement des répliques est très importante afin de maximiser les gains de la réplication des données.

Dans ce chapitre, nous proposons une stratégie hybride pour gérer la tolérance aux pannes des données et d'assurer un bon équilibrage de charge pour les environnements du Cloud, notre stratégie permet de d'assurer les qualités de services exigées par les utilisateurs et de garantir un profit du fournisseur simultanément. Dans notre stratégie, nous estimons le temps de réponse moyen des requêtes, si le temps de réponse estimé n'est pas acceptable le goulot d'étranglement est identifié (donnée). Nous utilisons la réplication des données pour éliminer ces goulots d'étranglement.

3.3 Description de l'approche proposée

3.3.1 Topologie de cloud

Les fournisseurs de cloud établissent plusieurs installations dans des régions géographiques distinctes pour une multitude de raisons y compris la fourniture de services qui couvrent le monde. Chaque région peut contenir plusieurs sous-régions, Ces sous-régions sont des installations cloud qui hébergent un certain nombre de nœuds qui fournissent une puissance de calcul et de stockage aux locataires.

Le système de cloud que nous considérons se compose de régions géographiques avec chaque région contenant un certain nombre de centres de données (Datacenters) où chaque centre de données contient à son tour un certain nombre de serveurs i.e des machines virtuelles (VM) qui résident sur des hôtes physiques (Hosts). Chaque serveur est équipé de ressources de calcul e.g CPU, réseau, stockage, pour contribuer lors de l'exécution de la requête.

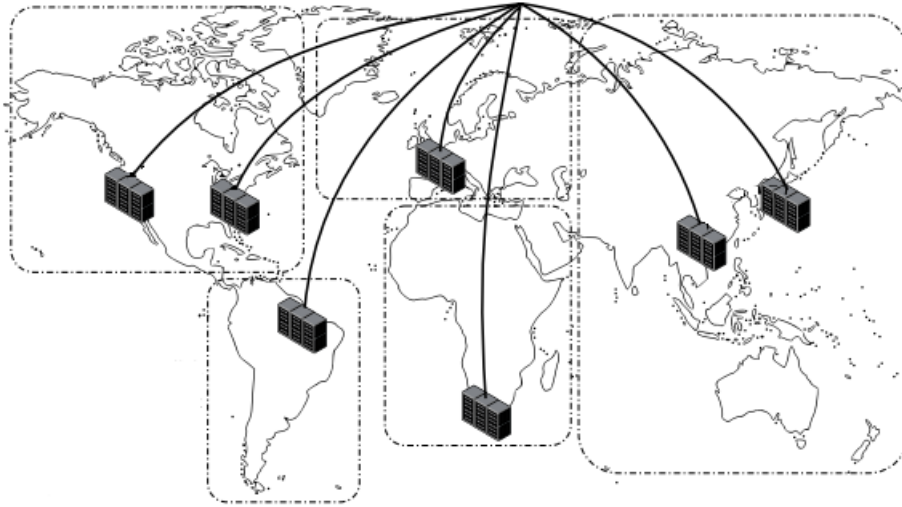


FIGURE 3.1 – Topologie de cloud

3.3.2 Modèle proposé

La stratégie de tolérance aux fautes proposée est basée sur deux modèles. Le premier modèle est basé sur une estimation du temps de réponse et le deuxième modèle est un modèle économique basé sur le coût de réplication, il est utilisé pour estimer le profit du fournisseur.

Toute stratégie de réplication de données doit répondre à quelques questions lors du processus de décision de réplication. Certaines de ces préoccupations ont été décrites dans de nombreux systèmes traditionnels telles que les grilles de données. Nous devons traiter les questions suivantes : quelles sont les données à répliquer, quand répliquer et où placer les répliques afin de constituer une stratégie complète qui convient à la gestion des données pour satisfaire les objectifs souhaités.

Cependant, dans les systèmes Cloud, ces décisions doivent être prises dans une perspective rentable, Nous étendons donc ces considérations dans le processus de réplication de données pour prendre en compte le point de vue du locataire et du fournisseur.

3.4 Décision de réplication (quand répliquer)

Tout d'abord la décision la plus fondamentale dans notre stratégie de tolérance aux pannes est de savoir si on effectue la réplication de données ou non, La décision

de réplication dépend de l'accomplissement des critères suivants : 1) d'assurer une fiabilité du system Cloud en cas de présence des pannes ; 2) Si le temps de réponse moyen estimé des requêtes est supérieur au temps de réponse convenu dans le SLA et 3) le traitement de ces requêtes peut générer un profit minimum pour le fournisseur, tout en prenant en compte le coût de création des nouvelles répliques.

Les différentes étapes de la décision de réplication sont décrites dans algorithme 1.

budgetR : budget de la requête Q.

coutServ : le cout de service.

coutServAPM : le cout de service + le profit minimum attendu.

Pg (Profit Global) : la somme des profits qui sont supérieurs au profit minimum attendu.

Lorsqu'un locataire envoie des requêtes au cloud pour les traiter, notre stratégie prédit si l'exécution de ces requêtes peut satisfaire l'objectif de performance par l'estimation du temps de réponse moyen des requêtes.

SLA :

Les exigences de la qualité de service sont extrêmement importantes pour le Cloud computing. Elles sont généralement formalisées sous forme de SLA. 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.

Algorithme 01 : Algorithme de création des répliques

```

arrivée des requêtes
T = Temps de réponse moyen estimé
if T > Tsla Then
    sélectionner la donnée à répliquer
    calculer le nombre des répliques NR
    P_moyen = le profit moyen estimé après l'exécution des requêtes avec les nouvelles répliques
    if P_moyen > P_min then          /* P_min : le profit minimum attendu */
        placer les répliques
        exécuter les requêtes
    else
        for chaque requête Q
            if si budgetR > coutServAPM then
                ajouter Q a la liste L
                Pg = Pg + (budgetR - coutServAPM)
            else
                ajouter Q a la liste d'attente
        end for
        calculer le nombre des répliques NR pour les requêtes dans L
        placer les répliques
        exécuter les requêtes
        if la liste d'attente != vide et Pg > 0 then
            Pg = Pg * 0.40 ;
            trier la liste par ordre décroissant selon le budgetR
            for chaque requête Q dans la liste d'attente
                manque = coutServ - budgetR
                if Pg > manque then
                    exécuter la requête Q
                    Pg = Pg - manque
                else
                    refuser la requête Q
            end for
        end for
    else
        exécuter les requêtes

```

Estimation du temps de réponse moyen :

Les requêtes envoyées par l'utilisateur sont exécutées par les machines virtuelles, ces requêtes ont besoins des données pour terminer leurs exécutions. Le temps de réponse de chaque requête est estimé par la formule suivante :

$$\text{Temps de réponse} = T1 + T2 + T3$$

T1 : est le temps d'attente.

T2 : est le temps d'exécution.

T3 : est le temps d'accès au réplique.

Le temps d'attente : Pour déterminer le temps d'attente moyen d'une requête, nous devons déterminer en premier temps le nombre d'accès autorisé simultanément à une réplique (NA). Généralement ce nombre est déterminé par le fournisseur de cloud, puis nous devons calculer le nombre de répliques minimum (NR) de cette donnée dans le Datacenter afin d'assurer les exigences de QoS.

Pour connaître le nombre des requêtes qui peuvent accéder aux répliques sans attendre (NRSA) nous utilisons la formule suivante :

$$NRSA = NR * NA$$

Le temps d'attente moyenne d'une requête est le temps moyen restant pour libérer les ressources (répliques) à partir des requêtes qui ont accès aux répliques sans attendre.

Le temps d'exécution :

Le temps d'exécution d'une requête est estimé par la formule suivante :

$$tempsd'exécution = \frac{tailledecloudlet}{lavitesse}$$

Où la taille de Cloudlet représente le nombre d'instruction de la requête Et la vitesse, c'est la vitesse d'exécution de la machine virtuelle donnée, elle est exprimée en MIPS.

Le temps d'accès : Le temps d'utilisation de la donnée.

Si le temps de réponse est inférieur au temps convenu dans le SLA, c'est à dire la qualité de service est vérifiée avec les ressources actuelles. Donc, pas besoin de créer des nouvelles répliques. Par contre, si le temps de réponse des requêtes dépasse le temps défini dans le SLA, dans ce cas, la qualité de service n'est pas assurée avec le nombre des répliques actuelles et donc nous avons une violation de SLA, dans ce cas, le processus de la réplication est déclenché afin d'augmenter la disponibilité et d'assurer les performances souhaitées.

