REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE

LA RECHERCHE SCIENTIFIQUE

UNIVERSITE MOULOUD MAMMERI TIZI-OUZOU

FACULTE DE GENIE ELECTRIQUE ET INFORMATIQUE

DEPARTEMENT D'INFORMATIQUE

Memoire

DE FIN D'ETUDES

D'un MASTER ACADEMIQUE

Domaine : Mathématiques et Informatiques

Filière : Informatique

Spécialité : Réseaux mobilités et systèmes embarqués

Thème :

Ordonnancement de tâches dans le Cloud

Proposé et dirigé par :

M me OUKFIF Karima.

Réalisé par :

M elle AILAM Melisa.

M elle BAFDEL Melissa.

Membres de jury :

M r DAOUI Mehammed

M me KHIALI Lynda

Année universitaire 2018/2019

Remerciements

Nous tenons à remercier Dieu tout puissant qui nous a armés de courage, de volonté et surtout de patience.

Nous exprimons notre sincère gratitude à toutes les personnes qui ont rendu ce mémoire possible par leurs aides et leurs contributions.

Nos premiers remerciements sont adressés tout d'abord à notre promotrice Mme OUKFIF pour nous avoir proposé ce sujet et de l'avoir dirigé, ses conseils et ses encouragements nous ont été précieux.

Nous remercions aussi notre professeur Mr HAMMACHE qui nous a accueillis à tout moment.

Nous tenons également à remercier les membres de jury qui ont eu l'amabilité d'examiner et de juger notre mémoire.

Nous gardons une place toute particulière à nos familles, nous leurs exprimons
toute notre profonde reconnaissance pour leurs encouragements et leurs soutient.

Dédicaces

Nous dédions ce modeste travail à
Nos chers parents
Nos frères et sœurs
Nos amis et camarades

Melissa, Melisa

Table des matières

Table des matières

Table des matières

Table des matières

Liste des Figures

CHAPITRE I : GENERALITE SUR LE CLOUD.

Chapitre II : Ordonnancement de tâches et Optimisation.

Chapitre III : La métaheuristique PSO.

Liste des Figures

Chapitre IV : Réalisation et tests.

Liste des tableaux

Introduction générale

Le cloud computing ou informatique en nuage est une infrastructure dans laquelle la puissance de calcul et le stockage sont gérés par des serveurs distants auxquels les usages se connectent via une liaison internet sécurisée. L'ordinateur de bureau ou portable, le téléphone mobile, la tablette tactile et autres objets connectés deviennent des points d'accès pour exécuter des applications ou consulter des données qui sont hébergées sur les serveurs.

La théorie d'ordonnancement de tâches dans les systèmes de Cloud computing suscite une attention croissante avec l'augmentation de la popularité de Cloud. En général, l'ordonnancement de tâches est le processus d'affectation des tâches aux ressources disponibles sur la base des caractéristiques et des conditions des tâches.

Les problèmes d'ordonnancement de tâches dans un contexte hétérogène comme le Cloud sont des problèmes difficiles. Ces problèmes deviennent encore plus difficiles lorsque les critères à prendre en considération pour l'optimisation sont multiples.

Dans ce travail, nous proposons une étude sur l'ordonnancement de tâches dans le cloud computing. Notre étude se focalise sur l'utilisation de la métaheuristique multi-objectifs PSO (Partical Swarm Optimisation), pour résoudre le problème de l'ordonnancement, la suite de ce mémoire est organisée comme suit : le chapitre 1 présentera les concepts du cloud et les notions fondamentales, le chapitre 2 abordera le principe de l'ordonnancement et ses différents algorithmes classiques et les différentes métaheuristiques d'optimisation, au niveau du chapitre 3 nous allons détailler la métaheuristique PSO et son fonctionnement, pour en finir avec un chapitre 4 qui portera l'implémentation et la réalisation et l'ensemble des tests effectués.

Pour conclure on aura une dernière partie Annexe, qui résume l'ensemble des outils et langages que nous avons utilisé tout au long de notre travail.

Introduction

La technologie de l'Internet se développe de manière exponentielle depuis son apparition. Actuellement, une nouvelle tendance a marqué sa naissance dans le monde des IT (Technologies de l'information et de la communication), il s'agit du Cloud computing. Cette technologie, offre des occasions aux sociétés de réduire les coûts d'exploitation des logiciels par leurs utilisations directement en ligne. En effet le cloud est un moyen par lequel nous pouvons accéder aux applications sur internet, il nous permet de créer, configurer et personnaliser des applications en ligne.

1 Définition et concepts du cloud Computing

1.1 Définition du cloud Computing

Le cloud computing ou l'informatique en nuage est l'exploitation de la puissance de calcul ou de stockage des serveurs informatiques distants par l'intermédiaire d'un réseau, généralement internet. Ces serveurs sont loués à la demande, le plus souvent par tranche d'utilisation selon les critères techniques (puissance, bande passante, etc) mais également au forfait. Le cloud computing se caractérise par sa grande souplesse : selon le niveau de compétence de l'utilisation client, il est possible de gérer soi-même son serveur ou de se contenter d'utiliser des applicatifs distants. Selon la définition du National Institute of Standards and Technlogy(NIST),Le cloud computing est l'accès via un réseau de télé-commutations, à la demande et libre-service, à des ressources informatiques partagées configurables .il s'agit d'une délocalisation de l'infrastructure informatique. **[I.1]**

Figure I. 1 : Architecture cloud.

1.2 L'évolution du cloud Computing

L'ère de l'information a connu de nombreuses évolutions. Nous avons débuté avec les mainframes, et évolué vers les mini-ordinateurs, puis sont venus les ordinateurs personnels jusqu'à atteindre le cloud computing. **[I.2]**

La figure ci-dessous illustre les différentes technologies dans l'ère de l'information :

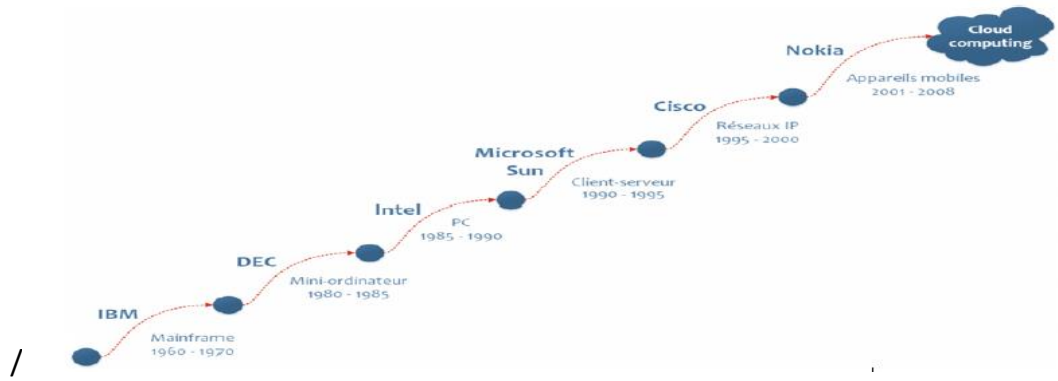


Figure I. 2 : Les différentes technologies dans l'ère de l'informatique.

Le cloud computing est une technologie qui n'est pas tout à fait récente comme on pourrait le penser ; ses traces remontent aux années soixante, à cette époque les utilisateurs accédaient depuis leurs terminaux à des applications fonctionnantes sur des systèmes centraux (mainframes) qui correspondaient aux serveurs du cloud. Cette idée continue d'évoluer dans l'ère du temps, elle prend naissance en 1990 et surtout en 1991 avec la naissance d'Internet et la mise sur le marché du logiciel CERN qui a été le premier logiciel accessible sur le Web. Cette idée continue de progresser avec la découverte d'EBay et d'Amazon en 1995. Le processus dévolution s'accélère ensuite, et en 2000, l'arrivée de Google fournit une réelle émergence du cloud computing.

1.3 Le modèle de déploiement du cloud Computing

Le cloud computing utilise un pool partagé de ressources informatiques (par exemple des réseaux, des serveurs, des espaces de stockage, des applications et des services) afin de fournir un accès réseau à la demande.

Comme le montre la **figure (I. 3),** le NIST (National Institute of Standards and Technology) a défini quatre types de modèles de déploiement du cloud **[I.3]**

- Public
- Privé
- Communautaire
- Hybride

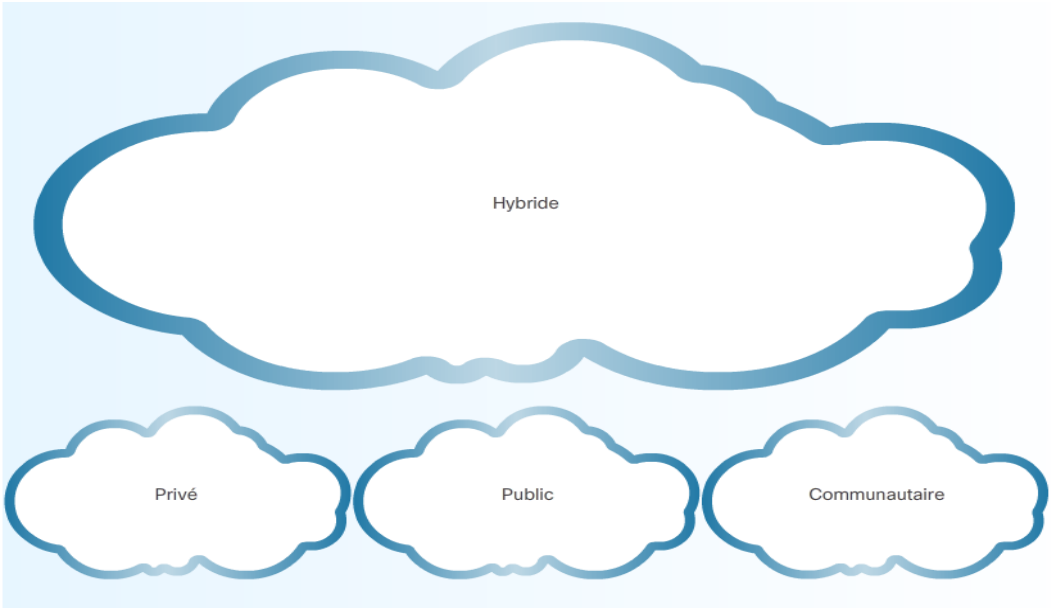


Figure I. 3: Le model de déploiement du cloud.

1.3.1 Cloud public

Destiné à être utilisé par le grand public. Son infrastructure est physiquement située sur le site du fournisseur, mais elle peut être détenue par une ou plusieurs organisations, comme des entreprises, des institutions universitaires ou des gouvernements. **[I.3]**

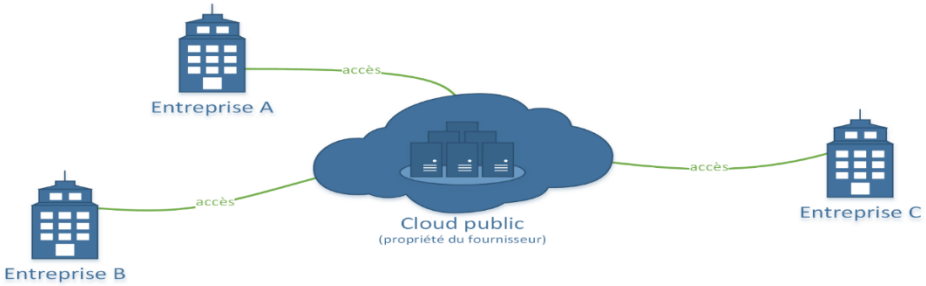


Figure I. 4: Modèle de déploiement d'un cloud public.

Dans la figure précédente on voit bien que l'entreprise A, l'entreprise B, et l'entreprise C ont un accès direct au cloud ; étant un cloud public toute entreprise ou tout individu a un accès direct. **[I.3]**

1.3.2 Cloud privé

Créé exclusivement pour une organisation unique. Son infrastructure peut être physiquement située sur le site ou en dehors de celui-ci, et elle peut appartenir à un fournisseur distinct. Un cloud privé n'offre de services qu'aux membres de cette seule organisation. **[I.3]**

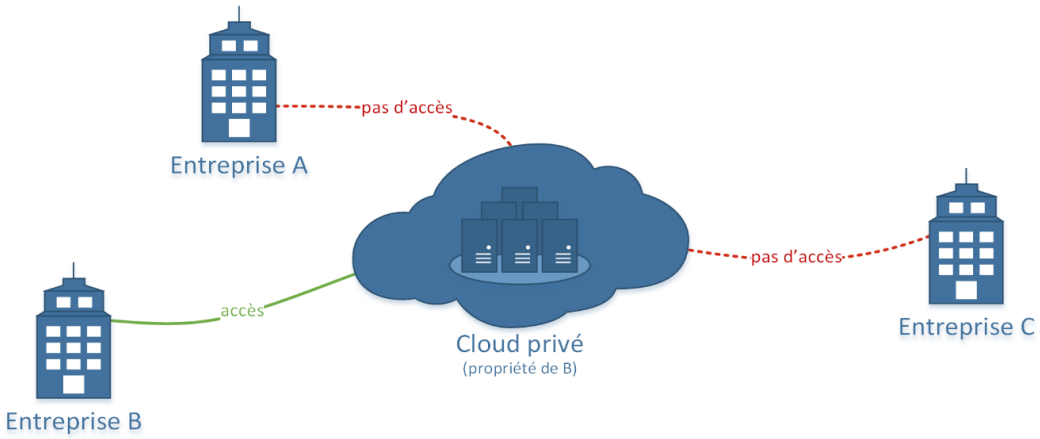


Figure I. 5: Modèle de déploiement d'un cloud privé.

La figure ci-dessus nous montre qu'uniquement l'entreprise B qui a un accès direct au cloud privé contrairement aux deux autres entreprises A et C, ce qui démontrent que ce cloud est propriétaire à l'entreprise B uniquement.

1.3.3 Cloud communautaire

Créé pour une utilisation exclusive par une communauté spécifique. La communauté se compose de plusieurs organisations partageant les mêmes préoccupations (par exemple en matière de mission, d'exigences de sécurité, de stratégie ou de critères de conformité). L'infrastructure peut être physiquement située sur le site ou en dehors de celui-ci, et elle peut appartenir à un fournisseur distinct ou à une ou plusieurs des organisations de la communauté. La différence entre le cloud public et le cloud communautaire se réfère aux besoins fonctionnels qui ont été personnalisés pour la communauté. **[I.3]**

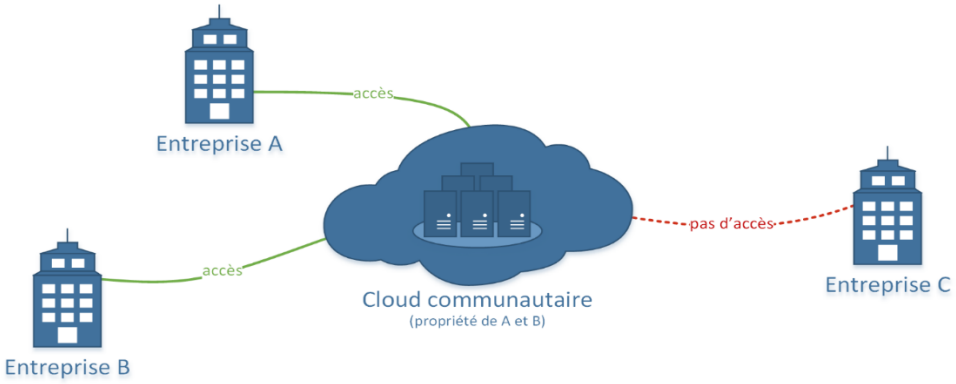


Figure I. 6: Modèle de déploiement d'un cloud communautaire.

Dans la figure ci-dessus on voit que seulement l'entreprise A et l'entreprise B qui ont un accès direct au cloud ce qui démontre que ces deux entreprises appartiennent à la même communauté qui partagent les mêmes intérêts et objectifs contrairement à l'entreprise C qui n'a aucun accès à ce type du cloud du fait qu'elle n'est pas de la même communauté.

1.3.4 Cloud hybride

C'est la combinaison d'au minimum deux infrastructures de cloud distinctes (cloud privé, communautaire ou public), représentant des entités uniques. Ces entités sont reliées par le biais d'une technologie permettant la portabilité des données et des applications. Cette portabilité permet à une organisation de conserver un point de vue unique en matière de solution du cloud, tout en profitant des avantages offerts par différents fournisseurs du cloud. Par exemple, la zone géographique (emplacement des utilisateurs finaux), la bande passante, les exigences en matière de stratégie ou de législation, la sécurité et le coût sont des caractéristiques susceptibles de différer d'un fournisseur à l'autre. Un cloud hybride offre une flexibilité permettant de s'adapter et de réagir aux services offerts par ces fournisseurs, et ce, à la demande. **[I.3]**

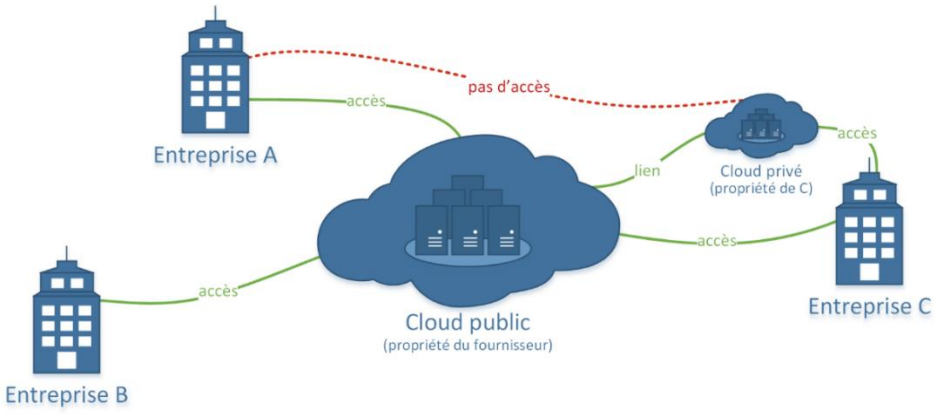


Figure I. 7: Modèle de déploiement d'un cloud hybride.

contrairement aux autres cloud on a dit que le cloud hybride est une combinaison d'au moins deux types du cloud et dans notre exemple on a le cloud hybride qui est composé d'un cloud public et un cloud privé comme le montre la figure ci-dessus donc dans ce cas toutes les entreprises A ,B, C auront un accès direct à la partie du cloud public par contre concernant la partie du cloud privé elle n'est accessible que par une seule entreprise qui est l'entreprise C , de plus dans le cas du cloud hybride on a la deuxième partie du cloud qui est dans notre cas est le cloud privé aura un accès direct au cloud public tout comme les différentes entreprises.

1.4 Les modèles de services du cloud Computing

Les modèles de services sont des modèles de références sur lesquels le Cloud Computing est basé. Ceux-ci peuvent être catégorisés en trois modèles de services de base **[I.4]**, comme indiqué ci-dessous :

- Infrastructure en tant que service (IaaS)
- Plate-forme en tant que service (PaaS)
- Logiciel en tant que service (SaaS)

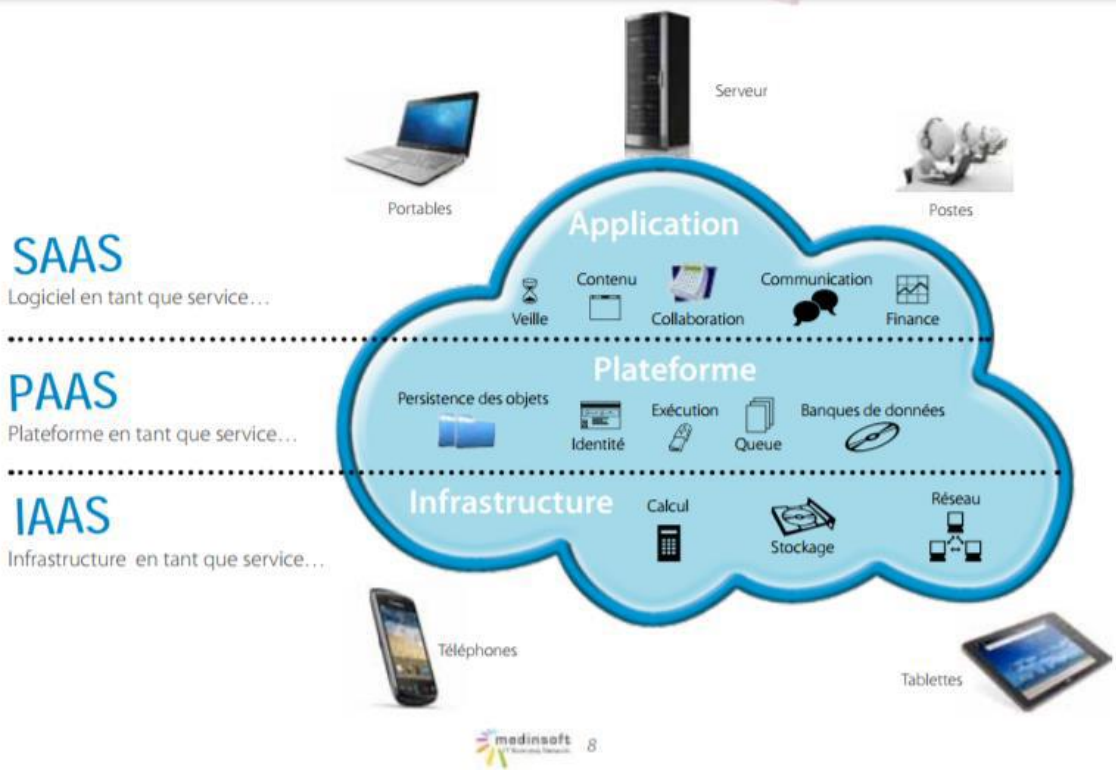


Figure I. 8: Les différentes couches des services du cloud computing.

1.4.1 IaaS (Infrastructure as a Services)

Les services IaaS du Cloud computing permettent de mettre à la disposition des utilisateurs des ressources matérielles (réseau, stockage, systèmes d'exploitation) accessibles sous format virtuelle. **[I.4]**

- **Avantage de (IaaS)**
  - Grande flexibilité.
  - Personnalisation

- **Inconvénient de (IaaS)**
  - Besoin d'administrateurs système comme pour les solutions de serveurs classiques sur site.
  - Besoin de sécurité.

Exemple : hébergement de serveurs virtuels

1.4.2 PaaS (Platform as a Service)

Les services du type PaaS disposent des environnements spécialisés au développement d'applications comprenant les outils et les modules nécessaires pour ce type de travail, il s'agit des environnements d'exécution qui permettent de gérer le cycle de vie des applications. Ce cycle de vie comprend notamment les phases de conception, de déploiement et plus généralement d'administration des applications. **[I.4]**

- **Avantage de (PaaS)**
  - Le déploiement est automatisé.
  - Pas de logiciel supplémentaire à acheter ou à installer.
- **Inconvénient de (PaaS)**
  - Limitation à une ou deux technologies (ex. Python ou Java).
  - Pas de contrôle des machines virtuelles.

Exemple : hébergement des sites web

1.4.3 SaaS (Software as a Service)

Les services SaaS du Cloud computing permettent de mettre à la disposition des utilisateurs des applications prêtes à l'emploi ; à la différence des applications Web ordinaires, les services SaaS du Cloud computing sont caractérisés par un haut niveau d'abstraction qui Permet d'adapter l'application à un cas particulier d'usage. **[I.4]**

- **Avantage de (Saas)**
  - Plus d'installation.
  - Plus de licence.
  - Migration de données.
  - Paiement à l'usage.

- **Inconvénient de (Saas)**
  - Limitation du logiciel proposé.
  - Pas de contrôle sur le stockage et la sécurisation des données associées au logiciel.

Exemple : messagerie, sauvegarde en ligne.

1.5 Caractéristiques du cloud Computing

Il existe cinq caractéristiques clés du cloud computing et qui sont affichées dans le schéma suivant : **[I.5]**

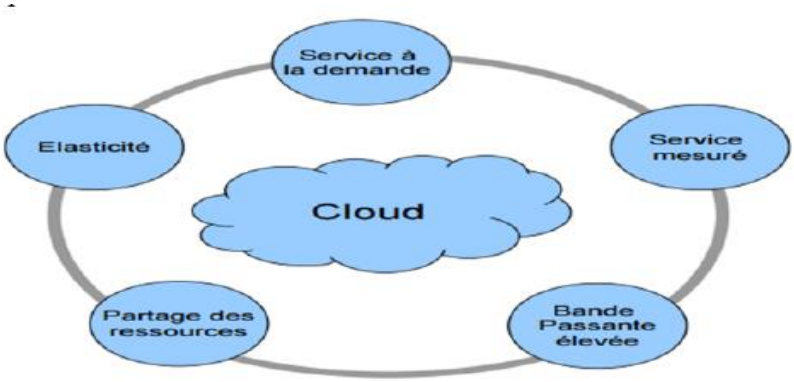


Figure I. 9 : Les cinq caractéristiques du cloud Computing.

1. **Service à la demande** : Capacité à fournir une ressource automatiquement, sans requérir d'interaction humaine coté fournisseur.
Le modèle du cloud computing permet aux utilisateurs d'utiliser les services Web et les ressources sur demande. On peut se connecter à un site Web à tout moment et les utiliser.

2. Mesurable : capacité de mesurer le service fourni. L'utilisation des ressources peut être surveillée, contrôlée et signalée, offrant la transparence pour le fournisseur et le consommateur du service utilisé. Cette caractéristique permet en particulier de payer le service à l'usage « pay as you go ».

3. **Bande passante élevée et accès réseau universel** : les ressources mises à la disposition des utilisateurs sont accessibles via Internet de n'importe quel endroit et en utilisant des plateformes hétérogènes (PC, téléphone, mobile, ordinateur portable, etc.) grâce à la capacité actuelle des réseaux.

4. Ressource partagées, mises en commun (pooling) : le Cloud Computing permet à plusieurs locataires de partager un pool de ressources. On peut partager une instance physique, de base de données et d'infrastructure de base.

5. Elasticité : possibilité de faire évoluer très rapidement la capacité fournie, que ce soit en plus ou en moins.

1.6 Principe du service du Cloud

- **Service-Level Agreement (SLA)**

SLA est un accord au niveau de service, c'est un contrat passé entre un fournisseur de service et ses clients internes ou externes. Ce contrat documente les services que le fournisseur met à la disposition de ses clients et les différents paramètres de chaque service, comme la disponibilité, le temps de réponse…etc.

- **Le pay as you go**

Les utilisateurs paient les ressources qu'ils utilisent en fonction de leur consommation réelle et précise, tous en suivant les modalités des services fournis.

2 Les technologies du cloud Computing

Certaines technologies fonctionnent derrière les plates-formes du cloud computing, rendant le cloud computing flexible, fiable et utilisable. **[I.6]**, Ces technologies sont énumérées ci-dessous :

- Virtualisation.
- Architecture orienté service (SOA).

