REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENTSUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE ABBAS LAGHROUR KHENCHELA

DEPARTEMENT DE MATHEMATIQUE ET INFORMATIQUE

Mémoire de fin d'étude pour l'obtention de diplôme

De master informatique

Option

Sécurité et Technologie Web

Thème

Sélection des chemins minimaux nécessaires et planification pour la tolérance aux fautes des services composés.

Réalisé par :

- Bouhlala Nessrine
- Mordjane Rania

Encadré par :

-Dr Hioual Ouassila

Année Universitaire 2019/2020

Résumé

Résumé :

Le Cloud Computing fait référence à l'utilisation des capacités de calcul des ordinateurs distants, où l'utilisateur dispose d'une puissance informatique considérable sans avoir à posséder des unités puissantes. Les services Cloud sont considérés comme des solutions partielles qui doivent être composés pour pouvoir répondre à la demande d'un client. La probabilité de voir une panne intervenir durant l'exécution devient forte lorsque le nombre de nœuds augmente. Puisqu'il est impossible d'empêcher totalement les pannes, une solution consiste à mettre en place des mécanismes de tolérance aux pannes. Dans ce travail, nous proposons une méthode basée sur le recouvrement arrière par reprise et la planification multi-agents en cherchant à chaque fois les chemins minimaux. La reprise est la technique la plus couramment utilisée : des points de reprise sont sauvegardés régulièrement ; l'exécution est ramenée dans un état sauvegardé survenu avant l'occurrence d'erreur. Cette exécution est distribuée en utilisant un système multi-agents qui se compose de deux agents : un agent planificateur et un agent contrôleur. L'agent planificateur a une connaissance complète de tous les services déployés dans le Cloud. Le rôle de cet agent est de créer les différents plans de composition des services pour répondre à la demande de l'utilisateur. Ces différents plans forment un graphe où les nœuds sont les services Cloud composants, et les arcs représentent l'ordre de composition des services. Ces arcs ont des poids qui sont des valeurs résultats de la pondération entre : le coût, le temps d'exécution, et la fiabilité. L'agent planificateur utilise ces poids pour déterminer le chemin minimal. Cet agent sauvegarde périodiquement un point de reprise (sommet d'un autre chemin minimal) dans une mémoire stable à condition qu'il y ait au moins deux chemins possibles. L'agent contrôleur de plan contrôle et assure le bon fonctionnement de chemin choisi et en cas de panne, il informe l'AP (Agent Planificateur) afin qu'il applique la technique de recouvrement arrière et de choisir un autre sous plan minimal.

Mots-clés : Cloud Computing, services Cloud, planification, SMA, chemins minimaux, tolérances aux pannes, recouvrement arrière, point de reprise.

Remerciements

Nous tenons tout d'abord à remercier Allah de nous a donné la force, le courage, le pouvoir et la volonté d'accomplir ce modeste travail.

Nos remerciements s'étendent à nos formidables parents qui par Leur prières et leurs encouragements, on a pu surmonter tous les obstacles.

Nous tenons à remercier, bien sûr, en priorité, notre encadreur Dr Hioual Ouassila pour ses conseils, ses orientations et son aide.

Nos sincères remerciements vont aux membres du jury qui nous ont fait l'honneur d'évaluer ce travail.

On remercie aussi tous les enseignants, qui m'ont suivi le long de mon cycle d'études.

Enfin ; on présente nos remerciements à toutes les personnes qui nous ont encouragé à construire et élaborer ce mémoire.

Dédicaces

Merci Allah de m'avoir donné le pouvoir et la patience et surtout la santé nécessaire pour achever ce travail.

Je dédie ce travail à :

Mes très chers parents « Kamissi » et « Sonia », qui m'ont beaucoup aidé et soutenus et je leur dis un grand merci pour tout votre soutien, votre amour, tous vos sacrifices et votre présence dans ma vie, que Allah les garde et les protège pour moi.

A mes chers et adorables frères « Raid » la prunelle de mes yeux, et « Borhane » mon cœur que j'adore.

A ma grand-mère « Mebarka » qui m'accompagné par ses prières et sa douceur, que Allah la protège et la donne sa santé. Et à mon cher grand-père « Fatmi ».

A mes proches amis de toujours pour leurs encouragements et leur soutien moral.

Et sans oublier mon binôme et ma chère amie « Nessrine ».

Mordjane Rania

Dédicaces

Merci ALLAH de m'avoir donné la capacité décrire et de réfléchir, la force d'y croire, la patience d'aller jusqu'au bout du rêve et bonheur de lever mes mains vers le ciel et de dire

« Ya Allh ».

Je dédie ce modeste travail :

A mes chers parents « Miloud »et « Saida » pour tous leurs sacrifices, leur amour, leur tendresse, leur soutien et leurs prières tout au long de mes études, que dieu vous bénisses et vous protèges.

A mes chères sœurs « Ibtissam », « Radia » et « Maroua », source d'espoir et motivations

A mes chers frères « Aymen », « Haithem », « Fares » et « Fouad », source de vie et d'amour.

A ma chère amie avant d'être binôme « Mordjane Rania ».

A mes chères copines « Abidi Nihad » et « Dakhli Ikram ».

A tous mes amis et mes camarades.

A tous mes enseignants, pour leur bienveillance et pour leur contribution à notre solide formation.

Bouhlala Nessrine

Table des matières

Liste de figures :

Liste des tableaux :

Introduction générale :

Le Cloud Computing offre à l'heure actuelle des services de pointe que nous pouvons exploiter à volonté avec une connexion internet : sauvegarde de données, logiciels en ligne, etc. Le Cloud représente ainsi un basculement de tendance, car au lieu de faire l'acquisition de nouveaux matériels et logiciels pour l'obtention de fonctionnalités dans un système isolé, on préfère se servir de la puissance de calcul et de stockage offerts par un fournisseur dans un système distribué. Ainsi, les systèmes multi-agents constituent une nouvelle technologie pour la conception et le contrôle de systèmes complexes grâce aux caractéristiques de ses entités.

Les services Cloud considérés comme des solutions partielles qui doivent être composées afin de fournir aux utilisateurs un service virtuel. Ce service composé doit être automatisé et dynamique pour pouvoir répondre rapidement aux besoins des utilisateurs, mais dans certains cas, une faute peut survenir. Afin de remédier à ce problème, il est indispensable d'appliquer des techniques de tolérance aux pannes qui, se réfère à un fonctionnement correct et continu même en présence de composants défectueux. C'est l'art et la science de la construction de failles. Un système tolérant aux pannes peut être capable de tolérer un ou plusieurs types de pannes, y compris les défauts matériels transitoires, intermittents ou permanents, les erreurs de logiciel et de conception, les erreurs de l'opérateur ou les perturbations induites ou les dommages physiques. [1]

La tolérance aux pannes dans les systèmes répartis est un domaine qui a été très largement étudié depuis une trentaine d'années. Diverses techniques peuvent être utilisées pour résoudre les fautes comme le recouvrement d'erreur par reprise.

Dans ce mémoire, nous avons proposé une méthode qui se base sur la notion de recouvrement arrière et la planification multi-agents ainsi que la sélection des chemins minimaux nécessaires à partir d'un graphe représentant les différents plans possibles d'exécution pour un-service composé demandé.

Notre architecture est basée sur les SMAs car les agents ont la possibilité de résoudre les problèmes de manière indépendante et peuvent collaborer les uns avec les autres pour atteindre leurs objectifs. Nous supposons qu'on a deux agents : l'agent planificateur (AP) et l'agent contrôleur (AC). Dans le reste de ce rapport on va expliquer le rôle de ces agents.

Le reste de ce manuscrit est organisé comme suit :

- *Chapitre 1* : Ce chapitre introduit en détail les deux premiers domaines sur lesquels se base notre travail. Dans la première partie : nous présentons la notion d'agent, de système Multi agents et les concepts qui y sont rattachés. Et la deuxième partie fournit une vue globale sur le concept de planification.
- *Chapitre 2* : qui est organisé en deux sections. En premier lieu, nous présentons la notion de tolérance aux fautes. Et en deuxième lieu, nous présentons le Cloud Computing avec ses concepts.

- *Chapitre3* : qui présente l'architecture proposée de notre système ainsi que la méthode proposée en détail.
- Chapitre 4 : décrit les outils d'implémentations utilisés et expose un ensemble d'interfaces qui explique notre application.

Enfin, nous terminons avec une conclusion générale qui résume l'apport essentiel de notre travail et qui comporte quelques perspectives.

CHAPITRE1 : SMA ET PLANIFICATION

I. Introduction :

Depuis quelques années, les systèmes multi-agents (SMA) ont pris une place de plus en plus importante en informatique, que ce soit dans le domaine de l'intelligence artificielle, dans ceux des systèmes distribués, de la robotique, ou même dans la vie artificielle.
L'intelligence Artificielle classique modélise le comportement intelligent d'un seul agent, le passage du comportement individuel aux comportements collectifs pour combler les limites de l'Intelligence Artificielle (I.A) classique à résoudre des problèmes complexes a conduit à la distribution de l'intelligence sur plusieurs entités (agents).

Ce chapitre est divisé en deux parties ; la première partie s'articule autour tout ce qui est en relation avec les notions théoriques des SMA et les agents. La deuxième partie, se concentre plutôt sur la planification.

II. Partie I. Système Multi-Agents :

A. Agent:

1. Définition :

Un agent est une entité autonome, réelle ou abstraite, qui est capable d'agir sur elle-même et sur son environnement, qui, dans un univers multi-agents, peut communiquer avec d'autres agents, et dont le comportement est la conséquence de ses observations, de ses connaissances et des interactions avec les autres agents. [2]

Cette interaction est réalisée grâce aux capteurs et aux effecteurs. Les capteurs servent à détecter (ou à lire) le contenu de l'environnement de façon partielle ou semi-partielle. Les effecteurs modifient ce contenu en concordance avec les objectifs de l'agent. [3] (cf. figure 1)

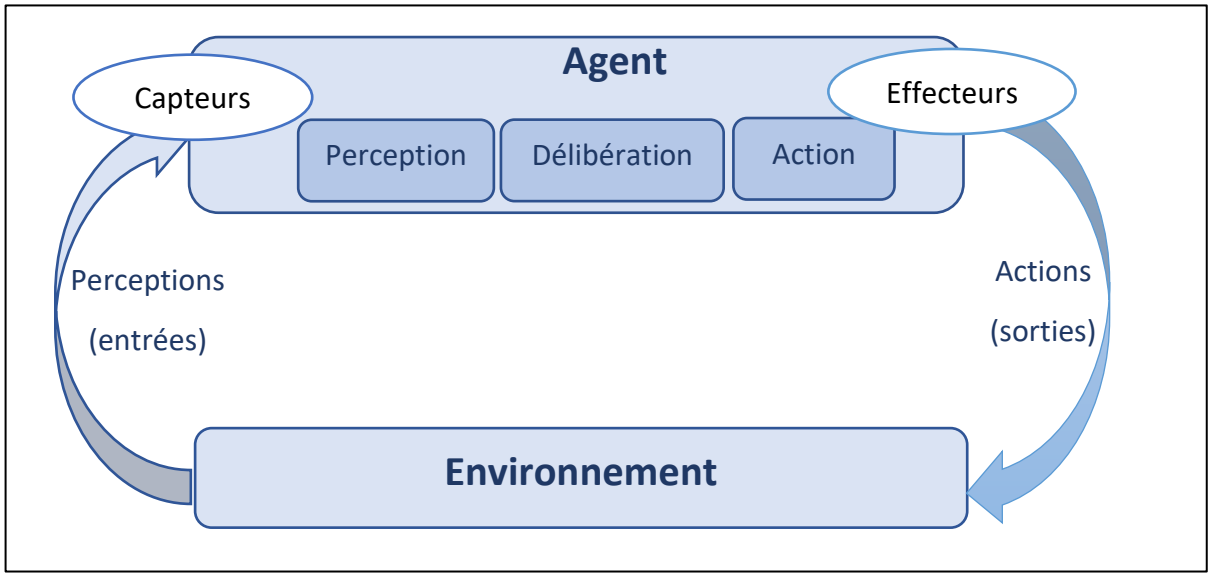

Figure 1: Interaction agent-environnement.

L'agent :

a) Est capable d'agir dans un environnement,

b) Peut communiquer directement avec d'autres agents,

c) Est mue par un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'il cherche à optimiser),

d) Possède des ressources propres,

e) Est capable de percevoir (mais de manière limitée) son environnement,

f) Ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune),

g) Possède des compétences et offre des services,

h) Peut éventuellement se reproduire. [4]

2. Caractéristiques d'un agent :

Un agent est un système informatique, situé dans un environnement, et qui agit d'une façon autonome et flexible pour atteindre les objectifs pour lesquels il a été conçu.

Les notions "***situé***", "***autonomie***" et "***flexible***" sont définies comme suit :

- **Situé :** l'agent est capable d'agir sur son environnement à partir des entrées sensorielles qu'il reçoit de ce même environnement. [2]
- **Autonome :** l'agent doit pouvoir prendre des initiatives et agir sans l'intervention d'un tiers (humain ou agent) et contrôler ses propres actions ainsi que son état interne. [2] [4]
- **Flexible :** l'agent dans ce cas est : capable de répondre à temps, l'agent doit être capable de percevoir son environnement et élaborer une réponse dans les temps requis, c.-à-d. L'agent réagit aux changements dans l'environnement et s'adapte selon la disponibilité des ressources. [4]

En plus, on peut définir d'autres caractéristiques :

- **Proactif :** l'agent doit exhiber un comportement proactif et opportuniste, génère et satisfait des buts, tout en étant capable de prendre l'initiative au "bon" moment. [2]
- **Social :** l'agent doit être capable d'interagir avec les autres agents (logiciels et humains) quand la situation l'exige afin de compléter ses tâches ou aider ces agents à accomplir les leurs. [2]
- **Interactivité :** l'agent doit pouvoir exercer des actions sur son environnement et réciproquement. [5]
- **Réactivité :** l'agent doit être capable de percevoir son environnement et d'élaborer une réponse dans un temps requis. Par exemple un agent de sauvegarde doit exécuter sa tâche suite à un événement survenu dans son environnement et qui lui demande de sauvegarder. [4]

- **Capacité d'apprendre :** un agent aura la capacité d'apprendre s'il sait acquérir de la connaissance, de l'information ou des habitudes. [5]
- **Coordination :** l'agent aura la capacité de coordonner ses actions par rapport à un utilisateur ou un autre agent. [5]
- **Compétition :** l'agent est capable d'agir dans un environnement où d'autres agents interviennent, le but est le même pour tous les agents présents mais un seul l'atteindra, les autres échoueront. [5]
- **Mobilité :** généralement on parle de deux types de mobilité d'agent : la mobilité relative ou bien par requête (il n'y a pas un réel déplacement de l'agent) et la mobilité réelle (le processus agent se déplace sur le réseau d'un serveur à un autre). [5] [6]
- **Délégation :** ce mécanisme permet de donner des autorisations à un agent, il accomplit un ensemble de tâches à la demande d'un utilisateur ou d'un autre agent. [5] [6]
- **Communication :** l'agent a des capacités d'interaction avec l'utilisateur au moyen d'interface utilisateur ou bien inter-agents.

3. Types d'agents :

On peut classer les agents selon leur comportement et leur granularité ; elle exprime la complexité de raisonnement d'un agent afin de séparer les agents dits intelligents et des agents moins intelligents. On parle d'agents ***cognitifs*** et d'agents ***réactifs***. [4]

a) Agent cognitif :

Les agents cognitifs peuvent anticiper, prévoir le futur, mémoriser des choses ..., etc. ils réfléchissent. Ce type d'agent est intelligent par lui-même c'est-à-dire qu'il peut effectuer un certain raisonnement pour choisir ses actions [4]. Les agents cognitifs disposent d'une base de connaissances comprenant les diverses informations liées à leurs domaines d'expertise et à la gestion des interactions avec les autres agents et leur environnement. Les agents sont généralement intentionnels c'est-à-dire qu'ils possèdent des buts et des plans explicites leur permettant d'accomplir leurs buts. [7]

b) Agent réactif :

Un agent réactif réagit directement à l'environnement perçu. Un tel agent se contente simplement d'acquérir des perceptions et de réagir à celles-ci en appliquant certaines règles prédéfinies. Il possède une représentation très simplifiée (partielle mais sophistiquée) de son environnement, a des buts explicites et est capable de planifier son comportement, de mémoriser ses actions passées, de communiquer par envoi de messages ou via des langages d'interaction élaborés, de négocier, etc. [4] [8]

Le tableau ci-dessous (tableau 1) montre la différence entre les deux types d'agent cités auparavant :

Systèmes d'agent cognitifs	Systèmes d'agent réactifs
Représentation explicite de l'environnement	Pas de représentation explicite
Peut tenir compte de son passé	Pas de mémoire de son historique
Agents complexes	Fonctionnement stimulus/action
Petit nombre d'agents	Grand nombre d'agent

Tableau 1:Différence entre un agent cognitif et un agent réactif [4]

c) Agent hybride :

Un agent hybride est le fruit de la combinaison des deux d'agents cités ci-dessus. Chaque agent hybride est caractérisé par la notion de couches (Voir Figure 2), et chaque couche représente soit les agents cognitifs, soit les agents réactifs. Un agent hybride est composé de plusieurs couches arrangées selon une hiérarchie. [4]

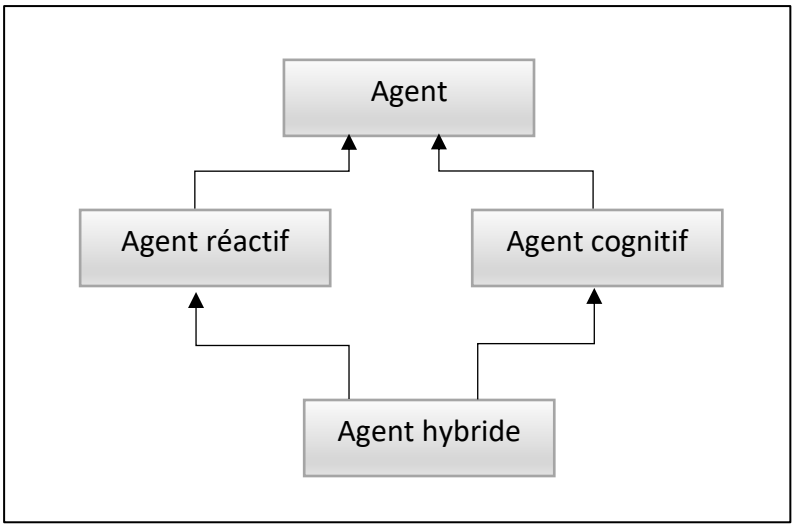

Figure 2: Structure des agents hybrides [4]

4. Communication entre agents :

La communication représente la forme la plus courante d'interaction dans les systèmes multi-agents. Elle permet l'échange des informations entre les agents, un acte de communication est défini comme étant l'envoi d'un message d'un émetteur à un ou plusieurs récepteurs.

Les communications dans les SMA, comme chez les humains, sont à la base des interactions et de l'organisation sociale. Sans communication, l'agent n'est qu'un individu isolé. C'est parce que les agents communiquent qu'ils peuvent coopérer, coordonner leurs actions et réaliser des tâches en commun. [8]

B. Les systèmes multi-agents (SMA) :

1. Définition :

On appelle système multi-agent (**SMA**), un système composé des éléments suivants :

1. Un environnement *E*, c'est-à-dire un espace disposant généralement d'une métrique.
2. Un ensemble d'objets *O*. Ces objets sont situés, c'est-à-dire que, pour tout objet, il est possible, à un moment donné, d'associer une position dans E. Ces objets sont passifs, c'est-à dire qu'ils peuvent être perçus, créés, détruits et modifiés par les agents.
3. Un ensemble *A* d'agents, qui sont des objets particuliers (A inclut O), lesquels représentent les entités actives du système.
4. Un ensemble de relations *R* qui unissent des objets (et donc des agents) entre eux.
5. Un ensemble d'opérations *Op* permettant aux agents de A de percevoir, produire consommer, transformer et manipuler des objets de O.
6. Des opérateurs chargés de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification, que l'on appellera les lois de l'univers. [4]

Les systèmes multi-agents mettent en œuvre des agents homogènes et hétérogènes ayant des buts communs ou distincts. Ils sont destinés à travailler ensemble afin de résoudre des problèmes qui dépassent leurs capacités ou connaissances individuelles et profitent des divers rôles de chacun des agents de systèmes.

Un système multi-agent est un système distribué composé d'un ensemble d'agents qui interagissent le plus souvent, selon des modes de coopération, de concurrence ou de coexistence. [4]

2. Caractéristiques d'un SMA :

Un SMA est généralement caractérisé par :

- Chaque agent a des informations ou des capacités de résolution de problèmes limitées.
- Chaque agent a un point de vue partiel.
- Il n'y aucun contrôle global du système multi-agents.
- Les données sont décentralisées
- Le calcul est asynchrone.

3. Architecture du SMA :

a) Organisation centralisée :

Dans une conception centralisée, un agent connait tous les autres agents.
Ainsi lorsque l'on a besoin d'une compétence particulière ou d'un agent particulier on s'en réfère à cet agent pour connaitre le/les agents concernés. (Voir figure 3)

Dans ce type d'organisation :

- Les agents ont tous la même structure : état et comportement.

- Emergence ou d'Intelligence Collective.
- Pas de communication directe
- Interaction via le partage d'un même espace de travail

Un SMA à contrôle centralisé ou à base de tableau noir est composé de trois éléments :

- Les connaissances
- Le tableau noir
- Le mécanisme de contrôle

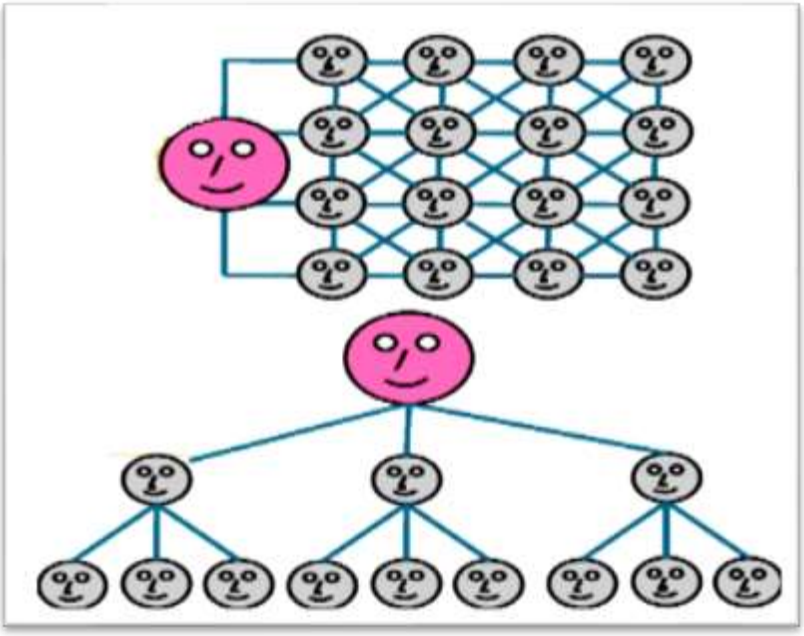

Figure 3:Architecture centralisé d'un SMA [4]

b) Organisation non centralisée (libre-Distribué) :

Ce type d'architecture est caractérisé par :

- Aucun agent ne connait tous les agents. Localement, un agent peut connaitre les agents qu'il est susceptible de traiter, mais personne n'a, a priori, de vision globale.
- Personne ne connait personne a priori et c'est en dialoguant que l'on trouve les autres.
- Les agents ont des structures différentes. (voir figure 4)
- Distribution totale des connaissances et du contrôle.
- Traitement local.
- Communication par envoi de message. (voir figure 4)

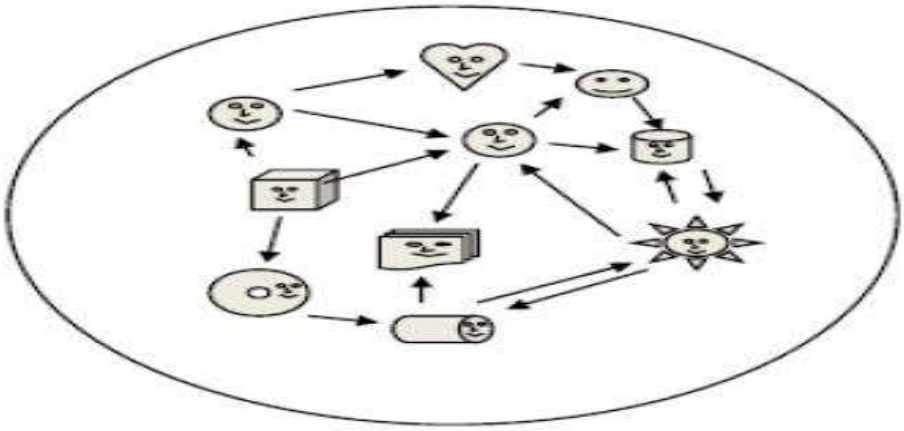

Figure 4:Architecture libre (non centralisée) du SMA [4]

4. Domaines d'applications des SMA :

Les systèmes multi-agents sont convenables, généralement, pour le développement des systèmes complexes et les systèmes répartis et hétérogènes où la solution est le résultat de l'interaction de plusieurs entités. Les systèmes multi-agents ont été appliqués dans les domaines suivants :

- **Simulation de phénomènes complexes :** dont l'objectif est la modélisation de phénomènes du monde réel, afin d'observer, de comprendre et d'expliquer leur comportement et leur évolution. On utilise les SMA pour simuler des interactions existantes entre agents autonomes, on cherche à déterminer l'évolution de ce système afin de prévoir l'organisation qui en résulte.
- **Les applications dans lesquelles les agents jouent le rôle d'êtres humains :** dans les systèmes de ventes aux enchères dans laquelle les agents jouent les rôles de commissaire-priseur et d'acheteurs, représentent une classe d'applications de cette catégorie.
- **La résolution distribuée de problèmes :** mettre en œuvre un ensemble de techniques pour que des agents, pertinents pour la résolution d'une partie ou l'ensemble du problème. Elle étudie comment distribuer des compétences au niveau de chaque partie du système, de façon à ce qu'il soit globalement plus compétent que chacune de ses parties.
- **La gestion des processus métier :** les décisions prises dans les grandes entreprises sont basées sur des informations provenant de plusieurs sources. Le défi réel est de prendre des décisions correctes avec des informations incomplètes ou obsolètes. Plusieurs recherches trouvent que les systèmes multi-agents sont idéaux pour travailler dans ces conditions.
- Le domaine médical.
- Les systèmes d'informations coopératifs.
- Sociologie, physique des particules.
- Chimie, Robotique.
- Biologie cellulaire, Ethologie, Ethnologie, …
- IAD pour la résolution de problèmes complexes.

