REPUBLIQUE ALGERRIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSTE MOULOUD MAMMERI DE TIZI-OUZOU
FACULTE DE GENIE ELECTRIQUE ET D'INFORMATIQUE
DEPARTEMENT D'INFORMATIQUE

Mémoire de Fin d'Etudes de Master Académique

Domaine : Mathématique et Informatique
Filière : Informatique
Spécialité : Réseaux, Mobilité et Systèmes Embarqués

Présenté par:

- ***Melle Berkani Dyhia***
- ***Melle Megherbi Kahina***

Thème :

Ordonnancement de workflows dans le cloud

Mémoire soutenu publiquement le 25/09/2017 Devant le jury composé de:

- ***Président : Mr Daoui***
- ***Encadreur : Mme Oukfif Karima***
- ***Examinateur : Mme Belkadi Malika***
- ***Examinateur : Mme Aouidjit Rachida***

Promotion 2016/2017

Remerciements

Au terme de ce travail,
Nous tenons en premier lieu à remercier le Bon Dieu pour le courage et la patience qu'il nous a donné afin de mener ce projet à terme.

Nous remercions vivement notre enseignante et promotrice M^me^ OUKFIF d'avoir accepté de nous encadrer, ainsi que pour son aide précieuse, son travail encourageant, et ses conseils judicieux.

Nous remercions également nos parents et tous les enseignants qui ont contribué à notre formation.

Sans oublier de remercier les membres de jury d'avoir accepter de juger notre travail.

Enfin, nos remerciements vont à tous ceux qui, de loin ou de prés ont aidé à l'élaboration de ce mémoire.

DEDICACES

J'ai le plaisir de dédier ce travaille à :

Mon cher père à qui j'invoque

la pitié ;

Ma chère mère qui m'a toujours été

présente pour m'encourager,

me conseiller et me soutenir tout au

long de mes années d'études que

dieu la garde pour moi ;

Mes chers frères Ali et Mokrane

que j'aime énormément ;

Ma meilleure tante Sabrina ;

Ma binôme Dyhia,

à qui je souhaite la réussite ;

Toutes les familles

« Megherbi »

et « Oulmas » ;

Tous mes amis (es)

sans exception ;

Et à tous ceux

qui m'aiment ;

Kahina qui vous aime ;

DEDICACES

Je dédie ce modeste travail

A ceux qui ont éclairé ma vie,

ma très chère mère et mon cher père

pour leur aide et leur soutien tout au long de mes études,

et qui ont fait de moi ce que je suis aujourd'hui.

A mes grands parents paternels et maternels.

A tous mes oncles et mes tentes.

A mes sœurs Lylia, Mellissa.

A mon frère Idir.

A ma binôme Kahina et à toute sa famille.

A tous mes ami(e)s.

A tous mes cousins et mes cousines.

Dyhia

Liste des figures

Figure I.1 : le cloud computing……...3
Figure I.2 : Modèles de services Cloud……..6
Figure I.3 : Les différents types d'un Cloud……..…9
Figure I.4 : Modèle de référence des systèmes de gestion de workflow (WfMC, 95).....20
Figure I.5 : Les algorithmes d'ordonnancement de tâches…….....22
Figure I.6 : Structure d'un DAG……...23
Figure II.1 : Espace décisionnel et espace objectif d'un MOP (cas de deux variables de décision X1 et X2 et de deux fonctions objectif f1 et f2)……..31
Figure II.2 : Relation de dominance (cas de deux objectifs à minimiser)…….32
Figure II.3 : Les méthodes de résolution d'un problème d'optimisation……..36
Figure II.4: Principes généraux d'une métaheuristique à base de: (a) solution unique, (b) population…….…...38
Figure II.5 : Principaux codages pour des problèmes d'optimisation……...40
Figure II.6.a : Rang dominance……43
Figure II.6.b : Compte dominance……43
Figure. II.6.c : Profondeur dominance……43
Figure II.7 : Illustration de la convergence et de la diversité en optimisation multi-objectif……..44
Figure II.8 : Squelette d'un algorithme évolutionnaire……...47
Figure III.1 : Logo Eclipse……53
Figure III.2: L'interface principale d'Eclipse……..54
Figure III.3 : Téléchargement du paquet tar.gz……...56
Figure III.4 : La structure du code source de JMetal……..57
Figure III.5 : La structure de packages JMetal……...58
Figure III.6 : Le diagramme de classe de JMetal……60
Figure III.7 : Exemple d'un fichier d'entrée……62
Figure III.8 : Calculer le rang des tâches……63
Figure III.9 : Exemple d'exécution de l'algorithme HEFT……64
Figure III.10 : Un exemple de workflow et des ressources…….…67
Figure III.11 : Le graphe des makspans par rapport aux nombres de tâches.……71
Figure III.12 : Le graphe des coûts par rapport aux nombres de tâches…….……72
Figure III.13 : Le graphe des makspans par rapport aux nombres de VMs…….…...73
Figure III.14 : Le graphe des coûts par rapport aux nombres de VMs……74

Liste des figures

Figure III.15 : Résultats d'exécution de l'algorithme HEFT pour 10 tâches……75
Figure III.16 : Résultats d'exécution de l'algorithme HEFT pour 20 tâches……76
Figure III.17 : Résultats d'exécution de l'algorithme HEFT pour 40 tâches……76
Figure III.18 : Résultats d'exécution de l'algorithme HEFT pour 60 tâches……....77
Figure III.19 : Graphe pour la Comparaison de makspan ……77

Liste des tableaux

Tableau I.1 : Les avantages et les inconvénients des services cloud ……..….6

Tableau I.2 : Les techniques de provisionnement des ressource……17

Tableau I.3 : Classification des algorithmes d'ordonnancement de workflows dans le cloud…….27

Tableau III.1 : Paramètres de l'algorithme génétique NSGA-II……70

Sommaire

Introduction générale

Chapitre I : environnement cloud et applications workflows .