2.1 Virtualisation

La virtualisation recouvre l'ensemble des techniques matérielles et ou logiciels qui permettent de faire fonctionner sur une seule machine plusieurs systèmes d'exploitation, plusieurs instances différentes et cloisonnées d'un même système ou plusieurs applications, séparément les uns des autres, comme s'ils fonctionnaient sur des machines physiques distinctes. Les intérêts de la virtualisation sont : l'utilisation optimale des ressources, l'économie sur le matériel, l'allocation dynamique de la puissance de calcul, la facilité d'installation, de déploiement et la migration des machines virtuelles. **[I. 7]**

2.1.1 Mécanisme de virtualisation

Un système d'exploitation principal appelé « système hôte » est installé sur un serveur physique unique. Ce système sert d'accueil à d'autres systèmes d'exploitation.

Un logiciel de virtualisation appelé hyperviseur (une plate-forme de virtualisation qui permet à plusieurs systèmes d'exploitation de travailler sur une même machine physique en même temps) est installé sur le système d'exploitation principal. Il permet la création d'environnements clos et indépendants sur lesquels seront installés d'autres systèmes d'exploitation « systèmes invités ». Ces environnements sont des « machines virtuelles ».

Un système invité est installé dans une machine virtuelle qui fonctionne indépendamment des autres systèmes invités dans d'autres machines virtuelles. Chaque machine virtuelle dispose d'un accès aux ressources du serveur physique (mémoire, espace disque...). **[I. 7]**

2.1.2 Avantages de la virtualisation

La virtualisation des systèmes informatiques présente de nombreux avantages citant : **[I. 7]**

- Déploiement rapide des applications.
- Niveaux de service supérieur et disponibilité accrue des applications.
- Évolutivité rapide et flexible.
- Réduction des coûts énergétiques, d'infrastructure et des installations.
- Diminution des frais de gestion.
- Accès aux applications et aux données des bureaux en tout lieu.
- Sécurité informatique renforcée.

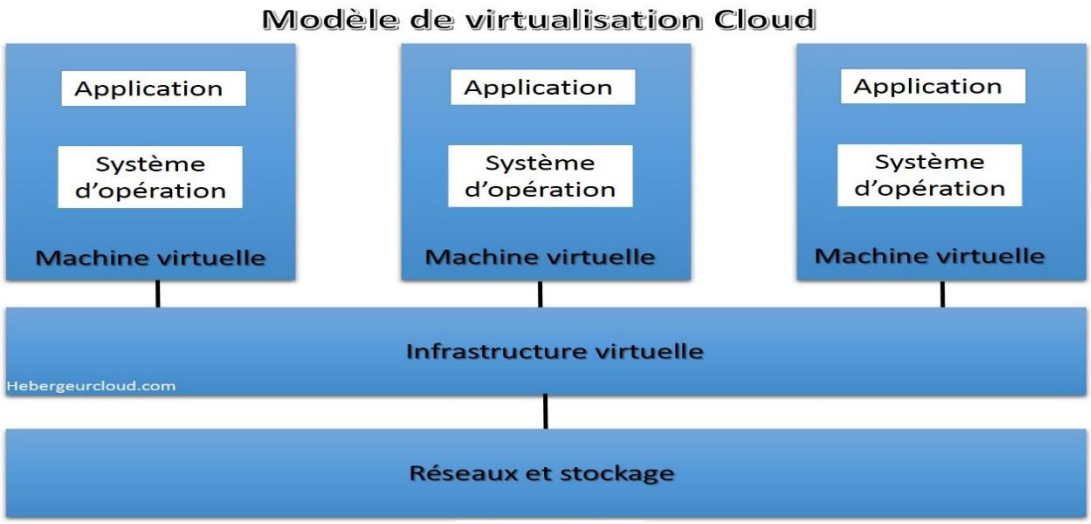


Figure I. 10: Model de nuage virtuel. [I. 6]

2.2 Architecture orienté services (SOA)

L'architecture orientée services permet d'utiliser les applications en tant que service pour d'autres applications, quel que soit le type de fournisseur, de produit ou de technologie. Par conséquent, il est possible d'échanger les données entre les applications de différents fournisseurs sans programmation supplémentaire ni modification des services. **[I.6].**

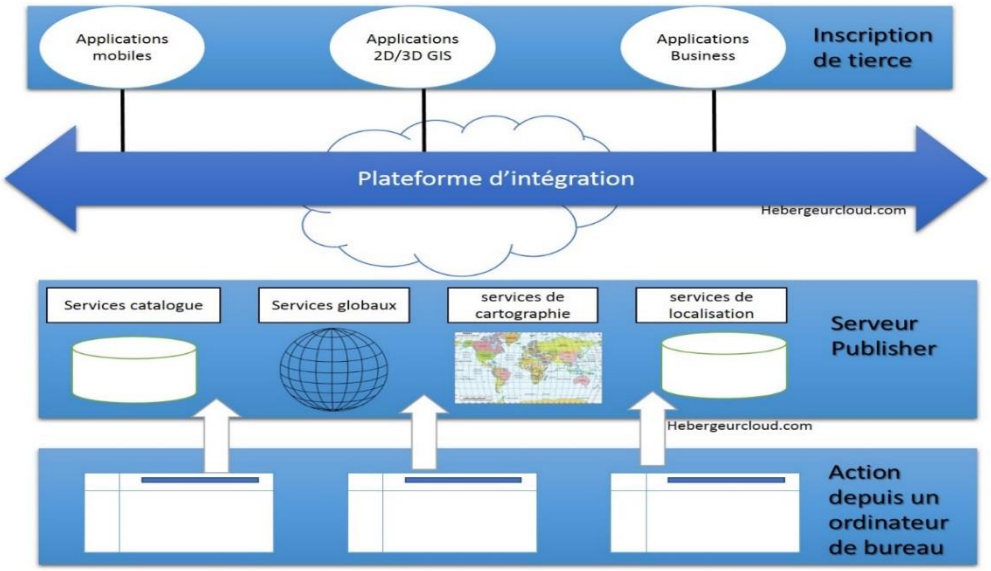


Figure I. 11 : Environnement SOA. [I. 6]

3 L'architecture d'une infrastructure cloud

L'architecture du cloud se réfère aux composants et sous-composants requis pour le cloud computing. Ils se compose généralement d'une plate-forme frontale ou extrémité avant (gros client, client léger, appareil mobile), d'une plate-forme d'extrémité arrière (serveurs, stockage), d'une livraison basée sur un cloud et d'un réseau (Internet, Intranet, Inter-cloud) ; Combinés, ces composants forment l'architecture du cloud computing. **[I.6]**

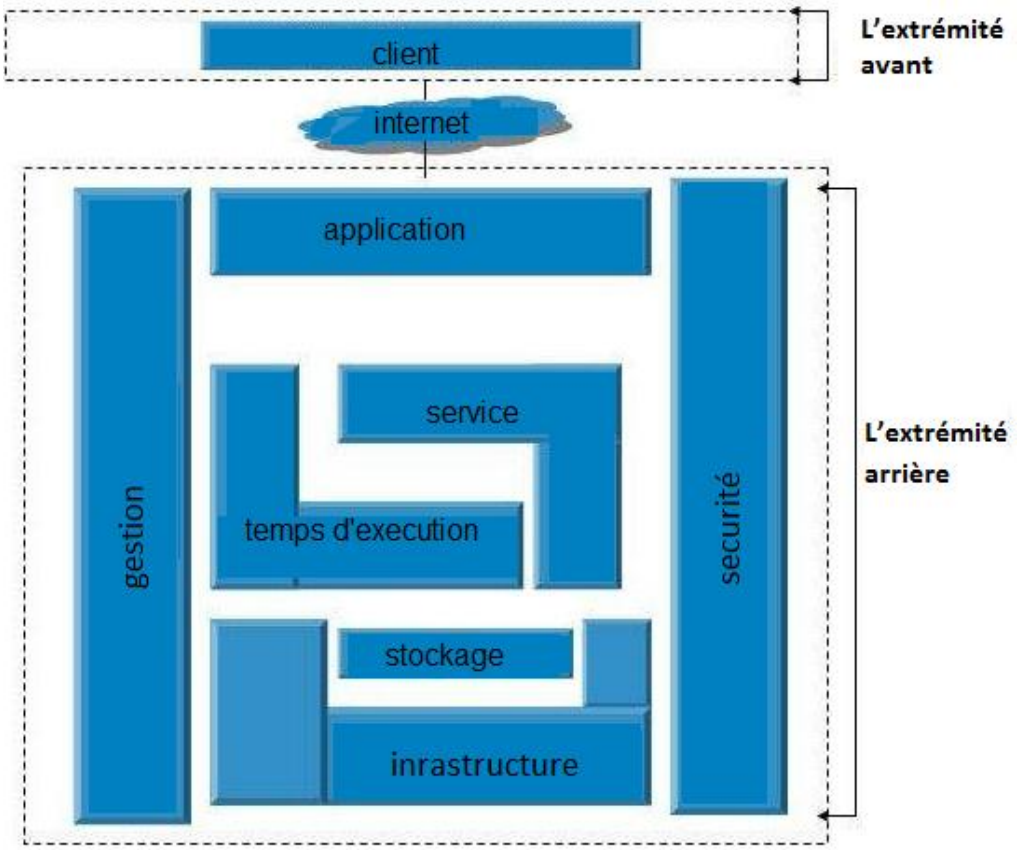


Figure I. 12: L'architecture d'une infrastructure cloud. [I.6]

3.1 L'extrémité avant

Le frontal ou l'extrémité avant désigne la partie cliente du système informatique en nuage. Il se compose d'interfaces et d'applications nécessaires pour accéder aux plates-formes du cloud computing, Exemple - Navigateur Web. **[I.6]**

3.2 L'extrémité arrière

L'extrémité arrière se réfère au nuage lui-même. Il comprend toutes les ressources nécessaires pour fournir les services de cloud computing. Il comprend un énorme stockage de données, des machines virtuelles, un mécanisme de sécurité, des services, des modèles de déploiement, des serveurs. etc. **[I.6]**

4 Avantages et risques du cloud computing

4.1 Avantages du cloud computing

Le Cloud Computing présente de nombreux avantages **[I.8] [I.9].** Certains d'entre eux sont énumérés ci-dessous :

- Service d'une grande disponibilité.
- Aucun investissement préalable.
- Facturation à la consommation.
- Accès simplifié utilisant des navigateurs web.

4.2 Risques du cloud computing

Les grandes catégories de risques liées à l'adoption du Cloud se déclinent selon les points suivants :

- **Sécurité et privacité :**

C'est la plus grande préoccupation concernant le cloud computing. Étant donné que la gestion des données et la gestion de l'infrastructure dans le cloud sont fournies par des tiers, il est toujours possible de transférer les informations sensibles à ces fournisseurs. Bien que les fournisseurs du cloud computing garantissent des comptes protégés par un mot de passe plus sécurisé, tout signe de violation de sécurité entraînerait la perte des clients et des entreprises.

- **Connexion**

C'est l'autre goulet d'étranglement. Si l'utilisateur n'a pas de connexion internet, ou une connexion insuffisante, il ne pourra pas accéder à sa plateforme de travail. L'idée dans ce cas

est de mettre le travail sur une application locale qui synchronise ensuite les données avec le serveur dès que l'utilisateur a à nouveau accès au réseau. Le problème de la sécurité des données en local se pose donc à nouveau.

- **Consommation d'électricité**

Le plus gros problème du cloud computing c'est la consommation élevée de l'électricité et l'énergie dépensé par les Datacenters.

- **Localisation des données**

La dématérialisation des données sur différents sites physique de stockage peut conduire à un éclatement des données et leurs répartitions sur différentes zones géographiques.

- **Budget élevé**

Les frais de transferts, peuvent s'avérer être importantes, selon l'utilisation que l'entreprise peut exploiter du Cloud, les besoins en bande passante peuvent faire exploser le budget.

Conclusion

Au niveau de ce chapitre nous avons clarifié le concept du cloud computing, en mettant en relief l'utilisation, les avantages et les inconvénients du Cloud computing ; A la base, il y a toujours et même de plus en plus de gros serveurs installés dans des data centers, qui vont abriter des logiciels puis stocker et diffuser des données, ce qui rend cette technologie indispensable aux grandes entreprises cela nous a conduits à déterminer notre besoin à voir la consommation d'énergie et coûts vis avis à l'utilisation du cloud computing.

Introduction

L'ordonnancement des tâches est souvent considéré comme un vrai challenge pour les gestionnaires dans les infrastructures distribuées, c'est ainsi que de nombreux travaux ont été consacrés à la recherche des solutions pour remédier à ces problèmes. Nous essayerons dans cette partie de présenter les différentes méthodes d'ordonnancement de taches dans l'infrastructure cloud.

1 Ordonnancement : concepts et définitions

1.1 Ordonnancement

Le problème d'ordonnancement consiste à organiser dans le temps l'exécution d'un ensemble de tâches, compte tenu des contraintes temporelle tel que le délai et d'autres contraintes portant sur l'utilisation et la disponibilité des ressources requises tout en satisfaisant un ou plusieurs objectifs (coût, qualité, délai, énergie, etc…)

Dans un problème d'ordonnancement plusieurs notions fondamentales interviennent. Parmi ces notions on trouve : les tâches, les ressources, les contraintes et les objectifs. **[II.1]**

- **Tâche**

Une tâche ou un job est une entité élémentaire localisée dans le temps par une date de début et une date de fin, et dont la réalisation nécessite une durée préalablement définie. Elle est constituée d'un ensemble d'opérations qui requiert pour son exécution certaines ressources et qu'il est nécessaires de programmer de façon à optimiser certains objectifs.

- **Ressource**

La ressource est un moyen technique ou humain destiné à être utilisé pour la réalisation d'une tâche et elle est disponible en quantité limitée.

- **Contraintes**

Les contraintes représentent les limitations imposées par l'environnement ou les ressources, ou les utilisateurs.

- Exemple : le budget, l'échéance.

- **Objectif**

Ce sont les critères à optimiser, c'est un ou plusieurs fonctions à minimiser ou à maximiser dans l'étude.

- Exemple : temps total d'exécution, cout monétaire, l'énergie consommée.

1.1.1 Type d'ordonnancement

Il existe deux types d'ordonnancement de tâches qui sont : **[II.2]**

1.1.1.1 Ordonnancement statique (hors-ligne)

Signifie que la séquence de l'ordonnancement est prédéterminée à l'avance, avant le début de l'exécution.

1.1.1.2 Ordonnancement dynamique

La séquence de l'ordonnancement est déterminée au préalable est mise à jour et réordonnancée en fonction des nouvelles tâches crées.

1.1.2 La résolution d'un problème d'ordonnancement

La résolution d'un problème d'ordonnancement consiste à déterminer :

- Le placement des tâches dans l'espace, c'est-à-dire sur les processeurs (la machines).
- Le placement des travaux dans le temps, c'est-à-dire les dates de début d'exécution de chaque tâche.
- La dépendance des tâches d'une application qui permet de définir le moment où une tâche peut être lancée sur une machine ; La dépendance de tâches a un impact crucial pour la conception des algorithmes d'ordonnancement, qu'on peut classer en deux catégories : **[II.2]**

A. Ordonnancement des tâches indépendantes

Dans ce type d'ordonnancement, toutes les informations sont connues à l'avance comme le temps d'exécution sur les différentes ressources. Dans ce cas, les tâches s'exécutent en parallèle, indépendamment les unes des autres

B. Ordonnancement des tâches dépendantes

Décrites par un DAG (Directed Acyclic Graph) qui se basent sur l'analyse des dépendances du graphe de tâches entier, afin de compléter les tâches interdépendantes au plus tôt.

1.2 L'ordonnancement dans une infrastructure cloud

L'ordonnancement dans le Cloud Computing se fait au niveau de l'utilisateur et au niveau du système.

Au niveau de l'utilisateur, la planification traite les problèmes soulevés par la prestation de services entre les fournisseurs et les clients.

Au niveau système c'est la gestion des ressources dans les centres de données (data centers) ; Le centre de donnée se compose de plusieurs machines physiques. Des millions de tâches des utilisateurs sont reçues ; l'attribution de ces tâches aux machines physiques se fait au niveau des centres de données. Cette affectation d'ordonnancement joue un rôle significatif sur les performances du centre de donnée. **[II.3]**

1.2.1 Le rôle de l'ordonnanceur

Un ordonnanceur (Scheduler) devra trouver la machine la plus appropriée pour traiter les tâches qui lui sont soumises. Les ordonnanceurs peuvent aller du plus simple (allocation de type round-robin) au plus compliqué (ordonnancement à base de priorités).

Les ordonnanceurs déterminent le taux de charge de chaque ressource afin de bien ordonnancer les prochaines tâches. Ils peuvent être organisés en hiérarchie avec certaines, interagissant directement avec les ressources, et d'autres (méta ordonnanceurs) interagissant avec les ordonnanceurs intermédiaires. Ils peuvent aussi superviser le déroulement d'une tâche jusqu'à sa terminaison, la soumettre à nouveau si elle est brusquement interrompue et la terminer prématurément si elle se trouve dans une boucle infinie d'exécution. **[II.4]**

Le problème d'ordonnancement est un problème majeur dans la littérature qui a poussé les auteurs à proposer différents algorithmes.

1.3 Les principaux algorithmes classiques d'ordonnancement dans la littérature

Nous présentons dans ce qui suit, les principaux algorithmes d'ordonnancement cités dans la littérature.

- ✓ **Algorithme Min-min**

L'algorithme commence par calculer le temps d'exécution minimale pour toutes les tâches puis la valeur minimale entre ces temps minimum est choisie ; elle représente le temps minimum d'exécution parmi toutes les tâches sur les ressources. Ensuite, en fonction de ce temps minimum, la tâche est ordonnancée sur la machine correspondante. Puis le temps d'exécution pour toutes les autres tâches sont mises à jour sur cette machine en ajoutant le temps d'exécution de la tâche assignée à des temps d'exécution des autres tâches sur cette machine/ressource et la tâche assignée est supprimée de la liste des tâches. Ensuite, la même procédure est répétée jusqu'à ce que toutes les tâches soient assignées sur les ressources. **[II.8]**

- ✓ **Algorithme Max-min**

L'algorithme Max-min suit le même principe que l'algorithme Min-min à l'exception des propriétés suivantes : Après avoir calculé les temps d'exécution minimum, la valeur maximale est sélectionnée, qui est la durée maximale parmi toutes les tâches sur les ressources. Ensuite, en fonction de ce temps maximum, la tâche est ordonnancée sur la machine correspondante. Puis le temps d'exécution pour toutes les autres tâches sont mises à jour sur cette machine en ajoutant le temps d'exécution de la tâche assignée à des temps d'exécution des autres tâches sur la machine qui a acquis la tâche sélectionnée et la tâche assignée est supprimée de la liste des tâches. La même procédure est répétée jusqu'à ce que toutes les tâches soient assignées sur les ressources. **[II.8]**

- ✓ **Round robin (RR)**

L'algorithme de scheduling du tourniquet, appelé aussi **Round Robin**, a été conçu pour des systèmes à temps partagé. Il alloue la ressource aux processus à tour de rôle, pendant une tranche de temps appelée **quantum**. **[II.10]**

- ✓ **First come first served (FCFS)**

L'algorithme FCFS est tout simplement la politique FIFO —First in First out, c'est l'un des algorithmes les plus simples, la stratégie c'est d'ajouter chaque tâche et ressource dans une file d'attente et d'exécuter chaque tâche et ressource par ordre d'arrivée. **[II.8]**

- ✓ **Earliest Deadline First scheduling (EDF)**

Dans le même ordre d'idée, on peut aussi choisir d'exécuter en premier la tâche qui nécessite d'être fini le plus rapidement. Cet algorithme est utilisé pour les systèmes temps réel. C'est un ordonnancement préemptif avec priorité dynamique : la tâche la plus prioritaire est celle dont la date de fin est la plus proche, c'est à dire que plus le travail doit être réalisé rapidement, plus elle est prioritaire. Cependant, il est assez complexe à le mettre en œuvre et il se comporte mal en cas de surcharge du système, c'est la raison pour laquelle il est peu utilisé. **[II.8]**

- ✓ **Heterogeneous-Earliest-Finish-Time (HEFT)**

Il est parmi les algorithmes d'ordonnancements de liste les plus répondus. Il détermine un ordonnancement totalement statique d'un DAG sur un environnement hétérogène de manière à minimiser la durée totale d'exécution de l'application. Il est caractérisé par :

- **EFT (Earliest Finish Time)** : qui est une fonction qui cherche la ressource qui exécute la tâche le plus tôt possible.

Lors de l'ordonnancement, la liste des tâches prêtes est triée en fonction du rang des tâches ; plus une tâche à un rang important, plus elle est prioritaire. L'objectif de cet ordre est d'exécuter les tâches les plus éloignées du puits du DAG en premier pour éviter d'importants temps d'inactivité en fin d'exécution c.à.d. les ressources en attente d'une nouvelle tâche. **[II.9]**

2 Etat de l'art des métaheuristiques d'optimisation

2.1 Problème d'optimisation

2.1.1 Définition

Le problème d'optimisation se définit comme la recherche du minimum ou du maximum (de l'optimum) d'une fonction donnée. La définition du problème d'optimisation est souvent complétée par

la donnée de contrainte, tous les paramètres (ou variables de décision) proposés doivent respecter ces contraintes. **[II.5]**

2.1.2 Approche d'optimisation

On distingue deux approches d'optimisation de problème qui sont l'optimisation mono-objective et l'optimisation multi objective. **[II.5]**

2.1.2.1 Optimisation mono-objective

L'optimisation mono-objectif considère l'optimisation d'un seul objectif ou l'optimisation individuelle de plus d'une fonction objective.

2.1.2.2 Optimisation multi-objective

Est l'optimisation qui traite le cas de la présence simultanée de plusieurs objectifs, cette optimisation autorise un certain degré de liberté qui manquaient en optimisation mono-objectif, la recherche nous ne donnera plus une solution unique mais plutôt une multitude de solution. Ces solutions sont appelées solutions de Pareto, et l'ensemble de solution que l'on obtient à la fin de la recherche est la surface de compromis.

2.1.3 Caractéristiques d'une optimisation

2.1.3.1 Fonction objective

La fonction objective est une fonction **f** définie dans un espace de recherche **S**, le but est de trouver une solution **s* ϵ S** de meilleure qualité **f(s)**, elle sert de critère pour déterminer la meilleure solution au problème d'optimisation.
L'équation suivante résume la définition : **[II.5]**

$$s^* = (\min/\max f(s) \backslash s \in S).$$

2.1.3.2 Variables de décision

Les variables sont regroupées dans un vecteur **X [1...n]** (n est le nombre de variables) ; et c'est en faisant varier ce vecteur que l'on recherche un optimum de la fonction objective **f** qui sera soit un minimum ou un maximum (selon l'étude effectuée). **[II.5]**

2.1.4 Classification des problèmes d'optimisation

On peut classer les différents problèmes d'optimisation que l'on rencontre dans la vie courante en fonction de leurs caractéristiques comme s'est illustré dans la figure suivante :

Critères		Classification
Nombre de variables de décision	Une variable de décision	Mono-variable
	Plusieurs variables de décision	Multi-variable
Type de variables de décisions	Nombre réel continu	Continu
	Nombre entier	Entier ou discret
	Permutation sur un ensemble fini de nombre	Combinatoire
Type de la fonction objectif	Fonction linéaire des variables de décisions	Linéaire
	Fonctions quadratiques des variables de décision	Quadratique
	Fonction non-linéaire des variables de décision	Non-linéaire
Formulation du problème	Avec contrainte(s)	Contraint
	Sans contrainte(s)	Non contraint

Figure II. 1: Classification des problèmes d'optimisation. [II.5]

2.1.5 Caractéristique d'une optimisation multi objective

2.1.5.1 La multiplicité des solutions

Lorsque on cherche à obtenir une solution optimale à un problème d'optimisation multi objectif donné, on s'attend souvent à trouver une multitude de solutions, du fait que certains des objectifs sont contradictoires et ces solutions, comme on peut s'en douter ne seront pas optimales au sens où elles

ne minimiseront pas tous les objectifs du problème. **[II.5]**

Un concept intéressant qui nous permettra de définir les solutions obtenues est le compromis, en effet les solutions que l'on obtient lorsque on a résolu le problème sont des solutions de compromis, elles minimisent un certain nombre d'objectifs tout en dégradant les performances sur d'autres objectifs.

La notion de compromis est effectuée selon trois méthodes d'optimisation qui sont définies en fonction de l'instant où l'on prend la décision d'effectuer ou non un compromis entre les fonctions objectives, ces méthodes sont les suivantes : **[II.5]**

A. Les méthodes d'optimisation à priori (décideur →recherche)

Dans ces méthodes, le compromis que l'on désire effectuer entre les différentes fonctions objectives a été déterminer avant l'exécution de la méthode d'optimisation.

Pour obtenir la solution à notre problème, il suffira de faire une et une seule recherche, et en fin d'optimisation, la solution obtenue reflétera le compromis que l'on désirait effectuer entre les fonctions objectives avant de lancer la recherche.

Cette méthode est intéressante dans le sens où il suffit d'une seule recherche pour trouver la solution, cependant, il ne faut pas oublier le travail de modélisation du compromis qui a été effectué avant d'obtenir ce résultat, aussi Il ne faut pas oublier non plus que le décideur est humain et qu'il sera susceptible de constater que la solution obtenue en fin d'optimisation ne le satisfait pas finalement et qu'il souhaite maintenant après avoir examiné cette solution, une solution qui opère un autre compromis entre les fonctions objectives.

B. Les méthodes à postériori : (recherche → décideur)

Dans ces méthodes, on va chercher un ensemble de solutions bien réparties dans l'espace de solutions ; le but sera ensuite de proposer ces solutions au décideur pour qu'il puisse sélectionner la solution qui le satisfait le plus en jugeant les différentes solutions proposées.

Ici, il n'est plus nécessaire de modéliser les préférences du décideur. On se contente de produire un ensemble de solutions que l'on transmettra au décideur. Il y'a donc un gain de temps non négligeable vis-à-vis de la phase de modélisation des préférences de la famille des méthodes à priori.

C. Les méthodes d'optimisation progressives : (décideur ↔ recherche)

Dans ces méthodes, on cherchera à questionner le décideur au cours de l'optimisation afin que celui-ci puisse réorienter la recherche vers des zones susceptibles de contenir des solutions qui satisfassent les compromis qu'il souhaite opérer entre les fonctions objectives.

Ces méthodes, bien qu'appliquent une technique originale pour modéliser les préférences du décideur, elles présentent l'inconvénient de monopoliser l'attention du décideur tout au long de l'optimisation.