- Systèmes parallèles et distribués.
- Robotique, Productique.
- Télécoms.
- Jeux vidéo, Cinéma (Animations 3D).
- Les agents ont des structures différentes
- E-commerce (Assistance du commerce électronique), E-learning.
- Urbanisation.
- Gestion du trafic routier.
- Finance : Trading automatique.
- Web Sémantique.
- Internet des objets (Web 3.0).

5. Avantages des SMA :

- **La modularité :** permet de rendre la programmation plus simple. Elle permet, de plus, aux systèmes multi-agents d'être facilement extensibles, parce qu'il est plus facile d'ajouter de nouveaux agents à un système multi-agent que d'ajouter de nouvelles capacités à un système monolithique.
- **La vitesse :** est principalement due au parallélisme, car plusieurs agents peuvent travailler en même temps pour la résolution d'un problème.
- **La fiabilité :** peut être également atteinte, dans la mesure où le contrôle et les responsabilités étant partagés entre les différents agents, le système peut tolérer la défaillance d'un ou de plusieurs agents. Si une seule entité contrôle tout, alors une seule défaillance de cette entité fera en sorte que tout le système tombera en panne (dû à la redondance).

Ils héritent aussi des bénéfices envisageables de l'Intelligence Artificielle comme :

- **Le traitement symbolique** (au niveau des connaissances) ;
- **La facilité de maintenance** ;
- **La réutilisation et la portabilité** mais surtout, ils ont l'avantage de faire intervenir des schémas d'interaction sophistiqués.
- **Les types courants d'interaction** incluent la coopération, la coordination et la négociation. [7]

III. Partie II. La planification :

La planification est un domaine central de l'intelligence artificielle qui est défini par le processus de délibération explicite qui choisit et organise des actions en anticipant leurs résultats attendus le mieux possible. [1]

La planification multi-agent tente de coordonner les actions de plusieurs agents de manière à ce qu'un but commun soit atteint (exemples classiques : la construction d'une maison, l'assemblage de composants...). Du fait que les agents établissent des plans qui spécifient leurs actions et interactions futures en fonction d'un certain objectif, tous les agents impliqués dans la planification s'entendent pour se comporter suivant ces plans.

La planification dans un SMA peut se décomposer en trois étapes : **la construction de plans**, la **synchronisation de plans (coordination)** et **l'exécution de plans**. Comme il est possible d'utiliser un ou plusieurs agents à chacune de ces étapes, il est possible d'obtenir un grand nombre d'organisations différentes. Un système de planification est alors composé d'un ensemble d'agents pouvant **planifier**, **synchroniser** ou **exécuter** des plans, un même agent pouvant accomplir une ou plusieurs tâches (voir figure 5). [7]

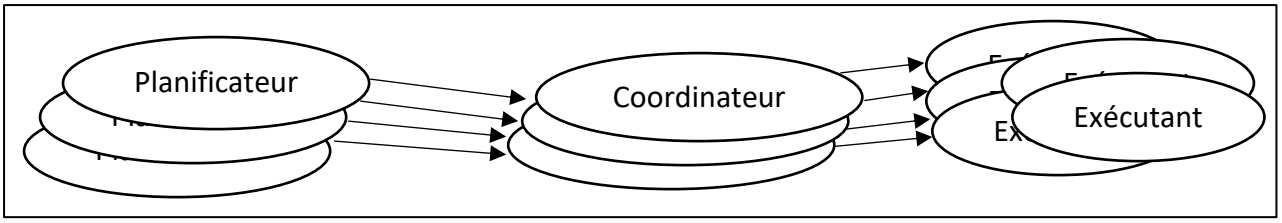

Figure 5:Système de planification multi-agent (exemple). [7]

1. Coordination d'actions et résolution de conflits :

La coordination peut se faire selon deux approches : la planification ou la négociation. Dans ce chapitre on s'intéresse à la première approche.

Pour la planification on a 2 types :

a) Planification centralisée :

Dans la planification centralisée, un agent central ayant une vue globale du système gère les conflits entre agent en établissant des plans selon un mécanisme de gestion des conflits

Principaux défauts :

- Besoin d'une vue globale,
- Coûteuse si l'agent superviseur est défaillant.

b) Planification distribuée :

Dans ce type de planification, chaque agent a son propre plan partiel, se communiquent leurs plans.

- Méthode coûteuse en communication.

2. Cadre général de la planification classique :

Un problème de planification classique se modélise sous la forme d'un système de transition d'états déterministes. Dans ce système de transition d'états, chaque état correspond à une représentation abstraite d'une situation à un instant donné dans l'environnement réel. De même, les transitions entre ces états correspondent aux actions applicables dans chacune de ces situations, et chaque action peut être associée à un coût, c'est-à-dire à une pénalité liée à son exécution, et qui est modélisée numériquement. Ces différents éléments constituent une sous-partie du problème de planification appelé « **domaine de planification** », et qui est formellement définie de la façon suivante : [9]

a) **Définition 1 (Domaine de planification** classique**).**

Un domaine de planification classique est un quadruplet D = (S, A, T, C) tel que :

- S est un ensemble fini d'états du monde.
- A est un ensemble fini d'actions.
- T ∈ S×A → S est une fonction de transition. Il s'agit d'une fonction partiellement définie sur S×A. Pour tout état s ∈ S et action a ∈ A, on dira que a est applicable à s si et seulement si T (s, a) est défini.
- C ∈ S × A → R+ est une fonction de coût. Le domaine de définition de cette fonction est identique à celui de la fonction T.

Pour compléter le modèle d'un problème de planification, on y adjoint un état initial, qui correspond à la situation dans laquelle se trouve l'agent, et on définit un but (aussi appelé objectif), correspondant à un ensemble d'états que l'on souhaite atteindre

b) Définition 2 (Problème de planification classique)

Un problème de planification classique est un triplet P = (D, s0 ,G) tel que :

- D est un domaine de planification classique.
- s0 ∈ S est l'état initial.
- G ⊆ S est l'ensemble des états correspondant au but à atteindre.

Une solution à ce problème est un plan, c'est-à-dire une séquence d'actions.

c) Définition 3 (Séquence d'actions exécutable)

Soient σ ∈ A∗ une séquence d'actions et un état s ∈ S. σ est exécutable en s si et seulement si :

1. σ = ϱ.
2. Il existe a ∈ A applicable à s et σ' ∈ A∗ exécutable en T (s, a) tels que σ = a ◦ σ'.

Pour le coût de a ◦ σ :

C(s, a ◦ σ) = C(s, a) + C(T (s, a), σ).

Une séquence d'actions est alors qualifiée de plan solution à un problème de planification, si et seulement si elle est exécutable dans l'état initial, et l'état final qu'elle atteint correspond au but spécifié.

d) Définition 4 (Solution à un problème de planification classique)

Soient P = (D, s0,G) un problème de planification classique, et σ ∈ A∗ une séquence d'actions. σ est un **plan solution** de P si et seulement si :

- σ est exécutable dans s0.
- T (s0, σ) ∈ G.

Parmi l'ensemble **des plans solutions** à un problème de planification, certains de ces plans sont moins coûteux que tous les autres plans solutions. On considère que ces plans sont des solutions optimales au problème de planification posé

e) Définition 5 (Solution optimale à un problème de planification classique)

Soient P = (D, s0, G) un problème de planification classique, et $\sigma* \in A*$ un plan solution de P. $\sigma*$ est une solution optimale de P si et seulement si :

$$C(s0, \sigma*) = \min\{C(s0, \sigma) \mid \sigma \text{ est une solution de P}\}$$

La figure 6 permet d'illustrer les différents éléments introduits dans un problème de planification qui est illustré sous la forme d'un système de transition d'états.

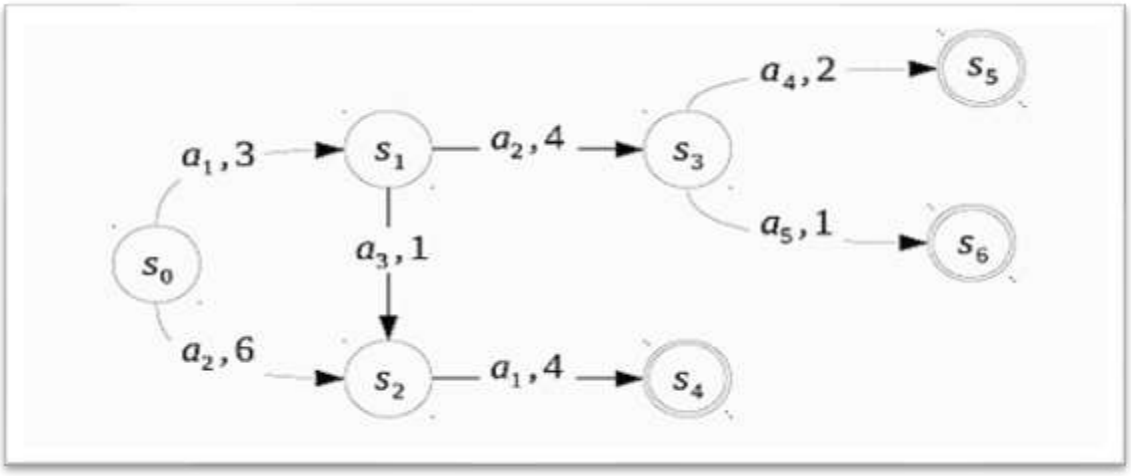

Figure 6:Représentation graphique d'un problème de planification [9]

s0 est l'état initial, tandis que s4, s5 et s6 sont des états correspondant au but à atteindre. Les séquences d'actions a1 ◦ a2 ◦ a4, a1 ◦ a2 ◦ a5, a1 ◦ a3 ◦ a1 et a2 ◦ a1 sont toutes des plans solutions, mais seules a1 ◦ a2 ◦ a5 et a1 ◦ a3 ◦ a1 sont des solutions optimales avec un coût total égal à 8, voir la figure 6.

IV. Conclusion :

Dans ce chapitre, nous avons présenté un état de l'art sur les systèmes multi-agents et la planification. La première partie de ce chapitre a été consacré à, la définition du concept d'agent, ainsi que la discussion des différents caractéristiques de ce dernier, puis nous avons présenté les types des agents existants qui sont classés en général en deux groupes : les agents cognitifs et les agents réactifs, à partir de deux types, nous avons distingué un autre type qui combine entre les deux, appelés « agents hybride ».Nous avons indiqué, également, qu'un système où évoluent plusieurs agents est appelé un **système multi agents** où les agents peuvent interagir entre eux à travers un ensemble d'évènements pendant lesquels les agents sont en relation les uns avec les autres soit directement, soit par le biais de l'environnement.

Dans la deuxième partie ; Nous avons défini le concept de planification, ensuite nous avons donné le cadre général de la planification classique. Enfin, nous avons terminé par une conclusion.

CHAPITRE2 : La Tolérance aux fautes et le Cloud

V. Introduction :

Le Cloud Computing fait référence à l'utilisation des capacités de calcul des ordinateurs distants, où l'utilisateur dispose d'une puissance informatique considérable sans avoir à posséder des unités puissantes. Il désigne un accès aux ressources informatiques qui sont quelque part, á travers internet. Depuis longtemps, tout le monde utilise le Cloud Computing sans même s'en rendre compte. Par exemple lorsque l'on utilise son mail, Hotmail, Gmail ou autre, nous faisons du Cloud. Les entreprises achètent ainsi des capacités qui sont par la suite facturées un peu comme le sont l'eau, le gaz ou l'électricité.

La probabilité de voir une panne, qui peut intervenir durant l'exécution d'une tâche donnée, est forte. La capacité d'un système à fonctionner malgré une défaillance d'une de ses composantes est appelée tolérance aux pannes (parfois nommée tolérance aux fautes, en anglais *fault tolerance*). Puisqu'il est impossible d'empêcher totalement les pannes, une solution courante consiste à mettre en place des mécanismes de redondance, en dupliquant les ressources critiques, ou des techniques de sauvegarde et de retour en arrière. [10] [11]

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

Dans ce chapitre nous présentons deux domaines qui ont une relation directe avec notre travail : La tolérance aux fautes et le Cloud. Dans un premier lieu, Nous allons donner les différents concepts liés aux tolérances aux pannes comme la sûreté de fonctionnement, ainsi que les techniques et mécanismes utilisées pour la gestion de fiabilité. Dans un deuxième lieu, nous introduisons ce qu'est le Cloud Computing, en expliquant ses différentes caractéristiques, ses composants, ses modèles de services ainsi que ces modèles de déploiement tout en citant quelques avantages et inconvénients de cette technique.

VI. Partie I. La tolérance aux fautes :

A. Sûreté de fonctionnement :

La sûreté de fonctionnement d'un système informatique est la propriété qui permet à ses utilisateurs de placer une confiance justifiée dans le service qu'il leur délivre. Mettre en œuvre la sûreté de fonctionnement d'un système correspond à lutter contre les défaillances du système. Cette notion se définit selon 3 axes différents : (voir figure 7) [12]

- *Ses attributs* : sont l'ensemble des propriétés que le système doit respecter pour rester sûr en fonctionnement.
- *Ses entraves* : sont les causes qui empêchent le système de fonctionner correctement.

- *Ses moyens* : sont les différentes possibilités qui permettent au système de rester sûr, même sous la contrainte d'un certain nombre d'entraves.

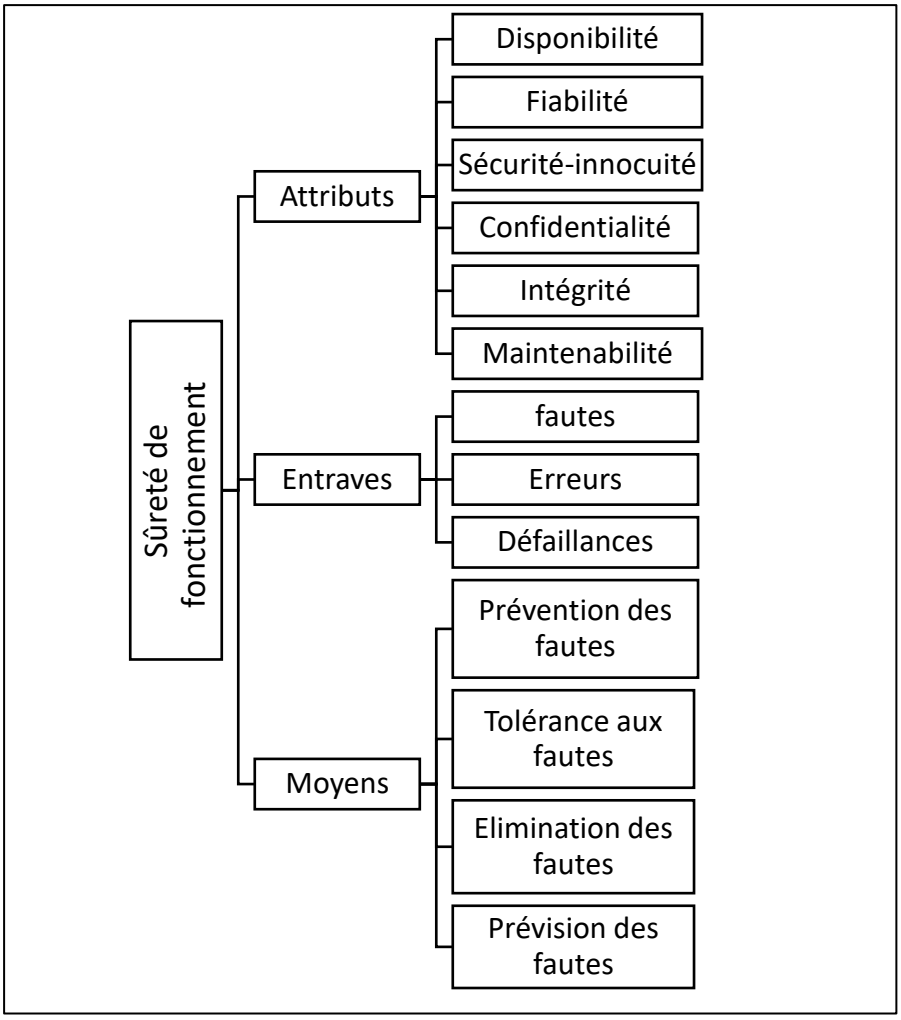

Figure 7:L'arbre de la sûreté de fonctionnement

1. Attributs de la sûreté de fonctionnement :

La sûreté de fonctionnement peut être considérée selon des points de vue différents mais complémentaires, ce qui permet de définir les attributs de la sûreté de fonctionnement, ces derniers correspondent aux propriétés que doit vérifier un système. Ces attributs permettent d'évaluer la qualité de service fournie par le système : [11] [12]

- **La disponibilité** : correspond à la capacité du système à être prêt à délivrer le service.
- **La fiabilité** : correspond à la continuité de service
- **La sécurité-innocuité :** correspond à l'évitement de conséquences catastrophiques sur l'environnement.

- **La sécurité** : correspond à la préservation de la confidentialité et de l'intégrité des informations.
- **La confidentialité** : correspond à l'absence de divulgation non autorisée de l'information.
- **L'intégrité** : indique l'absence d'altérations inappropriées de l'informations.
- **La maintenabilité** : correspond à l'aptitude aux réparations et aux évolutions.

2. Entraves à la sûreté de fonctionnement :

Lorsque le service délivré dévie du service spécifié, la spécification étant une description agréée (par exemple entre le fournisseur et l'utilisateur) du service attendu. Les entraves à la sûreté de fonctionnement sont en trois types : *les fautes, les erreurs, et les défaillances.* Ces trois types sont liées par des relations de causalité : (voir figure 8)

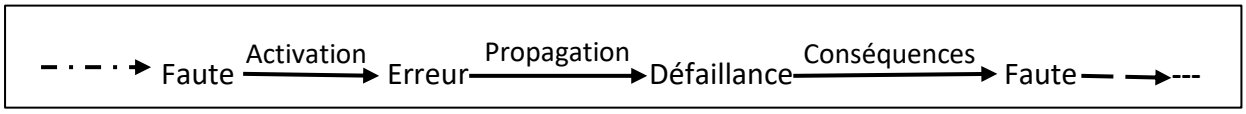

Figure 8:Relations entre fautes, erreurs et défaillances [1]

- La défaillance survient parce que le système a un comportement erroné.
- Une erreur est la partie de l'état du système (par rapport au processus de traitement) qui est susceptible d'entraîner une défaillance.
- La cause adjugée ou supposée de l'erreur est une faute.
- Une erreur est donc la manifestation d'une faute dans le système.
- Une défaillance est donc l'effet d'une erreur sur le service.
- Une faute est active lorsqu'elle produit une erreur.

3. Moyens de la sûreté de fonctionnement :

Les moyens de la sûreté de fonctionnement sont les méthodes, outils et solutions qui permettent de fournir au système l'aptitude à délivrer un service conforme au service spécifié (obtention de la sûreté de fonctionnement), et de donner confiance dans cette aptitude (validation de la sûreté de fonctionnement).

La conception et la réalisation d'un système informatique sûr de fonctionnement passent par l'utilisation combinée d'un ensemble de méthodes qui peuvent être classées de la manière suivante : [12]

- **Prévention des fautes :** comment empêcher, par construction, l'occurrence ou l'introduction de fautes.
- **Tolérance aux fautes :** comment fournir, par redondance, un service conforme à la spécification, en dépit des fautes.
- **Elimination des fautes :** comment réduire, par vérification, la présence de fautes (leur nombre et leur gravité).

- **Prévision des fautes :** comment estimer, par évaluation, la présence et la création des fautes et leurs conséquences.

B. La tolérance aux fautes:

1. Définition :

La tolérance aux fautes est la qualité qui permet d'assurer la délivrance d'un service correcte en dépit de fautes affectant les différentes ressources du système. Pour assurer le fonctionnement, il faut donc être capable de détecter les fautes et de diagnostiquer précisément leurs origines pour permette ensuite de délivrer, malgré les fautes, au mieux le service. [1]

2. Etapes de la tolérance aux fautes

Plusieurs phases successives, non obligatoirement toutes présentes, font partie d'un processus de tolérance aux fautes :

- *Détection :* Découvrir l'existence d'une faute (état incorrect) ou d'une défaillance (comportement incorrect).
- *Localisation :* Identifier le point précis (dans l'espace et le temps) où l'erreur (ou la défaillance) est apparue.
- *Isolation :* Confiner l'erreur pour éviter sa propagation à d'autres parties du système.
- *Réparation :* Remettre le système en état de fournir un service correct. Le composant défectueux est identifié et le système fonctionne comme si les composants défectueux ne sont pas utilisés ou sont utilisés d'une façon telle que la faute ne cause pas désormais une défaillance. [13]

3. Classement des pannes :

La capacité d'identification du modèle de pannes joue un rôle très important si l'on veut réaliser la tolérance aux pannes. En effet, les conséquences et le traitement d'une panne diffèrent selon son modèle. Les pannes peuvent être classées selon plusieurs critères : leur degré de gravité, leur degré de permanence et leur nature. [12] [14]

a) Classement selon le degré de gravité :

Le classement des pannes selon leur degré de gravité des défaillances conduit à des :

- **Pannes franches (crash fault) :** 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 chaque fois que cela est possible.
- **Pannes par omission :** des messages sont perdus en entrée et/ou en sortie. Ce type de panne est utilisé pour les réseaux.
- **Pannes de temporisation :** les déviations par rapport aux spécifications concernent uniquement le temps (typiquement le temps de réaction à un événement)

- **Pannes par valeurs :** les résultats produits par un composant défaillant sont incorrects.
- **Pannes byzantines :** le système peut faire n'importe quoi, y compris avoir un comportement malveillant.

b) Classement selon le degré de performance :

Un autre classement des pannes selon leur degré de permanence est également possible :

- **Panne transitoire :** elle se produit de manière isolée.
- **Panne discontinue :** elle se produit aléatoirement plusieurs fois.
- **Panne permanente :** elle persiste dès qu'elle apparaît jusqu'à réparation.

c) Classement selon la nature de la panne :

On utilise également un troisième classement selon la nature de la panne :

- **Pannes accidentelles :** elles se produisent de manière accidentelle ;
- **Pannes intentionnelles :** elles sont créées avec une intention qui peut être malicieuse.