I.1. Introduction …….....1

I.2. Le cloud….....2

I.2.1.Historiqu……...…….2

I.2.2.Définition du cloud...……...…….2

I.2.3.La virtualisation…...……...……..……3

I.2.4.Modèles de service cloud...…….……..4

I.2.5.Critères de classifications des clouds……..7

I.2.6.Types de clouds…...…….…….…7

I.2.7.Caractéristiques fondamentales du cloud.……..…….....10

I.2.8.Exemple de clouds « Amazon EC2 » et « Google App Engine »……....12

I.3. Etat de l'art sur les approches de réservation de ressources dans le cloud…….13

I.3.1. Introduction …….....13

I.3.2.Types de provisionnement des ressources……14

I.3.3. Paramètres pour l'approvisionnement…….....15

I.3.4. Comparaison des techniques de provisionnement des ressources……15

I.4. Workflow……...…….....17

I.4.1.Définition de workflow…….…….…17

I.4.2.Intérêts du cloud pour les workflow……..……18

I.4.3.Système de géstion de workflows pour les clouds……..…….……..19

I.5. Ordonnancement de workflows…….…….…21

I.5.1. Définition…….....21

I.5.2. La dépendance des tâches d'une application…….....21

I.5.2.1. Ordonnancement des tâches indépendantes……22

I.5.2.2.Ordonnancement des tâches dépendantes…….……22

I.5.2.2.1. Définition DAG (graphe acyclique dirigé)……..……...23

I.6. Méta-heuristiques d'ordonnancement de workflows dans le cloud computing 24

I.7. Conclusion …….....28

Sommaire

Chapitre II : Optimisation multi-objectif et méta-heuristique

II.1. Introduction……29

II.2. L'optimisation multi-objectif……30

II.2.1. Définition……30

II.2.2. La multiplicité des solutions……30

II.2.3. Concepts et définitions……30

II.2.3.1. Formulation générale d'un problème d'optimisation multi objectif……31

II.2.3.1.1. Définition……31

II.2.3.2. Notions de dominance……32

II.2.3.3. Optimalité de Pareto……32

II.2.4. Optimisation multi-objectif et aide à la décision……33

II.2.5. Méthodologies de résolution……35

II.2.6. Les métaheuristiques……37

II.2.6.1. Définition……37

II.2.7. Concepts communs à tout type de métaheuristique……39

II.2.8. Concepts communs à tout type de métaheuristique pour le cas multi objectif……41

II.2.9. Métaheuristiques à base de population de solutions……45

II.2.9.1. Les algorithmes évolutionnaires ……45

II.2.9.1.1. Les algorithmes génétiques (GA)……45

II.3. Conclusion……51

Chapitre III : Modélisation et tests

III.1. Introduction……52

III.2. Environnement de développement……53

III.2.1. Environnement Matériel……53

III.2.2. Environnement Logiciel……53

III.2.2.1. Présentation de l'outil de développement……53

III.2.2.2. Présentation du langage de programmation java ……54

III.2.2.3. Présentation du langage de programmation C++……55

Sommaire

III.2.2.4. Présentation du simulateur JMetal……...55
III.2.2.4.1. Installation……55
III.2.2.4.2. Structure d'emballage JMetal……58
III.2.2.4.3. L'architecture de JMetal……59
III.3. Présentation de l'algorithme d'ordonnancement de workflow HEFT……60
III.3.1. Définition……60
III.3.2. Etapes de la méthode HEFT……..61
III.3.3. Modèle du cloud computing……..…63
III.3.4. Modélisation du workflow……64
III.3.5. Les métriques mesurées……65
III.4. NSGA-II pour l'ordonnancement de *workflow*s……67
III.4.1. Etapes de l'algorithme NSGA-II……...67
III.4.2. Calcul de la distance de "Crowding"…….……69
III.4.3. Operateurs génétiques……..…..……...69
III.4.4. Paramètres de l'algorithme génétique NSGA-II…….……70
III.5. Test et résultats……..70
III.5.1. Exécution du programme NSGA-II.....…...70
III.5.2. Makespan et coût en fonction de nombre de tâches ……..……70
III.5.3. Comparaison de makspan……74
III.6. Conclusion……..77

Conclusion générale

Bibliographie

Introduction générale

Le cloud est de plus en plus reconnu comme une nouvelle façon d'utiliser, à la demande, les services de calcul, de stockage et de réseau de manière transparente et efficace. Il présente une technologie prometteuse qui facilite l'exécution des applications scientifiques et commerciales. Il fournit des services flexibles et évolutifs, à la demande des utilisateurs, via un modèle de paiement à l'usage. Ces services sont offerts avec différents niveaux de QoS (qualité de service) afin de répondre aux besoins spécifiques de différents utilisateurs. Ces paramètres de QoS peuvent être définis et proposés par différents contrats de service, SLA (Service Level Agreements). Un SLA spécifie les exigences négociées en termes de ressources, les QoS, les attentes minimales et les obligations qui existent entre les utilisateurs et les fournisseurs de cloud.

Plusieurs applications scientifiques dans de nombreux domaines contiennent généralement de nombreuses tâches contraintes par des relations de précédence. Ces applications sont souvent exprimées sous forme de workflows scientifiques comportant une série d'étapes de traitements (tâches) liées. Récemment, le cloud leur fournit, à la demande, ces infrastructures avec différents niveaux de QoS.

L'ordonnancement de tâches et principalement de workflow est un problème NP-complet qu'on peut résoudre par des méta-heuristiques, parmi elles on distingue principalement les algorithmes évolutionnaires, plus précisément les algorithmes génétiques.

En raison de l'importance des applications de workflows, plusieurs projets de recherche ont été menés pour concevoir des systèmes de gestion de ces derniers et des algorithmes d'ordonnancement appropriés. Pour résoudre le problème d'ordonnancement de workflow dans le cloud nous l'avons modéliser sous JMetal ensuite, nous avons réaliser plusieurs tests.

Ce mémoire est divisé en trois chapitres:
Le **chapitre I** présente les concepts liès aux domaines sur lesquels nous travaillons, à savoir: le cloud et le workflow, Le **chapitre II** s'intitule l'optimisation multi-objectifs et méta-heuristiques et , le **Chapitre III** détaille notre contribution pour l'ordonnancement bi-objectif de workflows sous l'environnement de travail JMetal, avec comme métrique QoS le temps de complétion (makespan) et le coût.

Chapitre I :

Environnement cloud et applications workflows

I.1.Introduction

L'ordonnancement de workflow reste un problème d'actualité, c'est l'action d'assigner des ressources au traitement des tâches. Beaucoup d'approches ont été proposées pour le résoudre dans différentes plates-formes.

Dans ce chapitre nous présentons d'abord la platforme cloud, ensuite nous présentons les approches de réservation de ressources dans le cloud. Enfin nous décrirons un ensemble de définitions de bases sur le concept d'ordonnancement de workflow dans le cloud.