Le deuxième critère important à vérifier dans le processus de réplication est la rentabilité du fournisseur. Ce dernier s'attend à générer un profit souhaité en exécutant les requêtes des clients. Le traitement des requêtes doit toujours être rentable avec l'inclusion des coûts associés à la création des nouvelles répliques.

Seule la satisfaction simultanée des deux critères mentionnés déclencherait la réplication.

Tandis que les fournisseurs visent à accroître ses bénéfices, les locataires exigent

une bonne performance à un prix raisonnable. Dans notre stratégie, nous visons à équilibrer entre les profits de fournisseur et les dépenses de locataire. Si l'exécution des requêtes de ces locataires est rentable pour le fournisseur alors les requêtes seront exécutées. Sinon, nous allons exécuter seulement les requêtes qui génèrent un profit et les autres requêtes seront mises en attente. Le profit retourner après l'exécution des requêtes doit être égal ou supérieur au profit minimum attendu P_{\min} . Dans le cas où le profit généré est supérieur à P_{\min} , nous ajoutons la différence entre le profit généré et P_{\min} au profit global (P_g).

Comme mentionné plus tôt, notre stratégie vise à minimiser les dépenses de locataire, Le profit collecté ne va pas tout à la poche du fournisseur mais un certain pourcentage (exemple 40% dans notre algorithme) Sera utilisé pour le bénéfice de l'utilisateur.

Dans notre stratégie, les requêtes qui sont en attente seront triées par ordre décroissant selon le budget. Nous favorisons les requêtes qui ont un budget grand afin de maximiser le nombre de requêtes exécutées. Une requête sera exécutée si son déficit est inférieur au profit collecté. Si le déficit est supérieur au profit collecté, cette requête sera refusée.

3.5 Degré de réplication (combien de répliques)

Déterminer le nombre de répliques pour chaque donnée est un aspect intéressant de la réplication de données, Trop de répliques peuvent entraîner un gaspillage de ressources tandis que trop peu de répliques ne suffisent pas à satisfaire le niveau de fiabilité attendu en cas des pannes. Notre stratégie de tolérance aux pannes calcule le nombre de répliques nécessaire pour assurer une certaine disponibilité de données et augmente la fiabilité du Cloud.

L'idée générale est de créer de nouvelles répliques à l'arrivée des requêtes. Cette création vérifie certaine condition comme le budget, le coût de réplication. Le nombre de répliques (NR) nécessaire est calculé comme suit :

NR = nombre de requêtes / nombre d'accès simultané à une réplique

Après l'estimation du nombre de répliques (NR), si la création de ces répliques génère un profit donc l'étape suivante consiste à placer les nouvelles répliques par un algorithme de placement. Sinon on exécute que les requêtes qui donne un profit et on calcule le nouveau nombre de répliques nécessaire à ces requêtes par la formule précédente.

Une fois cette étape terminée, on procède au placement des répliques.

3.6 Placement des répliques

Les deux modèles d'estimation permettent d'identifier les goulots d'étranglement des ressources pour déterminer ce qu'il faut répliquer et décider également quand la réplication est nécessaire. Un placement stratégique des répliques permet d'atteindre l'objectif de l'équilibrage de charge.

Algorithme 02 : Placement des répliques

```

Replica Placement (Datacenter , replica)
begin
  liste = vide
  For All hosts  $H \in \text{Datacenter}$  do
    if  $\text{freespace}(H) \geq \text{sizeof}(\text{replica})$  and  $\text{load}(H) < \text{Seuil}_{\text{load}}$  then
      add  $H$  to liste
    end if
  end for
  if  $\text{sizeof}(\text{liste}) = 0$  then
    Replica Placement(nextDatacenter , replica)
  else
    Target = premier Host dans la liste
    load = load (Target)
    For All Host  $H \in \text{liste}$  do
      if  $\text{load}(H) < \text{load}$ 
        Target =  $H$ 
        load = load( $H$ )
      end if
    end for
    Placer la réplique dans Target
  end if
end

```

3.7 L'équilibrage de charge

Le deuxième objectif de notre stratégie consiste à effectuer une distribution des requêtes à des machines de façon intelligente pour améliorer l'utilisation des ressources. le problème de placement des répliques consiste à choisir des localisations physiques pour les répliques afin de réduire les coûts d'accès aux données et d'augmenter la performance. Le serveur candidat pour placer la nouvelle réplique

doit avoir une charge faible, suffisamment d'espace de stockage et une bande passante réseau suffisante pour garantir un meilleur service aux demandeurs.

Après avoir calculer le nombre de répliques approprié, les nouvelles répliques doivent être placées sur des sites qui ont un degré de fiabilité élevé. Pour cela, ces répliques seront placées sur des sites moins chargé et qui ont une probabilité de défaillance faible.

Pour trouver le meilleur emplacement, tous les serveurs du Cloud doivent être évalués en termes de charge, de stockage et de ressources réseau. Notre stratégie cherche les serveurs qui ont une capacité de stockage supérieur à la taille de la réplique, parmi ces candidats on place la réplique dans le host le moins chargé avec un degré de fiabilité élevé.

3.8 Détection des pannes

La détection des pannes dans l'approche proposée est réalisée par l'envoi des messages de vie (MV). Dans chaque Datacenter, on configure un processus responsable de la surveillance. Ce dernier envoie régulièrement des messages de vie à tous les nœuds. L'agent attends les messages de vie de chaque nœud N. Si le message de vie d'un nœud N n'arrive pas après un certain temps, le nœud N sera déclaré en panne et sera ajouté à la liste des nœuds en pannes.

Les étapes de détection de pannes sont décrites dans algorithme 3, nous utilisons les notations suivantes :

- **MV** : le message de vie
- **ListPan** : liste des nœuds en pannes
- **RécepMV(N)** : est une valeur booléenne qui indique si le MV de nœud N est reçu ou pas.

L'algorithme est constitué de deux parties. La première partie consiste à envoyer des messages de vie a chaque période. La deuxième partie consiste à attendre les messages de vie des nœuds.

Lorsqu'une défaillance d'un nœud est détectée, notre stratégie récupère la liste des machines virtuelles et la listes des tâches (Cloudlets) qui ont été exécutés sur le nœud défaillant et les redémarre. Dans notre stratégie, nous avons défini deux états pour un Datacenter :

Algorithme 03 : Détection des pannes

```

for all nœud N do
    Send MV TO N
end for
Wait(delai)
for all nœud N do
    if RécepMV(N) == false then
        ListPan ← ListPan + N
    end if
end for

```

- **État normal** : le Datacenter est moins chargé et le nombre des nœuds défaillant est faible.
- **État chargé** : il n'existe pas suffisamment de ressources, ou bien le nombre des nœuds défaillant est très important.

Pour redémarrer les tâches qui ont subi la panne, nous devons identifier l'état du Datacenter hébergeant ces tâches, si le Datacenter est en état normal, un nœud disponible du même Datacenter est sélectionné ensuite les tâches seront lancées. Si le Datacenter est en état chargé, le Datacenter le moins chargé est sélectionné pour assurer un équilibrage de charge. Ensuite nous choisissons un nœud disponible pour relancer les tâches.

3.9 Conclusion

Dans ce chapitre, nous avons proposé une solution pour la tolérance aux pannes et l'équilibrage de charge dans le Cloud Computing pour assurer la sûreté de fonctionnement et une répartition équilibrée des tâches.

Notre approche basée sur la réplication dynamique permet d'assurer la fiabilité des services cloud avec un minimum coût malgré l'occurrence des fautes.

Le prochain chapitre sera consacré à la partie implémentation et évaluation des services de notre proposition de tolérance aux pannes. Nous mettrons en évidence l'importance et l'efficacité de l'approche proposée.

CHAPITRE 4

IMPLÉMENTATION

Contents

4.1	Introduction	64
4.2	Langage et environnement de développement	64
4.3	Langage de programmation Java	64
4.4	Environnements de développement	65
4.4.1	Eclipse :	65
4.4.2	CloudSim :	65
4.5	Interface principale	69
4.5.1	Configuration des paramètres de simulation	69
4.6	Lancement de la simulation	72
4.7	Résultats expérimentaux	72
4.7.1	Expérience 1 : Temps de réponse moyen	73
4.7.2	Expérience 2 : nombre de répliques créés	73
4.7.3	Expérience 3 : Impact de la fréquence des pannes sur le comportement de notre approche	74
4.7.4	Expérience 4 : l'équilibrage de charge	76
4.7.5	Expérience 5 : Dépenses monétaires du fournisseur (Cost)	77
4.8	Conclusion	78

4.1 Introduction

Ce chapitre est consacré à la réalisation et la concrétisation des approches proposées, en étendant le simulateur CloudSim afin de gérer la réplication de données et la tolérance aux pannes dans les environnements de Cloud Computing. Dans un premier temps, nous présentons l'environnement de notre travail, puis nous définissons les différents services du simulateur CloudSim, et finalement nous présentons une série de simulations et leurs interprétations pour mettre en évidence nos propositions.