4. Mécanismes de la tolérance aux fautes :

L'objectif de la tolérance aux fautes est la préservation de la continuité du service malgré l'apparition de fautes qui s'effectue le plus souvent en deux temps : [11]

- *La détection d'erreurs* qui provoque la levée d'un signal ou d'un message d'erreur à l'intérieur du système.
- Et, *Le rétablissement du système*, qui est déclenché par le signal ou message, substitue un état jugé exempt d'erreurs et de fautes à l'état erroné détecté.

a) Mécanismes de détection d'erreurs :

Ils permettent de détecter la présence d'erreurs dans le système après l'activation d'une faute. Il existe deux catégories de mécanismes de détection d'erreurs :

- *La détection préventive :* elle a lieu pendant des phases particulières de l'activité du système, comme les phases de démarrage ou de maintenance par exemple. Couramment, elle consiste à vérifier des propriétés à l'aide de test.
- *La détection concurrente* : elle a lieu pendant l'exécution normale du système. Elle a pour l'objectif de détecter la présence d'une erreur afin d'y remédier. Dans un système compositionnel, les erreurs détectées sont le plus souvent les défaillances des composants constitutifs du système. [1]

b) Mécanismes de rétablissement du système :

Le rétablissement du système peut être assuré par deux méthodes complémentaires : *le traitement d'erreur*, qui vise à corriger l'état erroné du système, et *le traitement de faute*, qui vise à empêcher la faute à l'origine de l'erreur d'être à nouveau activée. [11] [15]

Parmi les techniques de *traitement d'erreur*, on distingue :

- *Le reprise d'erreur :* qui consiste à sauvegarder périodiquement l'état du système pour le ramener dans un état antérieur à la détection d'erreur ; les principaux inconvénients de la reprise sont la taille des sauvegardes, la difficulté d'effectuer des sauvegardes cohérentes, et le surcoût temporel nécessaire à leur établissement.
- *Le rétablissement par poursuite :* qui consiste à créer un nouvel état à partir duquel le système peut continuer à fonctionner de façon acceptable, éventuellement en mode dégradé ; une forme limite de cette technique est la mise du système dans un état sûr, par exemple en l'arrêtant.
- *La compensation d'erreurs :* qui consiste à utiliser des redondances présentes dans le système pour fournir un service correct en dépit des erreurs ; cette compensation peut être consécutive à une détection d'erreur (détection et compensation), ou appliquée systématiquement (masquage d'erreur).

Le traitement de faute est composé de quatre phases successives.

- *Le diagnostic* qui a pour but d'identifier la localisation et le type de la faute responsable de l'état erroné du système.
- *L'isolement* qui consiste à exclure la participation du composant erroné de la délivrance du service, par moyen physique ou logiciel.
- *La reconfiguration* qui cherche à compenser l'isolement du composant défaillant, soit en basculant sur des composants redondants, soit en réassignant ses tâches à d'autres composants.
- Et enfin*, La réinitialisation* qui consiste à vérifier et à mettre à jour la nouvelle configuration du système.

5. Techniques de tolérance aux fautes :

Le but des techniques de tolérance aux fautes est d'éviter les défaillances du système en présence d'erreurs qui demeurent après le développement. En plus, éliminer les fautes et leur effet sur l'état interne d'un logiciel avant qu'une défaillance ne se produise.

La tolérance aux pannes dans les systèmes répartis et parallèles est, le plus souvent, réalisée par l'emploi d'un mécanisme de redondance ou reprise. Les mécanismes de redondances appartiennent à deux catégories : les techniques basées sur la duplication et les techniques basées sur une mémoire stable [14]. La figure 9 résume les techniques de tolérance aux pannes et leurs types :

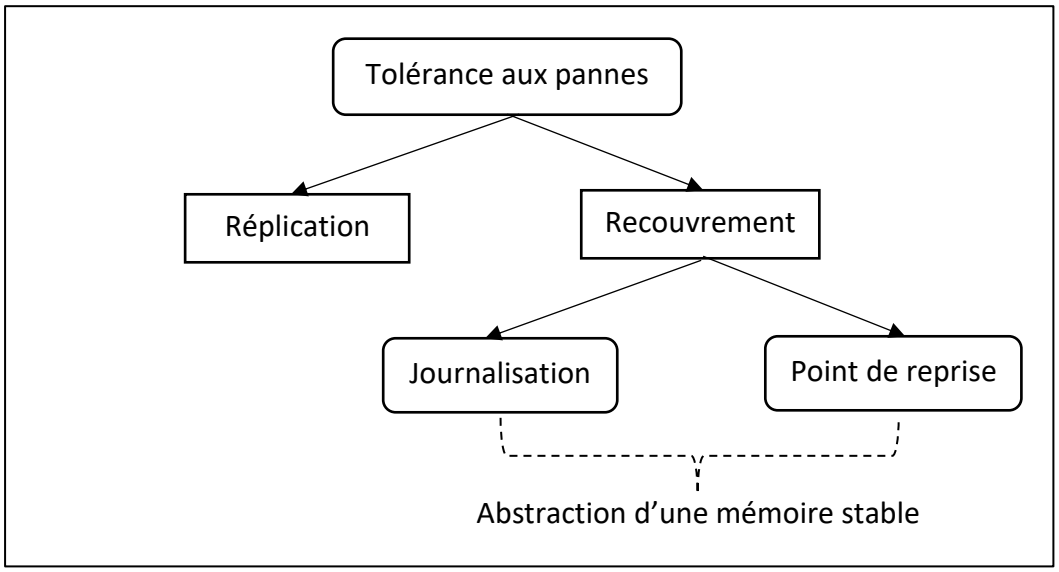

Figure 9:Les techniques de tolérance aux pannes dans les systèmes répartis

a) Tolérance aux fautes par duplication :

La tolérance aux fautes basée sur la duplication consiste à créer des copies multiples des processus sur des machines différentes. Lors de la panne d'un processus, il est remplacé par une de ses copies. La duplication dans un système réparti présente plusieurs avantages, elle apporte une meilleure sûreté de fonctionnement en augmentant la disponibilité et la fiabilité des données.

Trois approches fondamentales du problème de la tolérance aux fautes, basées sur la duplication sont proposées dans la littérature : la duplication active, la duplication passive et la duplication hybride à base de la duplication passive et active. [14] [16]

- ***Duplication active*** *:* la duplication active peut être utilisée de manière efficace pour masquer la faute de plusieurs composants matériels (capteurs, processeurs, média de communication et actionneurs) dans un système distribué. Elle désigne les stratégies dans lesquelles toutes les copies jouent un rôle identique. Toutes les copies reçoivent la même séquence ordonnée de requêtes, qui sont toutes traitées dans le même ordre. Cette stratégie évite d'utiliser des points de reprise couteux. En revanche, elle nécessite un mécanisme de diffusion atomique et requiert que l'exécution des requêtes soit déterministe pour garantir la cohérence.

- ***Duplication passive :*** dans ce type de duplication, on distingue la copie primaire et les copies secondaires. La copie primaire est la seule qui reçoit les requêtes et qui effectue toutes les opérations. La perte d'un message, dans le cas de la duplication passive des communications, à cause des fautes de déconnexion, peut être tolérée par la retransmission de ce message via des routes disjointes dans un réseau. Cette technique de duplication nécessite des mécanismes particuliers de détection de fautes, et le choix du primaire ainsi que son intégration après la réparation. Pour assurer la cohérence, la copie primaire diffuse son nouvel état aux copies secondaires après chaque modification. Cet état sert de point de reprise en cas de défaillance.

- ***Duplication semi-active:*** appelé aussi duplication hybride, une combinaison de la duplication active et passive, dans laquelle par exemple, on utilise la duplication active pour les tâches et la duplication passive pour les communications. à la différence la duplication active, les copies secondaires attendent une notification de la copie primaire avant de traiter la requête. Cette notification comporte les informations nécessaires qui permettent de résoudre le problème de l'indéterministe du traitement de requêtes.

b) Tolérance aux pannes par mémoire stable :

Une mémoire stable est considérée comme étant un support persistant de stockage, dont le rôle principal est *d'assurer une accessibilité et une protection des données contre les pannes* pouvant affecter le système. Lorsqu'une donnée est stockée sur la mémoire stable, on dit que cette donnée est stable.

Ainsi, suite à une panne, un état correct ayant été stocké antérieurement à cette panne sur la mémoire stable reste accessible, cela permet au système un retour à un état antérieur. La tolérance aux fautes par sauvegarde périodique de points de reprise sur mémoire stable est très aisément mise en œuvre dans le cas d'une application séquentielle (et donc non-répartie).

Les techniques de tolérances aux fautes basées sur la mémoire stable ont deux techniques : basées sur les points de reprise (Check-Point) ou celles basées sur les fichiers logs.

A. **Par point de reprise :** c'est une technique qui consiste à sauvegarder dans un support de stockage stable l'état d'un processus durant son exécution, qui est appelé ligne de recouvrement. En cas de panne, le processus est redémarré depuis son dernier checkpoint. (voir figure 10)

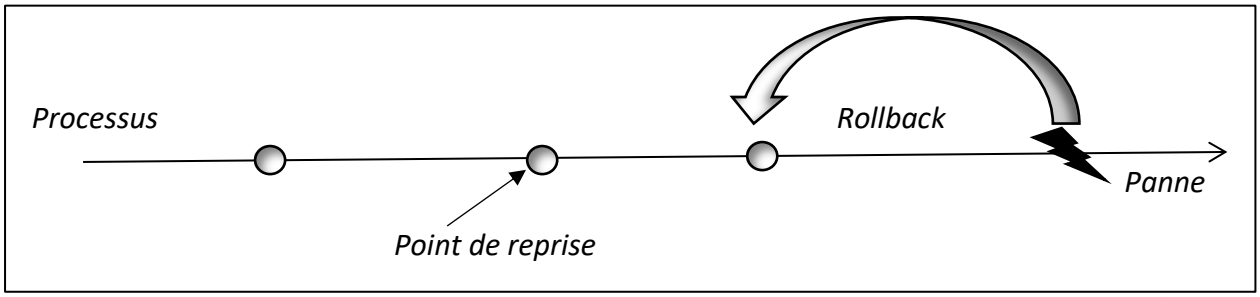

Figure 10:Tolérance aux pannes par recouvrement arrière (rollback-recovery) [16]

Il existe différentes approches pour assurer la cohérence de la ligne de recouvrement. Nous décrivons dans cette section les trois approches possibles : *points de reprise indépendants*, *synchronisés* et *induits par messages.*

- **Sauvegarde indépendant :** Cette méthode consiste à faire prendre aux différents processus du système des points de reprise de manière totalement indépendante. La cohérence des états globaux formés n'est donc pas assurée et c'est en cas de panne que le protocole recherche une ligne de recouvrement parmi tous les états globaux formés. Cette recherche s'effectue généralement par la création d'un graphe de

dépendance entres les points de reprise locaux. L'inconvénient principal de cette approche est le risque d'effet domino qui peut causer une perte importante du travail réalisé avant la panne et la sauvegarde inutile de points de reprise, ceux-ci ne faisant jamais partie d'un état global cohérent. Ces points de reprise augmentent le cout de l'exécution normale (i.e. en l'absence de panne) et ne servent pas à la reprise après une panne. De plus, la sauvegarde non coordonnée oblige chaque processus à maintenir plusieurs points de reprise. Afin de déterminer un état global cohérent (ligne de recouvrement), chaque processus dispose d'un journal en mémoire stable dans lequel il enregistre tout ou partie des messages échangés ainsi que leurs historiques. Lorsqu'une défaillance survient, l'algorithme de reprise utilise les points de reprise locaux et les journaux afin de déterminer une ligne de recouvrement. [11]

- **Sauvegarde synchronisé :** Les points de reprises sont réalisés de manière coordonnée sur tous les processus de l'exécution, de façon à assurer que l'état global résultant soit cohérent. Cette synchronisation peut être :
    1. *Bloquante* : la cohérence de l'état global est assurée par le fait que tous les processus sont stoppés lors de la prise de cet état global. Cette approche est aussi nommée sync-and-stop.
    2. *Non bloquante* : la cohérence est assurée par des messages "balais", qui vident les canaux de communication afin d'éviter les messages orphelins. Selon les approches, les canaux de communication sont supposés FIFO.
    3. *Sur horloge* : les horloges de chaque machine sont synchronisées, et les processus déclenchent les points de reprise en fonction de leur horloge locale.

    Dans les deux premiers cas, ce type de protocole nécessite l'utilisation de messages additionnels. L'approche synchronisée permet d'éviter l'effet domino puisque tous les états globaux crées sont forcément cohérents. [17]

- **Sauvegarde induite par messages :** la sauvegarde induite par les communications (communication-Induced Checkpointing : CCI), est un compromis entre la sauvegarde coordonnée. L'idée principale est d'une part, d'éviter la coordination des processus en permettant la sauvegarde non coordonnée, appelée sauvegarde locale, et d'autre part d'éviter l'effet domino en forçant un processus à sauvegarder son état en cas d'évaluation de la ligne de recouvrement. Cette sauvegarde est alors appelée sauvegarde forcée. Donc, le principe de CIC est d'étendre un ensemble de points de reprise locale (sauvegardes locales) de l'application (système) à un ensemble de points de reprise constituant un état global cohérent (sauvegarde globale). [1]

B. Tolérance aux pannes par journalisation : Le principe du mécanisme de journalisation est la sauvegarde de l'histoire d'exécution de l'application. En effet, le principe de la reprise basée sur la journalisation est que durant l'exécution normale, c-à-d sans panne, chaque processus stocke dans un journal sur mémoire stable les informations correspondant à tous ses événements qu'il observe (les points de reprise). Après une panne, le processus défaillant reconstruit son état d'avant la panne à partir de son état initial et en utilisant son journal, afin de rejouer les événements exactement comme ils se sont produits avant la panne (relancer depuis son point de reprise le plus récent).

Pour éviter la reconstruction de l'état d'un processus défaillant à partir de son état initial, des sauvegardes périodiques de l'état du processus sont effectuées. Traditionnellement, les événements sont associés aux messages et le mécanisme de journalisation est donc appelé *journalisation de messages.*

On peut distinguer trois approches différentes dans la tolérance aux pannes par journalisation : *pessimiste, optimiste* et *causale.*

- **Journalisation pessimiste** : Ce protocole a été conçu dans l'hypothèse qu'une panne peut se produire après n'importe quel événement non déterministe. Il enregistre en mémoire stable le déterminant de chaque message avant que ce dernier ne soit autorisé à interagir avec le système (Figure 11). Ces protocoles font souvent référence à la journalisation synchronisée parce que quand un processus enregistre un événement non déterministe sur mémoire stable, il attend de recevoir un acquittement pour continuer son exécution.

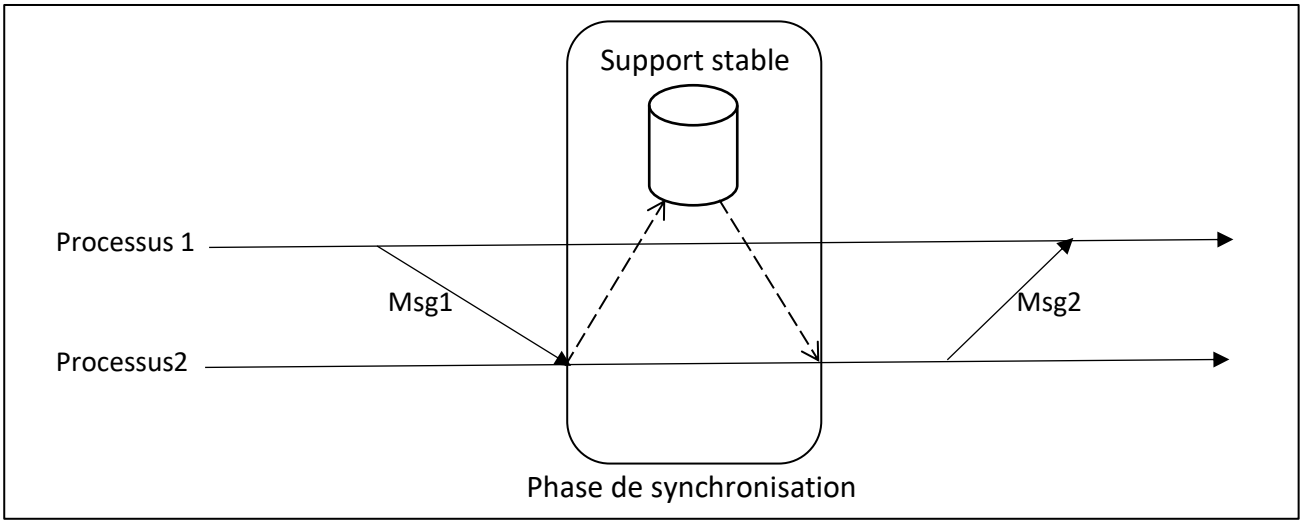

Figure 11:Processus de sauvegarde avec la journalisation pessimiste [16]

Dans un système à journalisation pessimiste, l'état de chaque processus est toujours récupérable. Cette propriété a quatre avantages :

- Les processus peuvent envoyer des messages à l'extérieur sans utiliser un protocole spécial.
- Lors d'une défaillance, les processus redémarrent au point de reprise le plus récent, et rejoue les messages stockés dans le journal.
- Le recouvrement est simple parce que les effets d'une panne se limitent seulement aux processus en panne.
- La procédure de récupération des points de reprise et des messages est simple, puisque les messages reçus avant le dernier point de reprise ne seront jamais utilisés car les processus redémarrent au processus le plus récent.

Le principal inconvénient reste la synchronisation qui entraine la dégradation des performances du système. Plusieurs approches ont été définies afin de minimiser les synchronisations, comme la technique utilisée par Johnson et al. [16]

- **Journalisation optimiste :** ce protocole utilise l'hypothèse selon laquelle la journalisation d'un message sur support fiable sera complète avant qu'une panne ne se produise. En effet, au cours de l'exécution des processus, les déterminants des messages, à savoir leur contenu et identifiants, sont conservés en mémoire volatile avant d'être périodiquement vidés sur support stable. Le stockage sur mémoire stable se fait de manière asynchrone : le protocole n'exige pas le blocage de l'application pendant la sauvegarde sur mémoire stable. La latence induite est alors très faible.
  Cependant une panne peut survenir avant que les messages ne soient enregistrés sur mémoire stable. Dans ce cas, les informations sauvegardées en mémoire volatile du processus en panne sont alors perdues. Le recouvrement devient compliqué puisque tous les processus ayant reçu un message provenant du processus défaillant deviennent orphelins (*Figure 12*). Ce qui peut produire l'effet domino. En plus, le protocole est obligé de maintenir plusieurs points de reprise en mémoire. La récupération des points de reprise devient plus complexe. Le calcul pour obtenir un état global cohérent peut avoir un coût important parce que les processus peuvent être amenés à redémarrer au point de reprise qui n'est pas le plus récent.
  Pour éviter l'effet domino, le protocole enregistre les informations de dépendances entre les messages durant l'exécution sans faute. Une solution consiste à utiliser des vecteurs de dépendances attaché aux messages et dont la taille est égale au nombre total de processus.
  Mais cela est très coûteuse pour les grands systèmes comportant un nombre important de processus. [16]

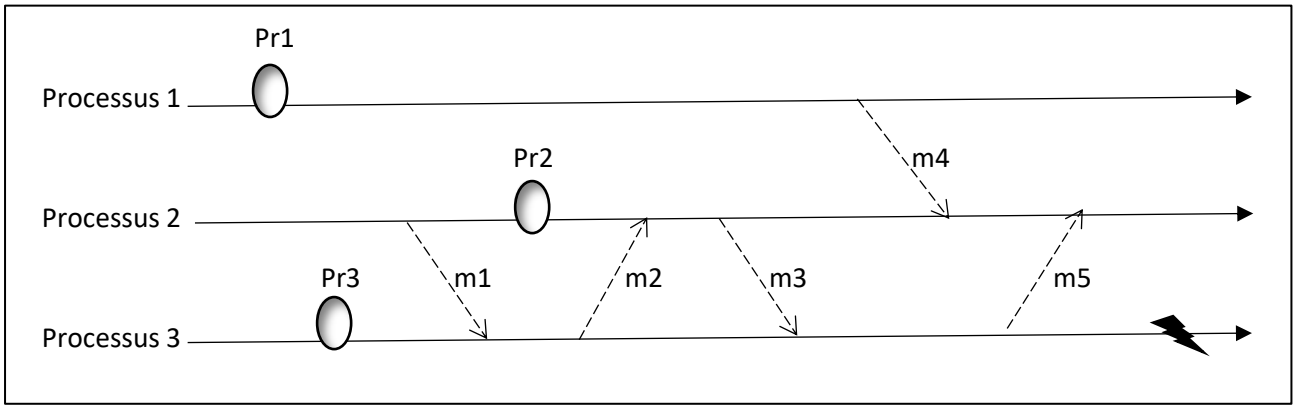

Figure 12:Enregistrement de messages optimiste [16]

Si le processus 3 tombe en panne avant la sauvegarde des messages m3 et m5, il va reprendre à partir du point de reprise pr3. Les messages m2 et m5 devenant orphelin, le processus 2 va reprendre à partir du point de reprise pr2.

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

La procédure de journalisation s'effectue de la manière suivante : (voir figure 13)

- Les messages sont enregistrés sur support stable, mais le processus n'attend pas un accusé réception pour continuer son exécution.
- Chaque processus détient une table de dépendances causales de messages. Tant que l'accusé de réception des messages n'est pas arrivé, les déterminants des messages sont ajoutés aux informations de dépendance de ces messages, en sachant que la table de dépendance causale de chaque message est attachée aux messages échangés.
  En cas de panne, le processus défaillant effectue un retour arrière au point de reprise le plus récent et rejoue les messages enregistrés sur support stable ou utilisent les tables de dépendances des processus.

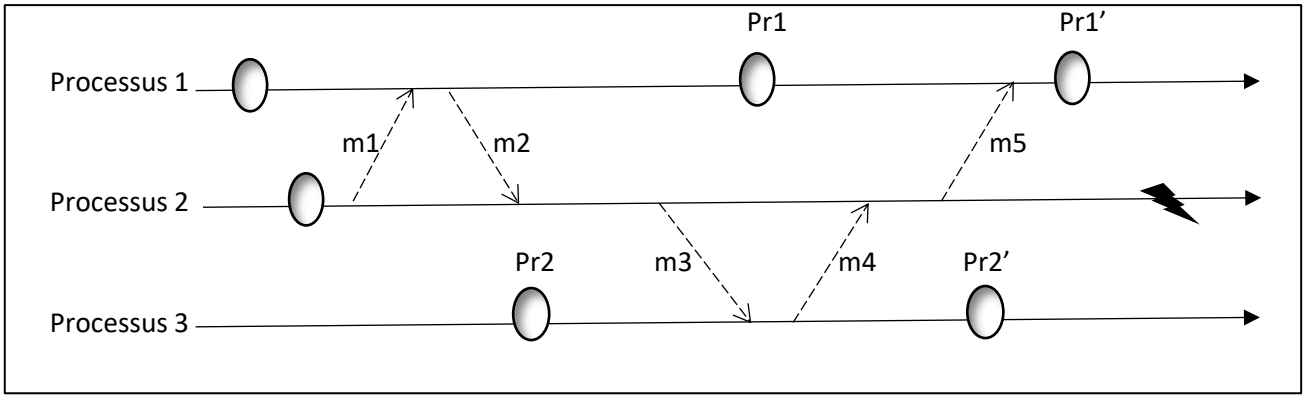

Figure 13:Recouvrement journalisation causale [16]

L'inconvénient de la journalisation causale est le surcoût sur la latence du réseau introduit par les informations de dépendance attachées aux messages. Pour éviter la dégradation des performances causée par les données supplémentaires de causalité, d'autres protocoles sont nés de l'étude des relations de causalité entre les messages envoyés dans un système par passage de messages.