I.2. Le cloud

I.2.1. Historique

L'idée principale du cloud est apparue dans les années 60, où le professeur John McCarthy avait imaginé que les ressources informatiques seront fournies comme des services d'utilité publique. C'est ensuite, vers la fin des années 90, que ce concept a pris de l'importance avec l'avènement du grid computing. Le grid computing est une métaphore exprimant la similarité avec le réseau électrique, dans lequel l'électricité est produite dans de grandes centrales, puis disséminée à travers un réseau jusqu'aux utilisateurs finaux. Le cloud computing n'est véritablement apparu qu'au cours de l'année 2006 avec l'apparition d'Amazon **EC2 (E**lastic **C**ompute cloud). C'est en 2009 que la réelle explosion du cloud survint avec l'arrivée sur le marché de sociétés comme **Google** (Google App Engine), **Microsoft** (Microsoft Azure), **IBM** (IBM Smart Business Service), **Sun** (Sun cloud) et **Canonical Ltd** (Ubuntu Enterprise cloud).

I.2.2. Définition du Cloud

Il existe de très nombreuses définitions du cloud computing, on prend par exemple la définition de **NIST** (**N**ational **I**nstitute of **S**tandards and **T**echnology) :

« Le cloud est un modèle d'accès a des ressources informatique partagées et configurable par exemple (réseaux, serveurs, stockage, applications et services) depuis un accès réseau, à la demande, de manière simple, à partir de n'importe quel type d'appareil et depuis n'importe quel endroit. Ces ressources sont disponibles rapidement et opérationnelles avec un minimum d'efforts et d'interactions avec le fournisseur de services ». **[I.1]**

Pour une définition global on peut dire que le cloud est un modèle informatique qui permet d'accéder, d'une façon transparente et à la demande, à un pool de ressources hétérogènes physiques ou virtualisées (serveurs, stockage, applications et services) à travers le réseau. Donc le client loue les services au lieu de les acheter, contribuant ainsi à transformer les dépenses de capital en dépenses opérationnelles. Il accède au service sans se soucier des exigences et des besoins en termes de plates-formes, de maintenance, du contrôle, de répartition de couts et de gestion du centre de données.

Les ressources sont délivrées sous forme de services reconfigurables et élastiques, à base d'un modèle de paiement à l'usage, dont les garanties sont offertes par le fournisseur via des contrats de niveau de service (**SLA**, **S**ervice **L**evel **A**greement).

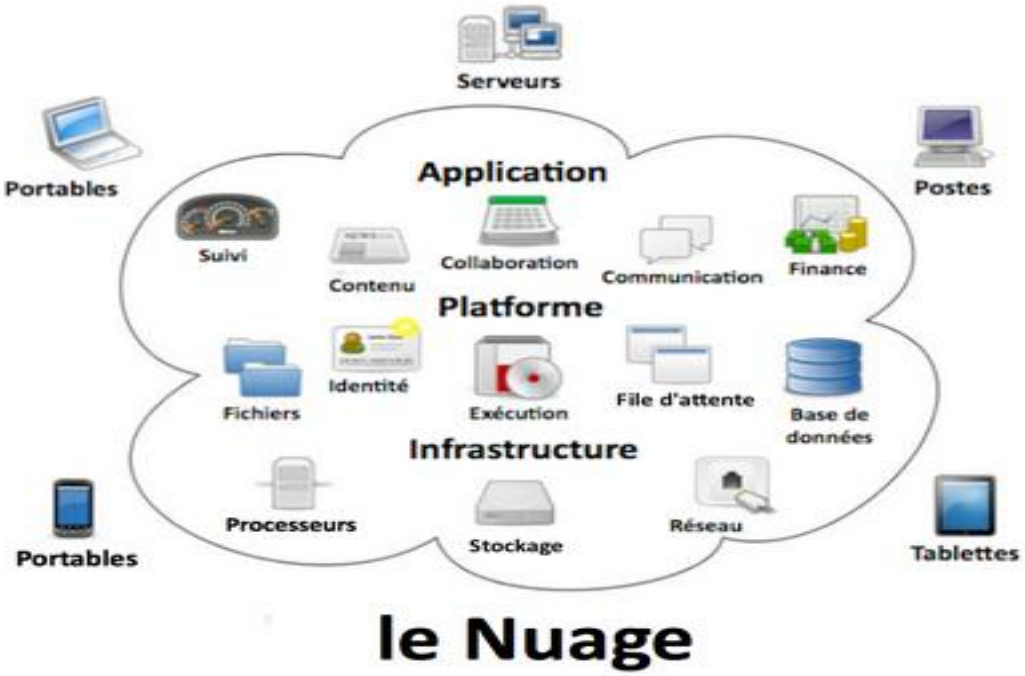


Figure I.1 : Le cloud computing.

I.2.3. La virtualisation

La virtualisation constitue le cœur du cloud computing, elle consiste à une couche logicielle qui permet de faire l'abstraction entre le matériel et le système d'exploitation. Ainsi, plusieurs systèmes peuvent être installés sur une même machine physique.

Elle a été adoptée par l'industrie informatique depuis 2004 et fait maintenant partie des piliers de la production des data centers. **[I.1]**

I.2.4. Modèles de services Cloud

Les fournisseurs de Cloud computing ont recours à de nombreux modèles de services. Les solutions Cloud existantes peuvent combiner différentes approches. Les trois modèles les plus courants **SaaS**, **PaaS**, et **IaaS**, dénommés **SPI** (**S**oftware, **P**latform, **I**nfrastructure), tels que définis par le **NIST**, sont définis ci-après.

I.2.4.1. Logiciel en tant que service (Software as a service (SaaS))

Le client peut avoir recours aux applications du fournisseur, exécutées sur une infrastructure Cloud. Les SaaS proposent les logiciels opérationnels, prêts à l'emploi, sans passer par une étape d'installation, et sans aucune tâche de maintenance. Dans ce modèle l'application est gérée par le client à travers une interface, généralement un navigateur internet. Les principales applications actuelles de ce modèle sont les applications de gestion de la relation client (**CRM** : **C**ustomer **R**elationship **M**anagement), la vidéo conférence, les communications unifiées, les outils collaboratifs, la messagerie. Les principaux fournisseurs de cette catégorie sont Salesforce.com (logiciels CRM) et Google (Gmail, Google Apps).

I.2.4.2. Plateforme en tant que service (Platform as a service (PaaS))

Ce modèle offre une plateforme entièrement configurée et gérée, sur laquelle l'utilisateur peut développer, tester et exécuter ses applications. Le déploiement des solutions PaaS est automatisé et évite à l'utilisateur d'avoir à acheter des logiciels ou d'avoir à réaliser des installations supplémentaires. Le PaaS offre une grande flexibilité, permettant notamment de tester rapidement un prototype ou encore d'assurer un service informatique sur une période de courte durée. Il favorise la mobilité des utilisateurs, puisque l'accès aux données et aux applications peut se faire à partir de n'importe quel périphérique connecté. Les principaux fournisseurs du PaaS sont SalesForce.com (Force.com), Google (Google App Engine), Microsoft (Windows Azure), Facebook (Facebook Platform).