4.2 Langage et environnement de développement

Les approches proposées dans ce travail ont été implémentées et testées dans un environnement possédant les caractéristiques suivantes : Un processeur Intel(R) Core(TM) i5-4200 U CPU @ 1.60Ghz 2.30Ghz , doté d'une capacité de mémoire de 6GB, sous Windows 7 64 bits. Nous avons utilisé l'environnement de développement Eclipse.

4.3 Langage de programmation Java

Java est à la fois un langage de programmation informatique orienté objet et un environnement d'exécution portable. Il est 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.

Le langage Java a la particularité principale que les logiciels écrits avec ce dernier sont très facilement portables sur plusieurs systèmes d'exploitation tels que : Unix, Microsoft Windows, Mac OS ou Linux avec ou sans modifications.

C'est la plate-forme qui garantit la portabilité des applications développées en Java.

Java est devenu aujourd'hui une direction incontournable dans le monde de la programmation parmi les différentes caractéristiques qui sont attribuées à son succès , nous avons :

- L'indépendance de toute plate-forme : le code reste indépendant de la machine sur laquelle il s'exécute. Il est possible d'exécuter des programmes

Java sur tous les environnements qui possèdent une Java Virtual Machine.

- Java est également portable, permettant à la simulation d'être distribuée facilement sans avoir à recompiler le code pour les différents systèmes.
- Le code est structuré dans plusieurs classes dont chacune traite une partie différente de la simulation.
- Java est multitâches : il permet l'utilisation de Threads qui sont des unités d'exécution isolées.

4.4 Environnements de développement

4.4.1 Eclipse :

Eclipse est un environnement de développement intégré, libre, extensible, universel et polyvalent, permettant de créer des projets de développement mettant en œuvre n'importe quel langage de programmation. Eclipse IDE est principalement écrit en Java (à l'aide de la bibliothèque graphique SWT, d'IBM), grâce à des bibliothèques spécifiques, ce langage est également utilisé pour écrire des extensions. La spécificité d'Eclipse IDE vient du fait de son architecture totalement développée autour de la notion de Plug-in (en conformité avec la norme OSGi), toutes les fonctionnalités de cet atelier logiciel sont développées en tant que Plug-in. Plusieurs logiciels commerciaux sont basés sur ce logiciel libre, comme par exemple IBM Lotus Notes 8, IBM Symphony ou WebSphere Studio Application Developer, etc.

4.4.2 CloudSim :

L'objectif principal de simulateur CloudSim est de fournir un cadre de simulation généralisé et extensible qui permet la modélisation, la simulation et l'expérimentation des nouvelles infrastructures du Cloud Computing et les services d'application, permettant aux utilisateurs de se concentrer sur des questions de conception du système qu'ils veulent étudier, sans être préoccupé aux détails relatifs aux services et infrastructures Cloud.

Nous avons utilisé pour la réalisation de notre travail la version du simulateur CloudSim 3.0.3.

Architecture générale de CloudSim :

La Figure 4.1 montre une architecture multicouche de la structure logicielle CloudSim et ses composantes.

La couche CloudSim fournit un support pour la modélisation et la simulation des Data Centers dans l'environnement des clouds computing y compris les interfaces de gestion dédiées aux machines virtuelles, la mémoire, le stockage et la bande passante. L'affectation des VMs à des hôtes, la gestion d'exécution des applications et le suivi de l'état du système dynamique sont traités par cette couche. Un fournisseur Cloud doit implémenter ses stratégies dans cette couche afin d'étudier l'efficacité des différentes politiques qui permettent l'attribution des VMs à ses hôtes.

Au niveau supérieur de la couche CloudSim, nous avons le code utilisateur (user code) fournissant les entités de base pour les hôtes (nombre de machines, leur spécification et ainsi de suite), les applications (nombre de tâches et leurs exigences), VMs, nombre d'utilisateurs, types d'application et les politiques d'ordonnancement du Broker. Un développeur d'applications Cloud peut générer plusieurs activités parmi lesquelles nous citons : Des Scénarios de disponibilité des modèles clouds, effectuer des tests robustes basés sur les configurations personnalisées supportées par le CloudSim et implémenter des techniques de provisionnement des applications personnalisées dans les clouds.

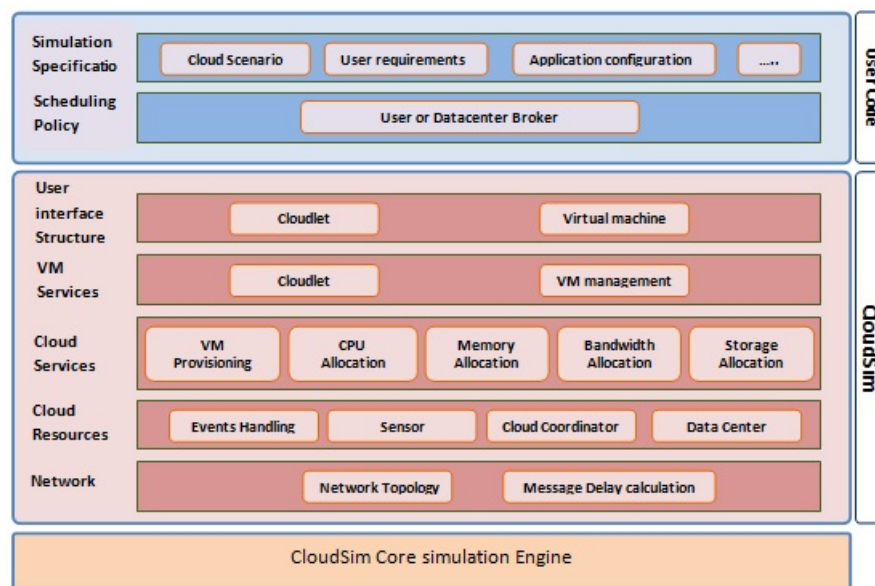


FIGURE 4.1 – Architecture générale de CloudSim

Les classes de CloudSim :

Le simulateur CloudSim est composé de plusieurs classes que nous pouvons classer en deux catégories : des classes qui modélisent les entités comme le Data Center, le Broker, etc. et des classes modélisant les politiques d'allocation.

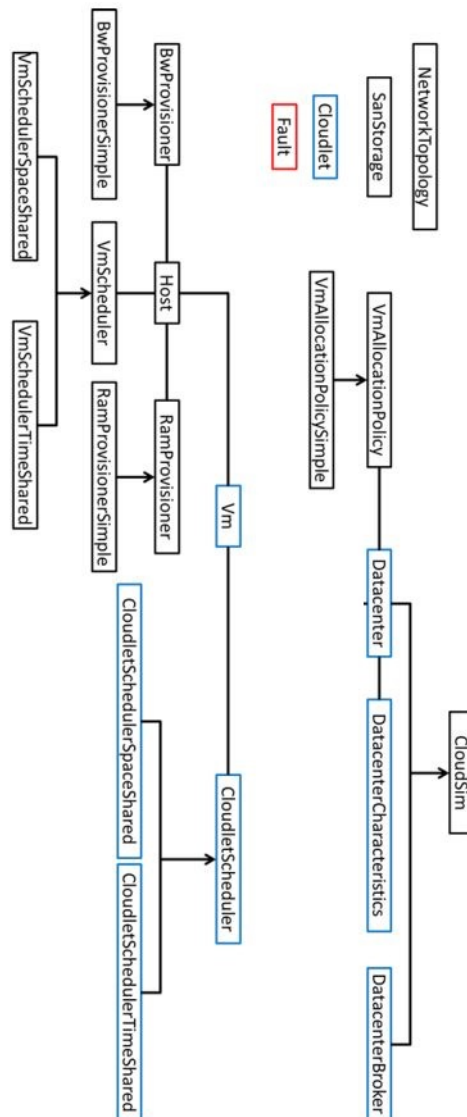


FIGURE 4.2 – Les classes de CloudSim

SimEntity : Il s'agit d'une classe abstraite, elle représente l'entité de simulation qui est capable d'envoyer des messages à d'autres entités et de gérer les messages reçus ainsi que les événements. Toutes les entités doivent étendre cette classe et redéfinir ses trois principales méthodes : `startEntity()`, `processEvent()` et `shutdownEntity()`. Ces méthodes définissent les actions pour l'initialisation de

l'entité, le traitement des événements et la destruction de l'entité.