En cas de panne, le processus fautif obtient les informations de causalité auprès de l'enregistreur d'événements. [16]

6. Politique de tolérances aux pannes :

Les techniques de tolérances aux pannes sont basées sur deux types de politiques standards : *proactive* et *réactive*.(voir figure 14) [12]

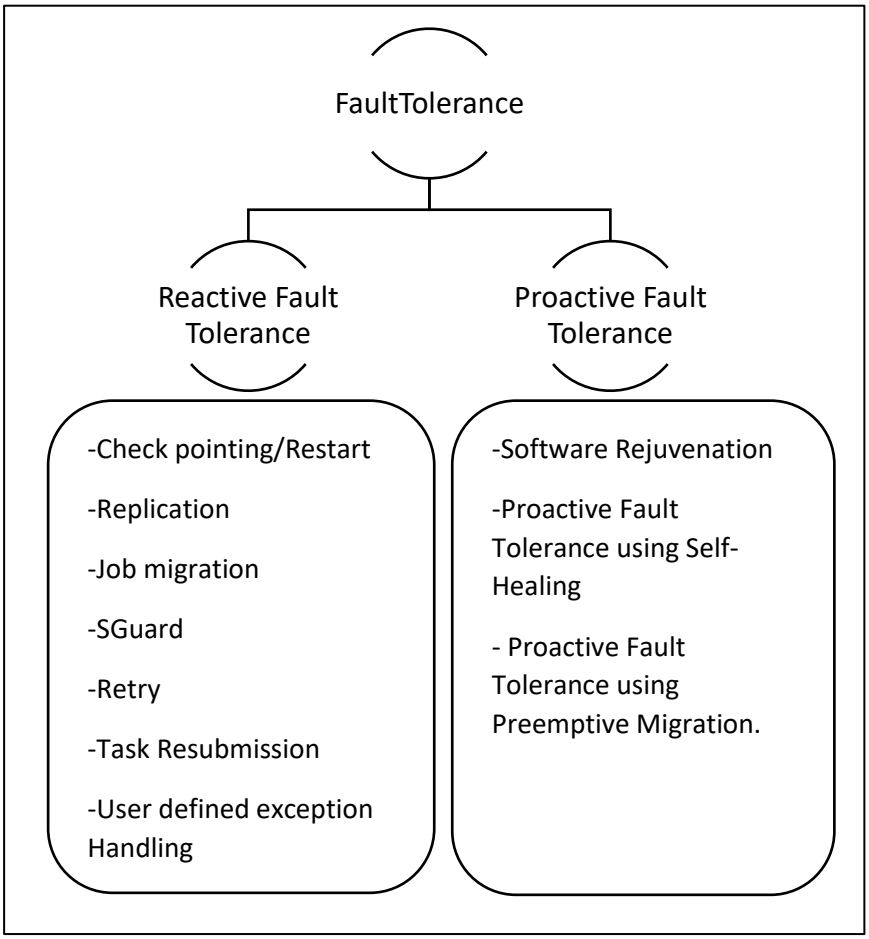

Figure 14:Techniques de la tolérance aux fautes [1]

a) Tolérance aux pannes réactive

Cette politique est utilisée pour réduire l'impact des défaillances sur un système lorsqu'une défaillance se produit effectivement. Les techniques basées sur cette politique sont :

- *Le Check pointing/Restart* : La tâche qui a échoué est relancée depuis le point de contrôle récent plutôt que depuis le début. C'est une technique efficace pour les applications de grandes envergures.
- *La réplication* : afin de rendre l'exécution réussite, plusieurs répliques de tâche sont exécutées sur différente ressources. De ce fait, la tâche entièrement répliquée ne tombe pas en panne. HAProxy, Hadoop et AmazonEc2 sont utilisés pour l'implémentation de la réplication.

b) Tolérance aux pannes proactive

Cette technique vise à prédire les pannes de manière proactive et à remplacer les composants soupçonnés par d'autres composants corrects en évitant ainsi la récupération des fautes et des erreurs. Les techniques basées sur cette politique sont :

- *Software Rejuvenation* : Le système est prévu pour les redémarrages périodiques et chaque fois que le système démarre avec un nouvel État.
- *Proactif tolérance de panne à l'aide d'auto-guérison* : Échec d'une instance d'une application s'exécutant sur plusieurs machines virtuelles est contrôlée automatiquement.

- *Proactif tolérance de panne à l'aide de Préemptive Migration* : Dans cette technique, une application est constamment observée et analysé. La migration préemptive d'une tâche dépend de mécanisme de contrôle de feed-back en boucle.

Partie II. Le Cloud Computing :

7. Définition de l'informatique en nuage (le Cloud Computing) :

Il existe plusieurs définitions du Cloud Computing, ce paradigme n'est pas encore normalisé, chacun peut le définir selon son point de vue. La plupart des experts définissent le Cloud Computing comme étant un modèle incluant la notion de services disponibles à la demande, plus facilement extensible, virtuels et illimités ne dépendant pas de l'infrastructure physique. (Voir figure 15)

Parmi les définitions du Cloud Computing (CC), nous citons certains des plus appropriés :

- Selon NIST (National Institute of Standards and Technology) : le Cloud est vu comme un modèle qui permet un accès omniprésent, pratique à la demande à un réseau partagé et à un ensemble de ressources informatiques configurables, à titre d'exemple des réseaux, des serveurs, du stockage, des applications et des services, qui peuvent être provisionnées et libérées avec un minimum d'administration. [12]

- Selon Génération NT : Le Cloud Computing est un concept d'organisation informatique qui place l'Internet au cœur de l'activité des entreprises, il permet d'utiliser des ressources matérielles distantes pour créer des services accessibles en ligne ». [12]

- Selon CISCO : Le Cloud Computing est une plateforme de mutualisation informatique fournissant aux entreprises des services à la demande avec l'illusion d'une infinité des ressources. [12]

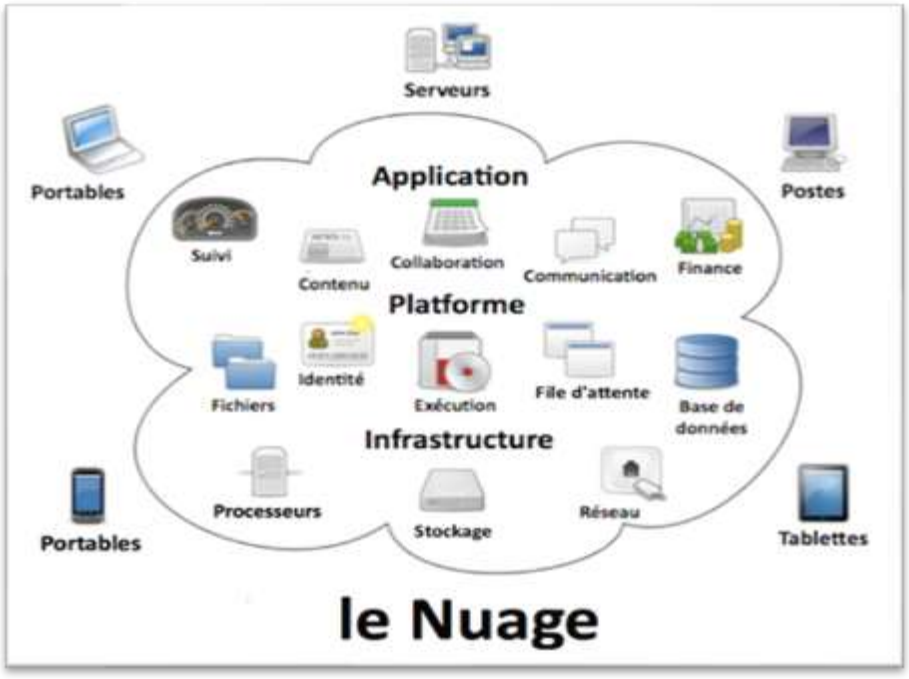

Figure 15:Cloud Computing

8. Virtualisation dans les Clouds :

La virtualisation est une technologie permettant l'exécution de plusieurs systèmes d'exploitation et/ou plusieurs applications sur le même ordinateur, ce qui accroît l'utilisation et la flexibilité du matériel. Un même ordinateur peut ainsi permettre l'exécution de plusieurs machines virtuelles contenant différents systèmes d'exploitation et/ou différentes applications isolées les unes des autres.

La mise en œuvre d'une machine virtuelle (Virtual machine ou VM) nécessite l'ajout d'une couche logicielle à la machine physique. L'emplacement de cette couche dépend du type d'architecture de la machine virtuelle. Il existe deux types de technologies de machine : les machines virtuelles applicatives (processus Virtual machines) et les machines virtuelles système (system Virtual machines).

La virtualisation repose sur trois éléments importants :

1) L'abstraction des ressources informatiques ;
2) La répartition des ressources par l'intermédiaire de différents outils, de manière à ce que celles-ci puissent être utilisées par plusieurs environnements virtuels ;
3) Et, la création d'environnements virtuels. [11]

La *virtualisation* a été la première pierre vers l'ère du Cloud Computing. En effet, cette notion permet une gestion optimisée des ressources matérielles dans le but de pouvoir y exécuter plusieurs systèmes virtuels a sur une seule ressource physique et fournir une couche supplémentaire d'abstraction du matériel. Le principe de virtualisation permet aussi d'être plus flexible dans l'allocation des ressources, par exemple si une machine virtuelle manque de ressources car le serveur physique sur lequel elle repose n'est pas ou plus assez puissant il est

dès lors très simple de virtualiser cette machine virtuelle sur un autre serveur plus puissant. [14]

9. Les composants de cloud :

Selon [18] les composants communs de cloud sont les suivants : (figure 16)

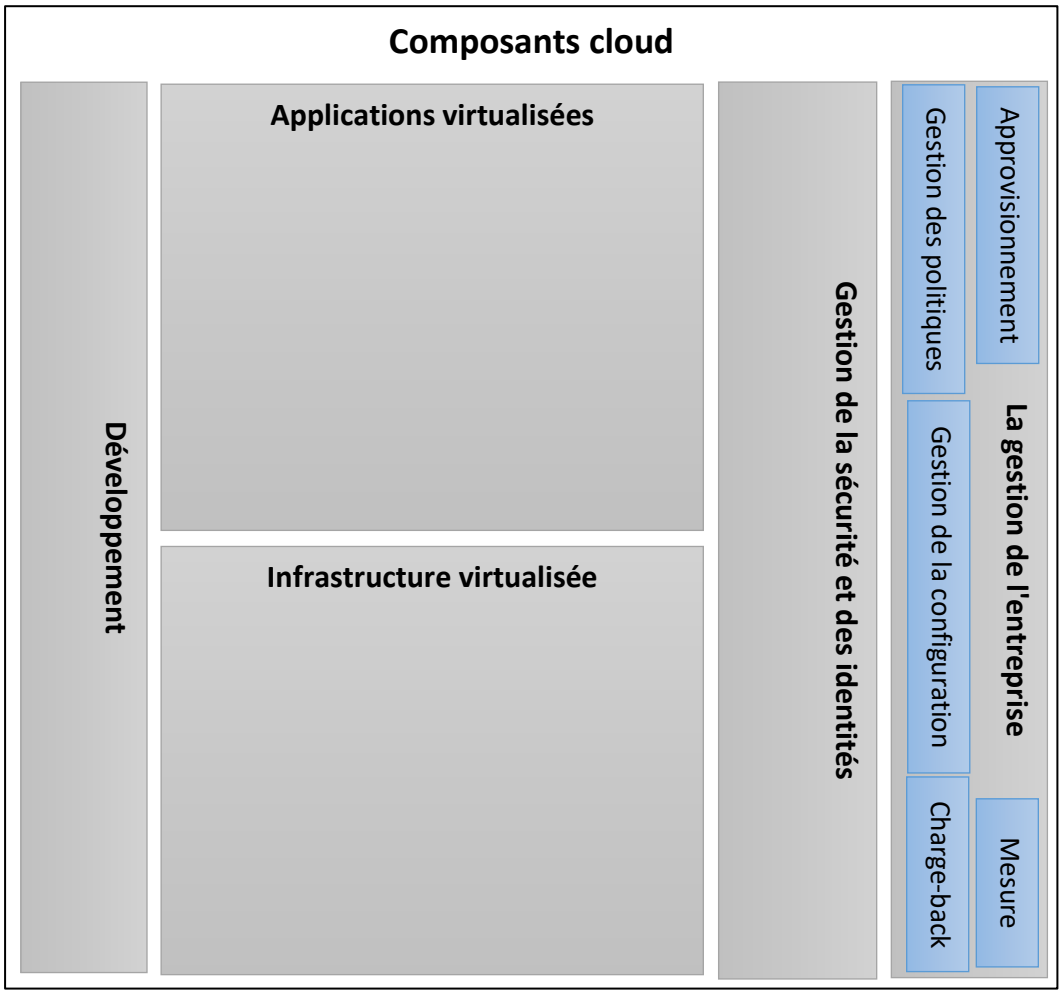

Figure 16:les composants de Cloud [18]

a) **Applications virtualisées :** elles rendent compatible les applications de l'utilisateur avec les hardwares, les systèmes d'exploitation, le réseau et le stockage pour permettra la flexibilité du déploiement.
b) **Infrastructure virtualisée :** elle fournit l'abstraction nécessaire pour s'assurer qu'une application ou un service ne soit pas directement attachée à l'infrastructure matérielle (serveurs, stockage ou réseaux). Ceci permet au service de se déplacer dynamiquement à travers les ressources virtualisées d'infrastructure.
c) **Gestion de sécurité et d'identité :** le système de gestion de sécurité fournie les commandes nécessaires pour assurer les informations sensibles (les protéger) et rependre l'exigence de conformité.
d) **Développement :** les infrastructures de développement facilitent non seulement l'orchestration de service mais permettent également aux processus d'être

développés. Ce sont les outils de développement comme le compilateur, SDK (Software Development Kit) et l'environnement de développement.

e) **Gestion d'entreprise :** la couche de gestion d'entreprise manipule le cycle de vie des ressources virtualisées et fournit les éléments additionnels d'infrastructure commune pour la gestion de taux de disponibilité, utilisation dosée, gestion de politique, gestion de permis, et recouvrement des pertes.

10. Comparaison avant et après l'apparition du Cloud Computing :

L'apparition du Cloud Computing a bouleversé le monde informatique. Cela crée un grand écart entre l'utilisation ou non de ce concept. Ci-dessous une figure illustrant les étapes de passage de l'évolution en informatique (voir figure 17). [12]

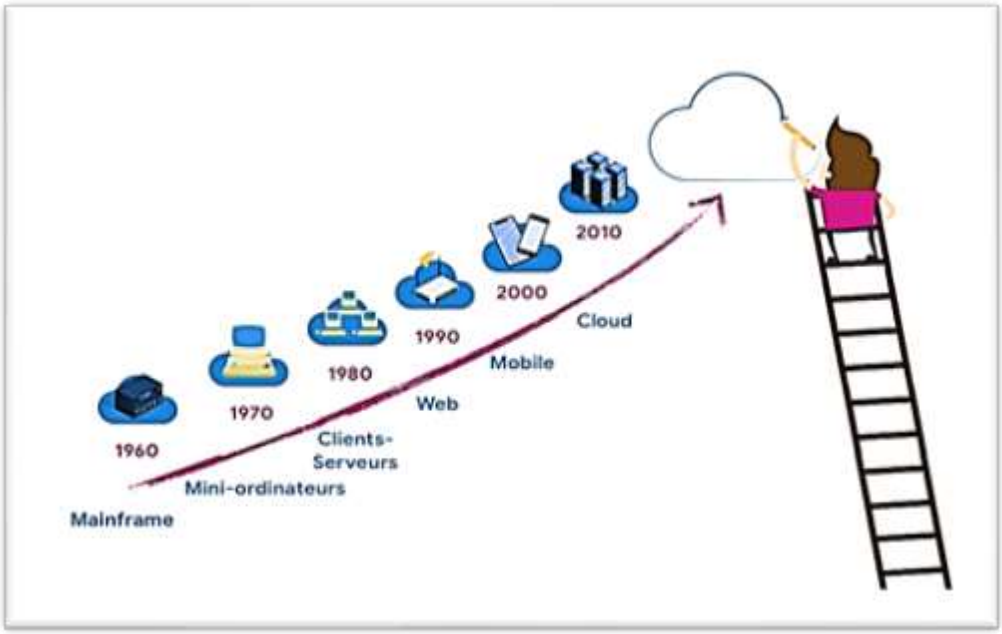

Figure 17: les étapes d'évolution de cloud computing

Le tableau ci- dessous (tableau 2) montre une petite comparaison avant et après l'apparition du Cloud Computing :

Avant l'apparition du Cloud Computing	***Après l'apparition du Cloud Computing***
-Les clients accèdent aux ressources : serveurs, applications, espaces de stockage et services via le réseau LAN ou intranet et internet. -Un groupe d'ingénieurs spécialisés est nécessaire pour garantir l'installation, la	-Les clients accèdent à des infrastructures informatiques mises à disposition par un prestataire de Cloud via Internet. -Le client ne gère aucun matériel, ni logiciels c'est le prestataire de Cloud qui s'en charge. Il peut donc se concentrer avant tout sur son travail et son savoir-faire.

configuration, la sécurité et la mise à jour du hardware et software. -L'accès au serveur ou à l'application sensible se fait depuis l'intranet en passant par l'authentification et selon des droits d'accès bien défini ou depuis Internet en utilisant les VPN « Virtual Private Network». -Si le serveur ou l'application tombe en panne et s'il n'existe pas de mécanisme de reprise après panne ou de sauvegarde toutes les données seront perdues -Pour faire la sauvegarde ou les backups de plusieurs serveurs cela nécessite une grande capacité de stockage -Si le serveur ou l'application n'est pas bien sécurisé, il est facile à l'attaqué. -Quand le nombre d'utilisateurs augmente dans les entreprises et les organisations…, celle-ci doit investir pour acheter plus d'équipements matériels qui peuvent être couteux. -Le client paie le support même s'il ne l'utilise pas à 100%. -On ne garantit pas l'accès de n'importe où car ceci représente une faille de sécurité et l'entreprise doit investir et acheter un grand matériel afin de garantir la haute disponibilité. -La connexion internet est indispensable que pour les clients qui utilisent les VPN, les clients dans l'entreprise accèdent à travers l'intranet. - Les données sont confidentiels et appartiennent totalement à l'entreprise.	-L'accès au serveur ou à l'application se fait de n'importe où et avec n'importe quel périphérique avec un accès prédéfini à travers le Web. -On ne se soucie pas des sauvegardes car la panne est transparente aux clients -On dispose d'une capacité de stockage illimité et qui peut augmenter selon les besoins, si le client oublie de sauvegarder ses données ou de faire les backups de ces serveurs, le prestataire de Cloud s'engage à prendre en main cette tache lourde afin de ne pas tomber en panne. -le Cloud Computing garantit la sécurité à travers les mécanismes de réplication des données, plan de reprise d'activité,… -Le Cloud Computing permet d'accéder plus rapidement à des ressources via un portail web et donc cela nous permet de réduire le cout d'investissement, on a plus à investir dans des équipements matériels très coûteuse en interne. -Le client paie uniquement ce qu'il consomme ou par abonnement mensuel. -Le Cloud Computing permet de garantir les accès et la haute disponibilité des services, se facteurs est très important pour les clients nomades. -La connexion internet est indispensable au client, il l'utilise pour accéder à sa plateforme de travail au niveau du Cloud. -Le client n'a pas un accès direct à ses données. Il dépend totale- ment du fournisseur et doit lui faire entièrement confiance «confidentialité des données, la vie n'est plus privée ».

Tableau 2:Comparaison avant et après l'apparition du Cloud Computing [12]

11. Caractéristiques du Cloud Computing :

Les caractéristiques du cloud computing sont : [14]

- **Auto-guérison :** Tout système Cloud doit contenir une ou plusieurs copies de chaque application déployée en lui, de telle façon qu'en cas de disfonctionnement de l'application en cours, l'application en copie vient la remplacer en prenant l'état actuel de l'application en échec. Les applications en copies doivent être maintenues et mise à jour à chaque fois que l'application en cours est modifiée.
- **Multi location (Multi-Tenancy) :** Sur le Cloud une même application peut être utilisée par plusieurs clients en même temps, en préservant la sécurité et les données privées de chaque client. Cela est possible en utilisant des outils de virtualisation qui permettent de partager un serveur sur plusieurs utilisateurs.
- **Evolutivité linéaire :** Un système basé Cloud computing technologie a la possibilité de découper les principales taches du système en un ensemble de petits morceaux, et de les distribuer sur l'infrastructure virtuelle du Cloud.
- **Orienté services :** Tout système Cloud est basé sur un ensemble de services indépendants les uns des autres. Ce qui donne de la puissance à la technologie Cloud c'est le rassemblement de ces services en une seule unité afin de présenter un service Cloud qui couvre tous les niveaux sur lesquels une application est basée (Niveau virtuel, niveau logiciel, une interface facile à utiliser… Etc.)
- **SLA (Service Level Agreement) :** appelé « accord de service » avec les services Cloud, un client peut négocier le niveau de service qu'il lui convient et il doit payer pour cela. Dans le cas où les ressources Cloud sont en surcharge, le système crée d'autres entités d'applications Cloud en utilisant les outils de virtualisation disponible afin de respecter les termes du contrat SLA.
- **Virtualisation :** Un système basé sur le Cloud Computing est un système complètement virtuel et indépendant de la couche physique sur laquelle les ressources et applications sont déployées.
- **Flexibilité :** Les services Cloud sont destinés à supporter les lourdes comme les petites charges de travail en augmentant ou réduisant automatiquement les ressources utilisées selon les tailles des taches à traiter.

12. Modèles de services du Cloud :

Le Cloud Computing est composé de plusieurs catégories de services, les trois principales catégories sont : IaaS (Infrastructure as a Service), PaaS (Platform as a Service) et SaaS (Software as a Service), chacun de ces types de service correspond à un niveau d'abstraction logiciel précis par rapport aux ressources informatiques matérielles accessibles via internet. (Voir figure 18)

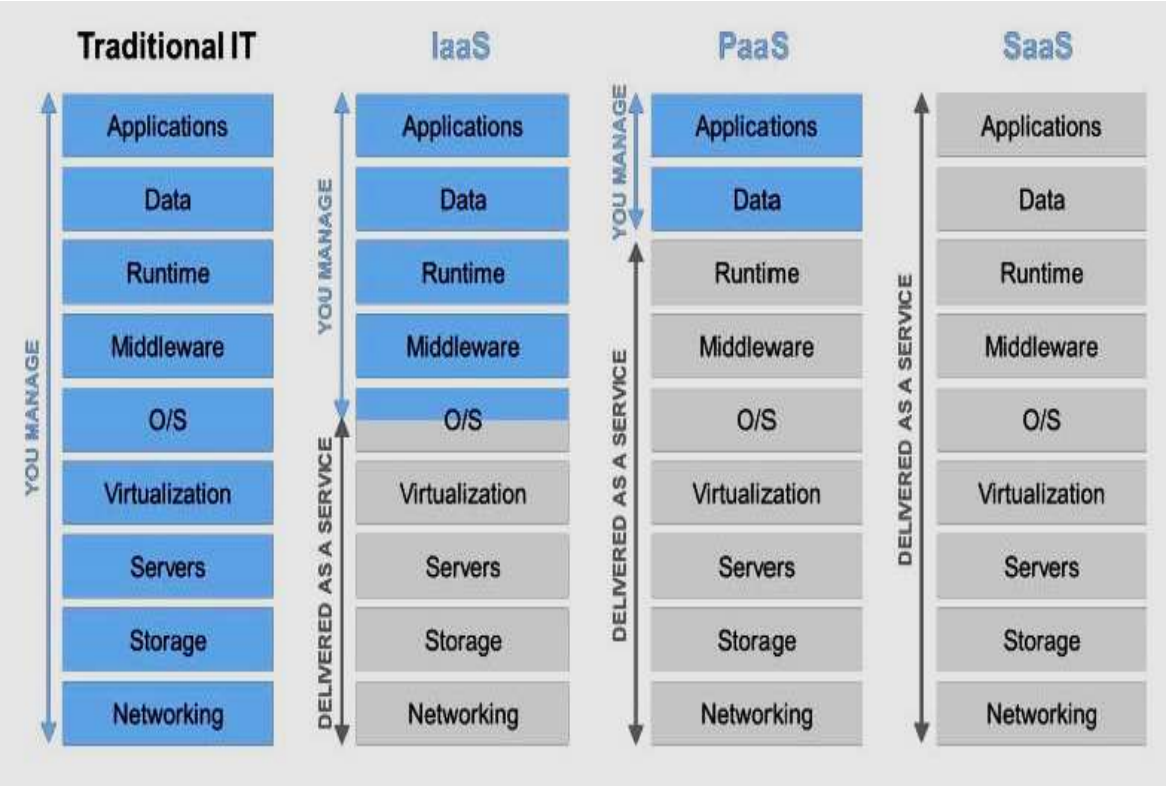

Figure 18:Les Modèles de Service du Cloud Computing.