Cet inconvénient n'est pas majeur lorsque l'on a affaire à des problèmes d'optimisation pour lesquels la durée d'une évaluation n'est pas importante mais pour les autres problèmes, l'utilisation d'une telle méthode peut être délicate car il est en effet difficile de monopoliser l'attention du décideur pendant une période qui peut s'étendre sur plusieurs heures en lui posant de plus en plus des questions toutes les dix minutes, voire même toutes les heures.

2.1.5.2 Approche de Pareto

Lorsqu'on résout un problème d'optimisation multi objectif, nous obtiendrons une multitude de solutions. Seul un nombre restreint de ces solutions va nous intéresser.

Pour qu'une solution soit intéressante, il faut qu'il existe une relation de **dominance** entre la solution considérée et les autres solutions. **[II.5]**

2.1.5.2.1 La dominance au sens de Pareto

Les approches Pareto utilisent la notion de dominance pour comparer les solutions et leurs affecter un score, ou sélectionner des solutions. Ces approches ont connu un important développement en conjonction avec les algorithmes évolutionnaires à population à partir de la seconde moitié des années 90 ; elles sont devenues la principale approche employée pour résoudre les problèmes multi objectifs du fait de leur capacité à trouver un ensemble potentiellement efficace via la recherche menée sur une population de solutions. **[II.5]**

La dominance Pareto ou l'optimalité de Pareto est définie comme suit : **[II.5]**

- **Une solution x domine (≤) au sens de Pareto une solution z :**

Si et seulement si : ∀ i ϵ {1…n} tel que fi(x) ≤ fi(z) et il existe i ϵ {1...n} tel que fi(x) < fi(z).

- **Une solution x domine fortement une solution z :**

Si et seulement si : ∀ i ϵ {1...n} tel que fi(x) < fi(z).

- **Les deux solutions sont incompatibles si et seulement**

Si et seulement si : ∀ i ϵ {1...n} tel que ni fi(x) < fi(z), ni fi(x)> fi(z).

La représentation de la dominance Pareto (Pareto optimal) dans l'espace objectif est appelée **front Pareto**.

Exemple :

Etant donné un problème multi-objectif (f, X) et son ensemble Pareto optimal X_e, le front Pareto est défini comme suit Z_N= {f(x)|x ∈ X_e comme c'est illustré sur la figure ci-dessous

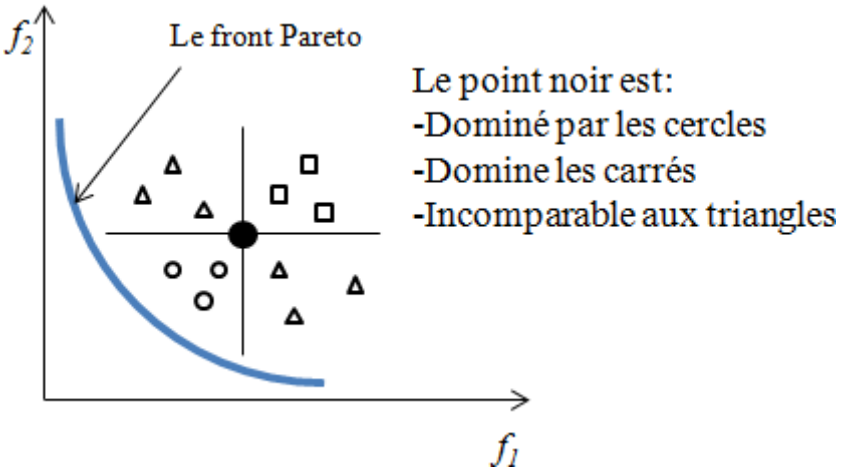


Figure II. 2: Relation de dominance. [II.5]

2.2 Les métaheuristiques

2.2.1 Définition

Les métaheuristiques sont une famille d'algorithmes stochastiques destinées à la résolution des problèmes d'optimisation. Leurs particularités résident dans le fait que celle-ci sont adaptables à un grand nombre de problèmes sans changement majeur dans leurs algorithmes, d'où le qualificatif "méta"; elles sont en général représentées sous la forme de concepts, elles reprennent des idées que l'on retrouve parfois dans la vie courante. **[II.6]**

2.2.2 Exemples de métaheuristiques

Nous présentons, dans cette section, quelques métaheuristiques qui sont largement utilisées :

2.2.2.1 Les algorithmes évolutionnaires

Les algorithmes évolutionnaires sont une famille d'algorithmes dont le principe s'inspire de la théorie de l'évolution naturelle de Charles Darwin, pour résoudre les problèmes divers. **[II.7]**

- ✓ **Les algorithmes génétiques**

Les algorithmes génétiques sont des algorithmes d'optimisation, cet algorithme reflète le processus de sélection naturelle où les individus les plus aptes sont sélectionnés pour la reproduction afin de produire la progéniture de la génération suivante.

L'algorithme génétique est constitué de Cinq phases importantes : **[II.7]**

A. Population initiale

Le processus commence avec un ensemble d'individus (solutions potentielles) qui s'appelle une population. Chaque individu est une solution au problème que vous voulez résoudre.

Un individu est caractérisé par un ensemble de paramètres (variables) appelés gènes. Les gènes

sont réunis en une chaîne pour former un chromosome (solution).

B. Fonction Fitness

La fonction de ‘’conditionnement physique’’ détermine le niveau de forme physique d'un individu (la capacité d'un individu à rivaliser avec d'autres individus). Il donne un score à chaque individu. La probabilité qu'une personne soit sélectionnée pour la reproduction est basée sur son score final. En informatique la fonction fitness ou bien la fonction objective représente la fonction d’évaluation des résultats obtenus pour un problème donné en prenant en considération un certain nombre d’objectif et elle est utilisée dans le but d’une optimisation pour aboutir à la solution la plus optimal.

C. Sélection

L'idée de la phase de sélection est de sélectionner les individus les plus aptes et de les laisser transmettre leurs gènes à la génération suivante.

D. Le croissement

Le croisement est la phase la plus importante d'un algorithme génétique. C’est le résultat obtenu lorsque deux chromosomes partagent leurs particularités.

E. Mutation

Les opérateurs de mutation produisent un enfant à partir d’un unique individu. En altérant un gène dans un individu selon un facteur de mutation, ce facteur est la probabilité qu’une mutation soit effectué sur un individu.

L'algorithme génétique est résumé par le pseudo-code suivant :

```
Début
    T=0
    Initialiser P(t)
    Evaluer la population P(t)
    Tant que condition de terminaison non satisfaite faire
    Début
        T = t+1
        Sélectionner P(t) à partir P(t-1)
        Mutations
        Croisements
        Evaluations
    Fin
Fin
```

Figure II. 3: pseudo-code de l'algorithme génétique.

2.2.2.2 L'intelligence du groupe (Swarm intelligence) :

2.2.2.2.1 Les algorithmes de colonies de fourmis

Les algorithmes de colonies de fourmis sont nés d'une constatation simple : les insectes sociaux et en particulier les fourmis résolvent naturellement des problèmes complexes. Un tel comportement est possible car les fourmis communiquent entre elles de manière indirecte par le dépôt de substances chimiques, appelées phéromones, sur le sol. Ce type de communication indirecte est appelée stigmergie. La principale illustration de ce constat est donnée par la figure suivante. **[II.6]**

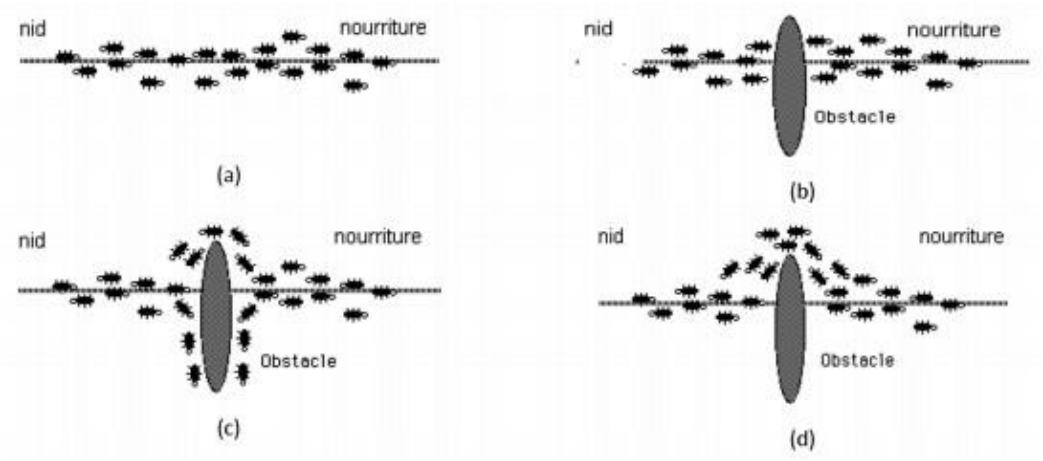


Figure II. 4: parcours des fourmis pour la recherche de nourriture.

On voit sur cette figure que si un obstacle est introduit sur le chemin des fourmis, celles-ci vont, après une phase de recherche, avoir tendance à toutes emprunter le plus court chemin entre le nid et l'obstacle. Plus le taux de phéromone à un endroit donné est important, plus une fourmi va avoir tendance à être attirée par cette zone. Les fourmis qui sont arrivées le plus rapidement au nid en passant par la source de nourriture sont celles qui ont emprunté la branche la plus courte du trajet. Il en découle donc que la quantité de phéromone sur ce trajet est plus importante que sur le trajet le plus long. De ce fait, le plus court chemin a une probabilité plus grande d'être emprunté par les fourmis que les autres chemins et sera donc, à terme, emprunté par toutes les fourmis.

2.2.2.2.2 Optimisation par Essaim de Particules

La méthode d'optimisation approchée PSO (partical swarm optimization) est basée sur les « Interactions sociales » entres des « agents » appelés « particules », Le but est d'atteindre un objectif donné dans un espace de recherche commun où chaque particule a une certaine capacité de mémorisation et de traitement de l'information, et permet de trouver l'optimum d'une fonction en un temps de traitement raisonnable. Cette métaheuristique sera l'objet du prochain chapitre. **[II.6]**

Conclusion

Dans ce chapitre, nous avons vu une définition du concept d'ordonnancement des tâches, élaborer les problèmes d'ordonnancement dans le cloud à base d'une optimisation multi-objective, et nous avons cité les variations d'algorithmes dédiés à sa résolution, dans le prochain chapitre nous allons détailler l'une des métaheuristique souvent utilisée qui est l'algorithme PSO (Partical Swarm Optimization).

Introduction

Parmi les métaheuristiques utilisées pour résoudre les problèmes d'ordonnancement de tâches nous avons cité PSO (Particle Swarm Optimisation) que nous avons choisi d'étudier et de tester dans notre travail.

Ce présent chapitre va être consacré justement à la présentation de cet algorithme avec ses différentes améliorations et les développements qui en découlent.

1 Présentation de l'algorithme PSO

1.1 Définition

L'Optimisation par Essaim de Particules (**OEP**), connu sous le nom anglophone de Particle Swarm Optimization **(PSO)**, est un algorithme qui utilise une population de solutions candidates pour développer une solution optimale au problème. **[III.1]**

Cet algorithme a été proposé par Russel Eberhart (ingénieur en électricité) et James Kennedy (socio-psychologue) en 1995. **[III.2]**

Il s'inspire à l'origine du monde du vivant, plus précisément du comportement social des animaux évoluant en essaim, tels que les bancs de poissons et les vols groupés d'oiseaux.

Autrement-dit, il s'inspire fortement de l'observation des relations grégaires d'oiseaux migrateurs, qui pour parcourir de « longues distances » (migration, quête de nourriture, parades aériennes, etc.), doivent optimiser leurs déplacements en termes d'énergie dépensée, de temps, (etc.).

Figure III.1: Essaim de particule.

L'essaim de particules correspond à une population d'agents simples, appelée particules, chaque particule est considérée comme une solution au problème, elle possède une position (le vecteur solution) et une vitesse.

Le déplacement en essaim est complexe, sa dynamique obéit à des règles et des facteurs bien spécifiques qu'il s'agit de cerner : **[III.1]**

- ✓ Chaque individu dispose d'une certaine intelligence « limitée » (qui lui permet de prendre une décision).
- ✓ Chaque individu doit connaitre sa position locale et disposer d'information locale de chaque individu se trouvant dans son voisinage.
- ✓ Obéir à ces trois règles simples : « rester proche des autres individus », « aller dans une même direction » ou « voler à la même vitesse ».

Tous ses facteurs et règles sont indispensables pour le maintien de la cohésion dans l'essaim, ceci par l'adoption d'un comportement collectif complexe et adaptatif. De plus, chaque particule possède une mémoire lui permettant de se souvenir de sa meilleure performance (en position et en valeur) et de la meilleure performance atteinte par les particules « voisines » (informatrices) : chaque particule dispose en effet d'un groupe d'informatrices, historiquement appelé son voisinage.

1.2 Mode de fonctionnement de l'algorithme PSO [III.1] [III.2]

L'algorithme PSO peut être aussi décrit en se plaçant du point de vue d'une Particule. Au départ de l'algorithme, un essaim est réparti au hasard dans l'espace de recherche. Le déplacement de toute particule est régi par des règles et conditions bien spécifiques, influencé par le mouvement des autres particules du voisinage.

Dans un tel contexte ce déplacement à une signification et doit parallèlement répondre à une logique, fondement même du PSO. Il consiste à chercher un optimum dans un voisinage donné, ce déplacement est influé par les trois composantes suivantes : **[III.3]**

- Une composante d'inertie :la particule s'efforce de suivre instinctivement son cap de déplacement.

- Une composante cognitive :la particule fait tout pour se diriger vers la meilleure position rencontrée jusqu'à présent.

- Une composante sociale : la particule s'inspire également de l'expérience, du parcours des autres particules, pour se diriger vers la meilleure position rencontrée par ses voisins.

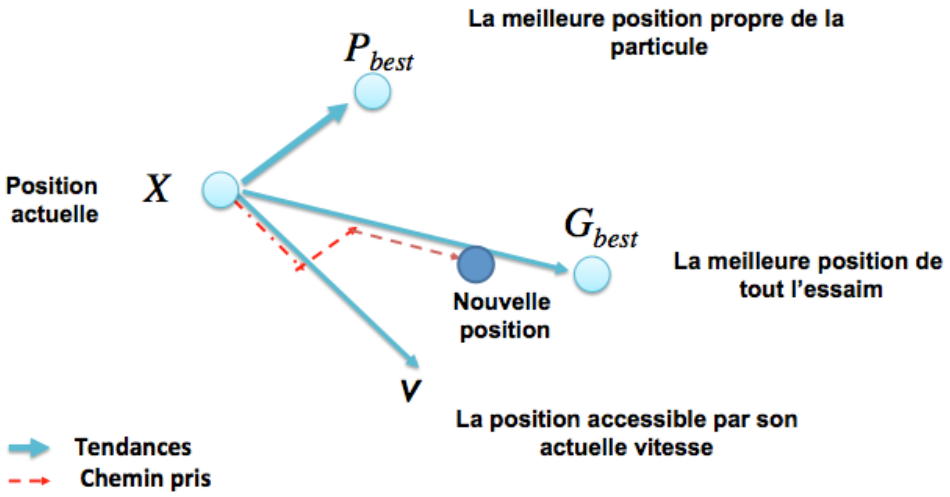


Figure III. 2: Déplacement d'une particule. [III.3]

La figure suivante présente un schéma de principe résumant les explications ci-dessus.

- I : est l'optimum de l'individu.
- P : est l'optimum de la population.
- + : est l'optimum absolu.

Figure III. 3: Règles simples d'application locale dans un essaim de particules. [III.3]

A terme, on se rend compte que toutes les particules, après un certain nombre d'itérations, convergent vers une même position, somme toute la meilleure rencontrée par l'ensemble des particules. Cela ne signifie pas pour autant, dans l'absolu, que c'est la meilleure position du voisinage, juste que c'est la meilleure rencontrée.

2 Composante de la PSO

Pour être en mesure d'utiliser le PSO, il est indispensable de définir un espace de recherche (composé de particules) et une fonction "objectif" à optimiser. La méthode de l'algorithme consiste alors à déplacer ces particules de telle sorte qu'elles trouvent l'optimum (comme expliqué précédemment), elles doivent disposer : **[III.4]**

- De données relatives à leurs positions, connaitre leurs coordonnées avec comme condition qu'elles soient comprises dans l'espace de définition.
- De la meilleure position qu'elles ont rencontrée.

- De la meilleure position rencontrée par leur voisinage et le résultat de leur fonction « objective ».
- De leur vitesse qui leur permet de se déplacer et de changer de position au fil des itérations.
- D'un voisinage, c'est le sous-ensemble de particules qui interagit directement avec la particule (surtout celle possédant la meilleure position).

D'après Maurice Clerc et Patrick Siarry**,** l'évolution d'une particule n'est finalement qu'une fusion de trois types de comportements :

- Egoïste : se déplacer suivant sa vitesse actuelle.
- Conservateur : revenir en arrière en prenant en compte sa meilleure performance.
- Panurgien : suivre aveuglement le meilleur de tous en considérant sa performance.

Finalement on remarque un compromis psycho-social entre d'une part la confiance en soi et d'autre part l'influence des relations sociales.

3 Notion de voisinage

Chaque particule dispose d'un sous-ensemble d'autres particules avec lequel elle est en interaction, c'est le voisinage de la particule. Cet entrelacement de rapports entre toutes les particules est assimilé à la topologie de l'essaim. On dénombre deux types de voisinage : **[III.5]**

3.1 Voisinage géographique

C'est un voisinage dynamique où les voisins sont les particules les plus proches. A chaque itération, les nouveaux voisins ou groupes doivent être réajustés en se référant à une distance prédéfinie dans l'espace de recherche. C'est donc bien un voisinage dynamique tel qu'illustré sur la Figure III.4 :

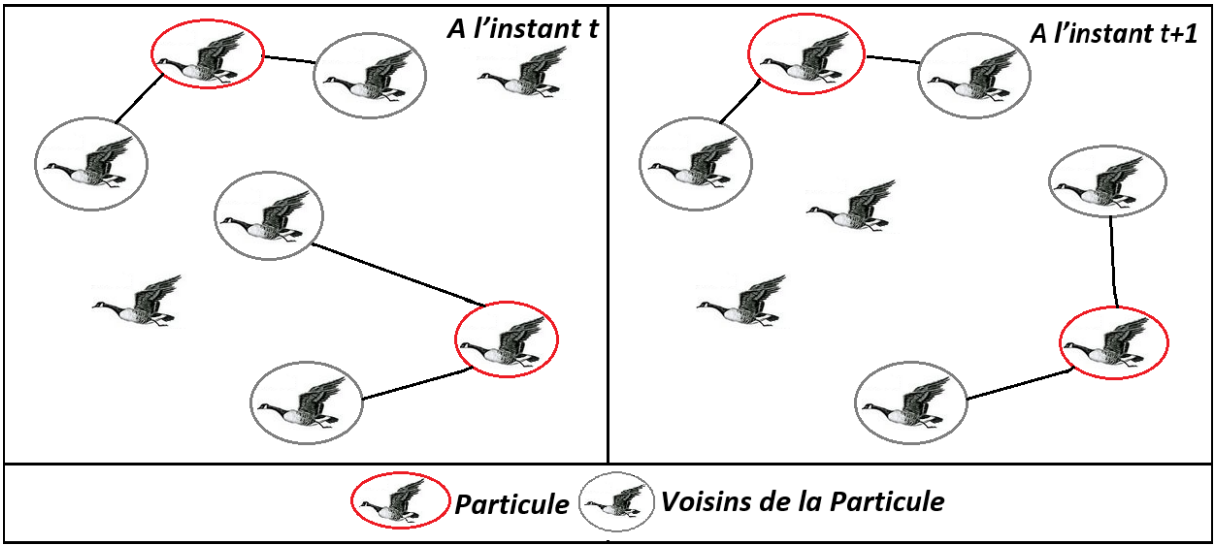


Figure III. 4: Voisinage géographique à l'instant t et t+1.

Dans cet exemple on va supposer que le voisinage d'une particule est un groupe formé des deux particules les plus proches. Dans la Figure III.4, la notion de voisinage dynamique est mise en évidence puisque pour un même essaim à l'instant **« t »** et à l'instant **« t+1 »** le voisinage n'est plus le même.

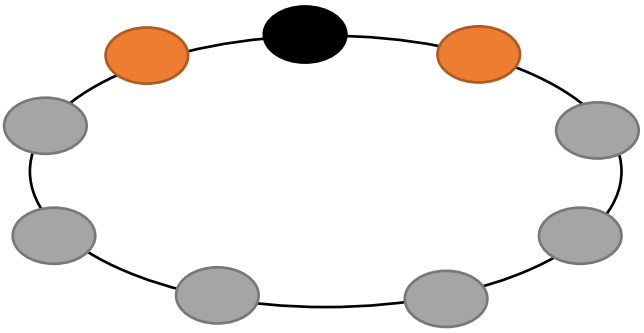
3.2 Voisinages sociaux

Ce type de voisinage est considéré comme statique, les voisins restent figés, autrement-dit, ils demeurent inchangés. C'est le voisinage auquel on a le plus souvent recours, en raison :

- De sa simplicité de programmation.
- Parce qu'il offre un meilleur rapport temps/coût, en termes de calcul.
- Dans un scénario de convergence, un voisinage social s'oriente forcement vers un voisinage géographique.

Dans la figure qui va suivre, les particules sont d'abord semées de manière fictive en forme de cercle, ensuite pour la particule étudiée, on insère au fur et à mesure dans ses informatrices ; dans un 1 ^{er} temps elle-même, dans un 2 ^{em} temps celles qui lui sont

adjacentes, puis de proche en proche jusqu'à atteindre la taille souhaitée, ainsi que définie dans la Figure **III.5** :



La particule principale

Les informatrices voisines directes

Les informatrices voisines successives

Figure III. 5: Voisinage en cercle (choix régulier des informatrices).

Dans cet exemple, la particule principale est placée en haut (en noir), ses informatrices correspondent aux deux particules directement à sa droite et à sa gauche (en orange dans notre figure).

Dans l'ébauche suivant par contre, les informatrices sont choisies de manière aléatoire, telle qu'exposée dans la Figure III.6 :

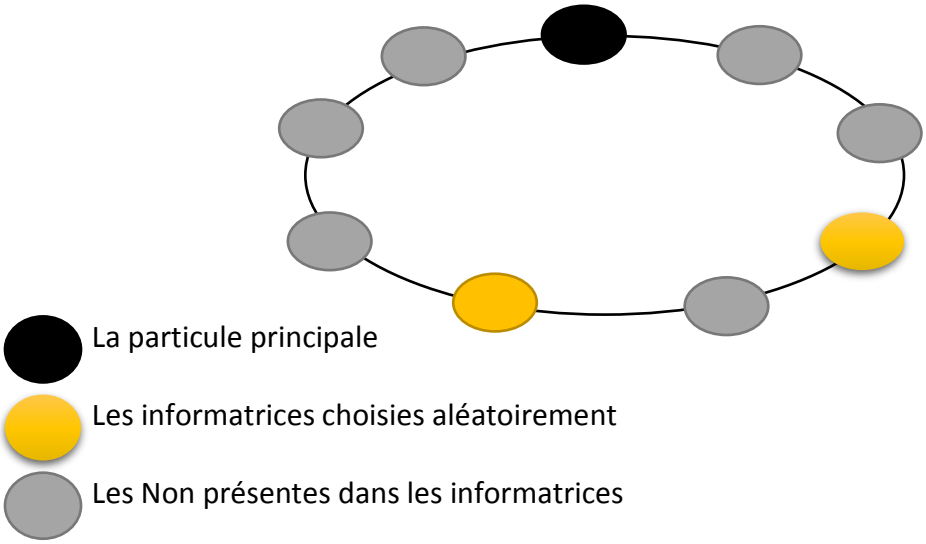


Figure III. 6: Voisinage en cercle (choix aléatoire des informatrices).

Ici, la particule principale est placée en haut (en noir) et ses informatrices sont choisies de façon aléatoire (en jaune par exemple dans notre figure).

4 Fonctionnement de l'algorithme PSO

En fonction de la nature du problème, il existe deux types d'algorithmes de PSO : le PSO discret, et le PSO continu. **[III.6]**

Nous allons présenter en premier lieu le type que nous allons étudier qui est le PSO continu.

- **PSO Continu**

Il est utilisé dans le cadre d'une optimisation continue, telle qu'elle est définie dans les travaux de Michel Gourgand et Sylverin Kemmoé Tchomté et sa représentation est la suivante : **[III.9]**

Dans un espace de recherche de dimension D, la particule i de l'essaim est modélisée par :

- Un vecteur position X_i= (x_{i1}, x_{i2}, …. x_{iD})
- Un vecteur vitesse V_i= (v_{i1}, v_{i2}, …, v_{iD}).

La qualité de sa position est déterminée par la valeur de la fonction objective en ce point.

Chaque particule garde en mémoire :

- La meilleure position par laquelle elle est déjà passée, notée :

  $P\ best$**i= (pbesti1, pbesti2, . . ., pbestiD).**

- La meilleure position atteinte par les particules de l'essaim notée

  $G\ best$ **= (gbest1, gbest2, . . ., gbestD).**

Au départ de l'algorithme, les particules de l'essaim sont initialisées de manière aléatoire dans l'espace de recherche. Par la suite, à chaque itération, les particules se déplacent, en fusionnant les trois composantes citées ci-dessus.

- $wv_{i,j}^{t}$: représente la composante d'inertie du déplacement, où le paramètre ***w*** gère l'influence de la direction de déplacement sur le déplacement futur.

- $c_1 r_{1_{i,j}}^{t}\ [pbest_{i,j}^{t} - x_{i,j}^{t}]$: représente la composante cognitive du déplacement, où le paramètre ***C1*** gère le comportement cognitif de la particule.

- $c_2 r_{2_{i,j}}^{t}\ [gbest_{j}^{t} - x_{i,j}^{t}]$: représente la composante sociale du déplacement où le paramètre ***C2*** gère l'aptitude sociale de la particule.

La fusion de ces trois composantes nous permet d'abord de dégager la vitesse qui elle-même nous offre la possibilité de calculer la position de la particule.

Ces deux variables sont obtenues par l'utilisation de l'équation (1) et de l'équation (2), successivement comme suit :

$$v_{i,j}^{t+1}=w v_{i,j}^{t} + c_1 r_{1\,i,j}[pbest_{i,j}^{t}-x_{i,j}^{t}] + c_2 r_{2\,i,j}[gbest_{j}^{t}-x_{i,j}^{t}],\ \ j\in\{1,2,\ldots D\} \qquad (1).$$

$$x_{i,j}^{t+1}=x_{i,j}^{t}+v_{i,j}^{t+1},\ j\in\{1,2,\ldots D\} \qquad (2).$$

- ***w*** est une constante, appelée coefficient/pourcentage d'inertie.
- ***c1*, *c2*** sont deux constantes, appelées coefficients/pourcentage d'accélération.
- ***r1, r2*** sont deux nombres aléatoires tirés uniformément dans **[0, 1]**, et ce à chaque itération ***t*** et pour chaque dimension ***j***.