I.2.4.3. Infrastructure en tant que service (Infrastructure as a service (Iaas))

Le client peut s'approvisionner en ressources de traitement, de stockage, de réseau et toute autre ressource informatique essentielle. Le client peut déployer et exécuter les logiciels qu'il souhaite, qu'il s'agisse de systèmes d'exploitation ou d'applications. Les utilisateurs d'IaaS sont libérés des charges causées par la possession, la gestion ou le contrôle du matériel sous-jacent. Par conséquent, ils n'ont pas accès directement aux machines physiques, mais indirectement à travers les machines virtuelles (**VMs**). Les utilisateurs ont la plupart du temps un contrôle presque complet sur les **VMs** qu'ils louent, ils peuvent choisir des images de systèmes d'exploitation préconfigurées par le fournisseur, ou bien des images de machines personnalisées contenant leurs propres applications, bibliothèques et paramètres de configuration. Les utilisateurs peuvent également choisir les différents ports d'Internet à travers lesquels les machines virtuelles seront accessibles, etc.

Il existe de nombreux fournisseurs commerciaux et open-source de l'IaaS. Parmi les plateformes commerciales, nous pouvons citer: Amazon EC2, Microsoft Azure.

➔ **Le tableau suivant présente les avantages et les inconvénients des services cloud :**

Services cloud	Avantages	Inconvénients
SaaS	• Pas d'installation • Plus de licence • Migration	• logiciel limité • sécurité • dépendance des prestataires
PaaS	• pas d'infrastructure nécessaire • pas d'installation • environnement hétérogène	• limitation des langages • pas de personnalisation dans la configuration des machines virtuelles
IaaS	• administration • personnalisation • flexibilité d'utilisation	• sécurité • besoin d'un administrateur système

Tableau I.1 : Les avantages et les inconvénients des services cloud. [I.3]

➔ **La figure ci-dessous résume les différents modèles de services Cloud :**

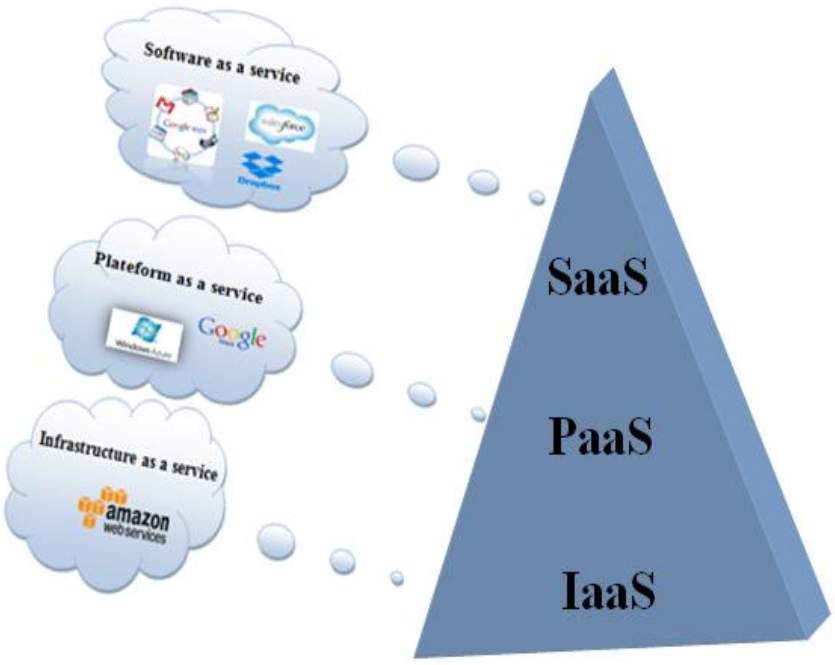


Figure I.2 : Modèles de services Cloud.

I.2.5. Critères de classifications des clouds

Avant de présenter les différents types de cloud, nous établissons dans un premier temps quelques critères de classification :

- **La raison de développement :** C'est la raison qui justifie la mise en place de la plateforme. Elle peut être commerciale, scientifique ou communautaire.
- **Le niveau de services :** C'est l'ambition du cloud à fournir aux entreprises une plateforme proche ou pas de leurs attentes.
- **L'accessibilité :** Le cloud peut être accessible pas tous (Cloud public) ou restreint à un publique particulier (Cloud prive).

I.2.6. Types de clouds

Les technologies du Cloud peuvent être proposées selon différents modèles de déploiement, les plus courants, tels que définis par le **NIST**, sont les suivants :

I.2.6.1. Cloud privé

Cloud privé est une plateforme gérée en interne. Cette infrastructure est fournie à l'usage exclusif d'une seule entreprise (le client), composée de plusieurs clients internes (les unités commerciales, par exemple). Elle peut être possédée, gérée et exploitée par le client, par un tiers, ou par les deux en des proportions variables, et peut être disponible sur site ou hors site.

I.2.6.2. Cloud communautaire

Cette infrastructure est fournie à l'usage exclusif d'une communauté donnée de clients faisant partie d'entreprises qui partagent des intérêts communs (missions, exigences de sécurité, politiques, règles de conformité, etc.). Elle peut être possédée, gérée et exploitée par une ou plusieurs de ces entreprises, par un tiers, ou par les deux en des proportions variables, et peut être disponible sur site ou hors site.

I.2.6.3. Cloud public

Les utilisateurs ont accès à des services cloud via l'Internet public sans savoir précisément où sont hébergées leurs données ni où sont exécutés leurs traitements. Cette infrastructure est fournie à l'usage du grand public. Elle peut être possédée, gérée et exploitée par une entreprise, un établissement d'éducation ou une administration publique, ou une combinaison des trois. Elle est disponible sur le site du fournisseur.

I.2.6.4. Cloud hybride

Cette infrastructure se compose d'au moins deux infrastructures Cloud distinctes (privée, communautaire ou publique) qui ne se mélangent pas, mais sont liées par une technologie standard ou propriétaire assurant la portabilité des données et des applications.