Chacun de ces types (IAAS, PAAS, SAAS) a une population cible (figure 19) :

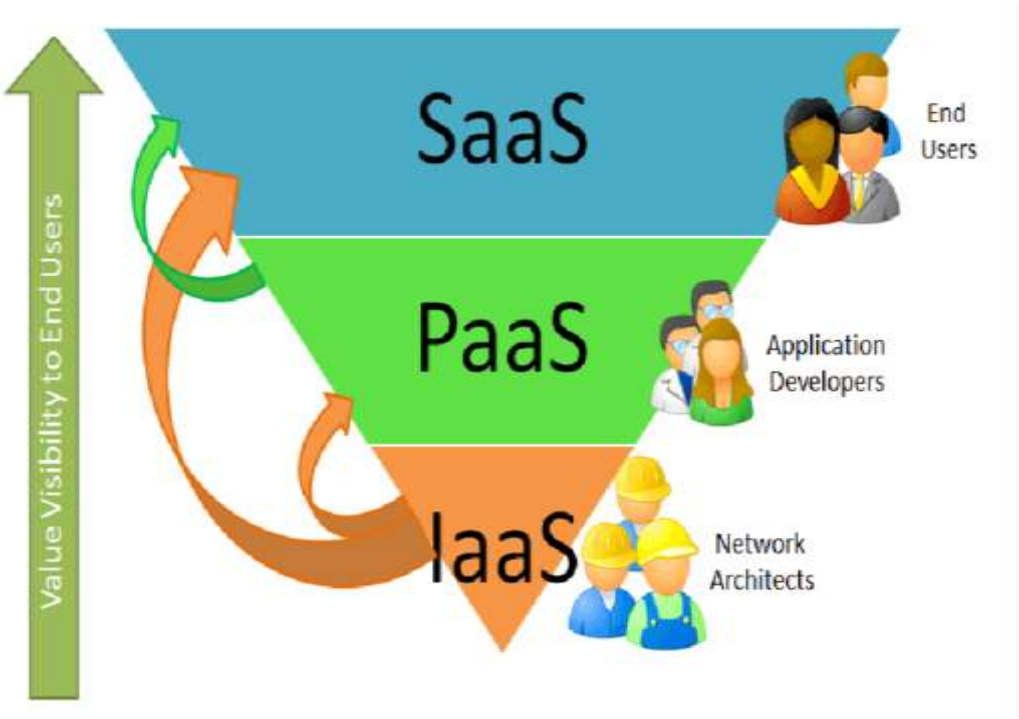

Figure 19:Les populations cibles des services Cloud

a) IaaS (Infrastructure As A Service):

Le fournisseur offre du matériels informatique vitalisé, tels que les réseaux informatiques, le stockage de données, les systèmes d'exploitation virtuels et la puissance de calcul, donc le client bénéfice de l'utilisation des ressources physiques (hardware) des serveurs, il peut installer son système d'exploitation et ces propres applications dans le Cloud sans se préoccuper des problèmes liés aux pannes et à la sécurité , ces ressources peuvent être allouées dynamiquement en fonction des besoins. Donc, IASS correspond à la partie infrastructure de Cloud, il fournit des instances d'OS (Operating System) et d'infrastructure sous-jacente (serveur, réseau, stockage). A titre d'exemple Amazon Elastic Comput Cloud, RackSpace, Eucalyptus, etc.

b) PaaS (Platform AS A Service):

Dans ce modèle, le prestataire du Cloud fournit une plate-forme composé d'un ensemble d'outils interactifs qui seront utilisés par les clients pour construire des applications puissantes et flexibles dans un système Cloud Computing. Le client peut installer ses propres applications si besoin. Il peut être utilisé pour la fourniture des bases de données, fourniture de serveurs d'application. A titer d'exemple Google App Engine, Windows Azure web role, etc.

c) SaaS (Software As A Service):

Le prestataire de Cloud fournit aux clients une solution logicielle livrée sur internet, Ces logiciels sont accessibles via différentes interfaces, clients légers, navigateur Web, terminaux mobiles. Les Saas regroupent les IaaS et les PaaS dans la même application. Pour le public : on a la messagerie électronique de Gmail, Yahoo, Outlook, Google Apps…Pour les particulier : Le plus utilisé est le stockage de documents (permettant également le partage) comme par exemple :Hubic, DropBox, Google Drive, SalesForce, Le Cloud fournit aussi des logiciels de création de documents, de traitement d'images, de gestion de dessins et bien d'autres qui seront utilisés par les clients selon leurs besoins. Autrement dit, un SaaS fournit des applications prêtes à l'emploi s'exécutant sur l'infrastructure du fournisseur Cloud et accessibles via le navigateur de client.

Les avantages et les inconvénients de chaque type de service :

Le tableau ci-dessous (tableau 3) illustre les avantages et les inconvénients les plus courants des trois types de services Cloud :

	Avantages	*Inconvénients*
Saas	-pas d'installation -plus de licence -migration	-logiciel limité -sécurité -dépendances des prestataires
PaaS	-pas d'infrastructure nécessaire -pas d'installation -environnement hétérogène	-limitation des langages -pas de personnalisation dans la configuration des machines virtuelles

IaaS	-administration -personnalisation -flexibilité d'utilisation	-sécurité -besoin d'un administrateur système

Tableau 3:Avantages et Inconvénients des services cloud. [14]

13. Modèles de déploiement du Cloud :

On utilise le terme Cloud pour représenter l'infrastructure gérée par un prestataire. Chaque prestataire propose une solution du Cloud différente, on peut distinguer quatre types principaux de modèles de déploiement des services de Cloud Computing (voir figure 20):

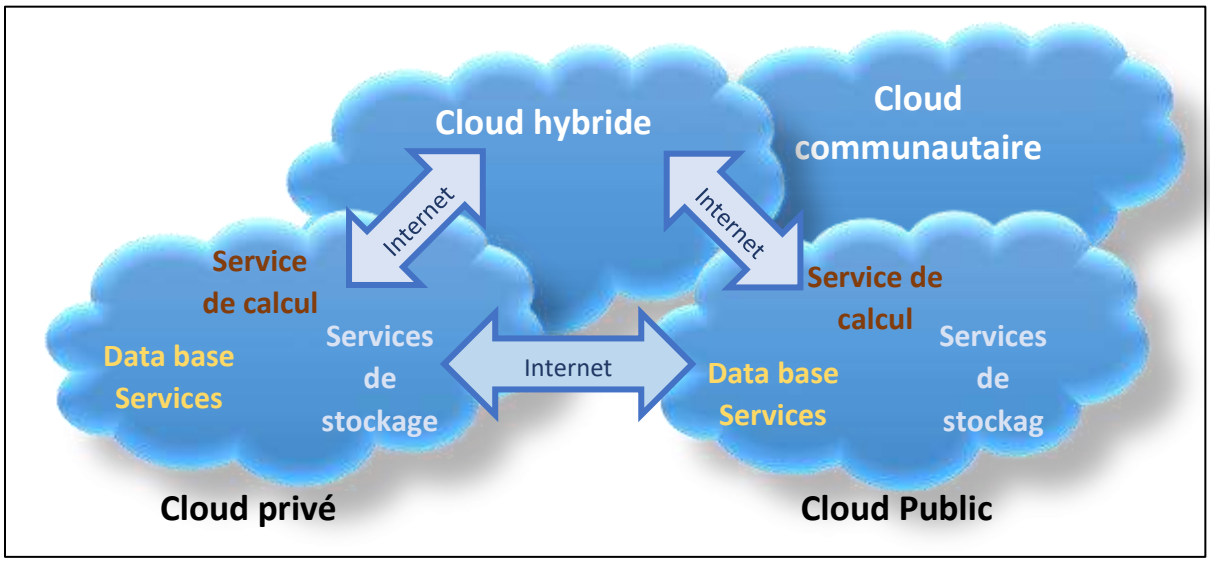

Figure 20:Les Modes de déploiement Cloud Computing. [11]

a) Le Cloud public :

Dans ce modèle de déploiement, le prestataire (ou fournisseur) du Cloud est externe, il possède sa propre infrastructure, la gère uniquement pour ses besoins, et ses services sont accessibles par le public. La facturation de ses services se fait selon les ressources allouées aux clients. A titre d'exemple : Amazon web services, Microsoft Azure, Eolas, etc.

b) Le Cloud privé :

Le Cloud est utilisé par un seul client (entreprise ou organisation..) c'est-à-dire que l'entreprise ou l'organisation est propriétaire du Cloud, cette infrastructure est accessible que par les membres de l'entreprise ou l'organisation. Il peut être géré par l'entreprise elle-même ou par le client, il est situé dans le local du client ou chez le prestataire de service. Ceci permet de garantir que les ressources allouées ne sont pas partagées avec d'autres clients. Dans ce cas on a deux types du Cloud privé :

a. *Cloud Privé interne* : Il est géré par l'entreprise ou l'organisation elle-même
b. *Cloud Privé externe* : Il est géré par le prestataire du Cloud, l'infrastructure est entièrement dédiée à l'entreprise et accessible via réseaux sécurisés de type VPN

c) Le Cloud communautaire :

L'infrastructure et les services sont partagés entre un ensemble d'entreprises ou d'organisations clientes indépendantes, réunies au sein d'une communauté (ville, académie, …) et partageant

des préoccupations spécifiques communes - les mêmes intérêts- (par exemple, la mission, les exigences de sécurité, des politiques et des considérations de conformité, des obligations légales). Ce type de Cloud peut être gérer par l'ensemble des clients ou par un prestataire de Cloud et peut être situées chez les clients qui la partage ou chez le prestataire. A titre d'exemple: UnivCloud.

d) Le Cloud hybride :

L'infrastructure d'un Cloud hybride est une composition de deux ou trois des types de nuages précédemment cités. Les différents Clouds qui la composent restent des entités indépendantes à part entière, mais sont reliés par des standards ou par des technologies propriétaires qui permettent le partage des données et des applications, et il permet aussi la portabilité des applications déployées sur les différents nuages. Lorsque l'utilisation des ressources du Cloud privé atteint le maximum, l'entreprise peut bénéficier d'autres ressources supplémentaires appartenant au Cloud public. Une autre propriété caractérise ce type de Cloud, c'est la répartition de la puissance de calcul ou traitement entre des Clouds publiques et privés ceci permet d'avoir des résultats dans un temps très rapide. Le Cloud Computing permet un équilibrage de charge et évite la surexploitation des ressources quelle que soit le modèle de déploiement choisi.

14. Bénéfices et Limites du Cloud :

Chaque technologie possède des avantages et des limites :

a) Bénéfices du Cloud Computing:

Voici quelques-uns des avantages possibles pour ceux qui offrent des services de Cloud et les applications :

- ✓ *Réduction des coûts :* Les caractéristiques du Cloud intéressantes pour les entreprises sont la réduction du coût total de possession des systèmes informatiques, la facilité d'augmenter ou de diminuer les ressources, puisqu'il élimine la nécessité d'investir dans du matériel et des logiciels. Le Cloud peut permettre d'effectuer des économies, notamment grâce à la mutualisation des services sur un grand nombre de clients. En ne payant que ce qu'elle consomme réellement, l'entreprise gère son budget au plus près de ses besoins. Par ailleurs, le Cloud Computing permet de bénéficier d'un haut niveau de service (24h/24 et 7j/7) pour un prix accessible aux petites structures.
- ✓ *Souplesse :* L'informatique dans les nuages est un modèle permettant d'établir un accès par le réseau à un réservoir partagé de ressources informatiques standard configurables (réseau, serveurs, stockage, applications et services) qui peuvent être rapidement mobilisées et mises à disposition en minimisant les efforts de gestion ou les contacts avec le fournisseur de service.
- ✓ *Recentrage sur le cœur de métier :* Il s'agit pour les entreprises clientes de rester concentrer sur leur métier, alors plus de disponibilité pour des activités à haute valeur ajoutée, et de disposer d'un système d'information toujours disponible grâce aux engagements que prendra le fournisseur en termes de qualité de service (la maintenance, la sécurisation et les évolutions).

- ✓ *Vitesse :* la plupart des services Cloud sont fournis en libre-service et à la demande. Dans quelques minutes et avec quelques clics, des énormes ressources de calcul peuvent mise en œuvre alors offrant aux entreprises un haut niveau de flexibilité.
- ✓ *Flexibilité :* accès aux services de n'importe quel poste de travail de l'entreprise voire de l'extérieur de l'entreprise et, pour certaines applications nomades, depuis un Smartphone.

b) Limites du Cloud Computing:

- *Menace sur la sécurité et la confidentialité :* La sécurité des données est l'un des freins les plus couramment évoqués par les entreprises car les données sont hébergées en dehors de l'entreprise dans la majorité des cas de cloud. Un grand risque lorsqu'on met une application qui donne des avantages compétitifs ou qui contient des informations clients dans le Cloud.
- *La bande passante peut faire exploser votre budget :* quand la bande passante qui serait nécessaire pour mettre cela dans le Cloud est gigantesque, les coûts seraient importants. Donc, il faut mieux d'acheter le stockage nous-mêmes plutôt que payer dans le Cloud.
- *Taille de l'entreprise :* Si on a une grande entreprise, alors nos ressources sont grandes, ce qui inclut une grande consommation du Cloud. Donc, on a plus d'intérêt á mettre au point notre propre Cloud plutôt que d'en utiliser un externalisé. Les gains sont bien plus importants quand on passe d'une petite consommation des ressources á une consommation plus importante.
- *Connexion :* si l'utilisateur n'a pas de connexion internet, alors il ne pourra pas accéder à ses ressources.

15. Sécurité dans le Cloud computing :

La sécurité et la conformité émergent systématiquement comme les principales préoccupations des responsables informatiques lorsqu'il est question de Cloud Computing, des préoccupations encore plus accentuées lorsqu'il s'agit de Cloud public.
La sécurité permet de garantir la confidentialité, l'intégrité, l'authenticité et la disponibilité des informations.
La mise sur pied d'une solution de Cloud Computing comporte des problèmes de sécurité inhérents á la solution elle-même. Le fait de centraliser toutes les informations sur un site pose un grand nombre de soucis. On peut citer comme problème potentiel :

- ➢ Une possible interruption massive du service.
- ➢ Une cible de choix pour les hackers
- ➢ Interface et API non sécurisé

Ce point de vulnérabilité du Cloud fait l'objet depuis quelques années l'objet de recherches avancées. Il a été créé un organisme chargé de mettre sur pied des normes en matière de sécurité dans le Cloud Computing. Cet organisme s'appelle CSA (Cloud Security Alliance).

VII. Conclusion :

Ce chapitre est divisé en deux sections dans lesquelles nous avons présenté un aperçu de la tolérance aux pannes pour les systèmes distribués et les Cloud Computing. Premièrement nous avons présenté la notion de sûreté de fonctionnement d'un système informatique, nous avons vu que la tolérance aux pannes n'est qu'un moyen pour assurer la sûreté de fonctionnement d'un système et qu'on utilise toujours la redondance pour l'obtenir, deux méthodes principales permettent de réaliser la tolérance aux pannes dans les systèmes répartis: la duplication et l'abstraction d'une mémoire stable. Le passage à l'échelle de ces systèmes, qui s'est accentué par l'apparition du Cloud Computing. Dans la deuxième partie de ce chapitre, nous avons introduit le Cloud Computing, nous avons abordé sa définition puis nous avons fait une comparaison avant et après l'apparition de ce concept en discutant les avantages et inconvénients. Nous avons discuté également ces caractéristiques fondamentales, ces services et les modèles de déploiements utilisés.

Notre but dans ce travail est de garantir la tolérance aux pannes dans les Cloud Computing. Pour cela nous proposons dans le prochain chapitre une architecture de tolérance aux pannes qui permet de garantir la continuité des services du Cloud Computing d'une façon transparent et efficace en cas de pannes.

CHAPITRE3 : Conception de la méthode proposée

VIII. Introduction

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

Le Cloud Computing est devenu une pierre angulaire dans l'architecture de la nouvelle génération des systèmes informatique dans l'entreprise. Par contre, les mesures de contrôle et de sécurité dans le Cloud sont restées similaires à ceux utilisés dans les systèmes informatiques traditionnels.

Dans ce chapitre, et afin de réponde à temps en cas d'erreur, nous présentons une méthode basée sur les SMA, la planification avec la recherche des chemins minimaux et le checkpointing. Ensuite, après cette petite introduction, nous présentons la problématique et les objectifs de ce travail. Nous discutons, par la suite, quelques travaux connexes. Puis, nous présentons le modèle proposé. Nous terminerons ce chapitre par une conclusion.

IX. Problématique et objectif :

Le Cloud Computing consiste en une interconnexion et une coopération de ressources informatiques, situées au sein d'une même entité ou dans diverses structures internes, externes ou mixtes, et dont le mode d'accès est basé sur les protocoles et standards Interne. Il permet de disposer d'applications, de puissance de calcul, de moyens de stockage comme autant de « services ». Ceux-ci seront mutualisés, dématérialisés donc indépendants de toutes contingences matérielles, logiciels et de communication, contractualisés (en terme de performances, niveau de sécurité, coûts) [19]. Les services de Cloud Computing sont normalement des solutions partielles qui doivent être composées pour fournir un seul service virtualisé aux consommateurs de Cloud. Cette composition de services doit être effectuée de manière automatisée et dynamique afin de répondre rapidement aux exigences du client.

Il est prévu que de nombreuses défaillances se produiront, ce qui prolongera, d'un côté, le temps prévu pour répondre aux demandes des clients, et épuisera les ressources du Cloud [20]. Notre objectif est de proposer une méthode permettant de remédier à ce problème. Elle se base sur la notion de recouvrement en arrière et la sélection des chemins minimaux nécessaires à partir d'un graphe. Ce dernier représente les différents plans possibles d'exécution pour un tel service composé.

X. Travaux connexes

Beaucoup de travaux ont étés proposés dans le domaine de la tolérance aux pannes dans les infrastructures du Cloud Computing, dont il y a beaucoup d'axes de recherche d'actualité dans ce domaine. Les infrastructures du Cloud ont introduit des nouvelles questions liées à la tolérance aux pannes. Dans ce contexte on va exposer quelques travaux similaires.

Le module de Checkpoint est chargé de sauvegarder périodiquement une copie d'état des machines virtuelles. Faire un Checkpoint de la tâche qui s'exécute dans une machine virtuelle doit inclure toutes les informations nécessaires pour reprendre l'exécution de la tâche dans un autre nœud. Ce qui nécessite la sauvegarde de l'état courant des tâches.

Kong et. Coll. [21] a présenté un modèle de tolérance aux pannes et de performance dans les infrastructures virtuelles. Mais il n'est pas bien adapté à la tolérance aux pannes des applications dans le Cloud.

Ravi Jhawar, Vincenzo Piuri, Marco Santambrogio, dans [22], ont décrit le Framework FTM (Fault Tolerance Manager). Il présente la gestion des techniques de fiabilité pour les utilisateurs dans laquelle il peut spécifier et appliquer le niveau désiré de tolérance de panne sans avoir besoin d'aucune connaissance sur sa mise en œuvre à base d'une couche de service utilisateur dédié.

Une autre étude vise à fournir une meilleure compréhension des problèmes de tolérance aux pannes et identifie les différents outils et techniques. Lorsque plusieurs instances d'une application sont en cours d'exécution sur plusieurs VM et l'un des serveurs tombe en panne, il est nécessaire de mettre en œuvre une technique de tolérance aux pannes qui peut gérer ces types de pannes. Pour résoudre ce problème, les auteurs de [23] ont proposé et mis en oeuvre la HAProxy.

Stratégie proposée dans [24] implique comment calculer le nombre optimal de points de contrôle en fonction de la distribution des événements de défaillance. Divers paramètres tels que la surcharge de pointage de contrôle, la temporisation ont un impact sur le système de Cloud. Pour cela, et afin d'optimiser l'impact et les meilleures performances, un algorithme adaptatif a été conçu. Dans [24], les paramètres pris en compte pour la mise en oeuvre du modèle sont : la demande de tâche multiple par l'utilisateur, le nombre de travaux, le coût du point de contrôle, la position du point de contrôle, la probabilité de défaillance, le temps d'exécution et le temps d'horloge. Les résultats expérimentaux ont montré la meilleure adéquation du système pour les applications à grande échelle.

Rajesh.S, KannigaDevi.R, dans [25] La méthode qu'ils ont proposée utilise deux ensembles différents de nœuds, l'un est l'ensemble des VM et l'autre est le nœud d'arbitrage principal (nœud serveur). La machine virtuelle utilise le teste d'acceptation pour validité sa logique. Le serveur contient le temps checker qui est l'évaluateur de la fiabilité et le mécanisme de décision pour trouver la VM fiable qui s'occupera pour traiter la demande du client. Pour fournir la tolérance aux pannes, les données peuvent être stockées sur de multiples Cloud utilisant la technique de virtualisation.

Un autre modèle de tolérance aux pannes dans le Cloud Computing est proposé par Anjali D.Meshram, A.S.Sambare, S.D.Zade dans [26]. Le modèle FTMC (Fault Tolerance Model for Cloud) tolère les pannes sur une base de fiabilité de chaque nœud, ce dernier peut être supprimé si les applications qu'il héberge ne fonctionnent pas bien.

Dans [27], la stratégie antérieure incluait un ordonnancement non préemptif avec un algorithme de migration de tâches. La méthode proposée avait un inconvénient majeur de recommencer la tâche dans une autre machine virtuelle. La solution proposée inclut un algorithme qui migre tâche abandonnée et démarre l'exécution à un point où le dernier point de contrôle a été enregistré. Cela conduit à de meilleures performances et atteint la QoS. Le principal défaut de cette méthode est d'augmenter considérablement le temps d'exécution de la tâche migrée.

XI. Discussion :

Le Cloud Computing fait référence à l'utilisation des capacités de calcul des ordinateurs Distants, où l'utilisateur dispose d'une puissance informatique considérable sans avoir à posséder désunîtes puissantes. L'approche du Cloud Computing s'appuie principalement sur le concept de virtualisation. Où ce concept est un ensemble de techniques permettant de faire fonctionner sur une seule machine plusieurs systèmes d'exploitation et/ou plusieurs applications, isolés les uns des autres.

Un Cloud est constitué d'un ensemble de machines virtuelles qui utilisent la même infrastructure physique. Notre architecture est basée sur le paradigme d'agent, car les agents sont capables de résoudre les problèmes de manière indépendante et peuvent collaborer les uns avec les autres pour atteindre les objectifs. Dans notre architecture, nous supposons qu'un agent planificateur (PA) qui conçoit les différents plans qui représentent les alternatifs nécessaires pour avoir la composition appropriée des services répondant à la demande de l'utilisateur. Ces plans constituent un graphe orienté lui permettant de définir **le chemin idéal (un sous plan minimal)** et il sauvegardera périodiquement un point de contrôle dans sa mémoire stable à condition qu'il existe un autre sous plan depuis le site du point de contrôle, et que le service est effectivement disponible au moment de la production de la panne. Et l'agent Contrôleur (AC) dont le rôle est de contrôler et d'assurer l'exécution correcte du plan choisi. En cas d'apparition d'un défaut lors de l'exécution, il informe l'AP afin de prendre les mesures nécessaires (une explication détaillée de comportement de ces agents sera introduite dans les sections qui suivent).