Datacenter : Cette classe modélise les services au niveau des infrastructures de base (matériel) qui sont offerts par les fournisseurs de Cloud (Amazon, Azure, App Engine). Elle encapsule un ensemble d'hôtes qui peuvent être homogènes ou hétérogènes par rapport à leurs configurations matérielles (mémoire, noyaux, capacité et stockage). La classe `PowerDataCenter.java` utilisée dans le package «Power» hérite de la classe `DataCenter.java` et elle permet la simulation des centres de données en calculant leurs énergie consommée.

DatacenterBroker : Cette classe modélise le Broker, qui est responsable de la médiation des négociations entre les utilisateurs et les prestataires de service selon les conditions de QoS des utilisateurs. Il déploie aussi les tâches de service à travers les Clouds. Le Broker, au nom des utilisateurs, agit sur les prestataires du service approprié du cloud par le service d'information du Cloud CIS (Cloud Information Services) et négocie avec eux pour une allocation des ressources qui répond aux besoins de QoS des utilisateurs.

VM : Cette classe représente une instance de machine virtuelle (VM) qui est gérée et hébergée par une machine physique (hôte). Chaque composant VM a accès à un composant qui stocke les caractéristiques suivantes liées à une VM telles que : mémoire accessible, le processeur, capacité de stockage et les politiques de provisionnement interne de la machine virtuelle. Dans le but de réduire l'énergie consommée par les Data Centers, la classe `VMPower.java` héritant de la classe `VM.java` enregistre l'historique de l'utilisation de CPU. Cet historique est utilisé dans les politiques de sélection et d'allocations des VMs.

Cloudlet : Cette classe modélise les services d'application du Cloud (comme la livraison, réseaux sociaux et le workflow d'affaires). `CloudSim` représente la complexité d'une application en fonction de ses besoins informatiques. Chaque service d'application a une taille d'instruction pré-assigne et la quantité de flux de transfert de données qu'il doit entreprendre au cours de son cycle de vie. Cette classe peut également être étendu pour supporter la modélisation de la performance et d'autres paramètres de composition pour les applications telles que les transactions dans les applications orientées base de données.

Fault injecter : Cette classe permet de modéliser l'inter-arrivée des pannes afin de créer les scénarios nécessaires pour évaluer le bon fonctionnement de nos approches de tolérance aux pannes. De nombreux scénarios de pannes sont possibles au cours d'exécution d'une application et le test exhaustif de tous ces

scénarios nécessite beaucoup de travail. Avec cette classe, il est possible de créer un scénario des pannes spécifique et surveiller le comportement de l'architecture de tolérance aux pannes dans un tel scénario.

4.5 Interface principale

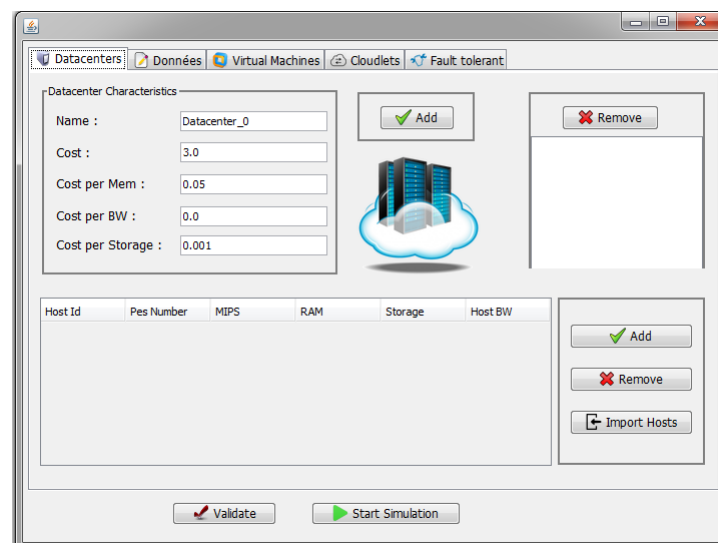
La version de Cloudsim n'a pas d'interface graphique, son exécution se fait sur console, donc nous avons créé une interface qui facilite l'accès à la configuration de simulateur.

4.5.1 Configuration des paramètres de simulation

Pour lancer la simulation d'un Cloud avec notre version étendu du simulateur CloudSim, nous devons configurer les paramètres de simulation dans un premier temps. Les Figures au-dessous montrent les différentes fenêtres de configuration de simulation qui contient 5 parties principales :

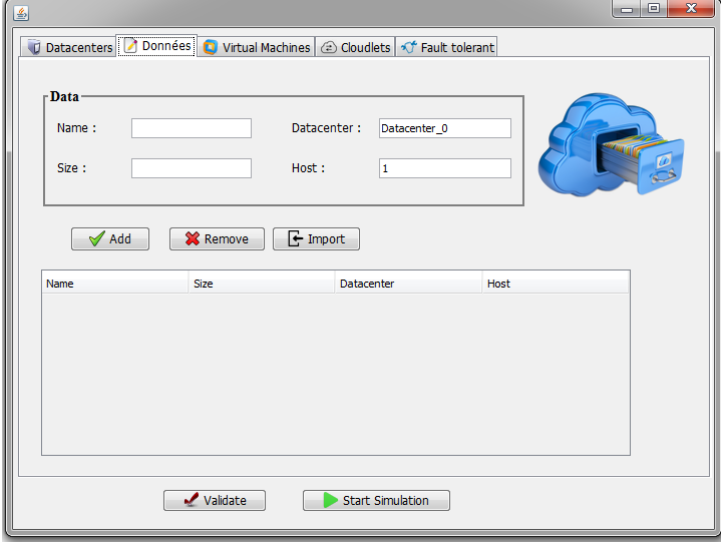
Datacenters :

Cette étape consiste à faire entrer les paramètres nécessaires propres à la topologie du réseau 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 coût de la mémoire, la taille du disque dur, le coût de stockage et la bande passante ainsi que son coût.



Configuration des données :

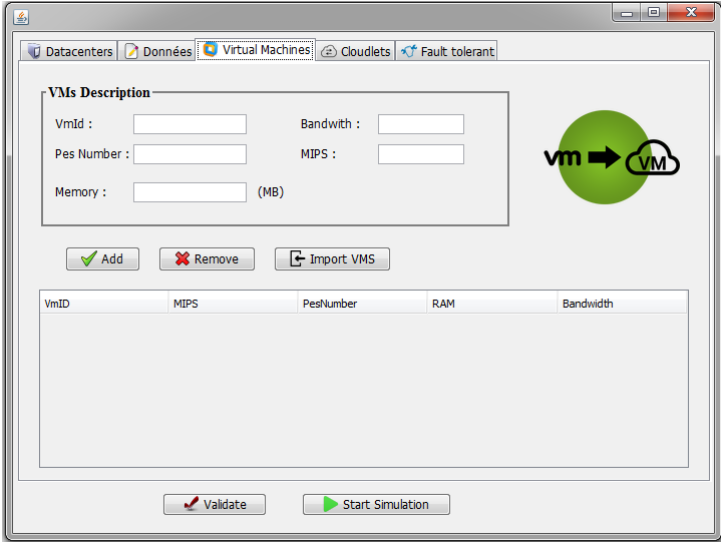
Cette étape consiste à créer des données. Pour ajouter une donnée, on doit configurer les paramètres suivants : le nom de la donnée, sa taille, le Datacenter qui héberge cette donnée et l'hôte où se trouve cette donnée.



Name	Size	Datacenter	Host
------	------	------------	------

Virtual Machines :

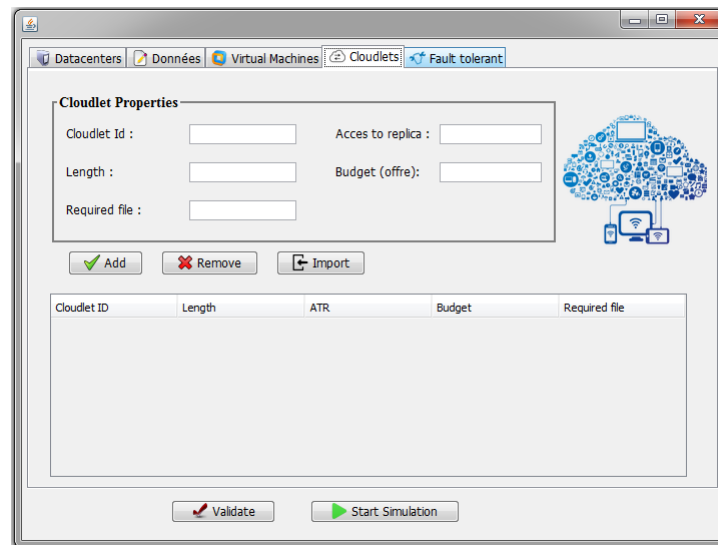
Permet de configurer les machines virtuelles et leur caractéristiques : le nombre de CPU dans une VM, la taille de la mémoire, la taille du disque dur et la bande passante.