➔ **La figure suivante représente les types de cloud :**

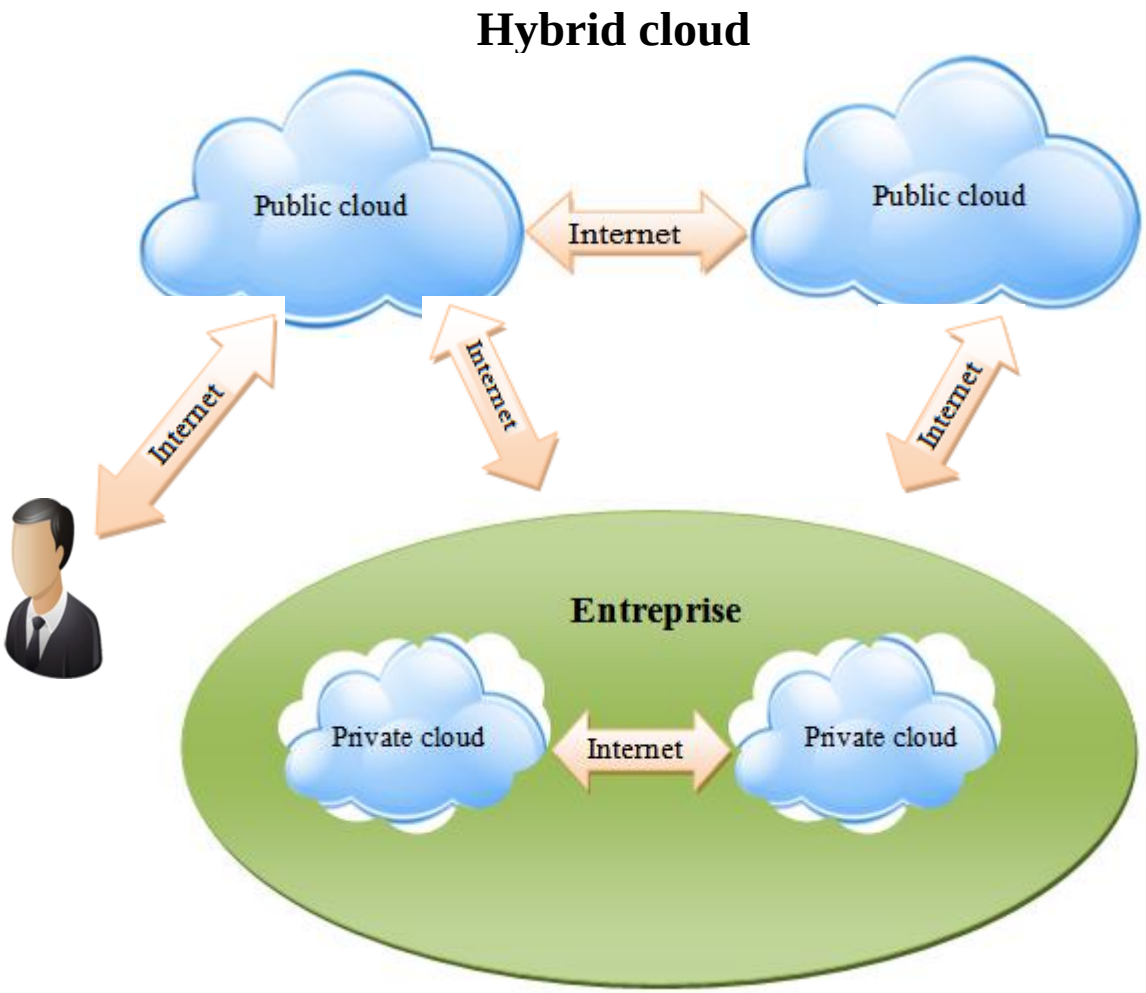


Figure I.3 : Les différents types d'un Cloud.

I.2.7. Caractéristiques fondamentales du cloud

Les caractéristiques fondamentales du cloud sont les suivantes :

- **Virtualisation :** Le Cloud repose largement sur la virtualisation des serveurs et du stockage, afin d'allouer ou de réallouer rapidement des services.

- **Accès en libre-service à la demande :** Le cloud offre des ressources et services aux utilisateurs à la demande. Un utilisateur accède depuis une interface web à des services informatiques sans intervention humaine de la part de chaque fournisseur de services.

- **Mutualisation de ressources (Pooling) :** Les ressources du cloud peuvent être regroupées pour servir des utilisateurs multiples, pour lesquels des ressources physiques et virtuelles sont automatiquement attribuées.

- **L'élasticité et la flexibilité des ressources :** Il est possible d'acquérir rapidement de nouvelles ressources en fonction de l'évolution des besoins du métier. La gestion de ressources peut être automatisée, ou bien un opérateur ajuste les ressources en fonction des besoins. L'utilisateur a l'illusion d'avoir accès à des ressources illimitées à n'importe quel moment, bien que le fournisseur en définisse généralement un seuil (par exemple : 20 instances par zone est le maximum possible pour Amazon EC2).

- **Paiement à l'usage :** La consommation des ressources dans le cloud s'adapte au plus près aux besoins de l'utilisateur. Le fournisseur est capable de mesurer de façon précise la consommation (en durée et en quantité) des différents services (CPU, stockage, bande passante,…) ; cela lui permettra de facturer l'utilisateur selon sa réelle consommation.

- **Fiabilité et tolérance aux pannes :** Les environnements cloud tirent parti de la redondance intégrée du grand nombre de serveurs qui les composent en permettant des niveaux élevés de disponibilité et de fiabilité pour les applications qui peuvent en bénéficier.

- **Garantie QoS (Quality of Service) :** Les environnements de cloud peuvent garantir la qualité de service pour les utilisateurs, par exemple, la performance du matériel, comme la bande passante du processeur et la taille de la mémoire.

- **Basé-SLA :** Les clouds sont gérés dynamiquement en fonction des contrats d'accord de niveau de service (SLA) entre le fournisseur et l'utilisateur. Le SLA définit des politiques, telles que les paramètres de livraison, les niveaux de disponibilité, la maintenabilité, la performance, l'exploitation, ou autres attributs du service, comme la facturation, et même des sanctions en cas de violation du contrat. Le SLA permet de rassurer les utilisateurs dans leur idée de déplacer leurs activités vers le cloud, en fournissant des garanties de QoS.

- **Sécurité de données et confidentialité :** Dans le cloud, les données doivent êtres transférées entre les dispositifs de l'utilisateur et les Datacenter des fournisseurs de services de cloud, ce qui les rendra cible facile pour les pirates. La sécurité des données et la confidentialité doivent être garanties, que ce soit sur le réseau ou encore dans les Datacenter de cloud où elles seront stockées.

- **Migration de machines virtuelles :** La virtualisation peut offrir des avantages importants dans le cloud en permettant la migration de machine virtuelle pour équilibrer la charge de travail entre les Datacenter. Elle permet un approvisionnement robuste et très réactif dans les Datacenter.