A la fin du déplacement des particules dans une itération donnée, les nouvelles positions sont évaluées et les deux vecteurs ***Pbesti*** et ***Gbest*** sont mis à jour, suivant les deux équations (3) dans le cas d'une minimisation d'une fonction objective puis (4) dans une version globale de PSO(PSOG).

$$\overrightarrow{P}best_i^{t+1} = \begin{cases} \overrightarrow{P}best_i^{t}, & si\ f(\overrightarrow{X}_i^{t+1}) \geq \overrightarrow{P}best_i^{t} \\ \overrightarrow{X}_i^{t+1}, & sinon \end{cases} \qquad (3).$$

$$\overrightarrow{G}best = \arg\, min_{\overrightarrow{P}best\,i}\, f(\overrightarrow{P}best_i^{t+1}),\ 1\leq i\leq N. \qquad (4).$$

Cette procédure est représentée dans l'Algorithme ci-dessous, où **N** représente le nombre de particules de l'essaim :

1. **Initialiser** aléatoirement N particules : position X [i] et vitesse V [i]
2. **Evaluer** les positions des particules en utilisant la fonction objective.
3. **Pour** chaque particule i, $P\ best$i = X[i]
4. **Calculer** Gbest selon (4)
5. **tant que** le critère d'arrêt n'est pas satisfait **faire**
6. **Déplacer** les particules selon (1) et (2)
7. **Evaluer** les positions des particules
8. **Mettre à jour** (3) et (4)
9. **fin**

Figure III. 7: Formalisation de l'algorithme PSO

Cet algorithme peut être modéliser par :

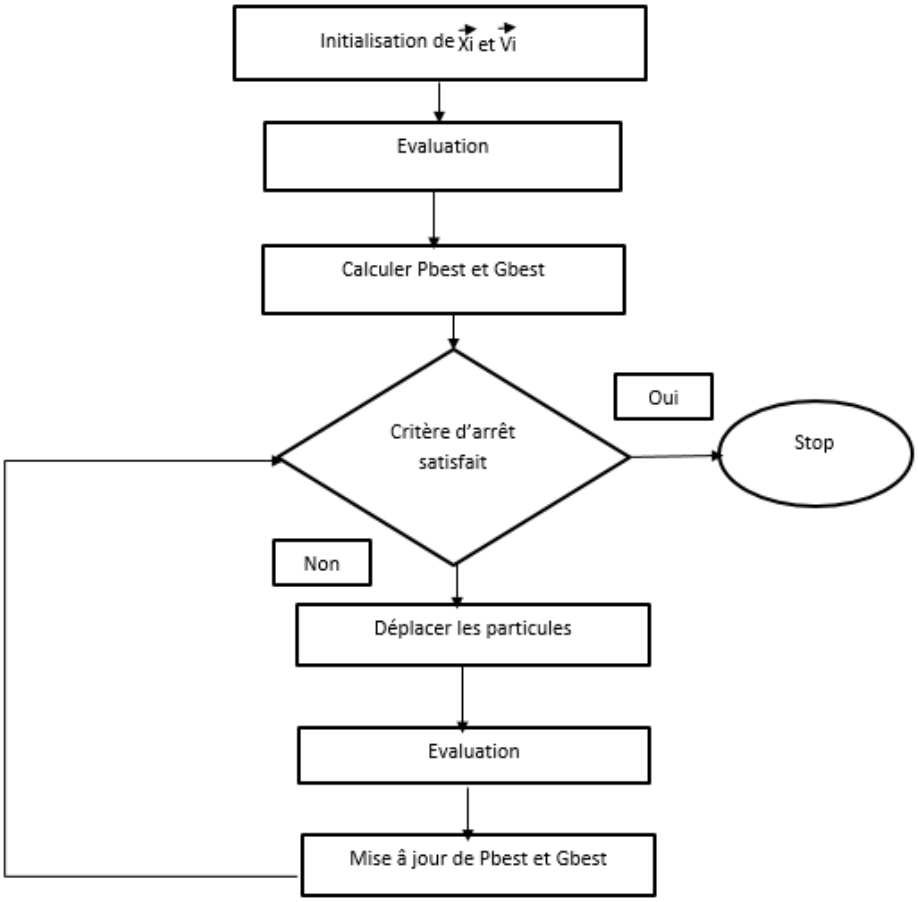


Figure III. 8 : Organigramme de PSO.

- **PSO discret [III.9]**

- Utilisé dans le cadre d'une optimisation discrète(combinatoire), telle qu'elle est définie dans les travaux de Maurice Clerc.
- N'est pas aussi efficace que certains algorithmes spécifiques.
- Très facile à adapter à n'importe quel autre problème combinatoire.
- Une position dans le PSO continu correspond à un ordonnancement en PSO discret.
- Changement de position en PSO continu correspond à des permutations en PSO discret en suivants des critères spécifiques.

5 Amélioration apportée au PSO

Depuis son apparition, plusieurs améliorations et des nouvelles notions ont été apportées à la version de PSO précédemment décrite appelée la version globale.

5.1 Cantonnement des particules

Dans l'hypothèse d'un déplacement très rapide de la particule, cela peut l'éjecter de l'espace de recherche, il serait judicieux d'introduire une nouvelle variable **Vmax**. Elle aura pour objet de limiter la vitesse sur chaque dimension et ainsi de parer à une éventuelle explosion du système. **[III.8]**

Cela n'amoindrit nullement les valeurs de **v[i]** à l'intervalle **[Vimin, Vimax]**, il borne seulement la distance maximale parcourue par une particule au cours d'une itération. Un tel procédé à l'avantage aussi de contrôler la divergence de l'algorithme et de réaliser un compromis efficace entre intensification (recherche locale) et diversification (recherche global).

Pareille l'amélioration peut être aussi enrichie par une stratégie de confinement/cantonnement des particules, afin d'éviter tout problème de leur sortie.
Parmi les stratégies proposées on peut trouver : **[III.8]**

- Une 1ère stratégie qui consiste à réintégrer une particule sortie de son espace de recherche.
- Une 2ème stratégie qui consiste à ce que la particule soit laissée à l'extérieur de l'espace de recherche, sans prendre en compte sa fonction "objectif". Par conséquent elle ne risque pas d'attirer les autres particules (risque d'attraction).
- Dans une 3ème stratégie, la particule est stoppée à la frontière et les composantes associées à sa vitesse sont annulées.

- Enfin dans une 4ème stratégie, la particule rebondit sur la frontière et se trouve bloquée ; les composantes de la vitesse sont multipliées par un coefficient tiré aléatoirement dans l'intervalle **[-1,0]**.

5.2 Coefficient d'inertie

Le coefficient d'inertie *w* contrôle l'influence de la direction de la particule sur son déplacement futur. Le but de l'introduction de ce paramètre est de réaliser un équilibre entre la recherche locale (exploitation) et la recherche globale (exploration). L'intensité de l'exploration de l'espace de recherche dépend de la valeur du poids d'inertie, une grande valeur de w facilitant une exploration globale, alors qu'une petite valeur facilite l'exploration locale. Du fait de son influence sur les performances de l'algorithme PSO, le poids d'inertie a suscité un grand intérêt de la part de la communauté des chercheurs. Dans la littérature, un coefficient d'inertie dynamique qui varie au cours du temps a été proposé. Il commence par une valeur proche de 0,9 et descend linéairement pour arriver à 0,4. Cette stratégie a beaucoup amélioré les performances de PSO pour plusieurs problèmes d'optimisation. Le coefficient d'inertie *w* varie linéairement avec le temps selon la formule suivante : **[III.7]**

$$W = wmin + (wmax - wmin) * (iter / maxiter) \quad (5)$$

Où :

- iter est l'itération courante.
- $maxiter$ est le nombre maximal d'itérations.
- $wmax$ et $wmin$ désignent respectivement les valeurs maximum et minimum du coefficient *w* (généralement, $wmax, wmin \in [0, 1]$).

5.3 Topologie du voisinage

Comme reprit plus haut le choix de la topologie (réseau de communication entre particules) a une influence notable sur la performance du PSO, plus précisément sur le déplacement des particules. Dans la version PSO (voir figure de l'algorithme PSO), les auteurs ont défini une topologie entièrement connectée (chaque particule est reliée à toute les

autres). D'où l'appellation PSO version globale **(Gbest)**, chaque particule dispose d'informations sur toutes les particules de l'essaim. Ces données effectivement utilisées sont représentées par le terme **Gbest**. Cependant cette version a l'inconvénient de ne pas pousser l'exploration suffisamment loin ; cela débouche sur une stagnation dans un optimum local et donc à une convergence prématurée.

Pour dénouer ce nœud, plusieurs variantes de PSO "originales", dénommées version locale **(Lbest)** ont été mises en place. On peut citer celle proposée dans [III.8], laquelle recourt à un graphe d'information statique sous forme d'anneau (cette version est connue comme étant la version locale classique). Dans PSOL, le terme **Gbest** est remplacé par **Lbest**. A chaque particule ***i*** a défini un voisinage **i.e**, l'information partagée doit être la meilleure solution trouvée dans le voisinage de chaque particule **Pbesti. [III.7]**

Conclusion

Dans ce chapitre, nous avons présenté l'algorithme d'optimisation par essaim de particule (PSO) inspiré du monde des animaux (espèces d'oiseaux). Depuis sa création, cette méthode a rencontré un franc succès, en raison de sa simplicité et de son efficacité sur une vaste gamme de problèmes.

Dans le chapitre qui suit nous verrons l'implémentation de cet algorithme dans l'environnement CloudSim pour le problème d'ordonnancement de tâches.

Introduction

Cette partie constitue le dernier volet de ce mémoire. Après avoir présenté le cloud et ses concepts et élaborer le problème d'ordonnancement de tâches et ses différentes méthodes de résolutions pour en spécifier la technique que nous avons utilisé ; c'est ainsi que nous allons la conclure par une réalisation qui est la partie importante consacrée à la présentation du projet et son environnement de développement.

En premier lieu nous abordons quelques définitions de base puis nous présenterons les différents objectifs que nous avons utilisés suivi de la présentation des outils utilisés dans le développement de notre projet, pour en finir avec la réalisation et les tests.

1 Définitions de base

1.1 Datacenter (DC)

Un **centre de données** (en anglais *data center*) appelé aussi un "centre de traitement de données" est un lieu ou un service sur lequel se trouve regrouper des équipements informatiques (ordinateurs centraux, serveur, équipements réseaux et de télécommunications….Etc ; comme exemple de datacenter nous citons les datacenters Microsoft, et Facebook (voir la figure (IV. 1)). **[IV.1]**

Les grands datacenters sont généralement conçus pour soutenir la demande croissante de calcul et de stockage de donnée d'entreprise en pleine croissance et de l'industrie mondiale.

Il ya environ 23000 datacenters dans le monde en 2014, le marché devrait croitre à environ 343,4 milliards de dollars ; nous prenons Facebook par exemple : le nombre de nœuds de calcul dans le centre de donnée est 60 000, ces centres de données à grande échelle consomment généralement une énorme quantité d'énergie électrique qui entraine des coûts élevés**. [IV.2] [IV.3]**

Figure IV. 1: les DCs google et facebook.

1.2 Host

Un hôte (en anglais host) est un terme général pour décrire tout composant relié à un réseau informatique, et il fournit des services à d'autres systèmes ou utilisateurs (serveur informatique) .

On utilise aussi le terme de **système hôte** pour désigner le système qui héberge un système virtuel qui utilise à son tour les ressources du système hôte. Cela s'appelle la virtualisation. **[IV.4]**

1.3 Virtual Machine (VM)

Une machine virtuelle (en anglais Virtual Machine) est une instance de la machine physique (PM) qui donne illusion aux utilisateurs d'avoir un accès direct à la machine physique. Les VM sont utilisées pour permettre le partage de ressources d'un matériel très coûteux et chaque VM est une copie protégée et isolée du système.

La virtualisation est donc utilisée pour réduire les coûts du matériel et d'améliorer la productivité globale en permettant à plusieurs utilisateurs de travailler simultanément sur une même PM et qui permet d'augmenter l'utilisations de la machine physique. **[IV.5]**

1.4 Cloudlets

Les cloudlets ou bien les tâches est le nombre d'instructions et de données connu à exécuter et à transférer. **[IV.6]**

1.5 Broker

Un courtier (en anglais broker), est une personne ou une entreprise tierce qui joue le rôle d'intermédiaire entre l'acheteur d'un service de cloud computing et les vendeurs de ce service. **[IV.7]**

En général, broker peut endosser trois rôles

- **Le négociateur :** il peut être mandaté pour négocier des contrats. Dans ce cas, il peut répartir les services entre divers fournisseurs dans un souci d'économie maximale, malgré les éventuelles complexités engendrées par les négociations impliquant plusieurs fournisseurs. Une fois la recherche terminée, le courtier présente au client une liste de fournisseurs recommandés ainsi qu'une comparaison des caractéristiques du service, des ventilations des coûts, et d'autres critères.
- **L'agrégateur de services :** il combine et intègre plusieurs services en un ou plusieurs nouveaux services. Il assure l'intégration des données et les connexions sécurisées entre son client et les multiples fournisseurs de cloud. Il peut choisir les services de plusieurs fournisseurs, selon les critères déterminés auparavant avec son client.
- **Le facilitateur :** il peut améliorer un service donné en optimisant certaines capacités spécifiques ou en proposant des services à valeur ajoutée ou personnalisés. Cette amélioration peut concerner la gestion de l'accès aux services cloud, le contrôle des identités, les rapports de performance, etc… **[IV.8]**

2 Processus d'exécution de tâches dans le cloud

Les étapes présentées ici sont celles qui concernent la simulation d'un environnement cloud sous CloudSim.

2.1 Création de la topologie du cloud [IV.6]

2.1.1 Création des DCs

Pour configurer un environnement Cloud, nous devons commencer par configurer sa topologie, donc nous devons configurer le nombre des Datacenter et leurs caractéristiques. Pour configurer un Datacenter, nous devons le créer et sa création est effectuée en deux parties de base : dans la première partie on doit spécifier les caractéristiques du Datacenter

tel que le nom du Datacenter, l'ID, le coût de traitement, coût d'utilisation de mémoire, coût de stockage, et le coût de la bande passante, on trouve aussi le nombre de Datacenter, etc.

Dans la deuxième partie, elle contient la liste des Datacenters que nous avons initialisé.

2.1.2 Création des hôtes

Une fois la création des différents Datacenter est effectuée, nous devrons créer les hôtes qui se trouvent dans chaque Datacenter Un datacenter peut contenir un ou plusieurs hôtes, mais avant la création nous devons attribuer les différentes caractéristiques de chaque hôte (machine physique) ; Parmi ces caractéristiques, on trouve le nombre de processeur, la vitesse de chaque processeur (MIPS), la ram, la capacité de stockage et aussi la bande passante. On peut aussi définir le nombre d'hôte qu'on va affecter pour chaque Datacenter.

2.1.3 Création des VMS

Pour exécuter les requêtes des utilisateurs, nous avons besoin de créer des machines virtuelles. Ces machines virtuelles doivent être crées dans un hôte, un hôte peut contenir une ou plusieurs machines virtuelles ; pour que la création de la machine virtuelle s'effectue avec succès dans un hôte, les caractéristiques de la VM doivent être inférieurs aux caractéristiques de l'hôte. Avant la création nous devons attribuer les différentes caractéristiques pour chaque VM ; parmi ces caractéristiques on peut citer l'identificateur de la VM, le nombre de processeur, la vitesse d'exécutions (MIPS), la ram, la bande passante et la capacité de stockage ;

1.*La VM 2 et 3 s'exécutent sur la même PM(Hôte) dont ID= 0*

Id VM	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Id Hôte	1	0	0	2	2	3	2	4	5	6	7	7	8	9	10	11	12	13	14

2.*CPU (VM 2) + CPU (VM 3) <= CPU (hôte 0)*

3. *RAM (VM 2) + RAM (VM 3) <= RAM (hôte 0)*

4. *BW (VM 2) + BW (VM 3) <= BW (hôte 0)*

Figure IV.2: Affectation des VMs aux Hôtes.

2.1.4 Préparation des cloudlets

Après la création des datacenters et l'affectation des VMs aux hôtes et aussi l'affectation des hôtes pour chaque datacenter nous devons configurer les requêtes (Cloudlets) des différents utilisateurs afin de les exécutée ; par exemple le nombre des cloudlets à simuler, la taille de la cloudlet, les données demandées et enfin l'id de la VM qui exécute cette cloudlet.

Après cette étape affecter chaque cloudlet à sa VM afin d'effectuer l'exécution

Dans notre cas toute cette procédure est effectuée par l'algorithme PSO.

2.2 Processus d'exécution des tâches

Le processus d'exécution des tâches dans le cloud peut être décrit comme suit :

1. Après la création du DC ses informations et ses caractéristiques seront envoyés vers le CSI (cloud information service) qui contient toutes les informations concernant les DCs (le nombre de DC disponibles les caractéristiques de chaque DC, les conditions des services de chaque DC et à chaque création d'un nouveau DC ses caractéristiques et informations seront enregistrées dans le CSI.
2. Un ensemble de cloudlet arrivent et elles seront envoyées vers le broker.
3. Communication et interaction du broker avec le CSI afin d'obtenir les informations nécessaires pour les services du cloud et les caractéristiques de chaque DC afin d'effectuer des négociations pour l'exécution des cloudlets.
4. Une fois la négociation est effectuée le broker transmet les cloudets pour le DC qui lui convient (qui satisfait les conditions nécessaires pour chaque cloudlet).
5. Apres l'exécution des différentes cloudlets par les VMs du DC choisit et traitement de toutes les requêtes et services des utilisateurs, le résultat sera envoyé vers le CSI.
6. Le CSI envoie à son tour le résultat vers le broker.

De ce fait la fin d'exécution de la tâche. **[IV.9]**

Ce processus est décrit dans le schéma suivant :

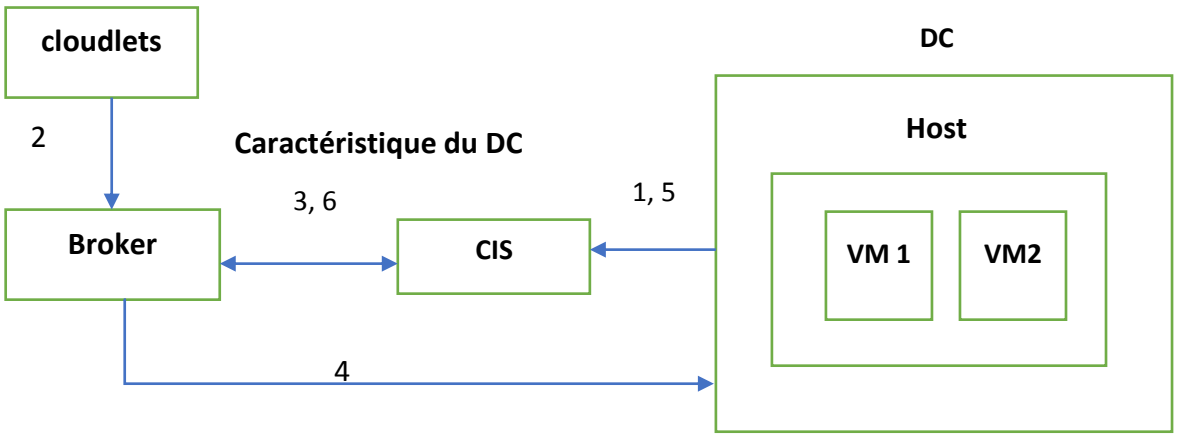


Figure IV.3:Processus d'exécution des tâches.

Dans la figure ci-dessus on a le processus d'exécution composé d'un seul DC qui contient un seul hôte et cet hôte est composé à son tour de deux VMs.

3 Objectif à optimiser

3.1 Makespan

L'une des mesures les plus utilisées dans l'évaluation des algorithmes d'ordonnancement de tâches est le temps de complétion ou makespan. Le makespan est la différence entre le temps de soumission des tâches et le temps de réception des résultats. Autrement dit, c'est le temps de fin d'exécution de la dernière tâche. **[IV.10]**

Le makespan est calculé selon l'équation (1.1) :

$$\textbf{Makespan= Max\{dcWorkingTime}_j\} \qquad \textbf{(1.1)}$$

Ou,

- **dcWorkingTime**$_j$ est le temps de travail du Datacenter j

$$\textbf{dcWorkingTime}_j=\textbf{TempsEx}_{ij}+\textbf{TempsCom}_{ij} \qquad \textbf{(1.2)}$$

Ou,

- **TempsEx**$_{ij}$ est le temps d'exécution de la tâche i par le Datacenter j
- **TempsCom**$_{ij}$: est le temps de communication de la tâche i avec le Datacenter j

$$TempsEx_{ij} = Math.random() * (max - min) + min/2 \quad (1.3)$$

$$TempsCom_{ij} = Math.random() * (max - min) + min \quad (1.4)$$

Où

- La méthode : **Math.random()** est une méthode qui permet de générer un nombre aléatoire.

Il existe différentes méthodes de le calculer, pour notre cas on a utilisé la manier décrite dans les formules (1.3) et (1.4)**.**

En général, le *makespan* est composé du temps d'exécution des tâches et du temps de transferts(communication) des données sur le réseau. Le temps d'exécution dépend à la fois de la charge de travail et des performances de machines. Le temps de transmission du réseau dépend, à la fois, de la latence du réseau et de la taille des données transmises.

3.2 Le coût

La plupart des fournisseurs de *cloud* ont fixé des prix pour l'utilisation de leurs services. Ils ont fixé le prix du transfert d'une unité de donnée entre deux services et le prix pour le traitement par unité de temps. **[IV.10]**

Le coût total d'exécution des tâches dans l'IaaS est défini dans l'équation **(2.1)** :

$$\boldsymbol{Coût} = \sum_{i,j=0}^{N,M} \boldsymbol{CoûtEx}(i,j) + \sum_{i,j=0}^{N,M} \boldsymbol{CoûtCom}(i,j) \quad (2.1)$$

Ou,

- **N** est le nombre de tâches à exécuter
- **M** est le nombre de DC
- **CoûtEx(i, j)** est le coût d'exécution de la tâche i par le DC j
- **CoûtCom(i,j)** est le coût de communication de la tâche i avec le DC j

$$CoûtEx(i,j) = TempsEx_{ij} * PrixU_{EX} \quad (2.2)$$

$$CoûtCom(i,j) = TempsCom_{ij} * PrixU_{Com} \quad (2.3)$$

Ou,

- **TempsEx(i,j)** est le temps d'exécution de la tâche i par le DC j calculé selon **(1.3)**
- **TempsCom(i,j)** est le temps de communication de la tâche i avec le Datacenter j calculé selon **(1.4)**

- **$PrixU_{EX}$** et **$PrixU_{Com}$** sont des prix unitaires attribuer à chaque temps que ce soit d'exécution ou de communication.

3.3 L'énergie

L'augmentation continue de la consommation d'énergie dans le cloud computing soulève une grande préoccupation pour les gouvernements et les fournisseurs de services. Leur objectif principal est de fournir des services aux clients tout en augmentant leurs revenus et en consommant plus efficacement l'énergie consommée par leurs infrastructures.

Le cloud offre alors plusieurs techniques qui permettent de réduire la consommation d'énergie comme par exemple la virtualisation qu'on a déjà cité et qui permet de faire marcher plusieurs systèmes dans un même hôte ; aussi on a la technique de migration des VMs qui peut aider à l'équilibrage de charge et qui fait le transfert des machines virtuelles dans d'autres machines physiques. **[IV.5]**

Pour calculer l'énergie consommée par l'ensemble des DCs on a qu'à appliquer
L'équation (3.1)

$$\mathbf{Energie} = \sum_{i,j=0}^{N,M} \boldsymbol{energieEx(i,j)} + \sum_{i,j=0}^{N,M} \boldsymbol{energieCom(i,j)} \qquad \textbf{(3.1)}$$

- **N** est le nombre de tâche à exécuter.
- **EnergieEx(i,j)** est l'énergie consommée lors d'une exécution d'une tâche i par le DC j
- **EnergieCom(i,j)** est l'énergie consommée lors d'une communication attribuée entre la tâche i et le DC j et elle est calculée

$$\textbf{EnergieEx(i,j)= TempsEx(i,j)*k} \qquad \textbf{(3.2)}$$

$$\textbf{EnergieCom}_{i,j}\textbf{=TempsEx}_{i,j}\textbf{*C} \qquad \textbf{(3.3)}$$

Avec

- **K** est une quantité énergétique consommée à chaque exécution.
- **C** est une quantité énergétique consommée lors d'une communication
- **$TempsEx_{ij}$** est le temps d'exécution de la tâche i par le DC j calculé selon **(1.3)**
- **$TempsCom_{ij}$** est le temps de communication de la tâche i avec le Datacenter j calculé selon **(1.4).**

4 Réalisation et tests

4.1 Outils de travail

- Une machine avec un processeur Intel® pentium ®CPU N3520@2.16Ghz, 2.16Ghz.
- Le langage de programmation Java.
- Un IDE Eclipse version Mars.
- Un simulateur CloudSim version 3.0.3 développés à base de Java.

L'installation et l'implémentation de l'environnement de travail est détaillée dans la partie Annexe.

4.2 Test et simulations

4.2.1 Principe de convergence

Pour démontrer le principe de convergence dans l'algorithme PSO on a opté pour cet exemple simple qui est une exécution d'un algorithme d'ordonnancement de 10 tâches selon 5 DCs sous une dimension de 10 particules et durant 7 itérations Successives, avec une fonction objectif calculée comme suit :

Fonction fitness = **A***makspan + **B***cout+ **C***Energie

Ou A=1, B, C=0

Les résultats obtenus sont représentés dans le tableau suivant :

Pa / Itérati	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9
Itération 0	**2030**	2521	**2030**	**2030**	2449	**2030**	2521	3162	**2030**	2521
Itération 1	**2030**	**2030**	**2030**	2521	**2030**	**2030**	**2030**	2883	2449	2449
Itération 2	**2030**	**2030**	**2030**	2521	**2030**	**2030**	**2030**	**2030**	2096	2695
Itération 3	**2030**	**2030**	**2030**	**2030**	**2030**	**2030**	**2030**	2521	2449	2695
Itération 4	**2030**	**2030**	**2030**	**2030**	**2030**	**2030**	**2030**	2639	**2009**	**2030**
Itération 5	2024	2017	2013	2630	2630	**2009**	2630	2009	**2009**	**2009**
Itération 6	**2009**	**2009**	**2009**	**2009**	**2009**	**2009**	**2009**	**2009**	**2009**	**2009**

Figure IV.4: résultat de l'étude convergence de PSO

A. Analyse des résultats

Selon les résultats obtenus on remarque qu'au niveau de :

- La 1ere itération : génération des valeurs différentes par certaines particule et une valeur qui est égale à 2030 ; choix de cette valeur comme l'optimal à suivre pour le moment.
- La 2eme et 3 eme itération : Ya encore la variation des résultats mais aussi on voit bien que les particules : P0, P1, P2, P4, P5, P6, P7 ont généré la même valeur qui est toujours 2030.
- La 4eme itération toutes les particules ont générées la même valeur sauf la particule P8, P9, aussi la particule P7 a changé sa valeur
- La 5 eme itération : génération d'une nouvelle valeur plus optimale que 2030 et elle est générée par la particule P8 ; de ce fait changement de l'optimum.
- La 6 eme itération génération de la nouvelle valeur de l'optimum par la particule P5 et P9.
- La 7 eme itération (la dernière) génération d'une valeur identique par toutes les particules.