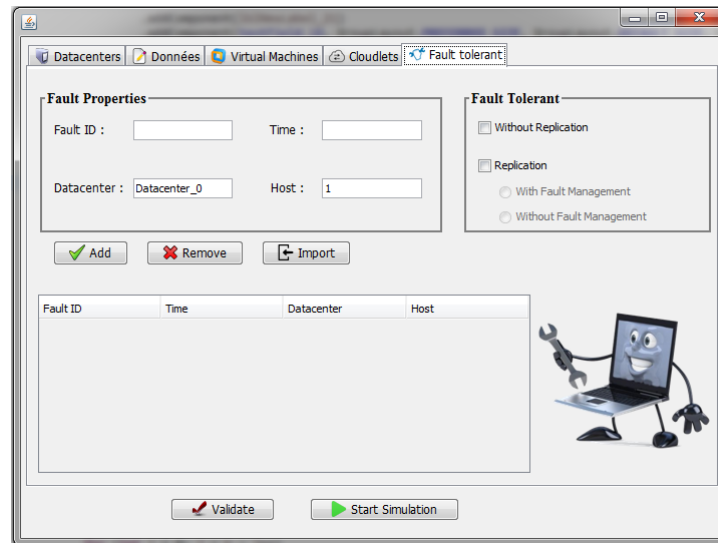
VmID	MIPS	PesNumber	RAM	Bandwidth
------	------	-----------	-----	-----------

Cloudlet :

Permet de configurer les propriétés de la Cloudlet et leur caractéristiques : la taille de la Cloudlet ,le fichier requis pour l'exécution ,le budget de cloudlet et le temps d'utilisation de fichier .

**Fault Tolerant :**

L'onglet Fault Tolerant permet à l'utilisateur d'injecter les pannes et de sélectionner le mécanisme de tolérance aux pannes. Without Replication représente l'exécution sans aucune approche de tolérance aux pannes. Replication est l'approche de tolérance aux pannes basée sur la réplication de données. Si l'option replication est sélectionnée, nous pouvons choisir l'option without fault management qui représente l'exécution sans gestion des pannes ou l'option with fault management pour la gestion des pannes (détection et correction).



4.6 Lancement de la simulation

Après avoir personnalisé les paramètres de simulation, l'utilisateur peut lancer la simulation en cliquant sur le bouton **Validate** pour valider les paramètres de configuration ensuite le bouton **Start Simulation**.



4.7 Résultats expérimentaux

Nous présentons dans cette partie quelques résultats des différentes expérimentations que nous avons obtenus, nous avons effectué plusieurs séries de simulations pour comparer les approches suivantes :

1. **Approche sans réplication (Without Replication)** : Représente la première stratégie dans laquelle aucune stratégie de réplication n'est implémentée. Dans cette approche, le nombre de données et leurs placement sont affectés d'une façon aléatoire et statique.
2. **Approche basée sur la réplication seulement (Replication)** : Représente notre stratégie dans laquelle la réplication des données est effectuée mais sans gestion des pannes.
3. **Approche basée sur la réplication avec la gestion des pannes** : c'est une amélioration de notre approche dans laquelle la réplication des données

est effectuée et la la gestion des pannes est considérée.

4.7.1 Expérience 1 : Temps de réponse moyen

Dans cette première simulation, nous avons calculé le temps de réponse moyen des trois approches, Cette simulation a été réalisée avec les paramètres de simulations suivants : 3 Datacenter, chaque Datacenter contient 40 machines et Chaque machine à un processeur d'une vitesse de 1000MIPS, 2Go de mémoire RAM. Le nombre de Cloudlets (tâches) varie entre 20 et 80. La Figure 4.3 montre les résultat de cette simulation : D'après ces résultats, nous remarquons que le temps

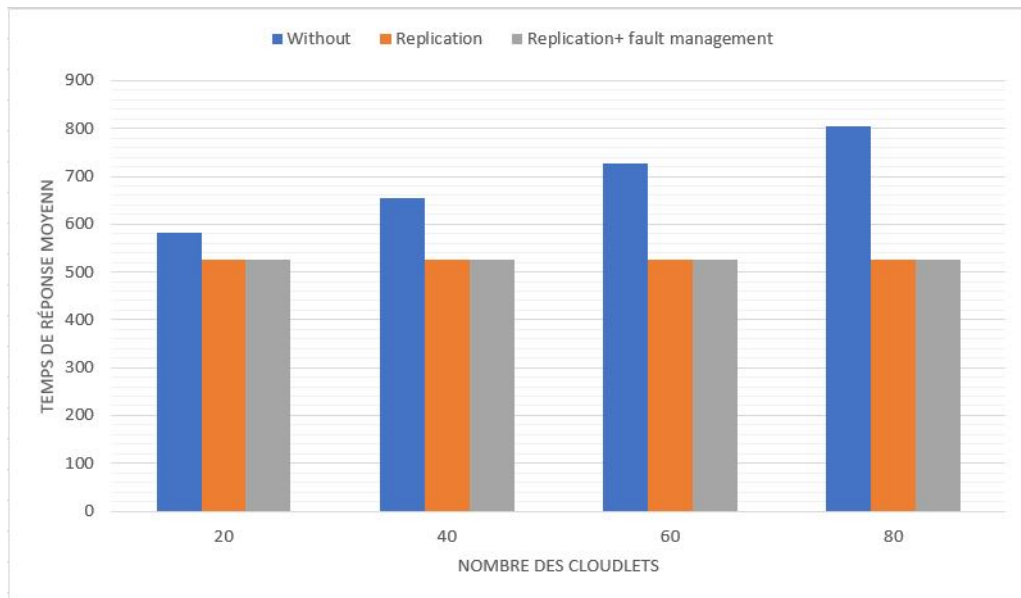


FIGURE 4.3 – Impact du nombre de Cloudlets sur le temps de réponse moyen

de réponse moyen dans la première approche augmente a chaque fois qu'on augmente le nombre de Cloudlets, car la stratégie sans réplication n'a créé aucune réplique donc plusieurs Cloudlets sont traitées à la fois avec un nombre de réplique statique alors l'exécution prend beaucoup de temps pour traiter toutes les Cloudlets, Par contre dans les autres approches le temps de réponse moyen est stable en raison d'avoir un plus grand nombre de répliques dans le nuage.

4.7.2 Expérience 2 : nombre de répliques créés

Comme son nom l'indique, la stratégie sans réplication n'a créé aucune réplique, par contre dans les autres approches basées sur la réplication, nous remarquons

que les deux stratégies ont effectué un nombre similaire de réplication et que le nombre des répliques créés augmente à chaque fois qu'on augmente le nombre de Cloudlets. La Figure 4.4 montre les résultats de simulation.

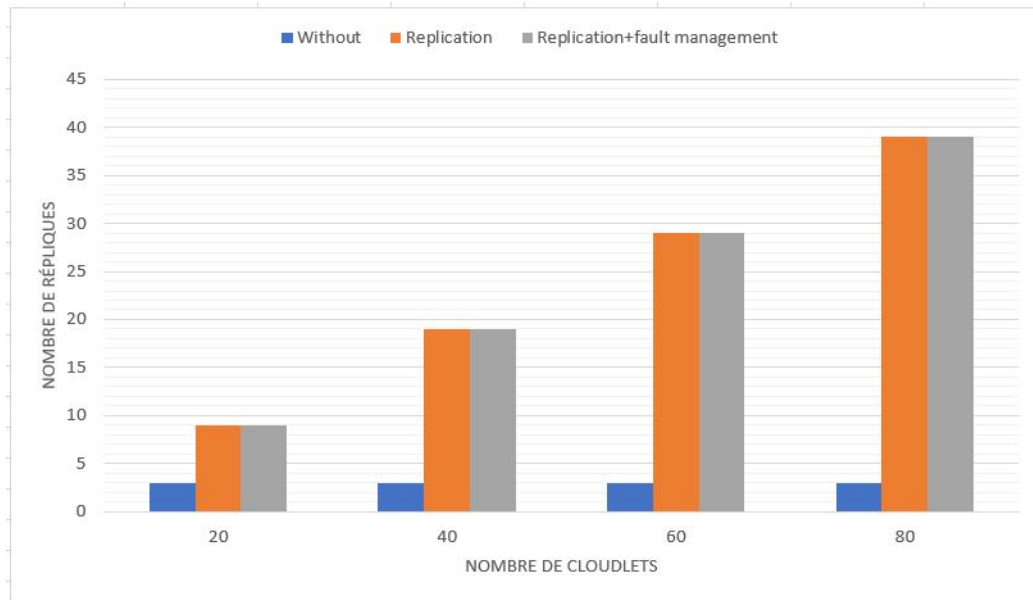


FIGURE 4.4 – Impact du nombre de cloudlet sur le nombre répliques.