- **Consolidation de serveurs :** La consolidation de serveurs est une approche efficace pour maximiser l'utilisation des ressources, tout en minimisant la consommation d'énergie dans un environnement de cloud. La technologie de migration de VM est souvent utilisée pour consolider les machines virtuelles se trouvant sur plusieurs serveurs sous-utilisés sur un seul serveur, de sorte à mettre ces derniers en mode d'économie d'énergie.

- **Gestion de l'énergie :** Améliorer l'efficacité énergétique est un autre enjeu majeur dans le cloud. Il a été estimé que le coût de l'alimentation et du refroidissement compte pour 53% des dépenses de fonctionnement total des Datacenter. Concevoir des

Datacenter économes en énergie a récemment reçu une attention considérable. Ce problème peut être abordé dans plusieurs directions. Par exemple, l'efficacité énergétique des architectures matérielles, permettant de ralentir la vitesse (fréquence et tension) du processeur et d'éteindre les composants matériels partiels, est devenue très pertinente.

- **Ordonnancement :** Est un enjeu important qui influence considérablement les performances de l'environnement de cloud. Il existe plusieurs niveaux d'ordonnancement dans le cloud, notamment : l'ordonnancement au niveau application et l'ordonnancement au niveau infrastructure.

I.2.8. Exemple de clouds « Amazon EC2 » et « Google App Engine »

➢ **Amazon EC2 :**

Amazone offre une solution de calcul très considérable avec le service Elastic Cloud Computing EC2, elle offre à l'utilisateur des puissances de calcul inégalé et des moyens de contrôle de son compte.

Une facilité de créations et de démarrage des machines virtuelles et des commandes simplifiée via une interface web.

Aujourd'hui EC2 fournit le contrôle complet sur les ressources informatique d'un client, ainsi de nouveaux exemples serveurs peuvent être installées et amorcés en quelques minutes, et leur capacité peut être mesurée rapidement par des utilitaires de la plateforme (Amazone Cloud Watch).

EC2 fournit beaucoup de fonctionnalités utiles pour des clients, y compris un système facturation très adapte et très précis en utilisant (le nombre d'heures, puissance de calcul, RAM des machines virtuelles, bande passante, espace disque alloué, etc.)Comme facteurs pour l'élaboration des couts.

Le déploiement entre les multiples lieux, les adresses IP élastiques, la connexion à l'infrastructure existante d'un client par les réseaux **VPN (V**irtual **P**rivate **N**etworks**)**, les services de contrôle par Amazone Cloud Watch, et l'équilibrage de charge.

EC2 a déployé une telle granularité et précision dans son nuage de calcul à tel point que c'est devenu une référence et un modèle dans le calcul dans le nuage.

Les utilisateurs peuvent démarrés, contrôlés, gèrent les règles du pare-feu, configurer à volonté et éteindre leurs instances de machines.

- **Google App Engine :**

Google **A**pp **E**ngine **(GAE)** est designé pour répondre aux aspirations et à la demande croissante de la communauté des utilisateurs des services Google, qui ne cesse de demander plus d'efficacité et de sécurités sur les plateformes web, en laissant les environnements Paython et Java actifs pour les applications serveur.

Les utilisateurs effectuent des requêtes HTTP pour du contenue statique (fichiers, images, etc.) ou pour du contenue dynamique (calcul, traitement de texte, etc.). Avec des stratégies d'équilibrage de charge et de routage basées sur le contenu.

Les systèmes d'exécution fonctionnant sur des serveurs d'applications traitent le traitement de logique des applications et fournissent le contenu de Web dynamique, alors que des pages statiques sont servies par l'infrastructure partagée de Google.

Pour découpler les données persistantes des serveurs d'applications, GAE les réplique dans ces serveurs de données à la place d'u n système de fichier local.

Les applications peuvent intégrer des services de données et d'autres services de Google et APP, telles que l'email, mémoire d'image et ainsi de suite par des apis fournis par GAE.

En plus de ces services, Google fournit également quelques outils pour aider les développeurs à intégrer facilement leurs applications via des APIS et SDK disponibles sur GAE.

I.3. Etat de l'art sur les approches de réservation de ressources dans le cloud

I.3.1. Introduction

La provision de ressources est un problème important et difficile dans les systèmes distribués à grande échelle tels que l'environnement Cloud computing .Il se réfère à la détermination,

l'établissement et la gestion du temps d'exécution des ressources logicielles et matérielles. Cette provision de ressources prend en considération l'accord de niveau de service (SLA) pour fournir un service aux utilisateurs du cloud. Il s'agit d'un accord initial entre les utilisateurs du cloud et les fournisseurs de services en nuage qui garantissent les paramètres de qualité de service (QoS). Une autre contrainte importante est la consommation d'énergie. Il faut prendre soin de réduire la consommation d'énergie, la dissipation de puissance et le placement de VM.

Ainsi, le but ultime de l'utilisateur du cloud est de minimiser les coûts en louant les ressources et du point de vue du fournisseur de services cloud pour maximiser les bénéfices en allouant efficacement les ressources. Afin d'atteindre l'objectif, l'utilisateur du nuage doit demander au fournisseur de services en nuage de fournir une provision pour les ressources de manière statique ou dynamique afin que le fournisseur de services en nuage connaisse le nombre d'instances des ressources et les ressources requises pour une application particulière.

I.3.2. Types de provisionnement des ressources

1) Approvisionnement statique: Pour les applications qui ont des demandes de travail prévisibles et généralement inexistantes, il est possible d'utiliser efficacement le «provisionnement statique». Avec l'approvisionnement anticipé, le client contracte avec le fournisseur pour les services et le fournisseur prépare les ressources appropriées avant Début du service. Le client reçoit une taxe forfaitaire ou est facturé sur une base mensuelle.

2) Approvisionnement dynamique: Dans les cas où la demande par les applications peut changer ou varier, des techniques de "provisionnement dynamique" ont été suggérés par lesquelles les VMs peuvent être migrées sur les nouveaux noeuds de calcul dans le cloud. Avec un provisionnement dynamique, le fournisseur alloue plus Les ressources requises et les supprime lorsqu'elles ne le sont pas. Le client est facturé sur une base de paiement par utilisation. Lorsque le provisionnement dynamique est utilisé pour créer un nuage hybride, il est parfois appelé éclatement des nuages.

3) Auto-provisionnement de l'utilisateur: Avec l'auto provisionnement de l'utilisateur (également appelé self service de nuage), le client achète des ressources du fournisseur de cloud via un formulaire Web, crée un compte client et paye des ressources avec une carte de crédit. Les ressources du fournisseur sont disponibles pour l'utilisation du client dans les heures, sinon les minutes.