A ce niveau on dit que les particules ont atteint le point de convergence.

B. Résultats des affectations obtenus :

- Pour **Gbest=2030** : l'algorithme PSO a effectué les affectations suivantes :
  - ✓ Un nombre de **4** tâches sont affectées au **DC0** et elles sont : **T3, T4, T5, T9.**
  - ✓ Un nombre de **1** tâche est affectée au **DC1** et elle est la tâche : **T1**.
  - ✓ Un nombre de **2** tâches sont affectées au **DC2** et elles sont : **T2, T7**.
  - ✓ Un nombre de **2** tâches sont affectées au **DC3** et elles sont : **T6, T8**.
  - ✓ Un nombre de **1** tâche est affectée au **DC4** et elles sont : **T0**.

- Pour **Gbest=2009** : l'algorithme PSO a effectué les affectations suivantes :
  - ✓ Un nombre de **2** tâches sont affectées au **DC0** et elles sont : **T2, T3.**
  - ✓ Un nombre de **2** tâches sont affectées au **DC1** et elles sont les tâches : **T5**, **T7.**
  - ✓ Un nombre de **3** tâches sont affectées au **DC2** et elles sont : **T1, T4, T9**.
  - ✓ Un nombre de **3** tâches sont affectées au **DC3** et elles sont : **T0, T6, T8**.
  - ✓ Un nombre de **0** tâche est affectée au **DC4**.

Remarque : le Gbest est la valeur de la solution optimale (résultat de la fonction objective) et dans notre cas :

Gbest= 2009 est obtenu comme suit

Fonction fitness = (A*2009) + (0*244.65)+(0*1616.20)=2009

Le graphique suivant traduit les résultats du précédent tableau

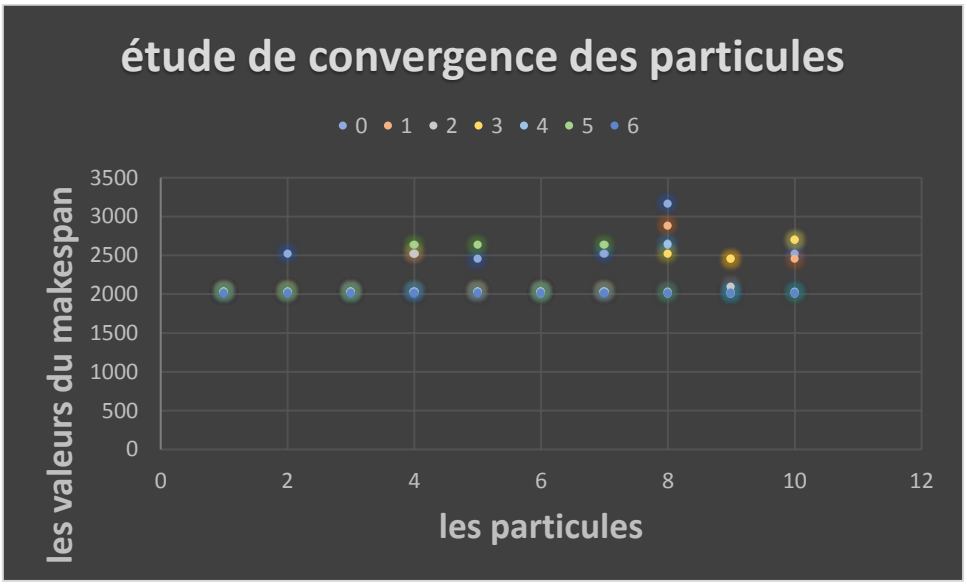


Figure IV.5: résultat de la convergence entre de PSO

4.2.2 Évaluations

Pour évaluer les performances de l'algorithme 'PSO', nous avons dû au premier lieu effectuer un choix portant sur les valeurs à affecter aux paramètres du PSO puis nous avons effectué une série de tests comparatifs entre l'algorithme PSO et les algorithmes FCFS et Round Robin.

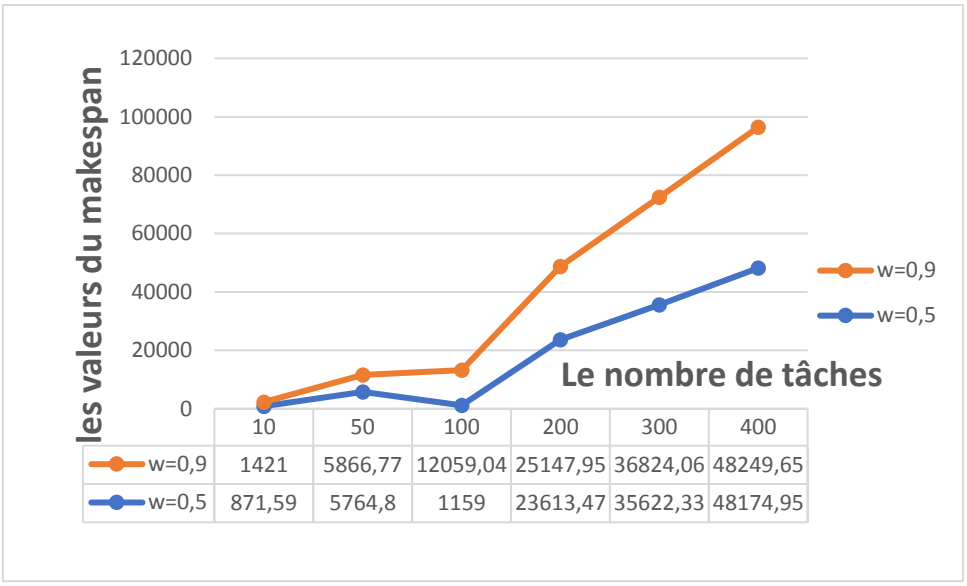
Nous avons réalisé ces tests avec un nombre différent de tâches : 10, 50, 100, 200, 300 et 400 tâches.

Evaluation1 : dans le tableau suivant nous avons attribuer les résultats obtenus durant les simulations des deux configurations effectuées sur PSO.

Le nombre de tâches	Choix1 : PSO avec W=0.5			Choix 2 : PSO avec W=0.9		
	makespan	Le coût	L'énergie	makespan	Le coût	L'énergie
10	871.59	181,64	1009.08	1421	246.49	1369.38
50	5764.8	1246.40	6924	5866.77	1326.27	7368.16
100	1159	2694.54	14969.68	12059	2763	15349.97
200	23613.47	5489.64	30497.99	25147.95	5612.04	31177
300	35622.33	8396.46	46647.03	36824.06	8715.40	48418.9
400	48174.95	11355.77	63087.60	48249.65	11624.06	64578.13

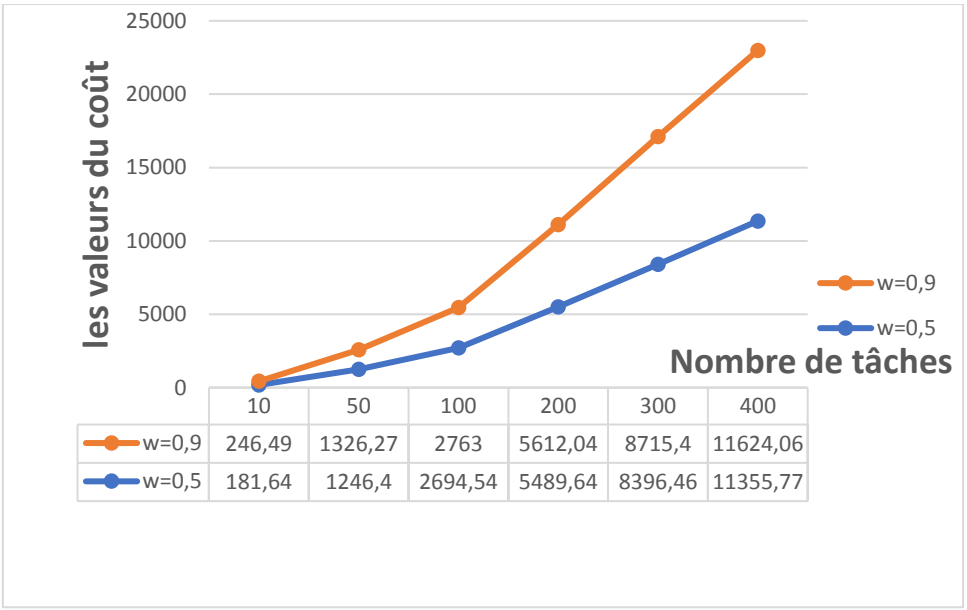
Figure IV. 6: Les valeurs du Makespan, le cout et l'énergie donné par PSO en fonction de nombre de tâches avec deux choix de paramètre d'inertie w.

Les courbes suivantes traduisent les résultats du précédent tableau et pour bien démontrer les différences entre les deux choix on a effectué une courbe spécifique pour chaque objectif de sorte que :la première représente les résultats du makespan, la deuxième représente les résultats du coût et la dernière les résultats de l'énergie :



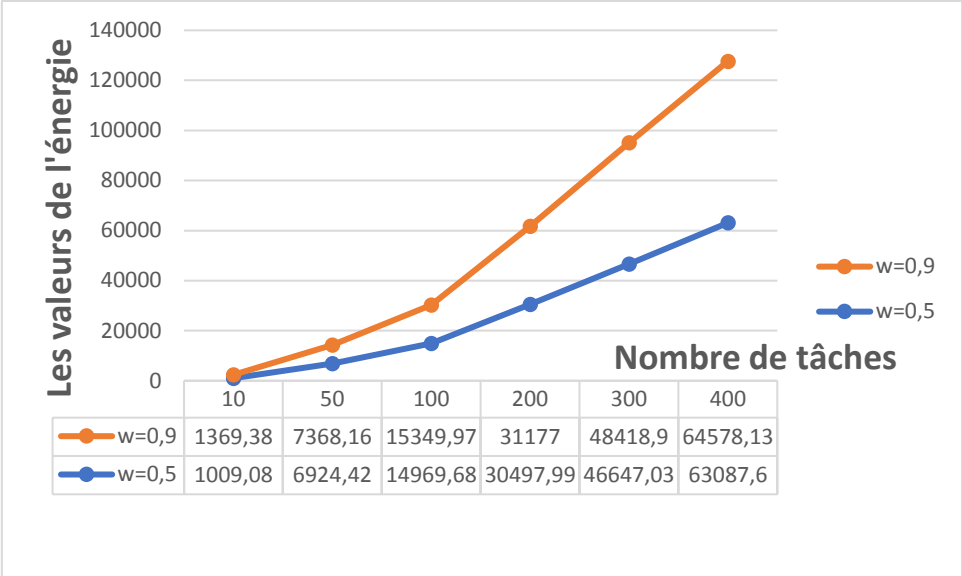
	10	50	100	200	300	400
w=0,9	1421	5866,77	12059,04	25147,95	36824,06	48249,65
w=0,5	871,59	5764,8	1159	23613,47	35622,33	48174,95

Figure IV.7: : les résultats du makespan



	10	50	100	200	300	400
w=0,9	246,49	1326,27	2763	5612,04	8715,4	11624,06
w=0,5	181,64	1246,4	2694,54	5489,64	8396,46	11355,77

Figure IV.8:: les résultats du coût



	10	50	100	200	300	400
w=0,9	1369,38	7368,16	15349,97	31177	48418,9	64578,13
w=0,5	1009,08	6924,42	14969,68	30497,99	46647,03	63087,6

Figure IV.9: les résultats de l'énergie

Nous constatons que les résultats du makespan, le coût et l'énergie obtenues par l'exécution du PSO avec la valeur de w=0.5 (choix 1) est toujours inferieurs par rapport à ceux obtenues par l'exécution du PSO avec la valeur du W=0.9 (choix 2).

D'où on peut dire que sur nos exemples, les résultats obtenus du PSO avec une composante d'inertie W=0.5 sont meilleurs que les résultats obtenus du PSO avec une composante d'inertie W=0.9.

Evaluation 2 : étude comparative

Dans ce qui suit, nous avons retenu les valeurs du choix 1 pour les paramètres de PSO et nous avons effectué des différentes simulations pour faire une comparaison entre l'algorithme PSO et deux autres algorithmes qui sont l'algorithme FCFS et l'algorithme RR.

Etude des résultats

- **Selon l'affectation des tâches et le temps d'exécution de chaque tâche** : nous avons remarqué que l'affectation des tâches aux VM et leurs exécutions diffèrent aussi d'un algorithme à un autre.

  Pour illustrer cette affectation nous avons choisi un exemple d'affectation de 10 tâches à 5 DCs et les résultats obtenus sont les suivants.

Au niveau de l'algorithme PSO :

Id Tâche	0	1	2	3	4	5	6	7	8	9
Id VM	2	3	3	4	5	6	6	2	2	2
Id DC	2	1	1	3	0	4	4	2	2	2
TempsEx	642.75	711.41	1323.79	239.79	89.138	494.37	1377.27	965.38	1460.69	1562.20

Au niveau de l'algorithme Round Robin :

Id Tâche	0	1	2	3	4	5	6	7	8	9
Id VM	5	4	6	5	4	5	6	3	4	4
Id DC	3	2	4	3	2	3	4	1	2	2
TempsEx	468.1	650.53	886.7	589.59	1090.24	1240.12	1425.48	244.40	1964.32	2227.83

Au niveau de l'algorithme FCFS :

Id Tâche	0	1	2	3	4	5	6	7	8	9
Id VM	6	5	6	6	6	5	6	4	2	5
Id DC	1	0	1	1	1	0	1	4	2	0
TempsEx	396.12	822.39	960.44	1444.26	1748.35	1748.35	2075.36	2075.36	2075.36	2075.36

Remarque : les tâches écrites par la même couleur seront exécutées par la même VM affectée au même DC.

Dans les illustrations suivantes (**Tableau 1, 2 et 3**) nous présentons les résultats de la simulation du makespan, Energie, le coût en fonction du nombre de tâches dans chacun des algorithmes PSO, Round Robin et FCFS :

Nbre tâche	10	50	100	200	300	400
PSO	1333,13	6098,02	11299,29	24669,87	36039,38	49776,3
Round Robin	2538,62	8686,26	13618,25	32535,54	39019,26	51074,95
FCFS	3204,74	6222,29	16504,41	27155,17	38133,86	50294,40

Tableau 1: Résultat de la simulation du makespan

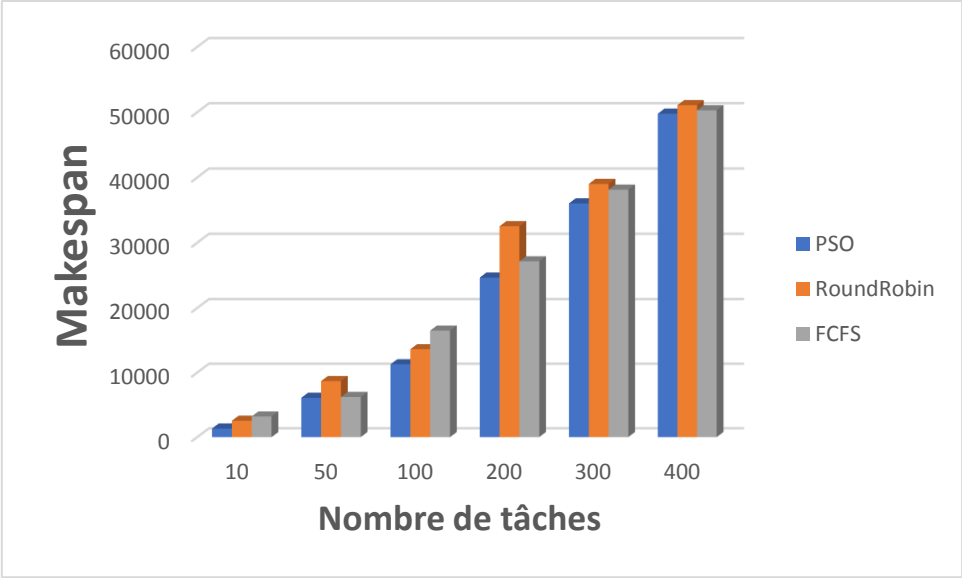


Figure IV.10: Résultat du Makespan selon les trois Algorithmes

Analyse 1 : à base des résultats obtenus on remarque bien que les valeurs du makespan générées au niveau de chaque exécution pour chacun des trois algorithmes exécutés sont variantes d'une exécution à une autre et aussi d'un algorithme à un autre ; de plus on remarque que l'algorithme PSO a généré les minimums de ces résultats par rapport à l'algorithme Round Robin et aussi à l'algorithme FCFS.

Nbre tâche	10	50	100	200	300	400
PSO	1340,84	7353,85	15012,44	31440,42	8484,27	62091,65
Round Robin	1479,86	7940,00	15805,89	34390,12	48444,54	63521,84
FCFS	1348,86	7777,74	16584,90	32119,76	47842,41	64732,84

Tableau 2: Résultat de la simulation de l'énergie

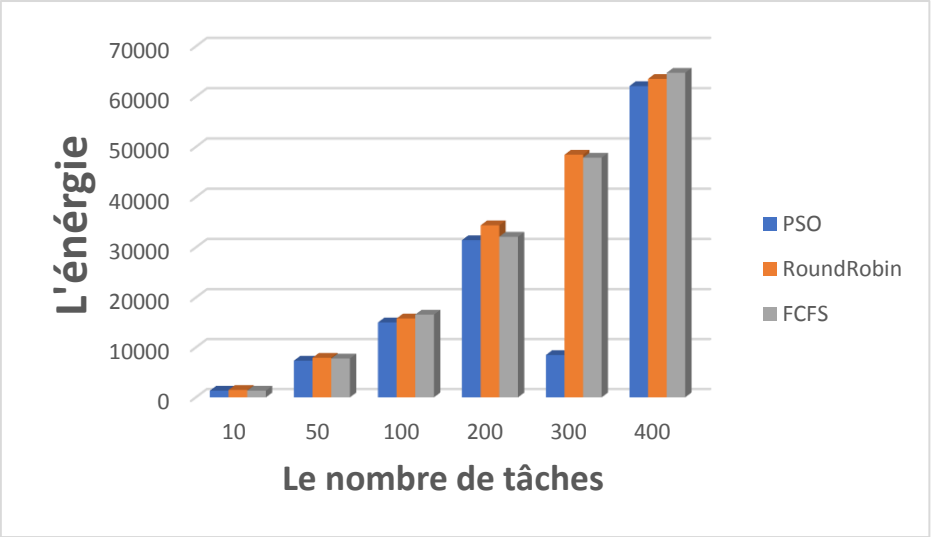


Figure IV.11:Résultat de l'énergie selon les trois Algorithmes

Analyse 2 : comme l'analyse précédente les valeurs de l'énergie générées au niveau de chaque exécution pour chacun des trois algorithmes exécutés sont variantes d'une exécution à une autre et aussi d'un algorithme à un autre de plus on remarque que l'algorithme PSO à générer les résultats minimums de l'énergie par rapport à l'algorithme Round Robin et FCFS.

Nbre tâches	10	50	100	200	300	400
PSO	193,08	1360,73	2622,34	5576,96	8484,27	11429,69
Round Robin	266,38	1429,26	2845,06	6190,22	8720,02	11433,93
FCFS	242,8	1399,99	2985,28	5781,56	8611,63	11651,91

Tableau 3:Résultats de la simulation du coût

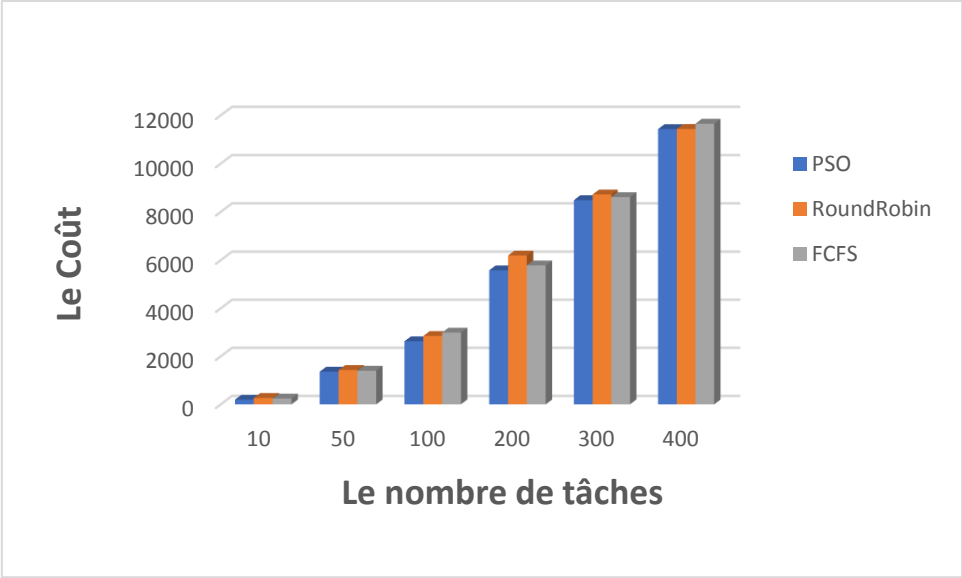


Figure IV.12:Résultat du coût selon les trois Algorithmes

Analyse 3 : même chose pour le coût ; l'algorithme PSO a généré les coûts minimums par rapport aux deux autres algorithmes.

Synthèse :

Apres l'analyse et comparaison entre les différents résultats nous constatons que les résultats de l'algorithme PSO sont toujours plus petits par rapport à l'algorithme Round Robin et l'algorithme FCFS et sa en termes des trois objectifs étudiés (Makespan, énergie et le coût) ; de ce fait on peut déduire qu'il meilleur et plus performant.

Conclusion

Au cours de cette dernière étape de notre travail, nous avons présenté l'implémentation ainsi que les outils utilisés pour résoudre le problème d'ordonnancement de tâches dans le cloud tout en prenant en considération différents objectifs et contraintes.

Conclusion générale

Pour ce projet de fin d'étude nous nous sommes intéressés à l'une des technologies les plus importantes au cours de ces dernières années dans le monde informatique et qui est le Cloud Computing.

Nous avons commencé notre projet par nous familiariser à ce concept récent, qui commence à prendre une ampleur considérable dans le monde de l'informatique avec ses plusieurs avantages qu'il peut apporter aux grandes entreprises.

La demande de ce service commence à devenir de plus en plus importante. Pour cela l'ordonnancement des tâches, est un aspect très important dans l'efficacité du fonctionnement du Cloud.

Notre travail avait pour objectif de résoudre un problème d'ordonnancement de tâches dans le cloud computing à l'aide de la métaheuristique PSO, dont le but est de satisfaire les exigences des fournisseurs du Cloud, tout en prenant en considération le makespan, l'énergie et le coût monétaire.

Pour les tests, nous avons utilisé le simulateur CloudSim implémenté sous Eclipse, puis nous avons effectué une étude comparative avec d'autres algorithmes, pour montrer et analyser les résultats que peut engendrer chacun d'eux après leurs exécutions.

Pour conclure, nous espérons que ce modeste travail puisse servir dans le futur aux étudiants.

Comme perspectives, on peut améliorer et élargir notre approche, et faire l'équilibrage de charge entre le Cloud et le Fog (informatique en brouillard) qui est une infrastructure intermédiaire entre les objets connectés et le Cloud.

Annexe

1 Langage de programmation java [A.1]

Java est un langage de programmation qui :

- Peut-être décrit en quelques mots : Orienté Objet, Simple, Robuste, Dynamique et Sécurisé, Indépendant de la Plateforme (VM), Semi Compilé/Semi Interprété, Bibliothèque Importante (JDK API).
- Se trouve sous 3 environnements d'exécutions différents
  - Java ME (Micro Edition) pour PDA, téléphone
  - Java SE (Standard Edition) pour desktop
  - Java EE (Entreprise Edition) pour serveur Servlet/JSP/JSTL, Portlet/JSF, JTA/JTS, JDO/EJB JavaMail, etc.
- Est OpenSource depuis novembre 2006 (2008 complètement)
- Toutes les sources sont disponibles : http://www.openjdk.org
- Il assure la portabilité entre différents environnements (machine/OS)

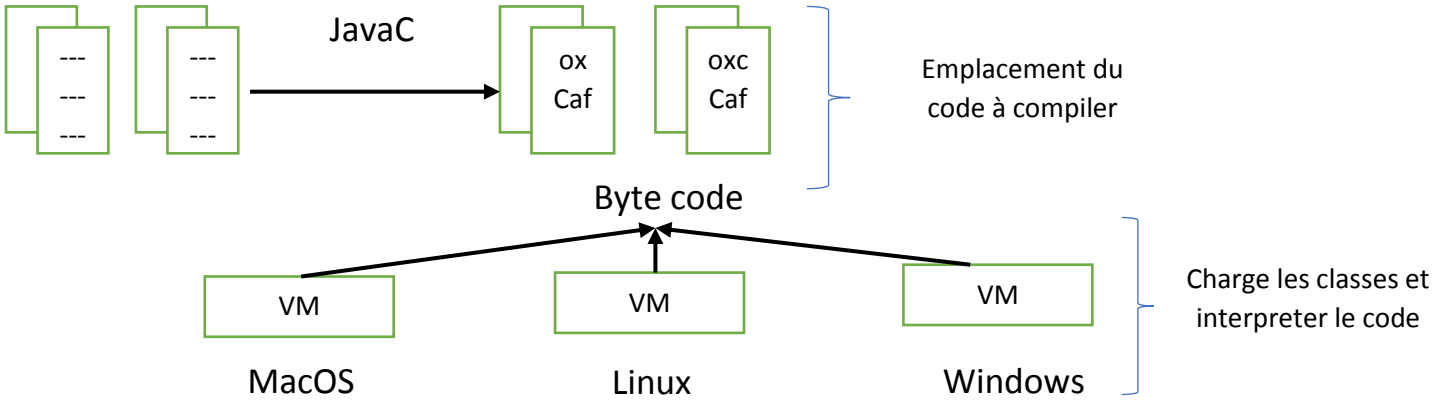


Figure A.1: La portabilité du langage Java. [A.1]

2 L'environnement de développement Eclipse [A.2]

Eclipse est une communauté open-source dont les projets visent à fournir une plateforme de développement ouverte, comprenant des espaces de travail modulaires, des outils et environnements d'exécution, pour construire, déployer et gérer des applications sur tout leur cycle de vie.
Eclipse est surtout (re)connu pour son IDE Java mais son champ d'action est devenu beaucoup plus large. Les projets Eclipse offrent des outils et environnements qui couvrent tout le cycle de développement : outils de modélisation, développement, déploiement, reporting, manipulation de données, tests, etc.