4.7.3 Expérience 3 : Impact de la fréquence des pannes sur le comportement de notre approche

Dans cette série de simulations nous avons testé l'impact du nombre de pannes sur le comportement du système. Nous avons lancé des simulations dans un Cloud constitué de 3 Datacenter, chaque Datacenter contient 40 machines. Les figure 4.5 et 4.6 présentent les résultats pour trois scénarios :

1. Sans réplication.
2. Avec réplication et sans gestion des pannes.
3. Avec réplication et gestion des pannes.

La seule différence entre les simulations est que le nombre des pannes varie entre 5 et 15.

Effet sur le temps d'exécution :

Dans cette simulation nous avons calculé le temps de réponse des cloudlets pour chaque scénario, la Figure 4.5 de manière évidente, la supériorité des ap-

proches basées sur la réplication par rapport à une exécution sans utiliser le mécanisme de réplication. Nous remarquons aussi, que la stratégie de réplication avec gestion des pannes atteint un meilleur temps de réponse par rapport l'approche sans gestion des pannes.

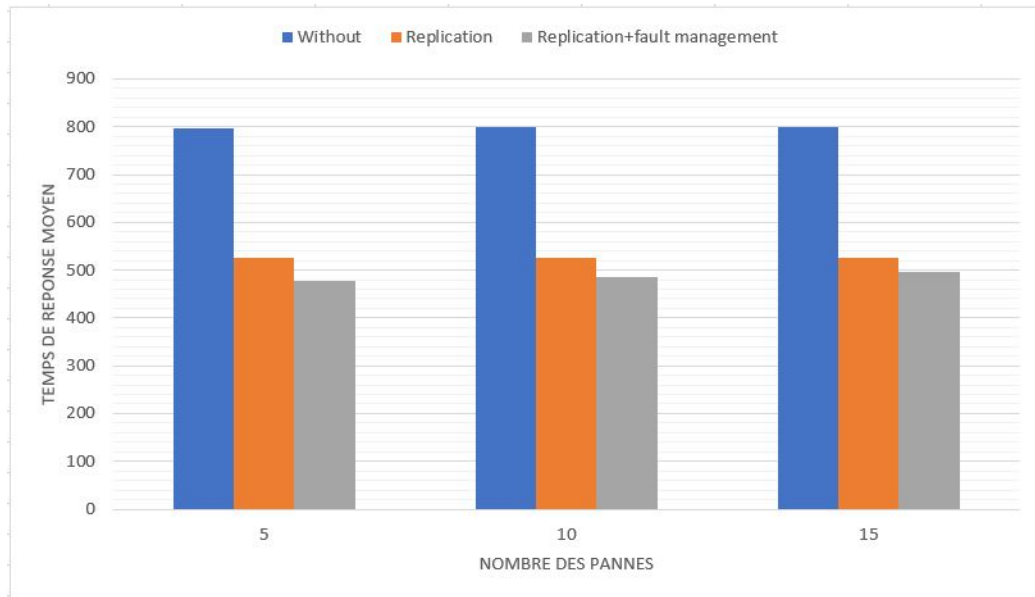


FIGURE 4.5 – Impact de nombre des pannes sur le temps de réponse.

Effet sur Nombre de cloudlets exécutées avec succès :

Nous avons mesuré le nombre de cloudlets exécutées avec succès et le nombre de cloudlets échouées pour chaque scénario, la Figure 4.6 montre le résultats de simulation.

Nous remarquons une diminution significative du nombre de cloudlet échouées en appliquant notre approche. Et cela est dû au mécanisme de tolérance aux pannes que nous avons appliqué pour relancer les Cloudlets échouées après la détection des pannes. Nous pouvons remarquer en présence des défaillances qu'avec notre approche a pu exécuter tous les cloudlets envoyés avec succès ce qui est équivalent à 100%.

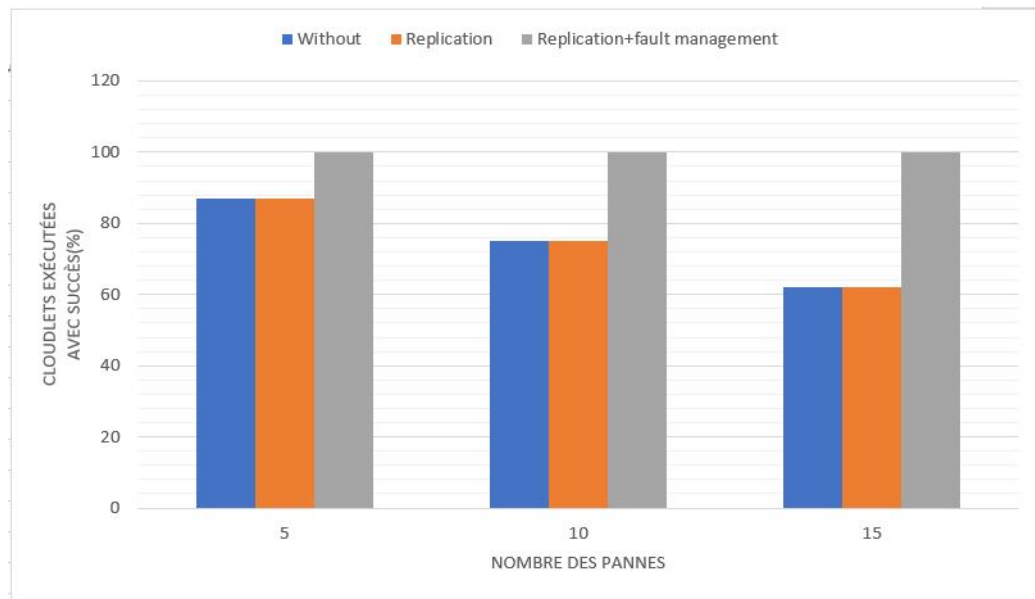


FIGURE 4.6 – L’impact des pannes sur le Nombre de cloudlets exécutées avec succès

4.7.4 Expérience 4 : l’équilibrage de charge

Dans cette simulation, nous avons calculé le degré d’équilibrage de charge des trois approches, Cette simulation a été réalisée avec les paramètres de simulations suivants : un Datacenter, contient 20 machines et Chaque machines a un processeur. La Figure 4.7 montre les résultats de simulation. Nous remarquons

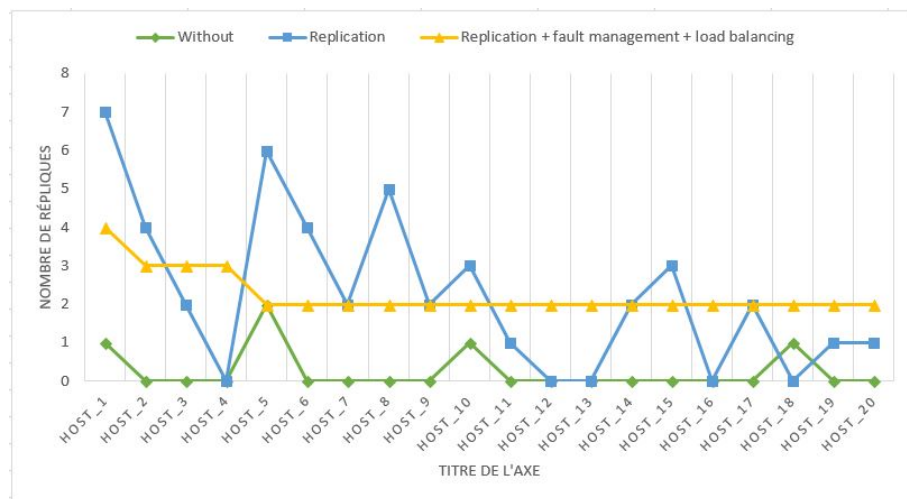


FIGURE 4.7 – L’équilibrage de charge

que à la différence des deux autres approches qui ne gère pas d’une manière fine l’équilibrage de charge, notre deuxième approche assure un bon équilibrage de

charge et une utilisation bien équilibrée des différents serveurs dans le datacenter.