XII. Méthode et architecture proposés :

1. Aperçu de l'architecture proposée du système :

La sûreté de fonctionnement est un élément de première importance, c'est pourquoi un mécanisme de tolérance aux pannes devient nécessaire pour en assurer certains aspects. Notre contribution consiste à proposer une approche permettant de masquer la défaillance d'un composant lors de l'exécution d'un service composé.

La dualité entre agent comme entité autonome et adaptive et SMA comme organisation décentralisée coopérative offre un cadre tout à fait privilégié pour résoudre un problème donné tel que la tolérance aux fautes dans le cloud computing. C'est pour ces raison on utilise les SMA dans notre travail.

L'architecture de notre système est illustrée dans la figure 21, elle est basée sur le concept d'agent et est inspirée d'un travail antérieur. Les principaux agents utilisés sont l'agent planificateur (AP) et l'agent contrôleur de plan (AC).

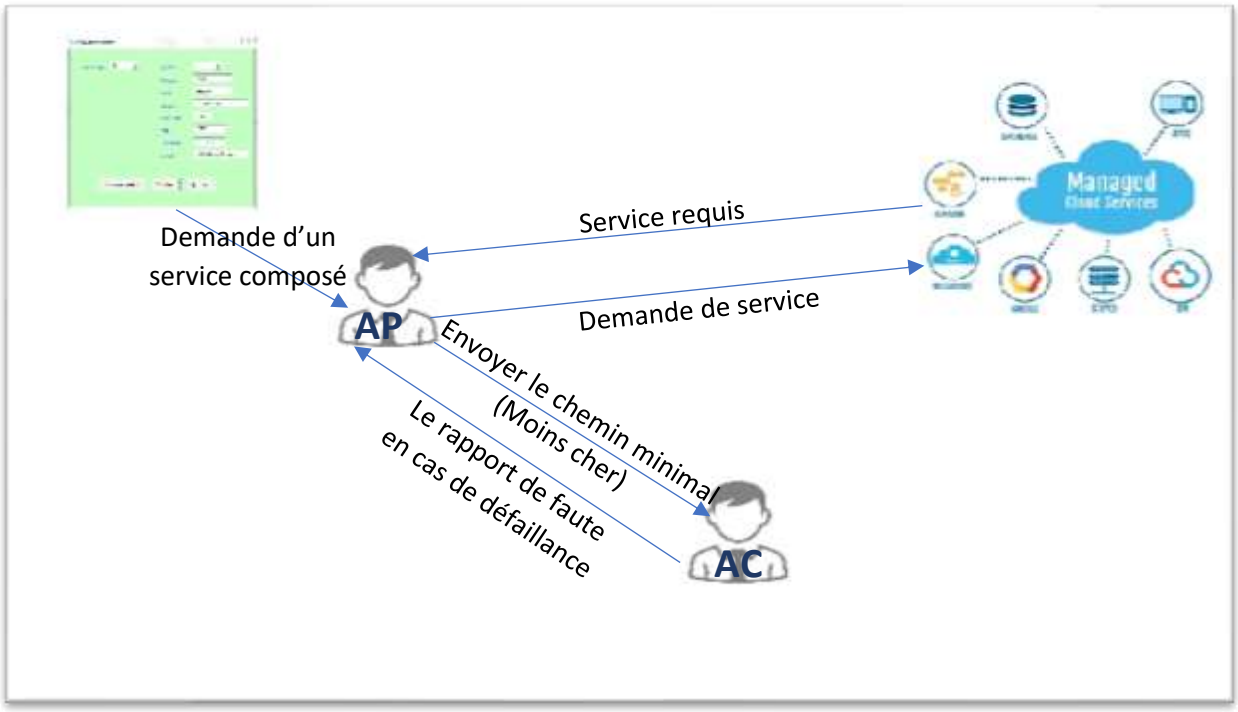

Figure 21: Aperçu global de l'architecture proposée

Dans l'architecture proposée, lorsque l'agent planificateur AP reçoit une demande d'un service composé depuis le client, il envoie à son tour une demande des services composants. Lorsqu'il reçoit ces derniers, il conçoit tous les plans possibles pour l'exécution de ce service composé. Il cherche le chemin minimal (le plan ayant un coût le moins cher), il transfère ce dernier à l'agent contrôleur (AC) qui est responsable du bon fonctionnement du chemin minimal choisi. En cours d'exécution du plan minimal choisi, l'AP enregistre, à chaque fois, un service comme un point de reprise à condition que depuis ce service il existe au moins un autre chemin minimal. En cas de détection de faute par l'AC, il informe l'AP par l'envoie d'un rapport de faute qui contient à quel niveau l'erreur s'est-elle produite et à quel service. Ainsi,

l'AP va retourner au dernier point de reprise enregistrer est choisir un autre sous plan minimal (sous chemin avec le coût minimal), donc on va reprendre l'exécution depuis ce service. Enfin, l'AP envoie le service requis au client.

2. Fonctionnement de la méthode proposée :

Dans cette partie, on va illustrer comment notre méthode fonctionne et répond aux objectifs fixés au début, à travers un diagramme de séquence illustré dans la figure 22. On utilise la technique de point de reprise (le check-pointing) basé sur le recouvrement arrière, et la planification multi agents en choisissant les chemins minimaux.

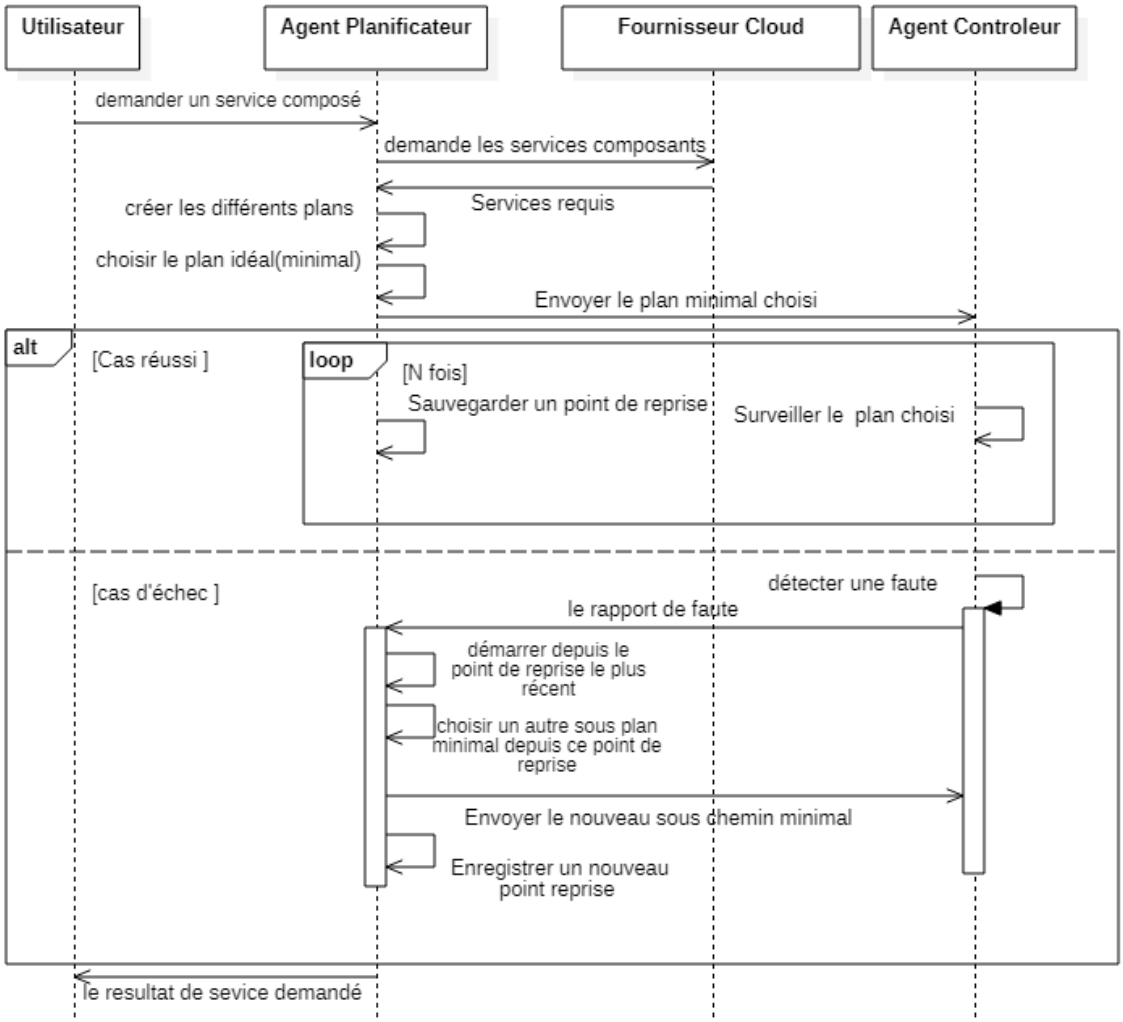

Figure 22: le diagramme de séquence qui représente le fonctionnement de la méthode proposée.

Selon le diagramme de séquence de la figure 22, l'agent planificateur (AP) après avoir reçu une demande d'un service composé, il la transmet au fournisseur Cloud, ainsi il crée les différents plans de composition des services dont chaque plan représente une solution possible pour répondre au besoin de client. Ensuite, il choisit le *chemin minimal* parmi eux et l'envoie à l'agent contrôleur (AC). Ce dernier assure le bon fonctionnement du plan sélectionné. Pendant ce temps, l'AP sauvegarde périodiquement un point de reprise. Lorsqu'une faute est détectée par l'AC, il informe l'AP pour choisir un autre sous plan idéal

(minimal) depuis le point de contrôle le plus récent, et ainsi de suite jusqu'à ce que le service composé sera exécuté avec succès.

3. Exemple illustratif :

On suppose que les différents plans d'un service composé constituent un graphe orienté tel que : (voir figure 23)

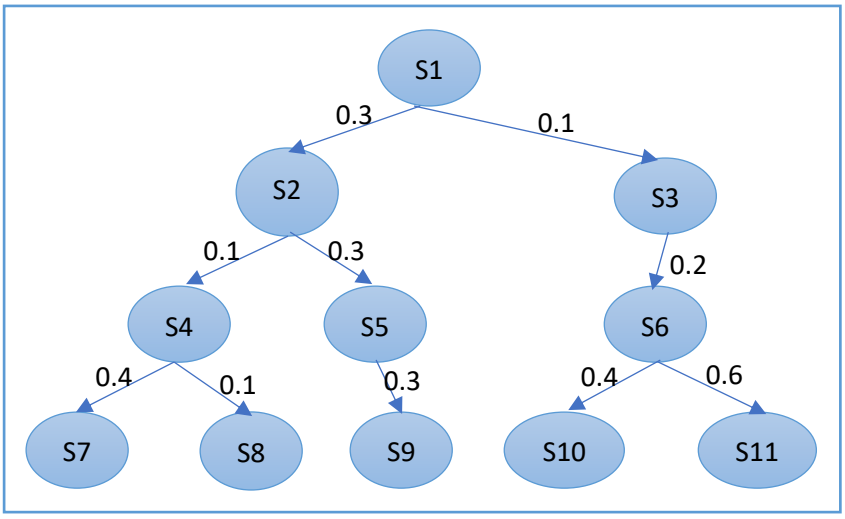

Figure 23: Plans d'un service composé

- La *racine* est le service initial (début pour tous les plans alternatifs d'un service composé donné), le service S1.
- Les *nœuds* sont les services composants, comme le service S2.
- Chaque *arc* orienté est un lien de succession entre 2 nœuds, il a une valeur pondérée entre : le coût de service, le temps d'exécution, et la fiabilité. Ces caractères sont utilisés par l'AP pour choisir le plan idéal.

Ces trois valeurs ne sont pas de même type, alors il faut les normaliser. Pour bien expliquer ce point, on fait référence aux trois tableaux suivants : tableau 4- tableau 5- tableau 6

Prix de service ($)	[50-100[	[100-150[	[150-200[	P>=200
La valeur normalisé	0.1	0.2	0.3	0.4

Tableau 4: comment estimer le cout d'un service

Nombre d'utilisateurs	[1k-2k[	[2k-3k[	[3k-4k[	Nb>=4k
La fiabilité	0.1	0.2	0.3	0.4

Tableau 5: Comment estimer la fiabilité

Temps d'exécution (s)	[0.1-.03[	[0.3-0.5[	[0.5-0.7[	T>=0.7
La valeur estimé	0.1	0.2	0.3	0.4

Tableau 6: comment estimer le temps d'exécution

Donc, pour estimer le poids de l'arc entre les deux service S1 et S2 par exemple (figure 22), il faut utiliser ces valeurs qui sont illustrer dans les tableaux (4,5,6) et appliquer cette formule :

$$Poinds(S1 - S2) = \frac{\frac{P(S1) + F(S1) + T(S1)}{3} + \frac{P(S2) + F(S2) + T(S2)}{3}}{2}$$

Tels que :

- P(Si) : la valeur estimée du prix d'un service.
- F(Si) : la valeur estimée de la fiabilité.
- T(Si) : la valeur estimée du temps d'exécution.

4. Comportement de l'agent planificateur (AP):

L'agent planificateur a une connaissance complète de tous les services déployés dans le Cloud, il est responsable de la création des différents plans d'un service composé sous forme d'un graphe orienté afin de répondre à la demande de l'utilisateur. Dans notre travail, on suppose que l'AP a un ensemble de plans (la création des plans par AP ne fait pas partie de ce travail).

L'AP enregistre périodiquement un point de reprise (ou on dit un point de contrôle) en utilisant le mécanisme de mémoire stable à condition que depuis ce point il existe au moins deux sous plans et le service est disponible au moment de la production de la panne. Les données à sauvegarder pour chaque point de reprise sont :

- Id de service
- La longueur de service (la taille de service à exécuter dans une ressource Cloud)
- Le service FileSize (la taille du service fichier avant l'exécution)
- Le service OutputSize (la taille de résultat d'exécution du service)
- L'état de ce service

En cas d'une panne, il applique la technique de recouvrement arrière et sélectionne un autre sous chemin minimal qui a un coût minimal. La tâche qui a échoué est relancée depuis le point de contrôle récent plutôt que depuis le début.

Afin de mieux comprendre le comportement de cet agent, on utilise le diagramme d'activité illustré par la figure 24 :

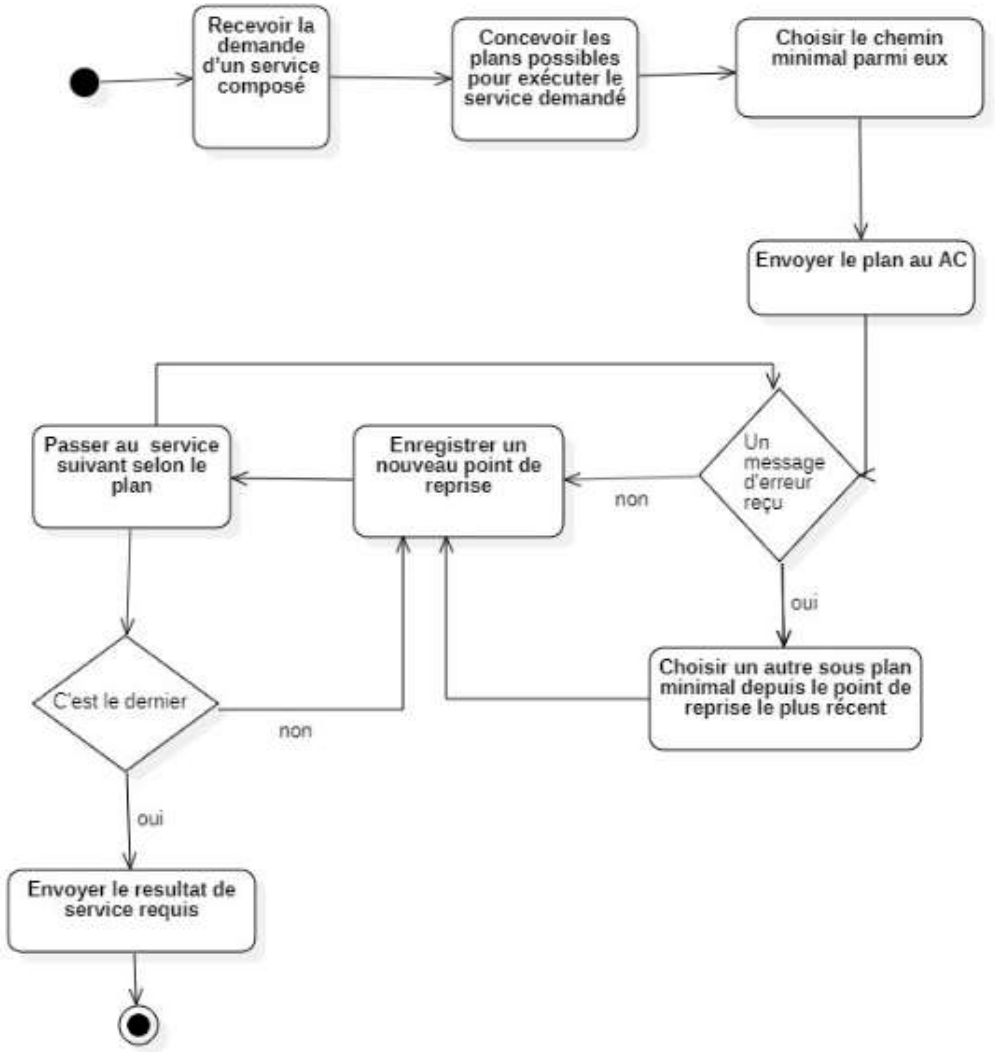

Figure 24: diagramme d'activité de l'agent planificateur

a) L'AP et l'algorithme de Dijkstra :

Pour sélectionner le plan minimal (le chemin minimal), l'agent planificateur utilise **l'algorithme de Dijkstra,** ce dernier permet de trouver le plus court chemin à partir un nœud initial.

Pour comprendre le déroulement de cet algorithme, on l'applique sur le plan orienté de la figure 25 :

1. On initialise le nœud de départ par 0et les autres par ∞ (voir tableau 7)
2. On se place au sommet de plus petit poids, ici le sommet S1.
3. On traite les arcs du sommet où on a placé.
4. Ensuite, on passe à un autre nœud qu'il a la plus petite étiquette.
5. Et ainsi de suite jusqu'à terminer tous les nœuds.

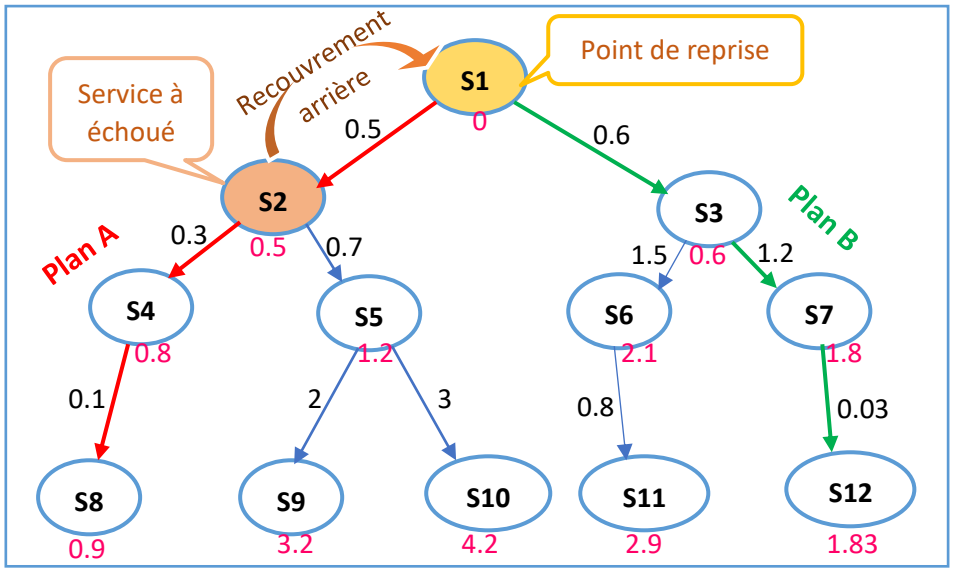

Figure 25: les différents plans d'un service composé en cas d'échec

S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
0	0.5	0.6	∞	∞	∞	∞	∞	∞	∞	∞	∞
0	0.5	0.6	0.8	1.2	∞	∞	∞	∞	∞	∞	∞
0	0.5	0.6	0.8	1.2	2.1	1.8	∞	∞	∞	∞	∞
0	0.5	0.6	0.8	1.2	2.1	1.8	0.9	∞	∞	∞	∞
0	0.5	0.6	0.8	1.2	2.1	1.8	0.9	3.2	4.2	∞	∞
0	0.5	0.6	0.8	1.2	2.1	1.8	0.9	3.2	4.2	∞	1.83
0	0.5	0.6	0.8	1.2	2.1	1.8	0.9	3.2	4.2	2.9	1.83

Tableau 7: Déroulement de l'algorithme de Dijkstra sur le plan précédent

Pour exécuter ce service composé (figure 24), on prend le chemin A (S1, S2, S4, S8) car il a la valeur minimal (0.9), si une panne est produite au niveau de service S2, on retourne vers le point de reprise le plus récent (S1), et on choisit un autre plan minimal : on prend le plan B (S1, S3, S7, S12).

b) Comment enregistrer les points de reprise et les utiliser par l'AP :

Le pseudo code ci-dessous permet de montrer comment l'AP procède pour enregistrer des points de reprise et les utiliser en cas d'une panne. Ce code s'exécute d'une manière répétitive afin de retourner comme résultat un service composé fiable.

```
Algorithme Enreg_point_rep
    chemin_choisi, Tab1, rapport_faute : tableau dynamique ;
    point_reprise : service ;
Début
 point_reprise=chemin_choisi[service1] ;
 Tab1=point_reprise ;
 Tant que (l'exécution de service composé n'est pas terminée) faire
      Si (chemin_choisi[service i]≠rapport_faute[serviceN])alors
           point_reprise=chemin_choisi[service i] ;
           Tab1=Tab1+point_reprise ;
      Si non
           Si (chemin_choisi[service i]==rapport_faute[serviceN])alors
                serviceJ=point_reprise ;
                Fonct_plus_court_chemin(liste_chemin(serviceComp),serviceJ) ;
           Finsi ;
      Finsi ;
 FinTq ;
 Afficher le contenu de Tab1 ; //tout les points de reprise
 Afficher le contenu de chemin_choisi;
Fin.
```

5. Comportement de l'agent contrôleur de plan (AC) :

Le rôle de cet agent est d'assurer et contrôler le bon fonctionnement du plan minimal choisi par l'AP en cours d'exécution. Si une faute est détectée, il informe l'AP par un rapport de faute qui contient les informations utiles tel qu'à quel niveau la faute est produite. Le diagramme d'activité ci-dessous permet de mieux illustrer le comportement de l'AC. (Voir figure 26)

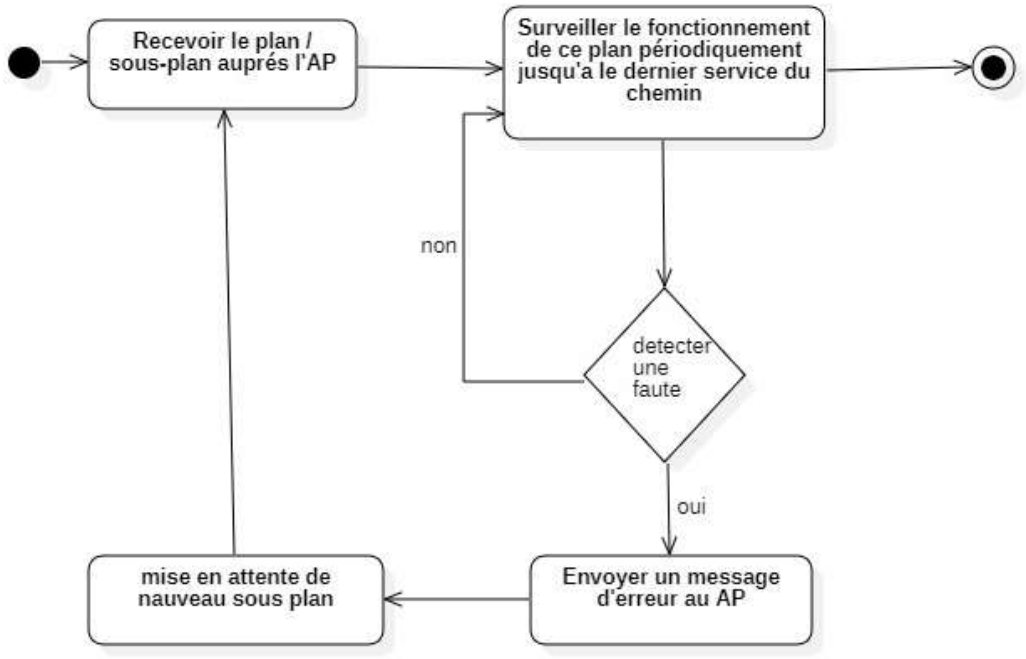

Figure 26: le diagramme d'activité de l'agent contrôleur

6. Communication inter-agents:

La communication permet aux agents de:

- Coordonner leurs actions et comportement.
- Essayer de modifier les états des autres agents.
- Essayer de persuader les autres agents de faire des certaines actions.