Eclipse IDE est principalement écrit en Java (à l'aide de la bibliothèque graphique SWT, d'IBM), et ce langage, grâce à des bibliothèques spécifiques, est également utilisé pour écrire des extensions.

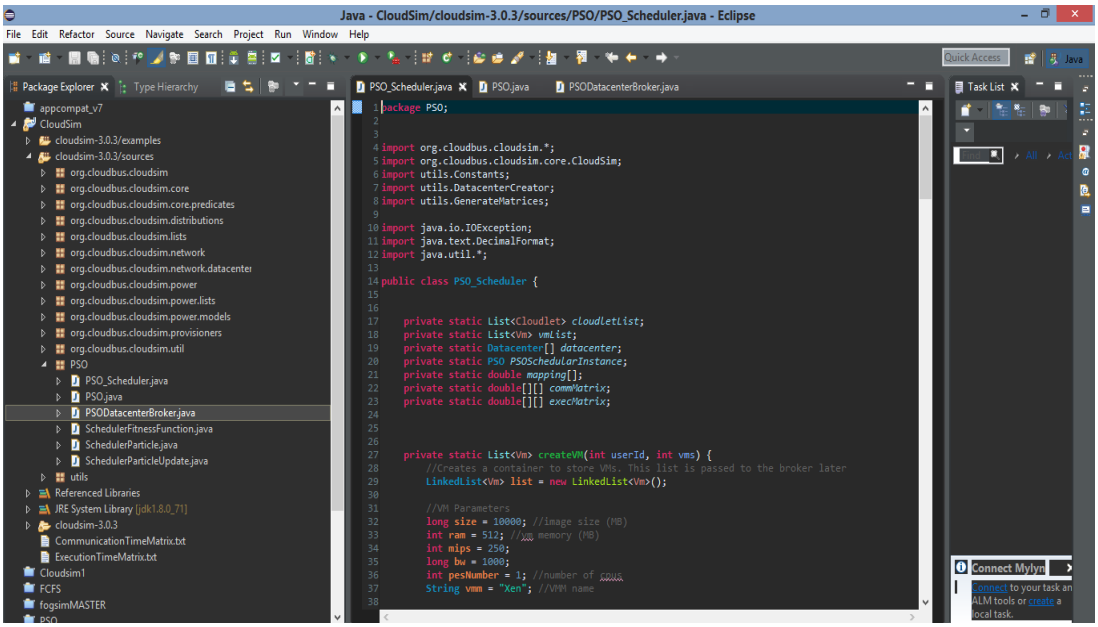


Figure A. 2 : Environnement de développement éclipse.

Il est téléchargeable de son site officiel https://www.eclipse.org/downloads/, et disponible sous Windows, Linux et MacOs. Il est puissant et compatible avec toutes les nouvelles technologies Java (Java EE, Bases de données UML, XML, etc.)

3 Le simulateur CloudSim

3.1 Définition

CloudSim est une infrastructure logicielle prenant en charge plusieurs fonctionnalités essentielles du cloud, telles que la file d'attente des tâches, le traitement des événements, la création d'entités du cloud, la communication entre entités, la mise en œuvre de règles de courtier, etc. ; il permet de :

- Testez les services d'application dans un environnement reproductible et contrôlable.
- Vérifier la performance avant d'utiliser les vrais nuages.
- Réglez les goulots d'étranglement du système avant de déployer des applications dans un nuage réel.

- Testez différents scénarios de répartition de la charge de travail et de performances des ressources sur une infrastructure simulée afin de développer et de tester des techniques de provisioning d'applications adaptatives.

3.2 Les fonctionnalités principales de CloudSim

Cloudsim offre pas mal de fonctionnalités dont on peut citer :

1. Prise en charge de la modélisation et de la simulation d'environnements informatiques à grande échelle.
2. Une plate-forme autonome pour la modélisation des clouds, des courtiers de services, des stratégies de provisioning et d'allocation.
3. Prise en charge de la simulation des connexions réseau entre les éléments système simulés.
4. Installation de simulation d'environnement de nuage fédéré, qui interconnecte des ressources de domaines privés et publics.
5. Disponibilité d'un moteur de virtualisation facilitant la création et la gestion de plusieurs services virtuels indépendants et Co-hébergés sur un nœud de centre de données.
6. Flexibilité permettant de basculer entre l'allocation d'espace partagé et l'attribution de temps partagé des cœurs de traitement aux services virtualisés.

3.3 L'architecture du CloudSim [A.3]

La structure logicielle de CloudSim et ses composants est représentée par une architecture en couche comme s'est illustré sur la figure **.**

Au niveau le plus bas se trouve le moteur de simulation des événements discrets SimJava, qui implémente les fonctionnalités de base requises, tels que les files d'attente, les traitements des événements, la création des entités du système Cloud (services, hôtes, Datacenter, Broker, VM…), la communication entre les composants et la gestion d'horloge de simulation.

CloudSim supporte la modélisation et la simulation de l'environnement de Datacenter basé sur le Cloud, tel que des interfaces de gestion dédiées aux VMs, la mémoire, le stockage et la bande passante. Elle 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 (nombre de machines, leurs spécifications), les politiques d'ordonnancement de Broker, applications (nombre de tâches et leurs besoins), VM, nombre d'utilisateurs. Elle contient les informations de base nécessaires au bon fonctionnement des hôtes et des applications comme le nombre de machines, leurs spécifications, le choix des stratégies d'ordonnancement du broker, etc.

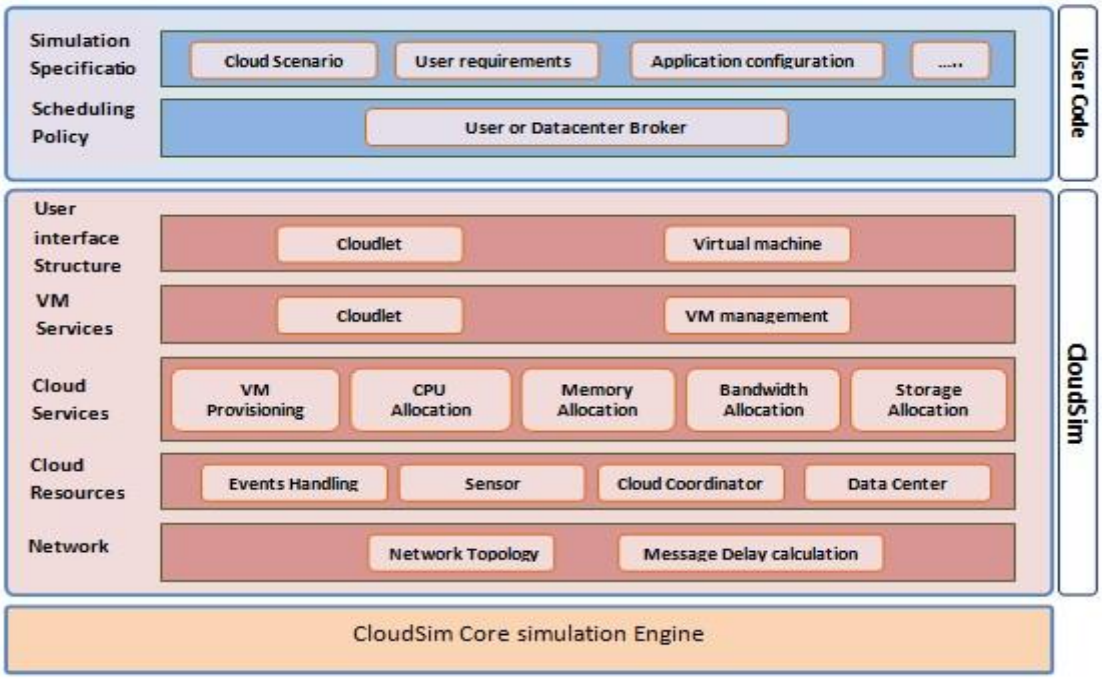


Figure A.3: Architecture du CloudSim.

3.3.1 Les différentes classes du CloudSim

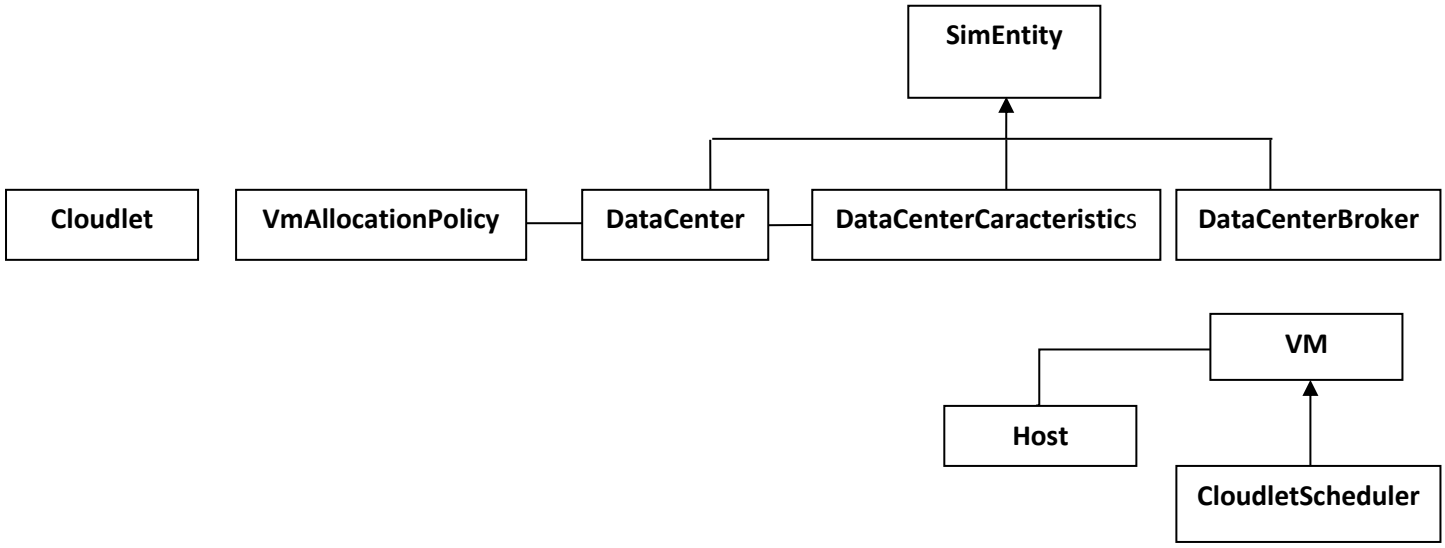


Figure A.4: les différentes classes du CloudSim.

Annexe

- **SimEntity**

Il s'agit d'une classe abstraite, elle représente l'entité de simulation qui est capable d'envoyer des messages à d'autres entités et de gérer les messages reçus ainsi les évènements.
Toutes les entités doivent étendre cette classe et redéfinir ses trois méthodes : **StartEntity**(), **processeEvent**() et **shutdownEntity().** Ces méthodes définissent les actions pour l'initialisation, le traitement des événements et la destruction de l'entité.

- **Cloudlet**

C'est une classe qui représente une tâche. Elle modélise les services d'application du Cloud (comme la livraison, réseaux sociaux et les sites d'affaires). Cette classe peut aussi être étendue pour supporter la modélisation des performances et d'autres paramètres de composition pour les applications telles que les transactions dans les applications orientées bases de données (Oracle, SQL).

- **Datacenter**

Cette classe modélise l'infrastructure du noyau (matériel, logiciel) offert par des fournisseurs de service dans un environnement de Cloud Computing. Il encapsule un ensemble de machines de calcul qui peuvent être homogènes ou hétérogènes en ce qui concerne leur configuration de ressources (mémoire, noyau, capacité et stockage). En outre, chaque composant du Datacenter instancie un composant généralisé d'approvisionnement en ressources qui implémente un ensemble de politiques d'allocation de bande passante, de mémoire et des dispositifs de stockage.

- **DatacenterBroker**

Cette classe modélise le courtier, qui est responsable de la médiation entre les utilisateurs et les prestataires de service ; selon les conditions de QoS des utilisateurs il déploie les tâches de service à travers les Clouds, le Broker agisse au nom des utilisateurs, identifie les prestataires de service appropriés du cloud par le service d'information du cloud CIS (cloud information services) en négociant avec eux pour une allocation des ressources qui répond aux besoins des utilisateurs par le respect des QOS demandées.

- **DatacenterCaracteristic**

C'est la classe qui contient les informations sur la configuration des ressources des Datacenters. Parmi eux le système d'exploitation, la liste des hôtes, le coût par seconde d'utilisation des ressources, etc.

- **Hôst**

Cette classe représente un serveur informatique physique dans un Cloud. L'host exécute des actions liées à la gestion des machines virtuelles et a une politique définie pour l'approvisionnement mémoire et bande passante, ainsi que d'une politique de répartition des PE (Processeur Élément) à des machines virtuelles. Un hôte est associé à un Datacenter. Il peut héberger un ou plusieurs VMs.

- **VM**

Cette classe modélise une instance de machine virtuelle, qui est gérée et hébergée pendant son cycle de vie par le composant Cloud Hôst. Chaque composant de VM a accès à un composant qui stocke les caractéristiques liées à elle telles que : l'accès mémoire, le processeur, la capacité de stockage et les politiques de provisionnement interne de la machine virtuelle.

- **VMAllocationPolicy**

C'est une classe abstraire qui représente la politique de provisionnement des hôtes aux machines virtuelles dans un Datacenter.

3.4 Installation et implémentation du CloudSim sous Eclipse

3.4.1 Procédure d'installation

Pour l'installation du CloudSim au niveau de l'IDE Eclipse,on a qu'à suivre les étapes suivantes :

Annexe

Etape1 : Télécharger les fichiers suivants au préalable :

- CloudSim 3.0.3 toolkit : https://github.com/Cloudslab/cloudsim/releases/tag/cloudsim-3.0.3

- CloudSim :Mathshttp://redrockdigimark.com/apachemirror//commons/math/binaries/commons-math3-3.6.1-bin.zip

Etape 2 : Décompresser les deux fichiers avant de commencer l'installation, et on aura les fichiers illustrés dans la figure qui suit :

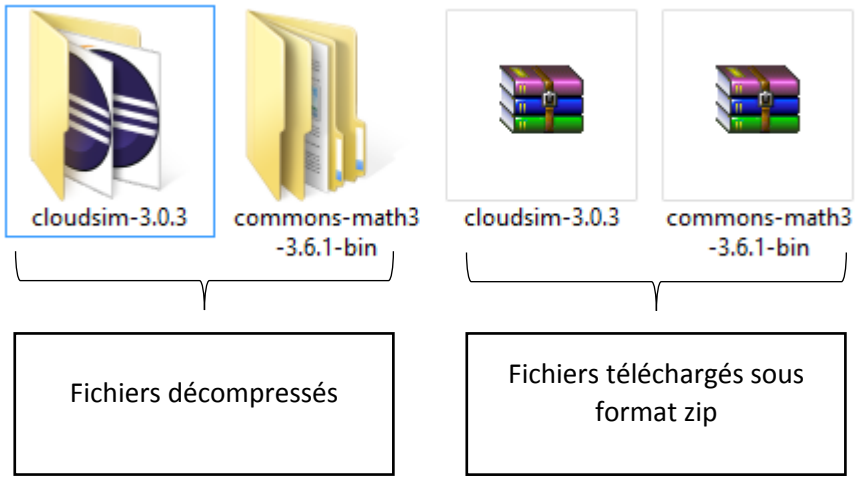


Figure A.5: les fichiers nécessaires pour l'installation de CloudSim.

Étape 3 : Ouvrez Eclipse Java IDE et cliquez sur **File** > **New**> ; une boîte de dialogue sera ouverte choisissez « **Java Projet** »

Annexe

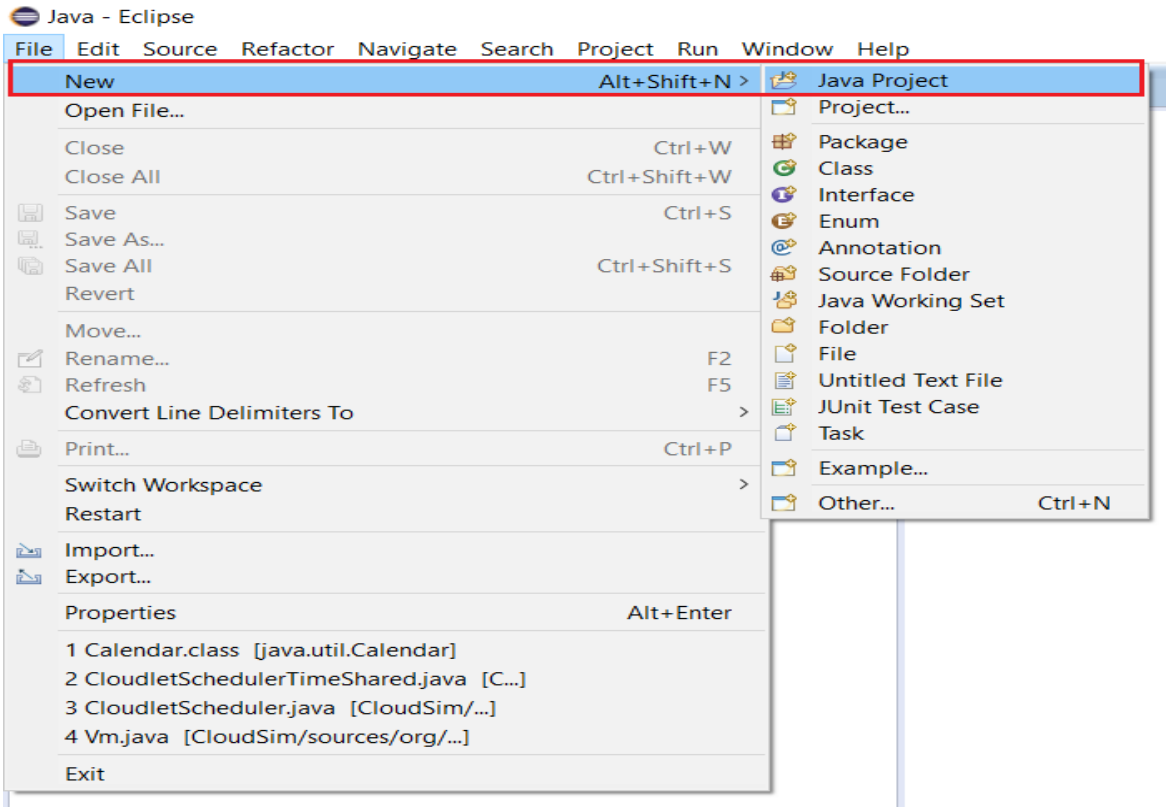


Figure A.6: Création d'un nouveau projet java.

Etape 4 : Dans la boîte de dialogue "**New Java Project** ", entrez " **projet Name**" désélectionnez "**Use default Location**" puis cliquez sur " **Browse** " et parcourez le chemin d'accès auquel vous avez extrait le dossier "**Cloudsim-3.0.3**" puis cliquez sur **NEXT**.

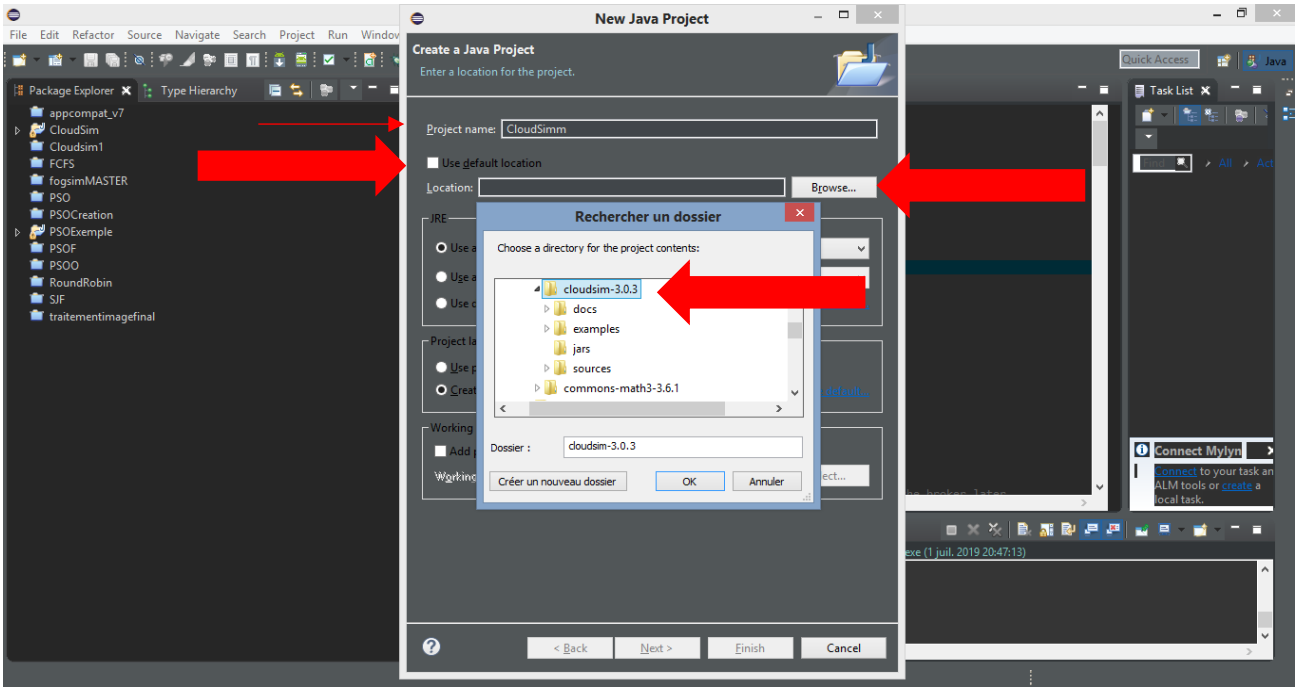


Figure A.7: Importation du fichier d'installation du CloudSim.

Annexe

Étape 5 : Une fois que vous cliquez sur **Next**, allez dans l'onglet "**Librairies**" pour ajouter les librairies « **commons-math3-3.6.1.jar** », nous devons cliquer sur "**Add external JARs**", puis parcourir le chemin sur lequel vous avez téléchargé et décompressé le fichier « **commun-math3-3.6.1.jar** ». Ajoutez-les à la liste en cliquant sur Ouvrir.

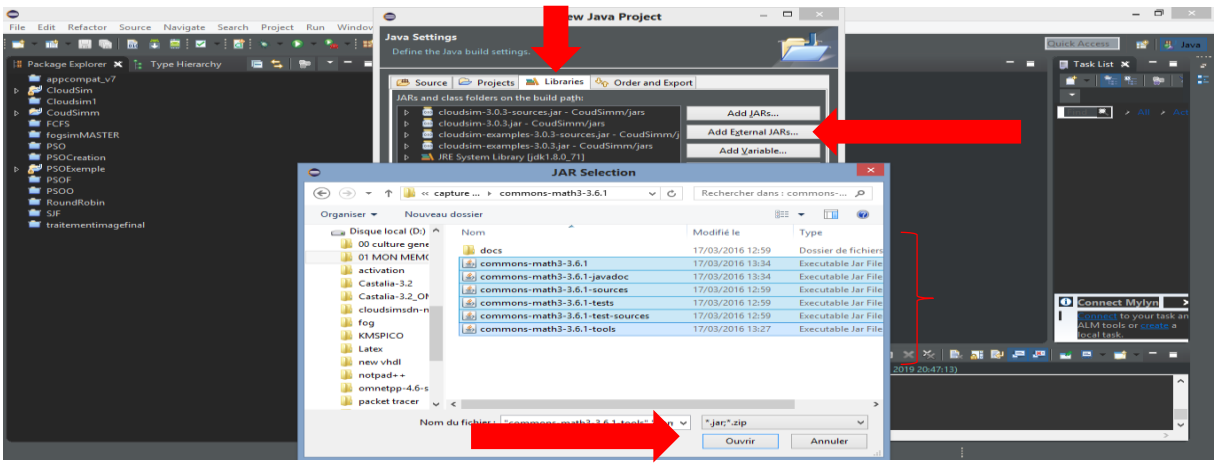


Figure A.8: importation des Librairies propores à CloudSim.

Enfin achever l'installation en cliquant sur le bouton « **Finish** » et clôturer l'installation.