4.7.5 Expérience 5 : Dépenses monétaires du fournisseur (Cost)

Dans cette simulation, nous avons calculé les Dépenses monétaires du fournisseur, la figure 4.8 montre les résultats de simulation.

Les transferts de données dans la stratégie sans réplication ont causé un coût de réseau important, De plus, la violation fréquente du SLA introduit un coût de pénalité élevé.

Dans les deux stratégies basées sur la réplication, le nombre de violations des SLA est très faible. En conséquence, le coût de pénalité est à un niveau plus acceptable.

En termes de coûts de stockage, la stratégie sans réplication n'a conservé que les ensembles de données initialement placés, d'où le coût de stockage est presque négligeable.

Les deux autres stratégies ont créé un grand nombre de répliques qui, à leur tour, a causé un coût de stockage important pour le fournisseur.

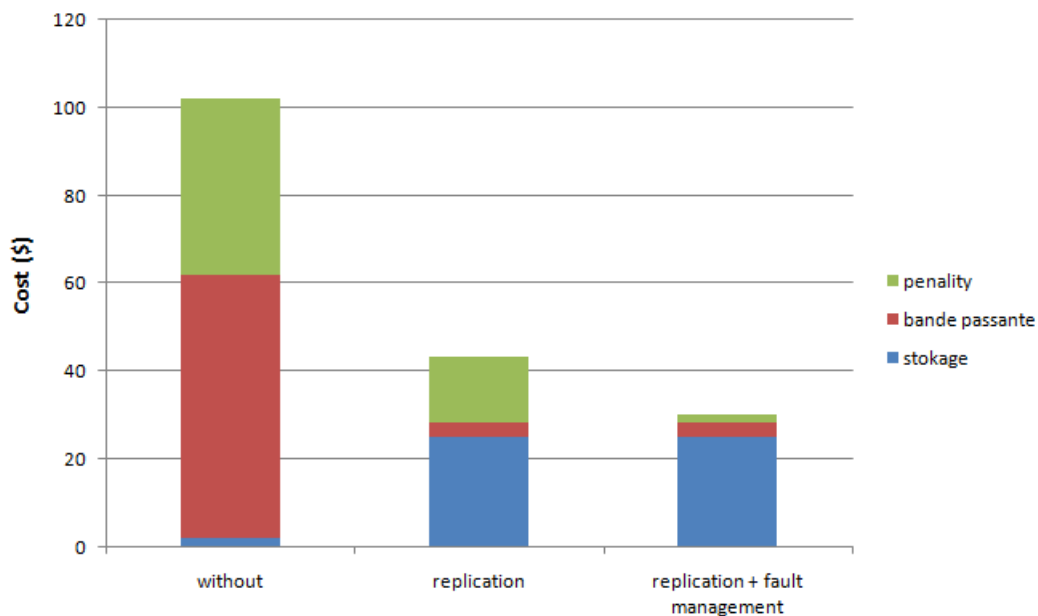


FIGURE 4.8 – Dépenses monétaires du fournisseur

4.8 Conclusion

Nous avons effectué une série d'expérimentations pour évaluer les performances de l'approche proposée, Nous avons réalisé plusieurs simulations en jouant sur différents paramètres comme : le nombre de cloudlets, le nombre de pannes.

Les résultats obtenus montrent que notre approche a réussi d'atteindre les objectifs désirés, Les performances de notre approche sont meilleures par rapport à l'approche statique.

CONCLUSION GÉNÉRALE ET PERSPECTIVES

Le cloud computing est basée généralement sur les services offerts à l'utilisateur selon les besoins et les contrats, donc La fiabilité et la robustesse de ces services deviennent des critères importants surtout dans un environnement qui exige un certain deadline pour satisfaire le SLA, pour cela un mécanisme de tolérance aux pannes est important.

Dans ce mémoire, nous avons présenté une stratégie pour gérer l'équilibrage de charge et la tolérance aux pannes dans le cloud computing, Cette approche est basée sur un mécanisme de réplication dynamique qui permet de garantir une certaine fiabilité, disponibilité et de distribuer la charge de travail entre les différents serveurs dans le Cloud. Nous avons intégré dans notre stratégie un module de gestion des pannes qui permet de garantir la continuité des services du système d'une façon transparente et efficace en cas des pannes.

Pour mettre en évidence l'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 de métriques telles que : nombre de tâches échouées, le temps de réponse, le nombre des répliques et l'équilibrage de charge. Les résultats montrent que l'approche proposée présente de bonnes performances en présence des pannes, elle permet d'améliorer le temps de réponse et minimiser les surcoûts causés par les pannes. En outre, elle garantit un bon équilibrage de charge entre les nœuds.

Un certain nombre de pistes de recherche sont envisageables pour poursuivre

notre travail. Parmi ces pistes de recherche, nous pouvons mentionner ce qui suit :

1. Mise en place d'un service permettre d'estimer la charge globale de système cloud.
2. Proposer une gestionnaire des pénalités en cas de violation de SLA.
3. Implémentation de cette approche et étudier ses performances dans une plateforme de Cloud Computing réelle.

BIBLIOGRAPHIE

- [1] Malvault W. Vers une architecture pair-à-pair pour l'informatique dans le nuage. Master's thesis, Université de Grenoble, France, 2011.
- [2] Samy Sadi. *Techniques de Checkpointing pour la Tolérance aux Fautes dans le Cloud Computing*. PhD thesis, Université d'Oran , Algérie, 2017.
- [3] Bouamama Samah. Gestion des ressources dans les cloud computing à base des modèles économiques. Master's thesis, Université d'Oran, Algérie, 2011.
- [4] A El Mahdaouy and M Oumsis. Evaluation et amélioration de performances des algorithmes d'équilibrage de charges dans un environnement cloud computing. casablanca, morocco, 2013.
- [5] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011.
- [6] Nicolas Grevet. Le cloudcomputing : évolution ou révolution. *Pourquoi, quand, comment et surtout faut-il prendre le risque*, 2009.
- [7] LIMAM Said. *Gestion de tolérance aux fautes dans le Cloud Computing avec CloudSim*. PhD thesis, Université Ahmed Ben Bella d'Oran1 Es Senia, 2012.
- [8] JF Pépin, S Bouteiller, AS Boissard, and J Watrinel. Fondamentaux du cloud computing-le point de vue des grandes entreprise. réseau de grandes entreprises (cigref). *Mars*, 2013.
- [9] Marie-Caroline Bonnet-Galzy Pascal Faure, Gabrielle Gauthey. Guide sur

- le cloud computing et les datacenters à l'attention des collectivités locales. 2015.
- [10] Dad Djouhra. *Optimisation des performances des data centers des cloud sous contrainte d'énergie consommée*. PhD thesis, Université d'Oran , Algérie, 2016.
- [11] Yannick Kuhn Allan Lefort Vincent Kherbache, Mohamed Moussalih. Cloud computing. projet tutoré en licence professionnelle asrall. 2010.
- [12] Civic Consulting. L'informatique en nuage, Étude d'Épartement thÉmatique a : Politiques Économiques et scientifiques. 2012.
- [13] L.F. Noumsl. Etude et mise en place d'une solution "cloud computing " privée dans une entreprise moderne : cas de camtel, ecole nationale supérieure des postes et télécommunications. 2012.
- [14] P.P. Codo. Conception d'une solution de cloud computing privé basée sur un algorithme de supervision distribué : Application aux services iaas. ecole polytechnique d'abomey-calavi (epac),. 2012.
- [15] Tomas G. Configuration réseau et mise en place de customer immersion expériences au microsoft innovation center de mons. 2010.
- [16] Y. Crouzet J. Arlat and Y. Deswarte. Encyclopédie de l'informatique et des systèmes d'information. 2006.
- [17] JC Laprie. Concepts de base de la tolérance aux fautes, journée de. l'ofta, informatique tolérante aux fautes. Paris, 1994.
- [18] Powell D. Failure mode assumption coverage. in proceedings of the 22th ieee surnposium on fault tolerant computing,. 1992.
- [19] Ali KALAKECH. Étalonnage de la sûreté de fonctionnement des systèmes d'exploitation – spécifications et mise en œuvre, 2005.
- [20] Haouati M. S. *Architectures innovantes de systèmes de commandes de vol*. PhD thesis, Université de Toulouse, 2010.
- [21] F. Salfner et al. A survey of online failure prediction methods, acm computing surveys (csur). 2010.