I.3.3. Paramètres pour l'approvisionnement

1. **Temps de réponse :** L'algorithme de provisionnement de ressources conçu doit prendre un temps minimum pour répondre lors de l'exécution de la tâche.

2. **Réduire le coût :** Du point de vue utilisateur Cloud le coût doit être réduit au minimum.

3. **Maximisation des revenus :** Cela doit être réalisé à partir de la vision du fournisseur de services Cloud.

4. **Tolérance aux pannes :** L'algorithme devrait continuer à fournir un service malgré l'échec des nœuds.

5. **Violation de SLA réduite :** L'algorithme conçu doit pouvoir réduire la violation de SLA.

6. **Réduction de la consommation d'énergie :** Les techniques de placement et de migration VM doivent réduire la consommation d'énergie.

I.3.4. Comparaison des techniques de provisionnement des ressources

Techniques de provisionnement des ressources	Mérite	Défis
1) Fourniture à terme de ressources pour des applications scientifiques dans des nuages hybrides avec Aneka.	✓ Capable d'allouer efficacement des ressources provenant de différentes sources afin de réduire les temps d'exécution des applications.	• Ne convient pas aux applications à forte intensité de données HPC.
2) Disposition dynamique dans les nuages de services multi-locataires.	✓ Correspond aux fonctionnalités des locataires avec les exigences du client.	• Ne fonctionne pas pour tester sur le système basé sur le cloud réel et sur plusieurs domaines.
3) Elastic Application Container: une approche légère pour l'approvisionnement en ressources Cloud.	✓ Surpasse en termes de flexibilité et d'efficacité des ressources**.**	• Ne convient pas aux applications Web et prend en charge un seul type de langage de programmation, Java.
4) Disposition de ressources sans échec pour l'infrastructure Cloud Cloud.	✓ Capable d'améliorer la QoS des utilisateurs environ 32% en termes de taux de violation du délai et 57% en termes de ralentissement avec un coût limité sur un nuage public.	• Ne sont pas capables d'exécuter des expériences réelles. • Ne peuvent pas déplacer des machines virtuelles entre des nuages publics et privés.
5) Méthode de provisionnement VM pour améliorer le bénéfice et la violation de SLA des fournisseurs de services en nuage.	✓ Réduit les violations de SLA et améliore le bénéfice.	• Augmente le problème de l'allocation des ressources et de l'équilibrage de charge parmi les centres de données.

6)	Conception et mise en œuvre d'un provisionneur adaptatif de machine virtuelle adaptatif (APA-VMP) utilisant l'intelligence de l'essaim.	✓ Emplacement efficace de la VM et réduction significative de la puissance.	• Ne convient pas pour conserver la puissance dans les centres de données modernes.
7)	Optimal Resource Provisioning pour Cloud Computing Environment.	✓ Fournit efficacement des ressources Cloud pour les utilisateurs de SaaS avec un budget limité et une échéance optimisant ainsi la QoS.	• Applicable uniquement pour les utilisateurs SaaS et les fournisseurs SaaS.

Tableau I.2 : Les techniques de provisionnement des ressources. [II.3]

I.4. Workflow

I.4.1. Définition de workflow

Diverses définitions ont été proposées, selon les catégories d'usages, En ce qui concerne le workflow scientifique, nous retiendrons la définition suivante.

Un workflow est composé d'un ensemble de tâches (traitements) organisées selon un ordre logique, afin de réaliser un traitement global, complexe et pertinent sur un ensemble de données sources. Ces données sont souvent complexes, tant au niveau de leurs structures que de leurs organisations. Elles sont souvent volumineuses.

La taille d'un workflow scientifique peut varier de quelques tâches à des millions de tâches, qui sont souvent de calcul intensif "Computation Intensive". Pour les grands workflows, il est souhaitable de répartir les tâches entre plusieurs ressources, afin d'optimiser les temps d'exécution.

Récemment, les environnements clouds computing sont considérés comme des plateformes d'exécution de workflows. L'exécution d'un workflow est gérée par un SGWf (Système de Gestion de Workflow).

I.4.2. Intérêts du cloud pour les workflows

Les clouds offrent plusieurs avantages pour les applications à base de workflows. Ces avantages facilitent :

• **L'approvisionnement de ressources :** Dans le cloud, au lieu de déléguer l'allocation au gestionnaire de ressources, l'utilisateur peut provisionner les ressources nécessaires et ordonnancer les tâches en utilisant un ordonnanceur contrôlé par l'utilisateur. Ce modèle d'approvisionnement est idéal pour les workflows, car il permet au système de gestion de workflow d'allouer une ressource une seule fois et de l'utiliser pour exécuter de nombreuses tâches.

• **L'allocation dynamique de ressources à la demande :** Les clouds donnent l'illusion que les ressources informatiques disponibles sont illimitées. Cela signifie que les utilisateurs peuvent demander, et s'attendre à obtenir des ressources suffisantes pour leurs besoins, à tout moment. L'approvisionnement à la demande est idéal pour les workflows et d'autres applications faiblement couplées, car il réduit le surcoût (overheads) d'ordonnancement total et peut améliorer considérablement les performances du workflow.

• **L'élasticité :** Outre l'approvisionnement des ressources à la demande, les clouds permettent aussi aux utilisateurs de libérer des ressources à la demande. La nature élastique de clouds facilite le changement des quantités et des caractéristiques de ressources lors de l'exécution, permettant ainsi d'augmenter le nombre de ressources, quand il y a un grand besoin, et d'en diminuer, lorsque la demande est faible. Cela permet aux systèmes de gestion de workflow de répondre facilement aux exigences de qualité de service (QoS) des applications, contrairement à l'approche traditionnelle, qui nécessite de réserver à l'avance des ressources dans les environnements de grilles.

• **La garantie des QoS via des SLA :** Avec l'arrivée des services de cloud computing provenant de grandes organisations commerciales, les accords de niveau de service (SLA) ont été une préoccupation importante pour les fournisseurs et les utilisateurs. En raison de compétitions entres les fournisseurs de services émergents, un grand soin est pris lors de la conception du SLA qui vise à offrir de meilleures garanties de QoS aux utilisateurs, et des termes clairs pour l'indemnisation, en cas de violation du contrat. Cela permet aux systèmes de gestion de workflow de fournir de meilleures garanties de bout en bout en "mappant" les utilisateurs aux fournisseurs de services selon les caractéristiques des SLA.