Chaque agent possède un modèle de communication qui gère la réception et l'interprétation des messages reçus. On peut citer, par exemple, le langage de communication ACL (Agent Communication Language) dont la spécification permet une interopérabilité maximale entre agents d'un SMA. ACL a été spécifié par la FIPA (Foundation for Intelligent Physical Agents) en 1996. ACL est basée sur la théorie des actes de langage, Une expression ACL est représentée sous la forme : <acte_de_communication> (<contenu>). [28]

Exemples :

- Affirmation (il pleut) ou (! x "il pleut")
- Question (il pleut) ou (? x "il pleut")

Les messages échangés entre les agents de notre architecture sont comme suit (voir figure 27 et figure 28)

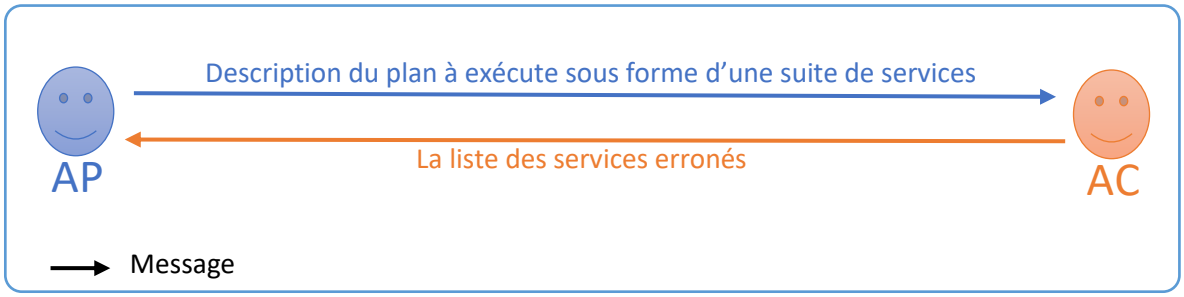

Figure 27: Echange de messages entre l'AP et l'AC

Figure 28: la communication entre agents en utilisant ACL Message

7. Scénario illustratif :

Le but de ce scénario est d'illustrer deux points:

- Le déroulement de notre méthode sur un cas réel.
- Et, l'interaction entre les différents agents de l'architecture proposée.

Pour cela, on a choisi l'exemple du service composé de voyage organisé. L'AP procède à la représentation des services sous la forme d'un graphe orienté (figure 29).

Tout d'abord, l'AP demande l'ensemble de services qui sont les composants du service composé demandé (voyage organisé) auprès d'un fournisseur Cloud, les services sont organisés en fonction de leur ordre chronologique.

L'AP applique l'algorithme de Dijkstra et il a obtenu les valeurs illustrées dans le graphe de la figure 29, afin de choisir le chemin le moins coûteux. On obtient le chemin minimal suivant : *Voyage organisé -> destination 1 -> billet d'avion -> hôtel 5 étoiles*.

A chaque fois, l'AP conserve un service comme un point de contrôle à condition que depuis ce service il existe au moins un autre chemin. Au même temps, l'AC veille sur le fonctionnement correct de plan choisi.

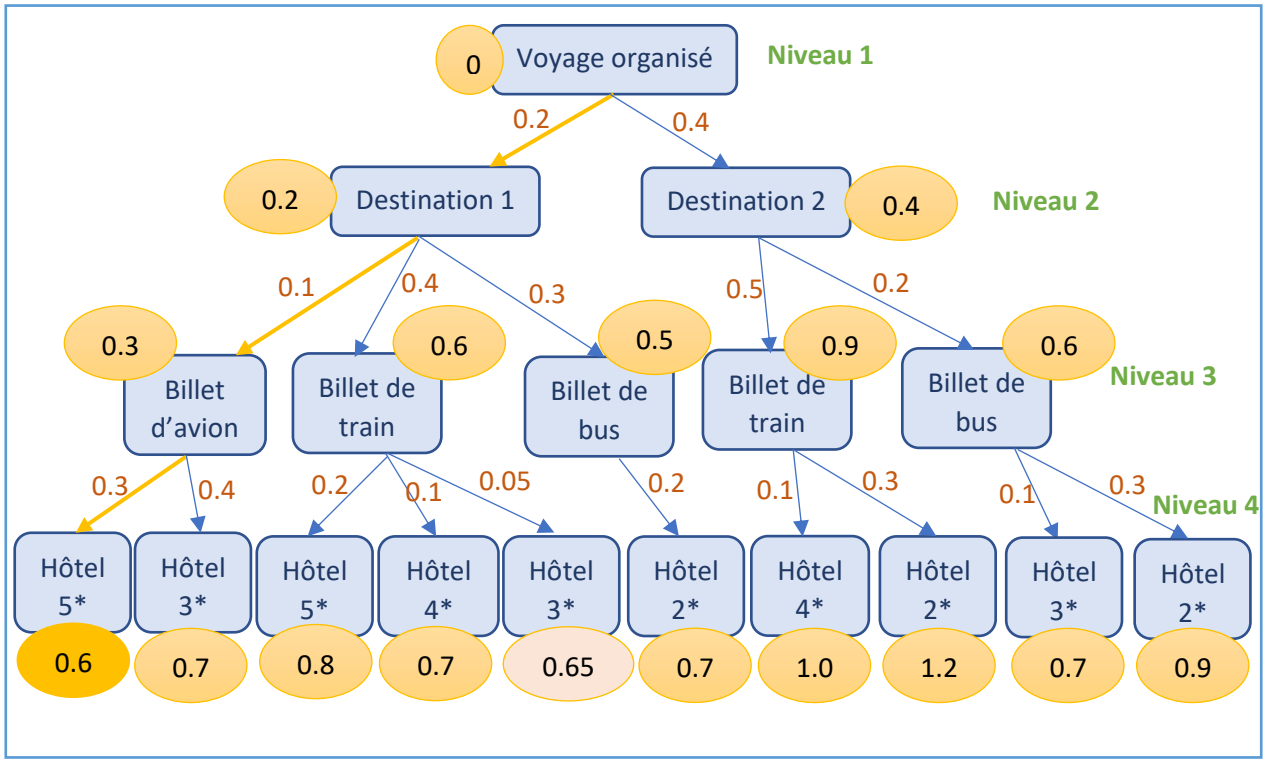

Figure 29: le graphe orienté des différents plans du service demandé

On suppose qu'un échec est produite au niveau 3 dans le service de réservation de billet d'avion : (voir figure 30)

Voyage organisé -> destination 1 -> billet d'avion -> hôtel 5 étoiles

Dans ce cas, l'AC envoie un rapport de faute à l'AP comme suit :

(Niveau3 : Service : réservation de billet d'avion)

Donc l'AP après qu'un message erreur reçu, il applique le mécanisme de recouvrement arrière (Back Recovery) au point de reprise le plus récent, et dans notre cas c'est :

(Niveau 2 : Service : destination 1)

On réapplique l'algorithme de Dijkstra pour trouver un nouveau sous plan qui soit différent de celui où la défaillance s'est produite. (Figure 30)

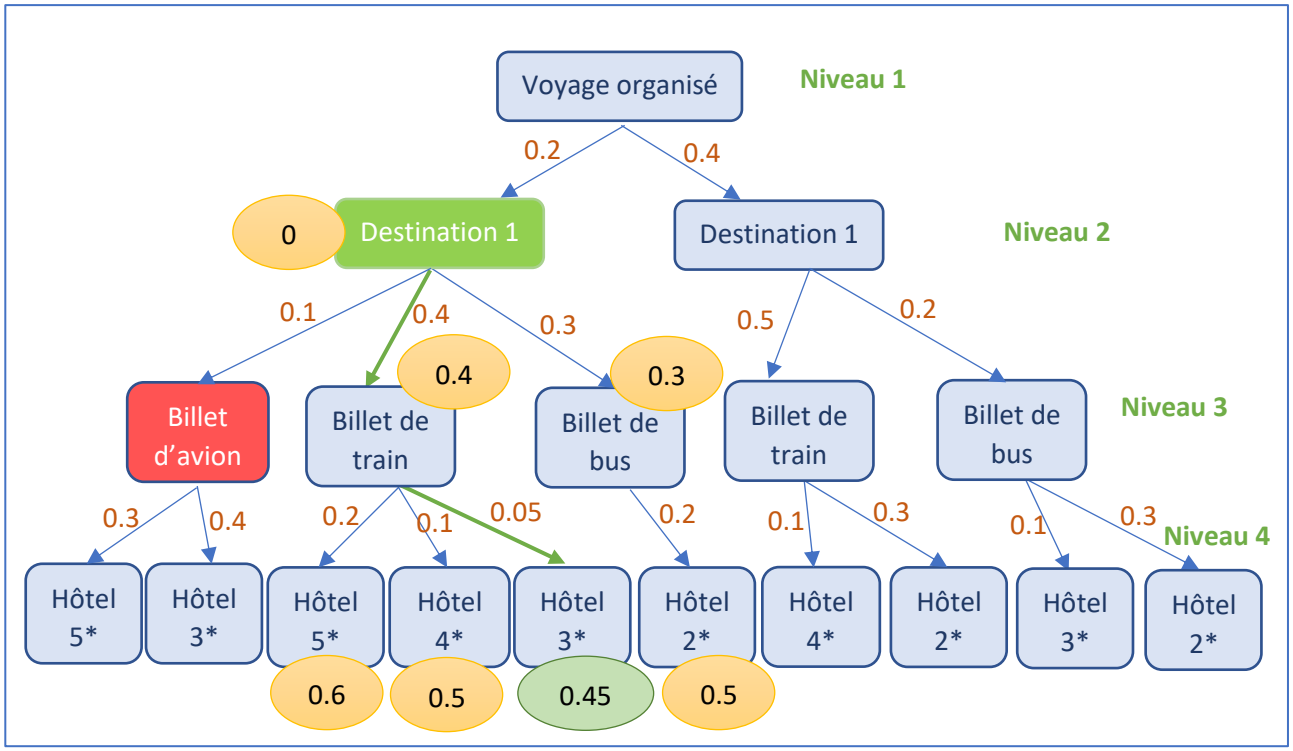

Figure 30: le graphe après la ré-application de l'algorithme de Dijkstra

Alors, le nouveau sous plan est : *Destination1 -> billet de train -> hôtel 3 étoiles.*

XIII. Conclusion :

Dans ce chapitre, nous avons proposé une méthode de tolérance aux pannes permettant d'assurer la sûreté de fonctionnement de Cloud Computing. Cette dernière oblige le système à délivrer correctement un service composé malgré l'occurrence des fautes en utilisant le mécanisme de récupération par retour arrière qui se base sur la vérification périodique des points de reprise. Nous sommes concentrés la description et la conception de la méthode proposée pour traiter la problématique que nous avons choisie.

Le modèle de notre architecture se base sur la planification multi agents. Elle se compose de deux principaux agents : AP et AC, nous avons présenté leurs rôles et leurs comportements à l'aide des diagrammes d'activités.

Le prochain chapitre sera consacré à la partie implémentation de notre proposition en utilisant la plateforme JADE et le simulateur CloudSim.

CHAPITRE4 : Mise en œuvre de la méthode proposée

Introduction :

Ce chapitre est consacré à la réalisation et la concrétisation de notre modèle proposé dans le chapitre précèdent, qui consiste à tolérer les pannes dans l'environnement Cloud. Dans un premier temps, nous présentons l'environnement de développement matériel et logiciel. Ensuite, nous décrivons quelques interfaces graphiques, pour mettre en évidence notre proposition, et finalement nous terminerons par une conclusion.

XIV. Langage et environnement de développement :

Nous avons réalisé ce travail sur une machine avec un processeur intel(R) Core(TM) i5-3210M CPU @ 2.50 GHz, doté d'une capacité mémoire de 4 Go, sous Windows 10 Professionnel de 64bits. Nous avons utilisé l'environnement de développement Eclipse.

1. Langage de programmation java :

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

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

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

Java permet de développer des applications autonomes mais aussi, et surtout, des applications clientserveur. Côté client, les applets sont à l'origine de la notoriété du langage. C'est surtout côté serveur que Java s'est imposé dans le milieu de l'entreprise grâce aux servlets, le pendant serveur des applets, et plus récemment les JSP (JavaServer Pages) qui peuvent se substituer à PHP, ASP et ASP.NET. Les applications Java peuvent être exécutées sur tous les systèmes d'exploitation pour lesquels a été développée une plate-forme Java, dont le nom technique est JRE (Java Runtime Environnement -Environnement d'exécution Java). Cette dernière est constituée d'une JVM (Java Virtual Machine -Machine Virtuelle Java), le programme qui interprète le code Java et le convertit en code natif. Mais le JRE est surtout constitué d'une bibliothèque standard à partir de laquelle doivent être développés tous les programmes en Java. C'est la garantie de portabilité qui a fait la réussite de Java dans les

architectures client-serveur en facilitant la migration entre serveurs, très difficile pour les gros systèmes. [14]

2. Environnement de développement : Eclipse

Eclipse est un projet, décliné et organisé en un ensemble de sous-projets de développements logiciels. Eclipse vise à développer un environnement de production de logiciels libre qui soit extensible, universel et polyvalent, en s'appuyant principalement sur Java.

Son objectif est de produire et fournir des outils pour la réalisation de logiciels, englobant les activités de programmation (notamment environnement de développement intégré et frameworks) mais aussi d'AGL recouvrant modélisation, conception, test, gestion de configuration, reporting… Son EDI, partie intégrante du projet, vise notamment à supporter tout langage de programmation à l'instar de Microsoft Visual Studio.

Bien qu'Eclipse ait d'abord été conçu uniquement pour produire des environnements de développement. Les utilisateurs et contributeurs se sont rapidement mis à réutiliser ses briques logicielles pour des applications clientes classiques. Cela a conduit à une extension du périmètre initial d'Eclipse à toute production de logiciel : c'est l'apparition du framework Eclipse RCP en 2004. Dans notre travail nous avons utilisé la version Eclipse IDE 2020 – 06. (Voir figure 31)

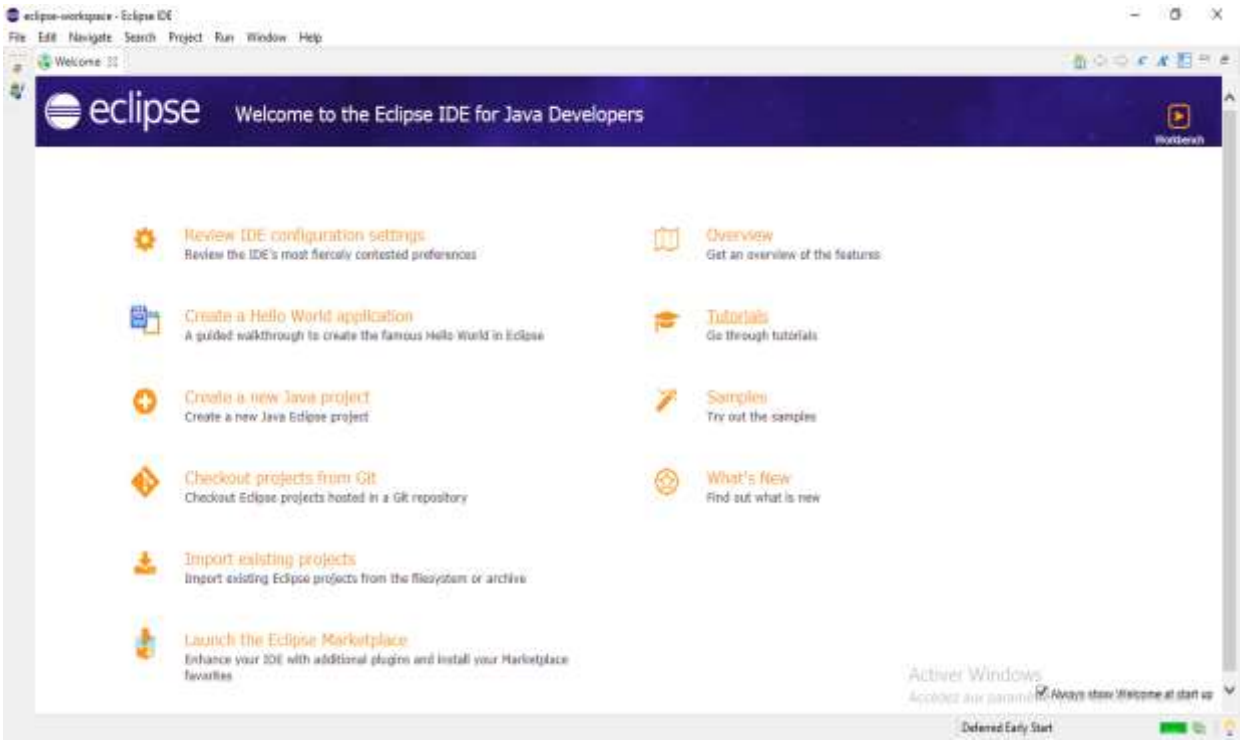

Figure 31: Environnement de développement d'Eclipse

3. CloudSim :

CloudSim est un nouveau cadre de simulation, général, et extensible, qui permet la modélisation, la simulation et l'expérimentation de nouvelles Cloud Computing infrastructures et de services d'application.

Dans le cas Cloud Computing, les outils de Simulations comme CloudSim donne une offre d'importants avantages aux clients et fournisseurs. Pour les clients, il permet de tester leurs Services dans un environnement contrôlable avec exempt du coût et de vérifier la performance avant de publier les vrais nuages. Cela permettra d'optimiser le coût de l'accès aux ressources en améliorant les bénéfices. Sans ces outils, les clients et les fournisseurs doivent s'appuyer sur des évaluations imprécises, ces approches peuvent conduire à l'inefficacité des performances des services et de réduire la génération des revenus. En outre, CloudSim aide les chercheurs et développeurs basés sur l'industrie pour tester la performance d'un service d'application développée dans un convenable environnement facile à installer. Il y a de nombreux avantages de l'utilisation de CloudSim pour tester les performances de départ, on peut citer :

- l'efficacité de temps : il prend très moins de temps et d'efforts pour mettre en oeuvre des applications de Coud Computing
- la flexibilité : les développeurs peuvent facilement modéliser et tester les performances de leurs applications et ses services dans des environnements hétérogènes (Microsoft Azure, Amazon EC2). [29]

Dans notre travail nous avons utilisé la version CloudSim 3.0.3.

L'architecture du CloudSim :

La structure logiciel de CloudSim et ses composant est représentée par une architecture en couche comme il est montré dans la figure 32. Au niveau le plus bas on trouve le moteur de simulation aux événements discrets SimJava SimJava implémente les fonctionnalités de base requises pour les cadres de simulation au niveau supérieur, telles que les files d'attente, le traitement des événements, la création de composants du système (services, hôte, Datacenter, Broker, les machines virtuelles), la communication entre les composants et la gestion de l'horloge de simulation.

CloudSim supporte la modélisation et la simulation de l'environnement de Datacenter basé sur Cloud, tel que des interfaces de gestion dédiées aux VMs, la mémoire, le stockage et la bande passante. La couche CloudSim gère l'instanciation et l'exécution des entités de base (VM, hôtes, Datacenters, applications) au cours de la période de simulation. Dans la couche plus haute de la pile de simulation, on trouve le code de l'utilisateur qui expose la configuration des fonctionnalités liées aux hôtes (ex: nombre de machines, et leurs spécifications), les politiques d'ordonnancement de Broker, applications (ex: nombre de tâches et leurs besoins), VM et le nombre d'utilisateurs. [29]

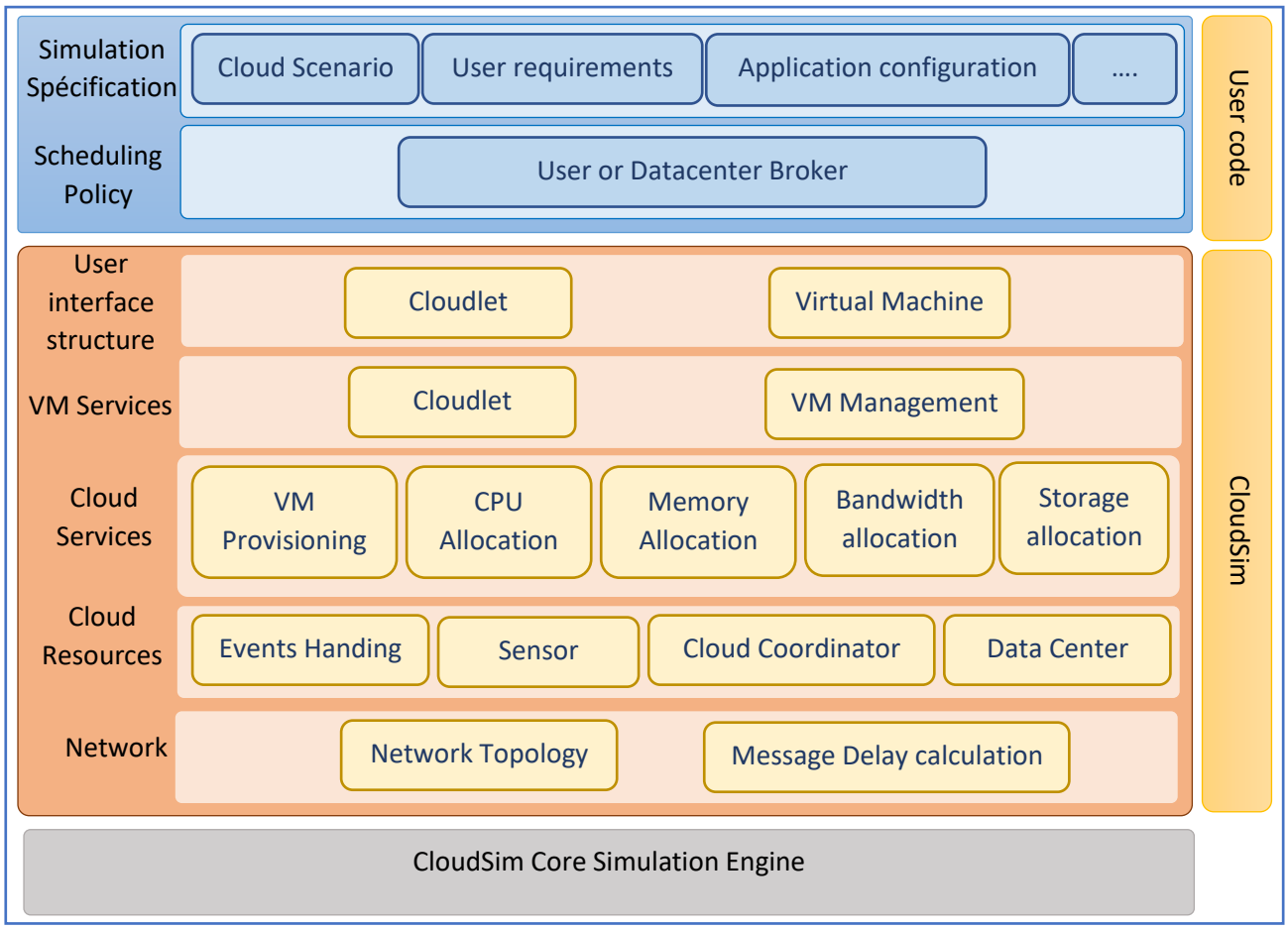

Figure 32: Les couches de l'architecture du CloudSim

a) Cloudlet:

Cette classe modélise les services d'application de base du Cloud (tels que la livraison, la gestion des réseaux sociaux et le déroulement des opérations métier de l'entreprise) qui sont généralement déployés dans des Datacenter. CloudSim orchestre de la complexité de la demande en fonction de ses exigences de calcul. Représente la complexité d'une application en termes de ce besoin informatique (taille en nombre d'instruction, temps de réponse..). Chaque service d'application a un transfert de longueur d'instruction et les données pré-attribué les frais généraux qu'il doit entreprendre au cours de son cycle de vie. Cette classe peut aussi être prorogé pour soutenir la modélisation d'autres mesures de performance et de composition pour des applications telles que les transactions dans les applications orientées base de données.

b) VirtualMachine :

Cette classe modélise une instance de virtuelle machine (VM), qui est géré et hébergé pendant son cycle de vie par le composant Cloud host. Un host peut, simultanément, instancier de multiples VMs et assigner des cœurs à base de politiques prédéfinies de partage de processeur (espace partagé, temps partagé).

c) DataCentre :

Cette classe modélise l'infrastructure du noyau du service (matériel, logiciel) offert par des fournisseurs de ressources dans un environnement de Cloud Computing. Il encapsule un ensemble de machines de calcul qui peuvent être homogènes ou hétérogènes en ce qui concerne leurs configurations de ressources (mémoire, noyau, capacité et stockage). En outre, chaque composant de Datacenter instancie un composant, généralisé, d'approvisionnement de ressource qui implémente un ensemble de politiques d'allocation de bande passante, de mémoire et des dispositifs de stockage. L'hôte représente un serveur informatique physique dans un Cloud. Il exécute des actions liées à la gestion des machines virtuelles et à une politique définie pour l'approvisionnement mémoire, ainsi que d'une politique de répartition à des machines virtuelles. Un hôte est associé à un DataCentre. Il peut héberger des machines virtuelles.

d) DataCentreBroker :

Cette classe modéliser le courtier (Broker), qui est responsable de la médiation entre l'utilisateur et les prestataires de service selon les conditions de QoS des utilisateurs et il déploie les tâches de service à travers les Clouds.