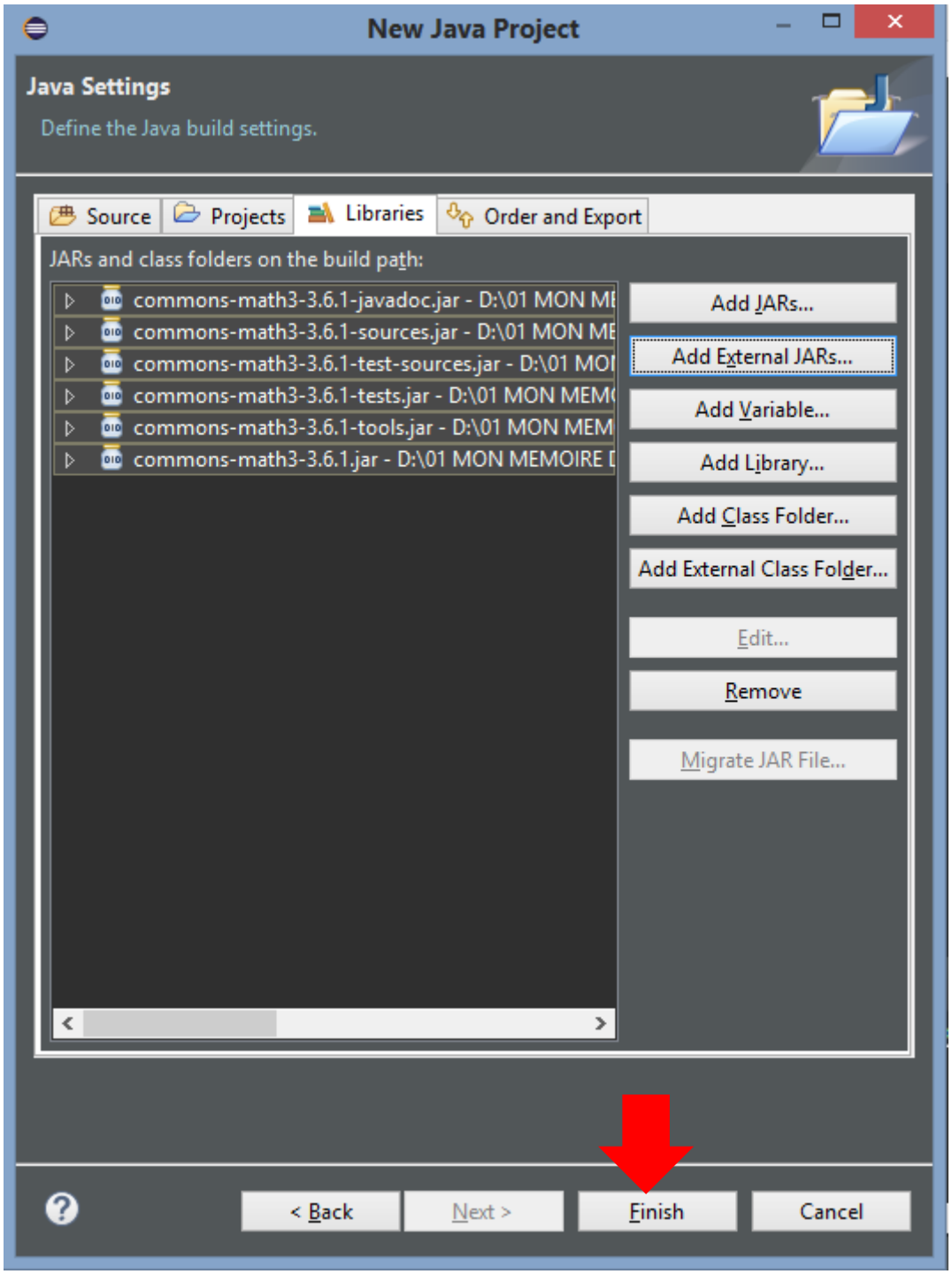


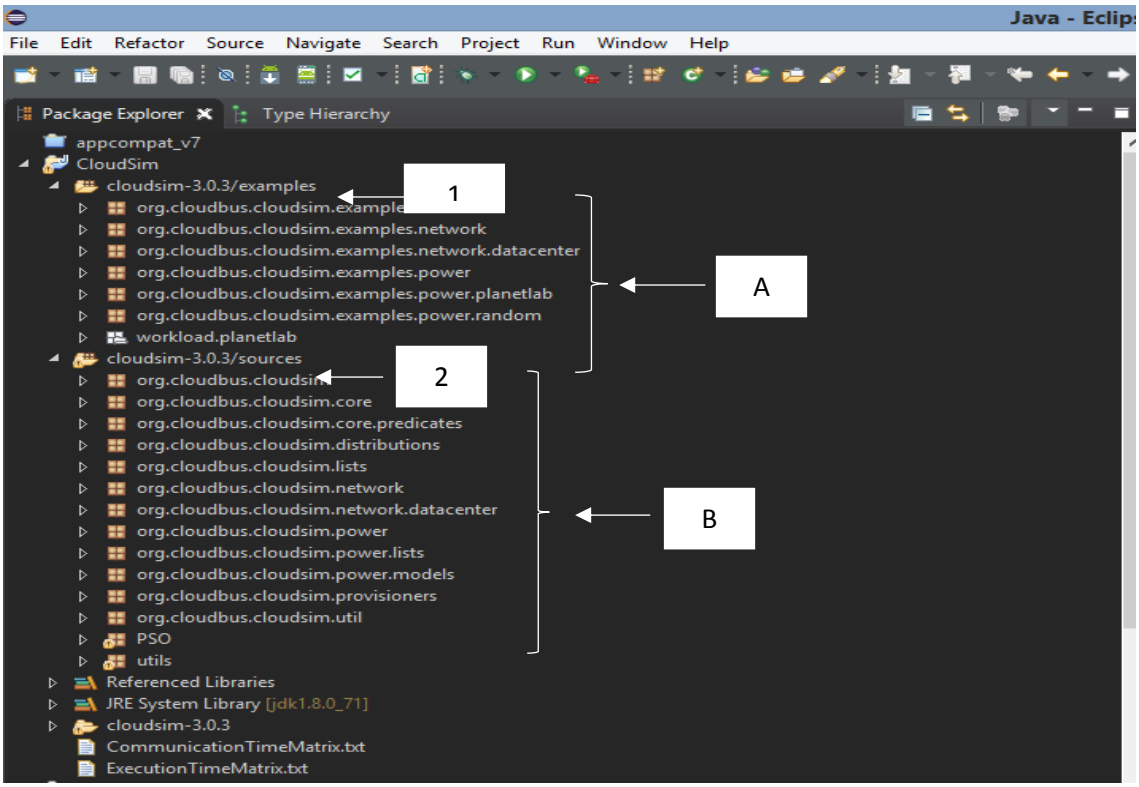
Figure A.9: fin de l'installation du CloudSim.

Le projet commencera à construire (compilation) automatiquement, ce qui peut prendre de 2 à 10 minutes selon la configuration de votre machine. Une fois le processus de construction est terminé, CloudSim est maintenant installé sous eclipse et vous pouvez commencer à travailler avec n'importe quel exemple CloudSim fourni dans le dossier '/ examples', ou bien implementre

d'autre algorithme comme pour d'autre simulation comme on l'a fait dans cas et qui sera l'etape suivante à expliquer.

3.5 Présentation des différents packages du CloudSim.

CloudSim est composé de deux packages principale : CloudSim-3.0.3/Examples et le package CloudSim-3.0.3/sources et chaque packages contient à son tour des sous packages comme le montre la figure suivante



- 1 : le package CloudSim-3.0.3/Examples
- 2 : le package CloudSim-3.0.3/sources.
- A : l'ensemble des sous packages du package 1
- B : l'ensemble des sous packages du package2

Figure A.10: les différents packages du CloudSim.

Le package **Examples** représente un ensemble d'exemple implémentés sous Cloudsim pour réaliser des tests et effectuer différentes simulations.

Pour le deuxième package sources qui est le plus important contient différents sous packages de base qui seront utilisés et appelés à chaque implémentation au niveau du CloudSim et nous citons quelques sous package :

- Le package **org.cloudbus.cludsim** est un package qui contient l'ensemble des classes de base du cloud sui sont par exemple les classes : coudlet, DataCenter, Host CloudletScheduler…etc comme le montre la figure suivante

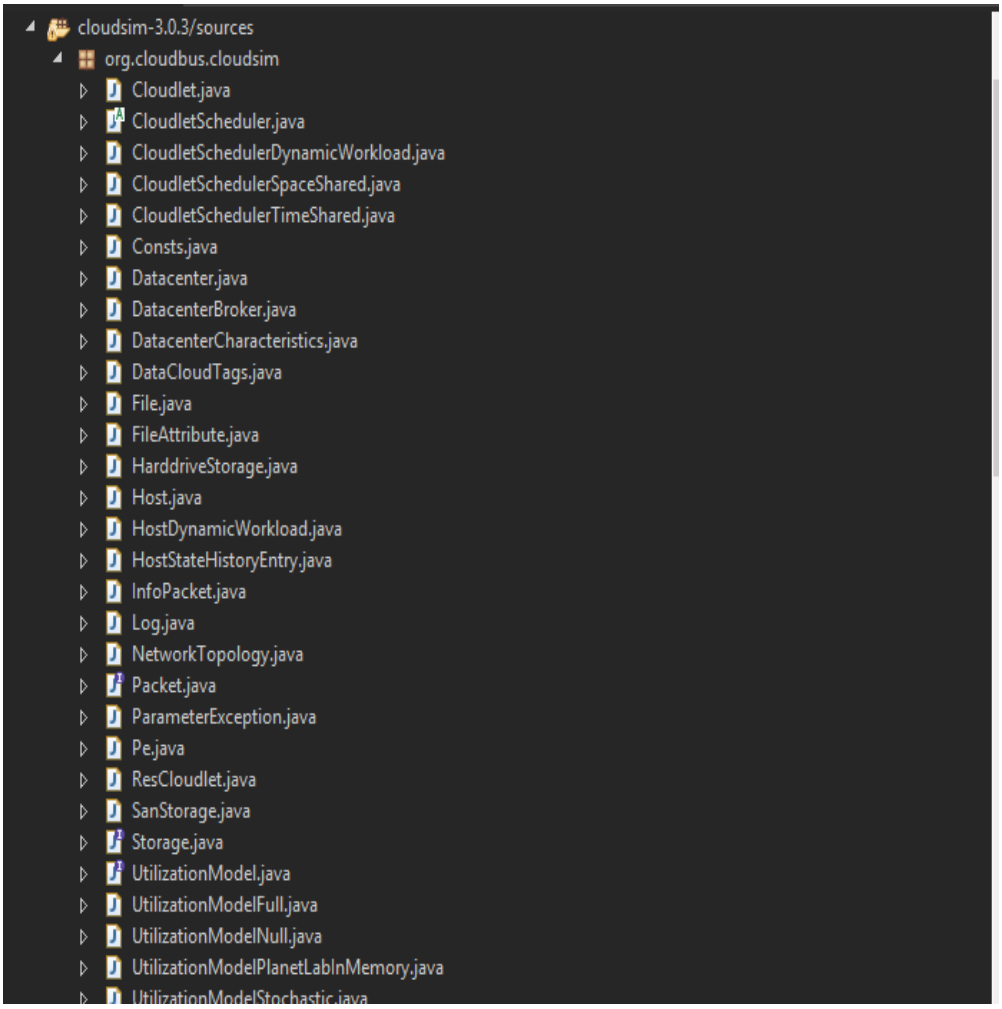


Figure A.11 : Les classes du package org.cloudbus.cloudsim.

- Le package **org.cloudbus.cloudsim.lists** : contient les différentes classes des listes des éléments du cloud qui sont la liste des cloudlets, la liste des Hosts, la liste des Vms…etc

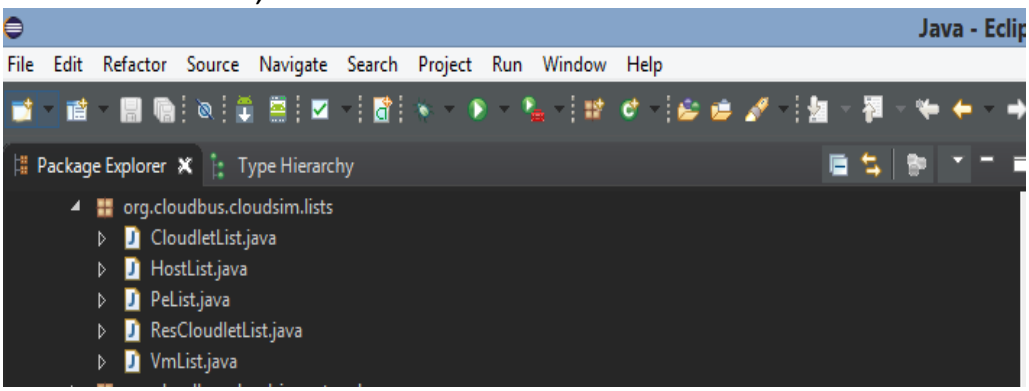


Figure A.12 : Les classes du package org.cloudbus.cloudsim.lists.

- Le package **org.cloudbus.cloudsim.provisioners** : contient les différentes classes nécessaires pour la réservation de la bande passante , la Ram et les Périphériques edges(PE)

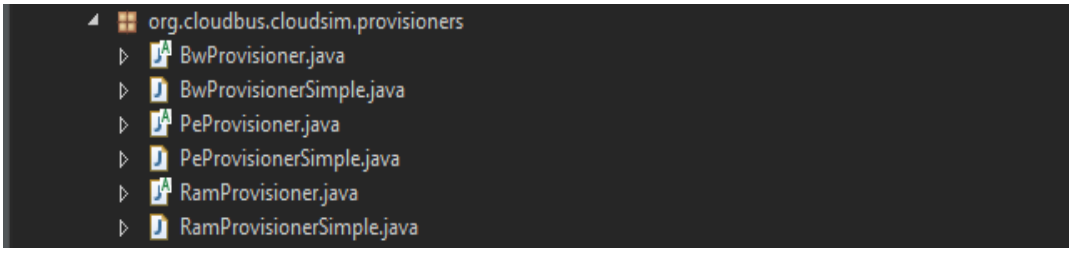


Figure A.13 : Les classes du package org.cloudbus.cloudsim.provisioners.

4 Implémentation de l'algorithme PSO

L'algorithme PSO est composé de deux packages principales :

- **Package PSO** : qui contient les différentes classes du PSO
- **Package utils** : qui contient les différentes classes des éléments utilisés comme les constantes, la création des DC et la génération des matrices qui seront utilisées pour les différents calculs, comme s'est illustré dans la figure suivante :

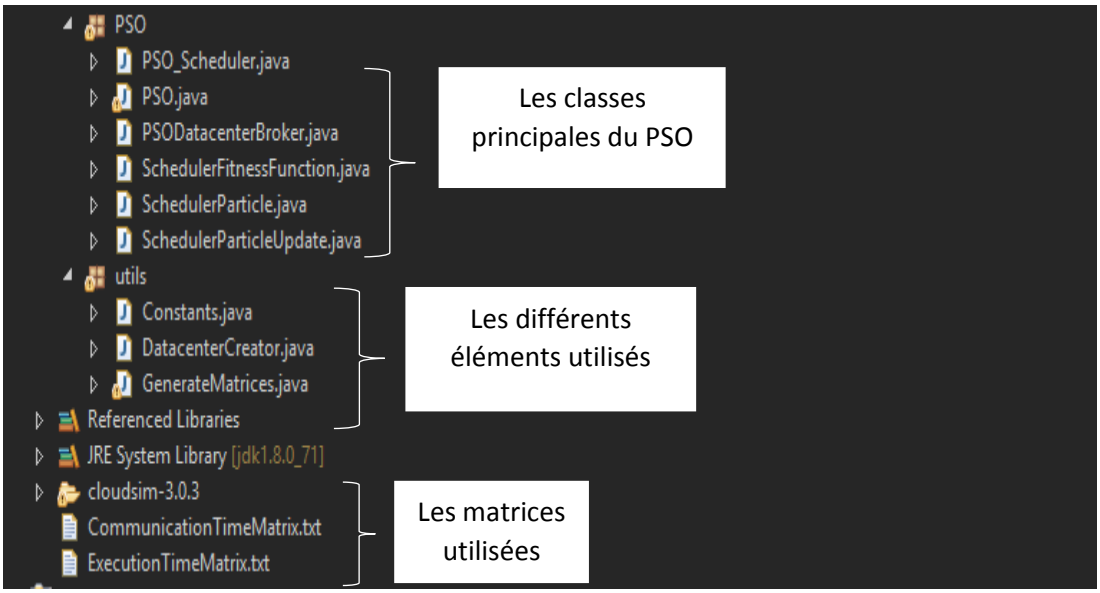


Figure A.14 : Les composant de PSO.

4.1 Les différentes classes du PSO

4.1.1 Le package PSO :

- **PSO_Scheduler :**

PSO_Scheduler.java

```java
package PSO;

import org.cloudbus.cloudsim.*;
import org.cloudbus.cloudsim.core.CloudSim;
import utils.Constants;
import utils.DatacenterCreator;
import utils.GenerateMatrices;
import java.io.IOException;
import java.util.*;

public class PSO_Scheduler {
    private static List<Cloudlet> cloudletList;
    private static List<Vm> vmlist;
    private static Datacenter[] datacenter;
    private static PSO PSOSchedularInstance;
    private static double mapping[];
    private static double[][] commMatrix;
    private static double[][] execMatrix;
    private static List<Vm> createVM(int userId, int vms) {
        //Creation d'un conteneaire pour stocker les VMs************************
        LinkedList<Vm> list = new LinkedList<Vm>();
        //parametres des VM
        long size = 10000; //image size (MB)
        int ram = 512; //vm memory (MB)
        int mips = 250;
        long bw = 1000;
        int pesNumber = 1; //number of cpus
        String vmm = "Xen"; //VMM name
        //creation des VMs
        Vm[] vm = new Vm[vms];
        for (int i = 0; i < vms; i++) {
            vm[i] = new Vm(datacenter[i].getId(), userId, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerSpaceShared());
            list.add(vm[i]);
        }
        return list;
    }
    public static List<Cloudlet> createCloudlet(int userId, int cloudlets, int idShift) {
        LinkedList<Cloudlet> list = new LinkedList<Cloudlet>();
```

Figure A.15: la classe PSO_Scheduler.

Annexe

```java
PSO_Scheduler.java
        LinkedList<Cloudlet> list = new LinkedList<Cloudlet>();
        //parametres des cloudlets**************************************************************************
        long fileSize = 300;
        long outputSize = 300;
        int pesNumber = 1;
        UtilizationModel utilizationModel = new UtilizationModelFull();
        Cloudlet[] cloudlet = new Cloudlet[cloudlets];
        for (int i = 0; i < cloudlets; i++) {
            int dcId = (int) (mapping[i]);
            long length = (long) (1e3 * (commMatrix[i][dcId] + execMatrix[i][dcId]));
            cloudlet[i] = new Cloudlet(idShift + i, length, pesNumber, fileSize, outputSize, utilizationModel, utilizationModel, utilizationModel);
            cloudlet[i].setUserId(userId);
            list.add(cloudlet[i]);
        }
        return list;
    }
    public static void main(String[] args) throws IOException {
        Log.printLine("***Debut***\n");
        new GenerateMatrices();
        commMatrix = GenerateMatrices.getCommMatrix();
        execMatrix = GenerateMatrices.getExecMatrix();
        PSOSchedularInstance = new PSO();
        mapping = PSOSchedularInstance.run();
        try {
            int num_user = 1;   // nombre utilisateur
            Calendar calendar = Calendar.getInstance();
            boolean trace_flag = false;  // mettre trace
            CloudSim.init(num_user, calendar, trace_flag);
            // Creation des DC**************************************************************
            datacenter = new Datacenter[Constants.NO_OF_DATA_CENTERS];
            for (int i = 0; i < Constants.NO_OF_DATA_CENTERS; i++) {
                datacenter[i] = DatacenterCreator.createDatacenter("Datacenter_" + i);
            }
            //creation du broker*************************************************************
            PSODatacenterBroker broker = createBroker("Broker_0");
            int brokerId = broker.getId();
            //Creation  des VMs et cloudlets au nom du meme broker*****************************
            vmlist = createVM(brokerId, Constants.NO_OF_DATA_CENTERS);
```

```java
PSO_Scheduler.java
            vmlist = createVM(brokerId, Constants.NO_OF_DATA_CENTERS);
            cloudletList = createCloudlet(brokerId, Constants.NO_OF_TASKS, 0);
            // ordonnancer nos dcIds au cloudsim dcIds*****************************************************
            HashSet<Integer> dcIds = new HashSet<>();
            HashMap<Integer, Integer> hm = new HashMap<>();
            for (Datacenter dc : datacenter) {
                if (!dcIds.contains(dc.getId()))
                    dcIds.add(dc.getId());
            }
            Iterator<Integer> it = dcIds.iterator();
            for (int i = 0; i < mapping.length; i++) {
                if (hm.containsKey((int) mapping[i])) continue;
                hm.put((int) mapping[i], it.next());
            }
            for (int i = 0; i < mapping.length; i++)
                mapping[i] = hm.containsKey((int) mapping[i]) ? hm.get((int) mapping[i]) : mapping[i];
            broker.submitVmList(vmlist);
            broker.setMapping(mapping);
            broker.submitCloudletList(cloudletList);
            //  Debut de la  simulation*************************************************************
            CloudSim.startSimulation();
            List<Cloudlet> newList = broker.getCloudletReceivedList();
            CloudSim.stopSimulation();
            printCloudletList(newList);
            Log.printLine(PSO_Scheduler.class.getName() + " \n***fin!***");
        } catch (Exception e) {
            e.printStackTrace();
            Log.printLine("simulation avec succes");
        }
    }
    private static PSODatacenterBroker createBroker(String name) throws Exception {
        return new PSODatacenterBroker(name);
    }

    public static void printCloudletList(List<Cloudlet> list) {
PSOSchedularInstance.printBestFitness();
    }
}
```

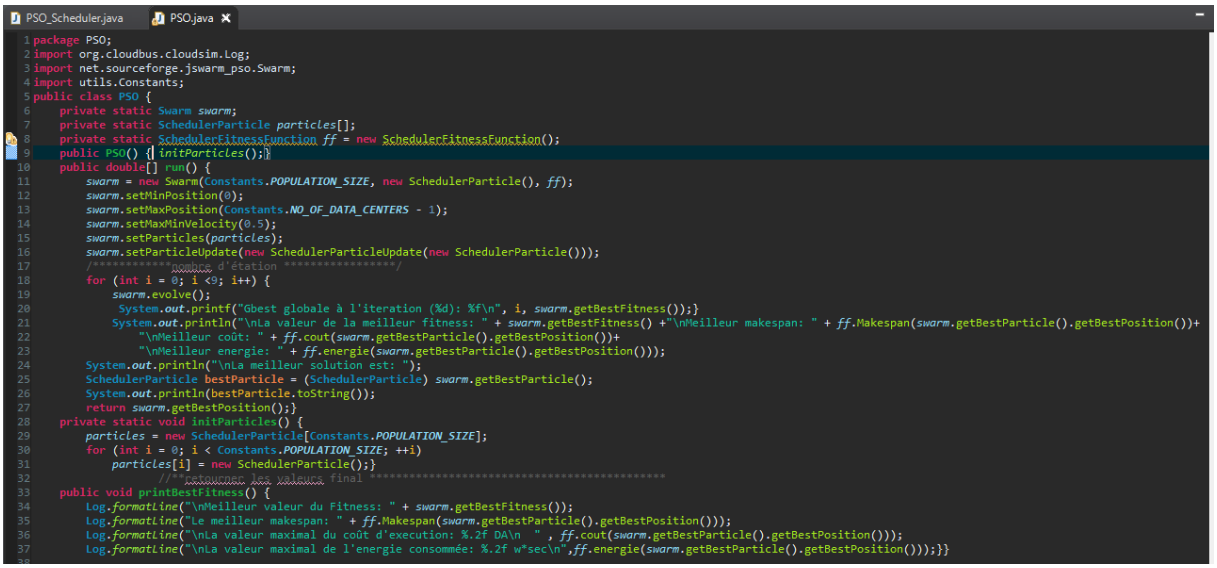
Figure A.16 : La classe PSO_Scheduler (suite).

Cette classe est :

- La classe principale elle contient la méthode main ().
- Elle effectue la création des VMs, des cloudlets, des DCs, le broker, lancer la simulation….

Annexe

- **La classe PSO :**



```
PSO_Scheduler.java   PSO.java
1  package PSO;
2  import org.cloudbus.cloudsim.Log;
3  import net.sourceforge.jswarm_pso.Swarm;
4  import utils.Constants;
5  public class PSO {
6      private static Swarm swarm;
7      private static SchedulerParticle particles[];
8      private static SchedulerFitnessFunction ff = new SchedulerFitnessFunction();
9      public PSO() { initParticles();}
10     public double[] run() {
11         swarm = new Swarm(Constants.POPULATION_SIZE, new SchedulerParticle(), ff);
12         swarm.setMinPosition(0);
13         swarm.setMaxPosition(Constants.NO_OF_DATA_CENTERS - 1);
14         swarm.setMaxMinVelocity(0.5);
15         swarm.setParticles(particles);
16         swarm.setParticleUpdate(new SchedulerParticleUpdate(new SchedulerParticle()));
17         /**********nombre d'étation ***************/
18         for (int i = 0; i <9; i++) {
19             swarm.evolve();
20              System.out.printf("Gbest globale à l'iteration (%d): %f\n", i, swarm.getBestFitness());}
21             System.out.println("\nLa valeur de la meilleur fitness: " + swarm.getBestFitness() +"\nMeilleur makespan: " + ff.Makespan(swarm.getBestParticle().getBestPosition())+
22                 "\nMeilleur coût: " + ff.cout(swarm.getBestParticle().getBestPosition())+
23                 "\nMeilleur energie: " + ff.energie(swarm.getBestParticle().getBestPosition()));
24         System.out.println("\nLa meilleur solution est: ");
25         SchedulerParticle bestParticle = (SchedulerParticle) swarm.getBestParticle();
26         System.out.println(bestParticle.toString());
27         return swarm.getBestPosition();}
28     private static void initParticles() {
29         particles = new SchedulerParticle[Constants.POPULATION_SIZE];
30         for (int i = 0; i < Constants.POPULATION_SIZE; ++i)
31             particles[i] = new SchedulerParticle();}
32                 //**retourner les valeurs final ***************************************
33     public void printBestFitness() {
34         Log.formatLine("\nMeilleur valeur du Fitness: " + swarm.getBestFitness());
35         Log.formatLine("Le meilleur makespan: " + ff.Makespan(swarm.getBestParticle().getBestPosition()));
36         Log.formatLine("\nLa valeur maximal du coût d'execution: %.2f DA\n  " , ff.cout(swarm.getBestParticle().getBestPosition()));
37         Log.formatLine("\nLa valeur maximal de l'energie consommée: %.2f w*sec\n",ff.energie(swarm.getBestParticle().getBestPosition()));}}
```

Figure A.17 : la classe PSO.