- [22] T.D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *j. acm.*, 1996.
- [23] S. Monnet. *Gestion des données dans les grilles de calcul : support pour la tolérance aux fautes et la cohérence des données*. PhD thesis, Rennes, France, 2006.
- [24] J Vial. *Test et Testabilité de Structures Numériques Tolérantes aux Fautes*. PhD thesis, Université de Montpellier, 2009.
- [25] Chafik ARAR. *Redondance logicielle pour la tolérance aux fautes des communications*. PhD thesis, Université BATNA, Algérie, 2016.
- [26] W. B. Ling Z. D. Cheng H. Hui, Z. Zhan and Y. Xiao Zong. A two-level application transparent checkpointing scheme in cloud computing environment. 2013.
- [27] Dawei Sun, Guiran Chang, Changsheng Miao, and Xingwei Wang. Analyzing, modeling and evaluating dynamic adaptive fault tolerance strategies in cloud computing environments. *The Journal of Supercomputing*, 66(1) :193–228, 2013.
- [28] Himanshu Agarwal and Anju Sharma. A comprehensive survey of fault tolerance techniques in cloud computing. In *Computing and Network Communications (CoCoNet), 2015 International Conference on*, pages 408–413. IEEE, 2015.
- [29] G Paul Koning, Peter C Hayden, Paula Long, Hsin H Lee, Vasudevan Subramanian, Lazarus J Vekiarides, and Satyanarayana R Goluguri. Distributed snapshot process, October 11 2011. US Patent 8,037,264.
- [30] Adam Moody, Greg Bronevetsky, Kathryn Mohror, and Bronis R de Supinski. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *Proceedings of the 2010 ACM/IEEE international conference for high performance computing, networking, storage and analysis*, pages 1–11. IEEE Computer Society, 2010.
- [31] E. N. (Mootaz) Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3) :375–408, September 2002.

- [32] Xiaolin Teng and Hoang Pham. Software fault tolerance. In *Handbook of Reliability Engineering*, pages 585–611. Springer, 2003.
- [33] Besson X. *Tolérance aux fautes et reconfiguration dynamique pour les applications distribuées à grande échelle*. PhD thesis, Université de Grenoble, France, 2010.
- [34] Netzer. Rand Raynal M. Hélary. J, Mostefaoui. A. Communication-based prevention of useless checkpoints in distributed computations. distrib. comput. 2000.
- [35] Joseph F Ruscio, Michael A Heffner, and Srinidhi Varadarajan. Dejavu : Transparent user-level checkpointing, migration, and recovery for distributed systems. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–10. IEEE, 2007.
- [36] Lorenzo Alvisi et Keith Marzullo. Message logging : Pessimistic, optimistic, causal, and optimal. *IEEE Transactions on Software Engineering*, 1998.
- [37] Manetho Elmootazbellah. Elnozahy. *fault tolerance in distributed systems using rollback-recovery and process replication*. PhD thesis, Rice University, Houston, TX, USA, 1993.
- [38] BOUKERRAM Abdellah DRIF Ahlem. Algorithme de diffusion génétique pour l'équilibrage de charge dans les grilles de calcul, ufa, sétif, algérie. 2007.
- [39] William Leinberger, George Karypis, Vipin Kumar, and Rupak Biswas. Load balancing across near-homogeneous multi-resource servers. In *Heterogeneous Computing Workshop, 2000.(HCW 2000) Proceedings. 9th*, pages 60–71. IEEE, 2000.
- [40] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7) :558–565, 1978.
- [41] O. Peres. *Construction de topologies auto stabilisante dans les systèmes à grande échelle*. PhD thesis, Université Paris-sud, France, 2008.
- [42] Kouidri Siham. gestion de la cohérence des répliques tolérante aux fautes dans une grille des données. Master's thesis, université d'Oran, 2011.

- [43] R. Buyya S. Venugopal and R. Kotagiri. A taxonomy of data grids for distributed data sharing, management and processing. *ACM computing survey*, 2007.
- [44] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid : Enabling scalable virtual organizations. *The International Journal of High Performance Computing Applications*, 15(3) :200–222, 2001.
- [45] Yaser Nemati, Faramarz Samsami, and Mehdi Nikhkhah. A novel data replication policy in data grid. *Australian Journal of Basic and Applied Sciences*, 6(7) :339–344, 2012.
- [46] Jean-Paul FIGER. Cloud computing-informatique en nuage. 2012.
- [47] <http://www.orange-business.com/fr/blogs/cloud-computing/transformation/le-potentiel-du-cloud-dans-l-educationmstechdays>. (consulté le 29/12/2016).
- [48] Lahdir Meriem Ait Ali Sylia. Gestion de la cohérence des données répliquées dans le cloud. Master’s thesis, Université A/Mira de Bejaia, 2016.
- [49] L. el houssine A. Eddaoui Y.Fahim, EL Habib.B. *La ré-allocation dynamique des tâches pour un équilibrage de charge dans le cloud computing*. PhD thesis, Casablanca, Maroc, 2014.
- [50] Han Y. S] Deng J, Huang S. C-H and Deng J. H. Fault-tolerant and reliable computation in cloud computing. *IEEE Globecom 2010 Workshop on Web and Pervasive Security (WPS 2010)*,, 2010.
- [51] Papaioannou T. G Bonvin N and Aberer. K. A self-organized, fault-tolerant and scalable replication scheme for cloud storage. In *ACM Symposium on Cloud Computing*, 2010.
- [52] Yagoubi Belabbas. Modèle arborescent pour l’équilibrage de charge dans les grilles de calcul. 2007.
- [53] Sandeep Sharma, Sarabjit Singh, and Meenakshi Sharma. Performance analysis of load balancing algorithms. *World Academy of Science, Engineering and Technology*, 38(3) :269–272, 2008.

-
- [54] Dr. Sandeep Sharma Meenakshi Sharma, Pankaj Sharma. Efficient load balancing algorithm in vm cloud environment. *International Journal on Computer Science and Technology*, 2012.

Résumé

Le cloud computing est un type de systèmes distribués dans lequel les pannes et la déséquilibre de charge existent et peuvent engendrer des pertes de ressources, ceci réduit la fiabilité de cloud, Dans ce cas, l'implémentation d'un service pour surmonter ces défis deviennent une exigence majeure. Dans ce travail nous avons présenté une approche pour gérer l'équilibrage de charge et la tolérance aux pannes dans les Cloud Computing. Notre approche qui est basée sur la réplication permet de garantir la continuité des services du Cloud Computing d'une façon transparent et efficace en présence des pannes et de masquer les effets négatifs de pannes. Les résultats obtenus par la simulation de notre proposition montrent l'efficacité de notre approche en termes de temps de réponse, de disponibilité et de répartition de charge.

Mots-clés : tolérance aux pannes, Cloud computing, virtualisation, réplication, équilibrage de charge , rentabilité.

Abstract

Cloud computing is a type of distributed systems in which failures and load imbalance exist and can cause resource losses, this reduces cloud reliability, in this case, the implementation of a service to overcome these challenges become a major requirement. In this work we presented an approach to manage load balancing and fault tolerance in Cloud Computing. Our replication-based approach ensures cloud service continuity in the presence of failures and masks the negative effects of failures. The results obtained by the simulation of our proposal show the efficiency of our approach in terms of response time, availability and load distribution.

Keywords : fault tolerance, cloud computing, virtualization, replication, load balancing, profitability.

المخلص

الحوسبة السحابية هي نوع من الأنظمة الموزعة أين الخلل وعدم توازن الحمل يمكن أن يؤدي إلى فقدان الموارد، مما يقلل من موثوقية السحابة ويحط من أدائها. في هذه الحالة، يصبح تنفيذ خدمة للتغلب على هذه التحديات مطلباً رئيسياً. في هذا العمل، اقترحنا طريقة لإدارة التسامح مع الخطأ وموازنة الأحمال في بيئة الحوسبة السحابية. يتضمن نهجنا القائم على النسخ المتماثل استمرارية الخدمة السحابية في وجود الفشل وإخفاء للتأثيرات السلبية للفشل. النتائج التي تم الحصول عليها من خلال محاكاة اقترحنا تظهر فعالية نهجنا من حيث زمن الاستجابة، وتوافر وتوزيع الحمل.

كلمات البحث: التسامح مع الخطأ، الحوسبة السحابية، المحاكاة الافتراضية، التكرار، توازن الحمل والربحية.