Le Broker agissant au nom des utilisateurs identifie les prestataires de service appropriés du Cloud par le service d'information du Cloud CIS (Cloud information services) en négocie avec eux pour une allocation des ressources qui répond aux besoins de QoS des utilisateurs.

4. La plateforme JADE

JADE (Java Agent Development framework) est une plateforme implémentée avec le langage JAVA par le laboratoire TILAB. Elle est destinée aux développeurs qui s'intéressent aux systèmes multi-agents distribués. La plateforme JADE fournit une interopérabilité sans limite pour les applications qu'elle prend en charge. Cette plateforme est indépendante du système d'exploitation ainsi que du matériel sur laquelle elle est implémentée [30]. Cette plateforme permet le développement des applications conformes aux normes FIPA qui propose le modèle AMRM (Agent Management Reference Model) comme le modèle de base de l'architecture de la plateforme JADE. Les principaux modules qui composent l'architecture JADE sont : le DF, l'AMS et également le MTS (Massage Transport Service), un moyen pour la communication entre plusieurs plateformes JADE [31].

Nous avons développé notre système à l'aide de JADE-S, la version 4.5.0 de JADE. JADE-S est formée par la combinaison de la version standard de JADE et du plug-in de sécurité [32]. JADE-S inclut des fonctionnalités de sécurité telles que: [30]

- *L'autorisation et stratégies* : l'autorisation est une caractéristique qui décrit la possibilité d'effectuer une action par un agent sur une ressource donnée. La stratégie spécifie les autorisations disponibles pour différentes entités.
- *Les certificats et autorité de certification* : l'autorité de certification (CA) est l'entité qui signe tous les certificats pour la plateforme, à l'aide d'une paire de clés publique/privée.

- *La délégation* : ce mécanisme permet de donner des autorisations à un agent. Outre le certificat d'identité, un agent peut également posséder d'autres certificats qui lui sont donnés par d'autres agents.
- *Et, la communication sécurisée* : la communication entre les agents de différents conteneurs / hôtes est réalisée à l'aide du protocole SSL (Secure Socket Layer). Cela permet une protection solide contre les tentatives malveillantes de détection de paquets.

5. Résultats de simulation :

Après la vérification depuis l'agent CP du chemin envoyé par l'AP et dans le cas où tous les services sont en état de « succès », le résultat sera comme montré ci-dessous :

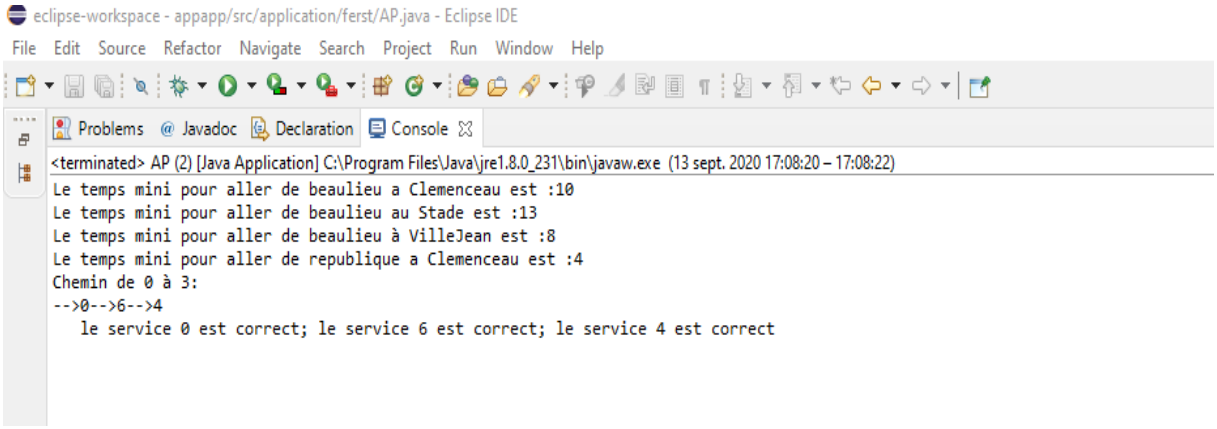

Figure 33:resultat d'exécution de cas de services corrects

Dans le cas de panne dans certains services du chemin, le résultat est :

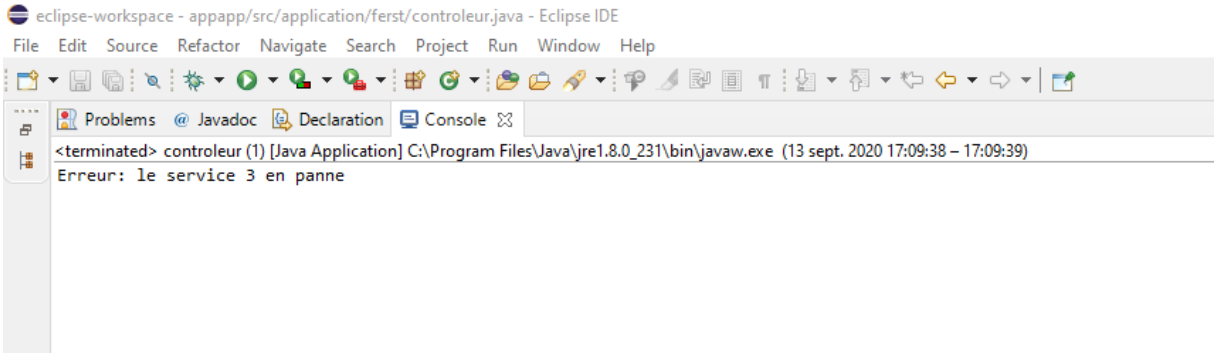

Figure 34: résultat d'exécution en cas de panne

6. Les différentes classes de notre projet :

a) La classe de l'agent planificateur :

controleur.java | AP.java

```
32    public static final int INFINITE = 1000;//Integer.MAX_VALUE;
33    public final static int ALPHA_NOTDEF = -999 ;// on met final psk c'est une constante
34    private int x0;
35    private int [] S;//ensemble de sommets dont les distances les plus courtes à la source sont c
36    private int [] R;//ensemble des prédécesseur des sommets de 0 à N-1;
37    private Graphe g0;
38    private int [] D;//tableau des valeurs du meilleur raccourci pour se rendre à chaque sommet
39    // rajout
40    private boolean [] noeudsMarqués;
41    private static int dimensionDeLaMatrice;//je rajoute ca pour simplifier le code.
42
43    public AP( int x, Graphe g){
44        x0 = x;
45        g0 = g;
46        dimensionDeLaMatrice = g0.nbSommet();
47        S = new int [dimensionDeLaMatrice];//sommets atteints
48        D = new int [dimensionDeLaMatrice];//distances
49        noeudsMarqués = new boolean[dimensionDeLaMatrice];
50        R = new int [dimensionDeLaMatrice];
51        calculePlusCourtChemin();
52    }
53
54    private void calculePlusCourtChemin(){
76
77    //implémentation de initDistMin
78    private void initDistMin(){
91
92    private void majDistMin(int n){
```

controleur.java | AP.java

```
54    private void calculePlusCourtChemin(){
76
77    //implémentation de initDistMin
78    private void initDistMin(){
91
92    private void majDistMin(int n){
103   private int distanceDsGraphe (int t, int s){
111
112   public int choixSommet(){
124
125
126   //fonction à définir :S.contains(i)
127   private boolean contains(int[] S, int i){
130
131   public int longueurChemin (int i){
134
135   //fonction à définir min
136   private int min (int i, int j){
141   public void afficheChemin(int i){
158
159   void computePath(int V) {
162
163   boolean getShortestPathTo(int V) {
166   public static void main(String[] args) {
216 }
217
218
```

Figure 35: le code source de la classe de l'agent planificateur

controleur.java | AP.java

```
2
3  import jade.core.Agent;
5
6  class controleur extends Agent {
7
8      static final int V = 9;
9      int minDistance(int dist[], Boolean sptSet[])
22
23
24     void printSolution(int dist[], int n)
30
31
32     void dijkstra(int graph[][], int src)
75
76
77     public static void main(String[] args)
78     {
79         int s ;
80         Graphe g = new Graphe();
81         g.degreEntrant(i);
82         /* Let us create the example graph discussed above */
83         int graph[][] = new int[][] { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
84                                      { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
85                                      { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
86                                      { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
87                                      { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
88                                      { -1, 0, 4, 14, 10, 0, 2, 0, 0 },
89                                      { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
```

Figure 36: le code source de la classe de l'agent contrôleur

Graphe.java

```
1  package application.ferst;
2
3
4  public class Graphe {
5
6      // Declaration de variable
7
8      private int [][] U;
9      private boolean [] valid; // sommet valide veut dire qu'il existe dans le graphe
10     public final static int ALPHA_NOTDEF = -999 ;// on met final psk c'est une constante
11
12
13      // Contructeur par defaut:
14
15     public Graphe () {
16         int NMax = 1000;
17         U = new int [NMax][NMax];
18         valid = new boolean [NMax];
19         for (int i=0 ; i<U.length ; i++){
20             valid[i] = false;
21             for (int j=0 ; j<U[i].length ; j++){
22                 U[i][j] = ALPHA_NOTDEF;
23             }
24         }
25     }
26     // constructeur via une matrice adjacence:
27
28     public Graphe (int [][] mat){
```

Graphe.java

```
28    public Graphe (int [][] mat){
29        int NMax = mat.length;
30        //int NMax = 1000 + mat.length;
31        U = new int [NMax][NMax];
32        valid= new boolean [NMax];
33        for (int i = 0 ; i<U.length ; i++){
34            if (i < mat.length){
35                valid [i] = true;
36            }
37            else {
38                valid[i] = false;
39            }
40            for (int j = 0 ; j<U.length ; j++){
41                if ((j<mat.length) && (i<mat.length)){
42                    U[i][j] = mat [i][j];
43                }
44                else {
45                    U[i][j] = ALPHA_NOTDEF;
46                }
47            }
48        }
49    }
50
51    public boolean existeSommet(int i){
52        boolean res = false;
53        if (!((i > valid.length) || (i<0))){
54            res = valid[i];
55        }
```

Graphe.java

```
56            return res;
57    }
58
59    public int nbSommet(){
60        int nb = 0;
61        for (int i = 0 ; i<U.length ; i++){
62            if (valid[i]){ nb ++;}
63        }
64        return nb;
65    }
66
67    public boolean existeArc(int i, int j){
68        return ((existeSommet(i)) && (existeSommet(j)) && (U[i][j] != ALPHA_NOTDEF));
69    }
70
71    public int getValArc(int i, int j){
72        if ((i<0) || (i >= U.length) || (j<0) || (j >= U.length)){
73            System.out.println("Graphe::getValArc : Erreur: hors de la matrice.");
74            System.exit(-1);
75        }
76        if (!existeArc( i, j)){
77            System.out.println("Graphe::getValArc : Erreur: aucun arc.");
78            System.exit(-1);
79        }
80        return U[i][j];
81    }
82
83
```

Graphe.java

```
                else
                {
                        if( degre(i)!=0)
                    {
                System.out.println("le sommet "+i+" est encore connectï¿½");
                System.exit(-1);
                 }

                valid[i]=false;
                      }

          return true;
     }


    public boolean ajoutArc(int i, int j, int val){
        if (!(existeSommet(i) && (existeSommet(j)))){
            return false;
        }
        if (existeArc(i,j)){
            return false;
        }

        U[i][j]=val;
        return true;

    }

```

Graphe.java

```

    public boolean supprimerArc(int i,int j){
        if (!(existeSommet(i) && (existeSommet(j)))){
            return false;
        }
        U[i][j]=ALPHA_NOTDEF;
        return true;
    }

    public int degreEntrant(int i){
        int x = 0;
        if ((i<0) || (i >= U.length)){
            System.out.println("Erreur: le service 3 en panne");
            System.exit(-1);
        }
        if (!existeSommet(i)){
            System.out.println("Erreur: le service 3 en panne");
            System.exit(-1);
        }
        for (int k=0 ; k<U.length ; k++){
            if (U[i][k] != ALPHA_NOTDEF){
                x++;
            }
        }
        return x;
    }

    public int degreSortant(int i){
```

Graphe.java

```
166    public int degreSortant(int i){
167        int x = 0;
168        if ((i<0) || (i >= U.length)){
169            System.out.println("Graphe::degreSortant : Erreur: hors de la matrice.");
170            System.exit(-1);
171        }
172        if (!existeSommet(i)){
173            System.out.println("Graphe::degreSortant : Erreur: ce sommet n'existe pas.");
174            System.exit(-1);
175        }
176        for (int k=0 ; k<U.length ; k++){
177            if (U[k][i] != ALPHA_NOTDEF){
178                x++;
179            }
180        }
181        return x;
182    }
183
184    public int degre(int i){
185        if ((i<0) || (i >= U.length)){
186            System.out.println("Graphe::degre : Erreur: hors de la matrice.");
187            System.exit(-1);
188        }
189        if (!existeSommet(i)){
190            System.out.println("Graphe::degre : Erreur: ce sommet n'existe pas.");
191            System.exit(-1);
192        }
193        return (degreEntrant(i)+degreSortant(i));
```

Graphe.java

```
195
196    public int [] lst_succ(int i){
197        if ((i<0) || (i >= U.length)){
198            System.out.println("Graphe::lst_succ : Erreur: hors de la matrice.");
199            System.exit(-1);
200        }
201        if (!existeSommet(i)){
202            System.out.println("Graphe::lst_succ : Erreur: ce sommet n'existe pas.");
203            System.exit(-1);
204        }
205        int[] tab = new int[nbSommet()];
206        int y=0;
207        while (y < nbSommet()){
208            for (int j=0; j<U[i].length;j++)//oubliï¿½ dans le code
209                if (U[i][j]!=ALPHA_NOTDEF){
210                    tab[y]=U[i][j];
211                    y++;
212                }
213        }
214        return tab; //oubliï¿½ dans le code
215    }
216
217    public String toString(){
218        int x;
219        int i=0;
220        int j=0;
221        x = nbSommet();
222        String chaine="";
```

Figure 37: le code source de la classe qui créer le graphe

7. Conclusion :

Dans ce chapitre, nous avons exposé quelques résultats donnés à l'implémentation de notre système, nous avons utilisé pour cela plusieurs outils tel que le CloudSim et l'Eclipse. Pour implémenter notre système, nous avons choisis la plateforme JADE. Cette plate-forme permet

de simplifier le développement des systèmes multi-agents tout en fournissant un ensemble complet de services et d'agents conformes aux spécifications FIPA. Les résultats obtenus montrent que l'utilisation de notre méthode permet d'améliorer la performance du système, elle minimise le pourcentage d'erreur et réduit le temps perdu causé par les pannes.

Conclusion générale

Le but de notre travail est la sélection des chemins minimaux nécessaires en la combinant avec la planification pour la tolérance aux fautes des services composés à partir d'un graphe donné. Ce dernier représente les différents plans, possibles, d'exécution pour un service composé demandé.

Pour valider notre travail, nous avons implémenté des Clouds et des services virtuels en utilisant le simulateur CloudSim, et on a intégré des agents à traves JADE (Java Agent Développement Framework) dans un environnement de développement Eclipse

Dans notre travail, on a représenté les services Cloud composants d'un service composé sous forme d'un graphe orienté tel que les nœuds sont de type Cloudlets. Pour sélectionner le chemin minimal, on a utilisé l'algorithme de Dijkstra. Ce dernier permet de trouver le plus court chemin à partir d'un nœud initial, cette phase est réalisée par l'agent planificateur (AP).

Ensuite, l'AP transfère le chemin minimal choisi à l'agent contrôleur AC qui est responsable de bon fonctionnement de ce dernier. En cours d'exécution, l'AP enregistre à chaque fois un service comme un point de reprise à condition que depuis ce service il existe au moins un autre chemin.

En cas de détection de faute par l'AC, il informe l'AP par l'envoie d'un rapport de faute qui contient à quel niveau l'erreur s'est-elle produite et à quel service. Ainsi l'AP va retourner au dernier point de reprise enregistré est choisir un autre sous plan minimal (qui a le coût minimal) par l'application de l'algorithme de Dijkstra (il prend le point de reprise comme paramètre), donc on va reprendre l'exécution depuis le dernier point de reprise enregistrer. Ainsi de suite jusqu'à la fin d'exécution de ce service composé. Enfin AP envoie le service requis au client.

Comme perspectives, on compte améliorer notre travail en l'appliquant sur des études de cas du monde réel plus compliqués. Ceci permettra de donner plus de réalité à nos résultats et, donc de mieux comparer notre proposition et nos résultats avec ce qui existe dans la littérature.

Bibliographie

[1] A. Aggoun et A. Mimouni, «proposition d'une méthode de recouvrement arriére avec planification pour la tolérance aux fautes des services Cloud composés,» 2018.

[2] C.-d. B, J. I et M. B, «Systèmes multiagents : Principes généraux et applications,» Département d'Informatique, Pavillon Pouliot, Université Laval, Ste-Foy, PQ, Canada, G1K 7P4, 2001.

[3] A. Grouls, «Agents et systémes multi-agents: vers une synthése de ces concepts,» Université du Québec, Montréal, 2013.

[4] M. L. Moussaoui , «Système Multi Agent pour planification de la production,» 2015/2016.

[5] B. Nachet, «modéle multi agents pour a conception de systémes d'aides à la décision collective,» Université d'Oran, Oran, 2013/2014.

[6] M. A. Yagoub, «Une approche basée agent pour la sécurité dans le Cloud Computing,» Université Mohamed Khider , Biskra, 2019.

[7] N. Brahmia et R. Boulahia, «Stratégie d'exploration multi agent pour les environnements inconnus,» 2013.

[8] I. Boussebough, «Les systémes multi-agents dynamiquement adaptables,» Université Mentouri, Constantine, 2011.

[9] A. Menif, «Planification d'actions hiérarchique pour la simulation,» École Doctorale de Dauphine, 2017.

[10] M. Berramdan, «Équilibrage de charge dans le cloud computing,» Université Mohamed Khider, Biskra, 2019.

[11] A. Mansouri et A. Tounsi, «Un mécanisme dynamique tolérant aux fautes des données dans le Cloud Computing,» 2017.

[12] L. Mezouer et N. Mekkioui, «Tolérances aux Pannes dans les infrastructures Cloud Computing,» 2016.

[13] M. Rebbah, «Tolérance aux fautes dans les grilles de calcul,» 2015.

[14] S. Zerdani, «Le Checkpointing Pour Les Bots Dans Le Cloud Computing,» 2017.

[15] B. Xavier, «Tolérance aux fautes et reconfiguration dynamique pour les applications distribuées à grande échelle,» Université de Grenoble, 2010.

[16] M. N. Ndeye, «Techniques de gestion des défaillances dans les grilles informatiques tolérantes aux fautes,» 2014.

[17] C. DELBÉ, «Tolérance aux pannes pour objets actifs asynchrones,» École Doctorale « Sciences et Technologies de l'Information et de la Communication », Nice, 2007.

[18] S. Limam, «Le Checkpointing Pour Les Bots Dans Le Cloud Computing avecCloudSIM,» 2012.

[19] M. Amoon, «Adaptive Framework for Reliable Cloud Computing Environment,» IEEE Access Journal, 2016.

[20] J. Gray, P. D et Siewiorek , «High-Availability Computer Systems,» IEEE Computer, 1991.

[21] X. Kong, J. Huang, C. Lin et P. D. Ungsunan, «Performance, Fault-tolerance and Scalability Analysis of Virtual Infrastructure Management System,» IEEE International Symposium on Parallel and Distributed Processing with Applications, Chengdu, China, 2009.

[22] R.Jhawar, V.Piuri et M. Santambrogio, «A Comprehensive Conceptual System-Level Approach to Fault Tolerance in Cloud Computing,» IEEE, 2012.

[23] A.Bala1 et Inderveer Chana2, «FaultTolerance- Challenges, Techniques and Implementation in Cloud Computing,» Computer Science and Engineering Department, Thapar University, Punjab, India, 2012.

[24] Sheng Di, Yves Robert, Frédéric Vivien et Cho-Li Wang, «Optimization of Cloud task processing with checkpoint-restart mechanism,» Proceedings of the International Conference on High Performance Computing, Networking, 2013.

[25] R. S et D. K.R, «Improving Fault Tolerance in Virtual Machine Based Cloud Infrastructure,» International Conference on Innovations in Engineering and Technology, 2014.

[26] M. A.D, S. A.S et Z. S.D, «Fault Tolerance Model for Reliable Cloud Computing,» International Journal of Recent and Innovation Trends in Computing and Communication, 2011.

[27] R. Santhosh et T. Ravichandran, «Non-Premptive Real Time Scheduling using checkpointing Algorithme for Cloud Computing,» Department of Computer Science and Engineering Faculty of Engineering, Karpagam University, 2013.

[28] F.-Y. Villemin, «Agent Communication Language,» 2012/2013.

[29] S. Lanani, «Une approche BPM (Business Process Managment) par composition d'application dans le cloud computing,» Université Mohamed Khidher, BISKRA.

[30] B. Fabio, P. Agostino et R. Giovanni, «JADE A FIBA compliant agent framework,» journal of PAAM, 1999.

[31] M. Huhns, «Agent web service,» IEEE internet computing , 2002.

[32] P. Agostino, R. Giovanni et T. Michele, «multi-user and security support Multi-agent Systems,» 2001.

[33] I. Boussebough, «les systemes multi-agents dynamiquement adaptables,» 2011.

BIBLIOGRAPHIE