Cette classe suit l'évolution au niveau du Swarm (afficher les Gbest à chaque itération, afficher le best Fitness…)

Annexe

➢ La classe PSODatacenterBroker :

```java
package PSO;
import org.cloudbus.cloudsim.*;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.core.CloudSimTags;
import org.cloudbus.cloudsim.core.SimEvent;
import org.cloudbus.cloudsim.lists.VmList;
import java.util.List;
public class PSODatacenterBroker extends DatacenterBroker {
    private double[] mapping;
    PSODatacenterBroker(String name) throws Exception {
        super(name);}
    public void setMapping(double[] mapping) {
        this.mapping = mapping;}
    private List<Cloudlet> assignCloudletsToVms(List<Cloudlet> cloudlist) {
        int idx = 0;
        for (Cloudlet cl : cloudlist) {
            cl.setVmId((int) mapping[idx++]);
        }
        return cloudlist;}
    @Override
    protected void submitCloudlets() {
        List<Cloudlet> tasks = assignCloudletsToVms(getCloudletList());
        int vmIndex = 0;
        for (Cloudlet cloudlet : tasks) {
            Vm vm;
            // if user didn't bind this cloudlet and it has not been executed yet
            if (cloudlet.getVmId() == -1) {
                vm = getVmsCreatedList().get(vmIndex);
            } else { // envoie des vm
                vm = VmList.getById(getVmsCreatedList(), cloudlet.getVmId());
                if (vm == null) { // vm was not created
                    Log.printLine(CloudSim.clock() + ": " + getName() + ": Postponing execution of cloudlet "
                            + cloudlet.getCloudletId() + ": bount VM not available");
                    continue;}}
            Log.printLine(CloudSim.clock() + ": " + getName() + ": Sending cloudlet "
                    + cloudlet.getCloudletId() + " to VM #" + vm.getId());
            cloudlet.setVmId(vm.getId());
            sendNow(getVmsToDatacentersMap().get(vm.getId()), CloudSimTags.CLOUDLET_SUBMIT, cloudlet);
```

PSO_Scheduler.java | PSO.java | PSODatacenterBroker.java

```java
        for (Cloudlet cloudlet : tasks) {
            Vm vm;
            // if user didn't bind this cloudlet and it has not been executed yet
            if (cloudlet.getVmId() == -1) {
                vm = getVmsCreatedList().get(vmIndex);
            } else { // envoie des vm
                vm = VmList.getById(getVmsCreatedList(), cloudlet.getVmId());
                if (vm == null) { // vm was not created
                    Log.printLine(CloudSim.clock() + ": " + getName() + ": Postponing execution of cloudlet "
                            + cloudlet.getCloudletId() + ": bount VM not available");
                    continue;}}
            Log.printLine(CloudSim.clock() + ": " + getName() + ": Sending cloudlet "
                    + cloudlet.getCloudletId() + " to VM #" + vm.getId());
            cloudlet.setVmId(vm.getId());
            sendNow(getVmsToDatacentersMap().get(vm.getId()), CloudSimTags.CLOUDLET_SUBMIT, cloudlet);
            cloudletsSubmitted++;
            vmIndex = (vmIndex + 1) % getVmsCreatedList().size();
            getCloudletSubmittedList().add(cloudlet);}}
    @Override
    protected void processResourceCharacteristics(SimEvent ev) {
        DatacenterCharacteristics characteristics = (DatacenterCharacteristics) ev.getData();
        getDatacenterCharacteristicsList().put(characteristics.getId(), characteristics);
        if (getDatacenterCharacteristicsList().size() == getDatacenterIdsList().size()) {
            distributeRequestsForNewVmsAcrossDatacenters(); } }
    protected void distributeRequestsForNewVmsAcrossDatacenters() {
        int numberOfVmsAllocated = 0;
        int i = 0;
        final List<Integer> availableDatacenters = getDatacenterIdsList();
        for (Vm vm : getVmList()) {
            int datacenterId = availableDatacenters.get(i++ % availableDatacenters.size());
            String datacenterName = CloudSim.getEntityName(datacenterId);
            if (!getVmsToDatacentersMap().containsKey(vm.getId())) {
                Log.printLine(CloudSim.clock() + ": " + getName() + ": Trying to Create VM #" + vm.getId() + " in " + datacenterName);
                sendNow(datacenterId, CloudSimTags.VM_CREATE_ACK, vm);
                numberOfVmsAllocated++;}}
        setVmsRequested(numberOfVmsAllocated);
        setVmsAcks(0); }}
```

Figure A.18 : La classe PSO_DataCenterBroker.

Annexe

➢ La classe SchedulerFitnessFunction

```java
package PSO;
import org.cloudbus.cloudsim.Log;
import net.sourceforge.jswarm_pso.FitnessFunction;
import utils.Constants;
import utils.GenerateMatrices;

public class SchedulerFitnessFunction<A> extends FitnessFunction {
    private static double[][] execMatrix, commMatrix;

    SchedulerFitnessFunction() {
        super(false);
        commMatrix = GenerateMatrices.getCommMatrix();
        execMatrix = GenerateMatrices.getExecMatrix();
    }
    @Override
    public double evaluate(double[] position) {
      double alpha = 0.4;
        double beta=0.3;
        double gama=0.3;
        return alpha* Makespan(position)+(beta)*cout(position)+gama*energie(position);
        }
    /************************calculer le makespan*****************************/
    public double Makespan(double[] position) {
        double makespan = 0;
        double[] dcWorkingTime = new double[Constants.NO_OF_DATA_CENTERS];

        Log.printLine("\nles differents resultats de temps d'execution:");
        for (int i = 0; i < Constants.NO_OF_TASKS; i++) {
            int dcId = (int) position[i];
            if(dcWorkingTime[dcId] != 0) --dcWorkingTime[dcId];
            dcWorkingTime[dcId] += execMatrix[i][dcId] + commMatrix[i][dcId];
            makespan = (Math.max(makespan, dcWorkingTime[dcId]));

            Log.printLine("le temps d'execution de la tâche  " + i + " exécutée par le  Datacenter " + dcId + " est:  " +makespan);
        }

        return makespan;
```

```java
    }
    /************************calculer le Cout*****************************/
    public double cout(double[] position) {
        double totalCost = 0;
        Log.printLine("\nLes differents résultats du coût:");
        int dcId_max=(int) position[0];
        double max_cost=(execMatrix[0][dcId_max]*5/100 + commMatrix[0][dcId_max])*8/100;
        for (int i = 0; i < Constants.NO_OF_TASKS; i++) {
            int dcId = (int) position[i];
            totalCost += execMatrix[i][dcId]*8/100 + commMatrix[i][dcId]*5/100;//chaque 100 mi  de communication coute 5DA et chaque 100 mi d'eecution coute 8DA
            Log.printLine("Le coût consommé par la tâche " + i + " exécutée par le DataCenter " + dcId + " est:  " + totalCost);
            if(max_cost<totalCost){
                max_cost=totalCost; }}
        return max_cost;}
    /************************calculer l'energie*****************************/
    public double energie(double[] position) {
        double DCEnergie = 0;
        Log.printLine("les differents résultats de l'énérgie:");
        int dcId_max=(int) position[0];
        double max_Energie=(execMatrix[0][dcId_max] + commMatrix[0][dcId_max])*1000/3600;
        for (int i = 0; i < Constants.NO_OF_TASKS; i++) {
            int dcId = (int) position[i];
            int K=5;
            int C=3;
            DCEnergie += (execMatrix[i][dcId]*K + commMatrix[i][dcId]*C)*1000/3600;//energie consommée
            //Log.printLine(DCEnergie);
            Log.printLine("l'energie consommée par la tâche " + i + " exécuté par le Datacenter " + dcId + " est:  " +DCEnergie );
            if(max_Energie<DCEnergie){
                max_Energie=DCEnergie;}}
        return max_Energie;}}
```

Figure A.19: la classe SchedulerFitnessFunction.

C'est au niveau de la classe SchedulerFitnessFunction qu'on a effectué les différents calculs des objectifs qu'on a pris en compte.

- **La classe SchedulerParticle :**

PSO_Scheduler.java | PSO.java | PSODatacenterBroker.java | SchedulerFitnessFunction.java | SchedulerParticle.java

```java
package PSO;
import net.sourceforge.jswarm_pso.Particle;
import utils.Constants;
import java.util.Random;

public class SchedulerParticle extends Particle {
    SchedulerParticle() {
        super(Constants.NO_OF_TASKS);
        double[] position = new double[Constants.NO_OF_TASKS];
        double[] velocity = new double[Constants.NO_OF_TASKS];

        for (int i = 0; i < Constants.NO_OF_TASKS; i++) {
            Random randObj = new Random();
            position[i] = randObj.nextInt(Constants.NO_OF_DATA_CENTERS);
            velocity[i] = Math.random();
        }
        setPosition(position);
        setVelocity(velocity);
    }
    @Override
    public String toString() {
        String output = "";
        for (int i = 0; i < Constants.NO_OF_DATA_CENTERS; i++) {
            String tasks = "";
            int no_of_tasks = 0;
            for (int j = 0; j < Constants.NO_OF_TASKS; j++) {
                if (i == (int) getPosition()[j]) {
                    tasks += (tasks.isEmpty() ? "" : " ") + j;
                    ++no_of_tasks;
                }
            }
            if (tasks.isEmpty()) output += "aucune tache associé au DataCenter " + i + "\n";
            else
                output += "un nombre de " + no_of_tasks + " est essocié au DtaCenter " + i + " et elles sont " + tasks + "\n";
        }
        return output;}
}
```

Figure A.20: La classe SchedulerParticle.

C’est au niveau de cette classe que les tâches sont affectées au différents DCs

- **La classe SchedulerParticleUpdate :**

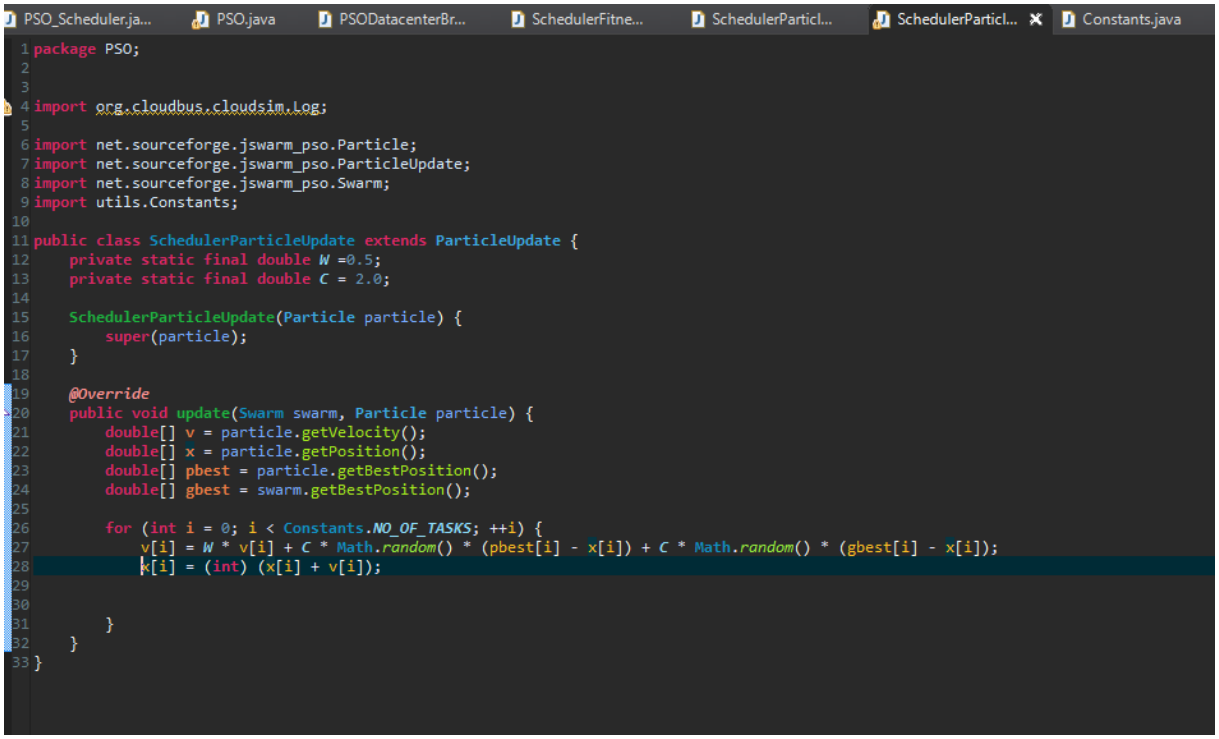


Figure A.21: La classe SchedulerParticleUpdate.

Cette classe effectue des mises à jour au niveau de chaque particule (son vecteur vitesse et position)

4.1.2 Le package utils

- **La classe constants :**

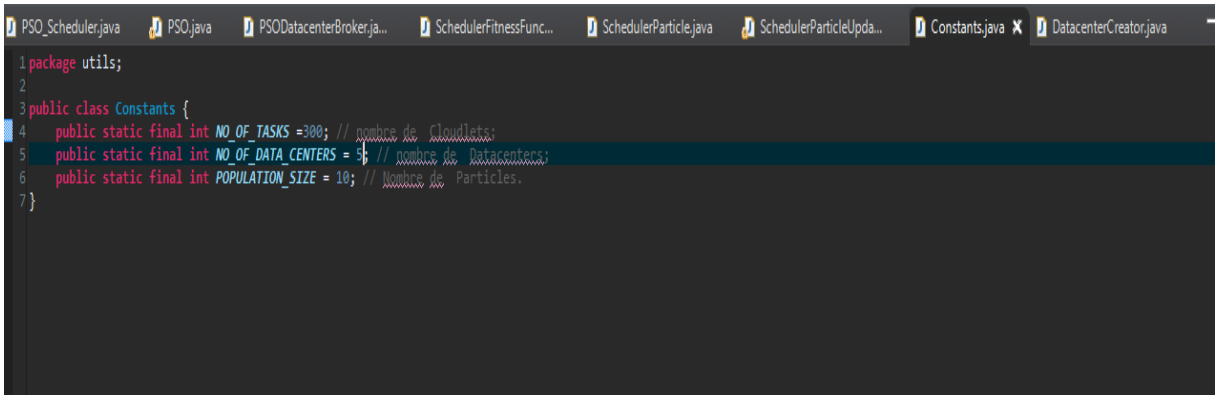


Figure A.22: La classe constants.

Elle contient le nombre de constante qu'on a utilisé dans notre simulation et dans notre cas on a utilisé des cloudlets, des datacenters et la taille de la population.

Annexe

- **La classe DatacenterCreator :**

DatacenterCreator.java

```java
package utils;

import org.cloudbus.cloudsim.*;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

public class DatacenterCreator {

    public static Datacenter createDatacenter(String name) {
        // Here are the steps needed to create a PowerDatacenter:*************************/
        // 1. We need to create a list to store one or more Machines*********************/
        List<Host> hostList = new ArrayList<Host>();
        // 2. A Machine contains one or more PEs or CPUs/Cores. Therefore, should*********/
        //    create a list to store these PEs before creating a Machine.***************/
        List<Pe> peList = new ArrayList<Pe>();
        int mips = 1000;
        // 3. Create PEs and add these into the list.
        peList.add(new Pe(0, new PeProvisionerSimple(mips)));
        //4. Create Hosts with its id and list of PEs and add them to the list of machines
        int hostId = 0;
        int ram = 2048; //host memory (MB)
        long storage = 1000000; //host storage
        int bw = 10000;

        hostList.add(
                new Host(
                        hostId,
                        new RamProvisionerSimple(ram),
                        new BwProvisionerSimple(bw),
                        storage,
                        peList,
                        new VmSchedulerTimeShared(peList)
                )
        ); // This is our first machine



        // 5. Create a DatacenterCharacteristics object that stores the
        //    properties of a data center: architecture, OS, list of
        //    Machines, allocation policy: time- or space-shared, time zone
        //    and its price (G$/Pe time unit).
        String arch = "x86";       // system architecture
        String os = "Linux";          // operating system
        String vmm = "Xen";
        double time_zone = 10.0;         // time zone this resource located
        double cost = 3.0;              // the cost of using processing in this resource
        double costPerMem = 0.05;        // the cost of using memory in this resource
        double costPerStorage = 0.1;    // the cost of using storage in this resource
        double costPerBw = 0.1;            // the cost of using bw in this resource
        LinkedList<Storage> storageList = new LinkedList<Storage>();    //we are not adding SAN devices by now

        DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
                arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);


        // 6. Finally, we need to create a PowerDatacenter object.
        Datacenter datacenter = null;
        try {
            datacenter = new Datacenter(name, characteristics, new VmAllocationPolicySimple(hostList), storageList, 0);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return datacenter;
    }
}
```

Figure A.23: La classe DatacenterCreator.

Cette classe effectue la création des datacenters et leurs caractéristiques.

- **La classe GenerateMatrices**

GenerateMatrices.java

```java
package utils;
import java.io.*;

public class GenerateMatrices {
    private static double[][] commMatrix, execMatrix;
    private File commFile = new File("CommunicationTimeMatrix.txt");
    private File execFile = new File("ExecutionTimeMatrix.txt");

    public GenerateMatrices() throws IOException {
        commMatrix = new double[Constants.NO_OF_TASKS][Constants.NO_OF_DATA_CENTERS];
        execMatrix = new double[Constants.NO_OF_TASKS][Constants.NO_OF_DATA_CENTERS];
        try {
            if (commFile.exists() && execFile.exists()) {
                readCostMatrix();
            } else {
                initCostMatrix();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        }
    private void initCostMatrix() throws IOException {
        System.out.println("Initializing new Matrices...");
        BufferedWriter commBufferedWriter = new BufferedWriter(new FileWriter(commFile));
        BufferedWriter execBufferedWriter = new BufferedWriter(new FileWriter(execFile));

        for (int i = 0; i < Constants.NO_OF_TASKS; i++) {
            for (int j = 0; j < Constants.NO_OF_DATA_CENTERS; j++) {
                commMatrix[i][j] = Math.random() * 600 + 20;
                execMatrix[i][j] = Math.random() * 500 + 10;
                commBufferedWriter.write(String.valueOf(commMatrix[i][j]) + ' ');
                execBufferedWriter.write(String.valueOf(execMatrix[i][j]) + ' ');
            }
            commBufferedWriter.write('\n');
            execBufferedWriter.write('\n');
        }
        commBufferedWriter.close();
        execBufferedWriter.close();
    }
    private void readCostMatrix() throws IOException  {
        System.out.println("Reading the Matrices...");
        BufferedReader commBufferedReader = new BufferedReader(new FileReader(commFile));
        int i = 0, j = 0;
        do {
            String line = commBufferedReader.readLine();
            for (String num : line.split(" ")) {
                commMatrix[i][j++] = new Double(num);
            }
            ++i;
            j = 0;
        } while (commBufferedReader.ready());
        BufferedReader execBufferedReader = new BufferedReader(new FileReader(execFile));
        i = j = 0;
        do {
            String line = execBufferedReader.readLine();
            for (String num : line.split(" ")) {
                execMatrix[i][j++] = new Double(num);
            }
            ++i;
            j = 0;
        } while (execBufferedReader.ready());
    }
    public static double[][] getCommMatrix() {
        return commMatrix;
    }
    public static double[][] getExecMatrix() {
        return execMatrix;
    }
}
```

Figure A.24: La classe GenerateMatrices.

Cette classe permet de générer les matrices d'exécution et les matrices de communication qu'on a utilisé pour nos calculs

4.1.3 Les Matrices générées

Pour la matrice on a donné l'exemple de 5 Datacenter (le nombre de ligne) et 2 tâches (nombre de colonne)

- **La matrice de communication**

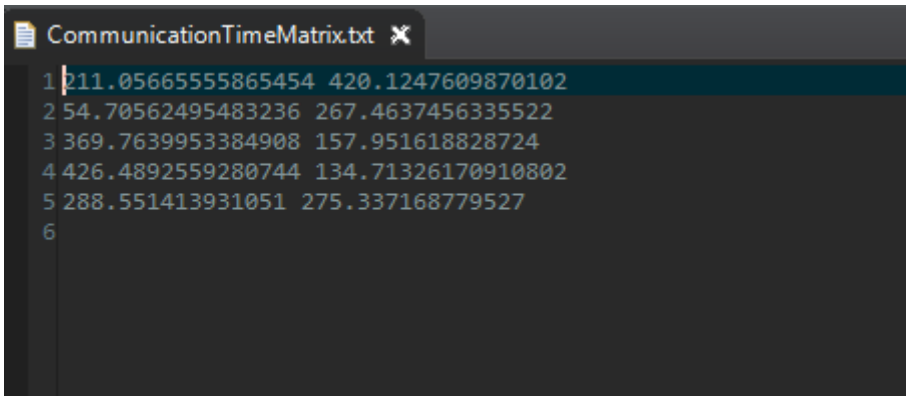


Figure A.25: La Matrice de communication de l'algorithme PSO.

- **La matrice d'exécution**

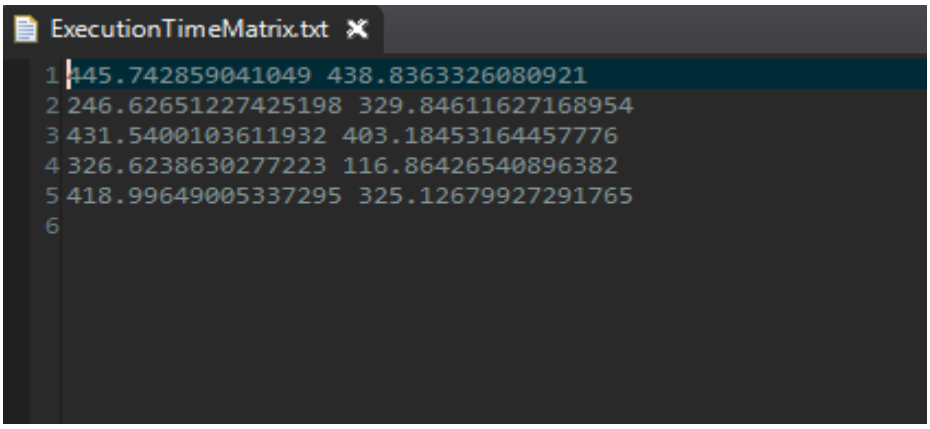


Figure A.26: La Matrice d'exécution de l'algorithme PSO.

Ces matrices seront utilisées pour effectuer les différents calculs au niveau de l'algorithme.

4.2 Résultat de la simulation

Les résultats de notre simulation seront démontrés dans la figure suivante :

```
La valeur de la meilleur fitness: 1757.1340038953986
Meilleur makespan: 1076.965830623874
Meilleur coût: 244.64453143304777
Meilleur energie: 4176.514374053116

La meilleur solution est:
un nombre de 3 tâches est essocié au DataCenter 0 et elles sont 2 4 7
un nombre de 1 tâches est essocié au DataCenter 1 et elles sont 0
un nombre de 2 tâches est essocié au DataCenter 2 et elles sont 6 9
un nombre de 2 tâches est essocié au DataCenter 3 et elles sont 3 5
un nombre de 2 tâches est essocié au DataCenter 4 et elles sont 1 8
```

Figure A.27: Résultat de la simulation.

Nous avons présenté uniquement les captures les plus importantes et l'environnement de notre travail et les différentes implémentations.

Références bibliographiques

[I.1] https://fr.wikipedia.org/wiki/Cloud_computing.

[I.2] https://www.safaribooksonline.com/library/view/cloud-security-and/9780596806453.

[I.3] https://static-course-assets.s3.amazonaws.com/ConnectNet6/fr/index.html. Cours (Cisco académique v 6)

[I.4] C. d. R. d. e. d. Production, Cloud Computing Définitions et Concepts, Enquête et Analyse des Tendances, France, 2010.

[I.5] http://blog.3li.com/les-caracteristiques-essentielles-du-cloud/

[I.6] https://www.tutorialspoint.com/cloud_computing/cloud_computing_overview.htm.

[I.7] https://fr.slideshare.net/Tsubichi/virtualisation-71519850

[I.8] https://www.renaudvenet.com/cloud-computing-avantages-et-inconvenients-2011-01-26.html.

[I.9] https://www.petite-entreprise.net/P-3714-83-G1-le-cloud-computing-les-avantages-et-les-inconvenients.html

[II.1] M. A. ALOULOU, Introduction aux problèmes d'ordonnancement,, LAMSADE université Paris Dauphine, 2005.

[II.2] H. SABBAH, Modélisation et dimensionnement d'une plate-forme hétérogène de service, thèse pour obtenir le grade de docteur, l'université de FERANCHE-COMTE, 2009.

[II.3] F. Teng., Resource allocation ands chelduling models for cloud computing, Ecole Centrale Paris, 2011.

[II.4] R. F. BERMAN, Adaptive Computing on the Grid Using AppLeS. IEEE Transactions on Parallel and Distributed System, 2002.

[II.5] Y. C. -. P. Siarry, Optimisation multiobjectif, BLD Saint Germain, 75240 Paris Cedex 05, 2002.

[II.6] Y. COOREN, Perfectionnement d'un algorithme adaptatif, d'Optimisation par Essaim de particules, thèse pour obtenir le grade de, l'université Paris 12 Val De Marne, 20058.

[II.7] https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-codee396e98d8bf3?fbclid=IwAR3qJWK6LvFZ0MLLpJ9PnTMXN75mLy_eRrYaki174679PZ-3yHHw1YaWmM.

[II.8] D. E. Insaf, optimisation d'ordonnancement et d'allocation de ressources dans le cloud computing, thèse de doctorat, 2016.

Références bibliographiques

[II.9] B. L. e. B. Halima, « hybridation d'euristiques pour le problème d'ordonnancement dans les grilles de calcul, » tizi Ouzou, 2014.

[II.10] Rapport sur système d'exploitation des ordinateurs

[III.1] A.Toumi, Restauration Adaptative d'images par Methodes Intelligentes, thése pour obtenir le grade de Docteur en science spécialité génie électrique, 2013.

[III.2] F.Hepner et a. U.Grenander, A Stochastic non Linear Odel for Coordinated Bird flocks, AAS Publication, Washington, DC, 199.

[III.3] M. Zemzami, N. Elhami, A. Makhloufi, M. Itmi et N. Hmina, Electrical power transmission optimizatin based on a new version of PSO Algorithm.

[III.4] GOURGAND et KEMMOE, Particle Swarm Optimization: A study of Paeticle deplacement for soloying continuous and combinatorial optimization problem, 2009.

[III.5] B. Lyes et B. Halima, Mémoire Master: Hybridation d'heuristique pour le probléme d'ordonnancement dans les grilles de calcul, Tizi_Ouzou, 2014.

[III.6] A. BESTAOUI ABDALLAH, Memoire: Gestion de spectre dans un réseau de radio cognitive en utilisant l'algorithme d'optimisation par essaim de particules, Tlemcen, 2015.

[III.7] R. Eberhat, P.Simpsom et a. R.Dobbins, Computational Intelligence pc Tools. AP Professionnal, 1996.

[III.8] F.Hepner et a. U.Grenander, A Stochastic non Linear Odel for Coordinated Bird flocks, AAS Publication, Washington, DC, 199.

[III.9] m. b. e. a. sene, «l'optimisation par essaim particulaire pour des problemes d'ordonnancement,» 63 173 AUBIERE cedex, 2011.

[IV.1] https://fr.wikipedia.org/wiki/Centre_de_données.

[IV.2] http://dspace.univ-tlemcen.dz/bitstream/112/13456/1/Etude-de-la-consommation-denergie-dans-le-Cloud-Computing..pdf.

[IV.3] M.Pedram, «Energy-efficient datacenter,» 2012.

[IV.4] https://fr.wikipedia.org/wiki/Host.

[IV.5] D. Djouhra, Memoire pour obtention d'un grade de docteur Optimisation des performances des data centers des clouds sous contrainte d'énergie consommée, oran, 2016.

Références bibliographiques

[IV.6] M. Yamina et D. S. Ismahén, mémoire Master: Ordonnancement et replication de données dans le cloud Computing, université DR.Tahar Moulay SAIDA, 2017.

[IV.7] https://www.lemagit.fr/definition/Cloud-Broker.

[IV.8] https://www.scalair.fr/blog/cloud-broker-definition.

[IV.9] Cloudsim Tutorial: Simulation Environnement Introduction, https://www.youtube.com/watch?v=qBFlB5puRRs&feature=share.

[IV.10] S. YASSA, Allocation optimale multi-contraintes des workflows aux ressources d'un environnement cloud computing, Cergy: Université de Cergy-Pontoise Ecole Doctorale Sciences et Ingénierie ETIS (CNRS UMR 8051) EISTI (Ecole Internationale des Sciences de Traitement de l'Information), 2014.

[A.1] R. Forax, Le langage java.

[A.2] https://www.opensourcemacsoftware.org/developpement/eclipse-ide-java-mac.html.

[A.3] https://graal.ens-lyon.fr/~ecaron/m2rts/2015/blogbazm/.