• **Le faible Coût d'exploitation :** Les fournisseurs de cloud profitent également des économies d'échelle, en fournissant des ressources de calcul, de stockage et de bande passante, à un coût très faible grâce, à la virtualisation. Ainsi l'utilisation des services de cloud public pourrait être économique et une alternative moins coûteuse, par rapport à l'utilisation de ressources dédiées, qui sont plus chères. Un des avantages de l'utilisation des ressources virtuelles pour l'exécution de workflow, plutôt que d'un accès direct à la machine physique, est le besoin réduit pour sécuriser les ressources physiques des codes malveillants

I.4.3. Systèmes de gestion de workflows

Un **SGWf** représente un système qui définit, implémente et gère l'exécution de workflows à l'aide d'un environnement logiciel fonctionnant avec un ou plusieurs moteurs de workflows et capable d'interpréter la définition d'un processus, de gérer la coordination des participants et d'invoquer des applications externes.

L'architecture de référence d'un **SGWf** proposée par la **WfMC** (**W**orkflow **M**anagement **C**oalition) en1995 est présentée dans la figure suivante.

Ce modèle inclut un service de déploiement, qui contrôle l'exécution des workflows et qui supporte cinq interfaces standardisées.

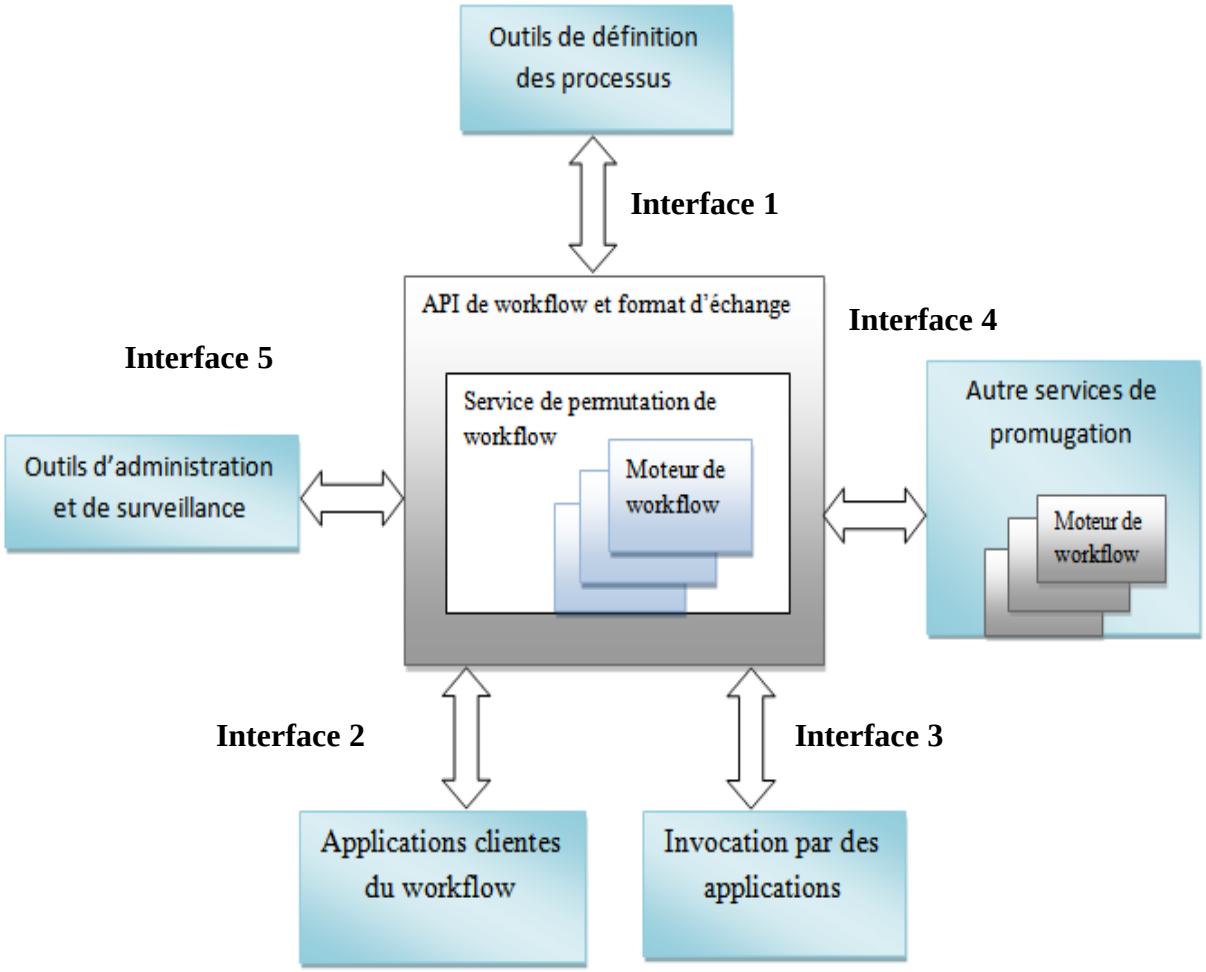


Figure I.4 : Modèle de référence des systèmes de gestion de workflow (WfMC, 95). [II.3]

- **Interface 1 :** Correspond à l'échange des modèles entre les moteurs de workflows et les différents outils de modélisation de processus.

- **Interface 2 :** Permet à des applications clientes de communiquer avec le moteur de workflows.

- **Interface 3 :** Permet au système de workflow d'invoquer des applications clientes.

- **Interface 4 :** Permet l'interopérabilité entre les différents moteurs de workflows.

- **Interface 5 :** Correspond à l'interaction entre les applications d'administration et de pilotage et le moteur de workflows.

Un des principaux modules de tous ces systèmes de gestion de workflow est l'ordonnancement de workflows.

I.5. Ordonnancement de workflows

I.5.1. Définition

L'ordonnancement de workflows est un processus qui "mappe" et gère l'exécution de tâches interdépendantes sur des ressources distribuées. Il alloue des ressources appropriées pour les tâches de workflow de sorte que l'exécution puisse être achevée, tout en satisfaisant les objectifs et contraintes imposés par l'utilisateur. Un ordonnancement adéquat peut avoir un impact significatif sur la performance du système. En général, le problème de mapping de tâches sur les services distribués appartient à une classe de problèmes connus comme NP-complets.

Les fournisseurs et les utilisateurs de services cloud ont des exigences et des objectifs conflictuels. Un bon ordonnanceur doit mettre en œuvre un compromis acceptable entre ces objectifs. Ainsi, l'ordonnancement de workflow dans le cloud devient un problème d'optimisation multi objectif **[I.4].**

I.5.2. La dépendance des tâches d'une application

Quand les relations entre les tâches dans une application sont prises en compte, une dichotomie commune utilisée est la dépendance et l'indépendance. Habituellement, la dépendance veut dire il y a la préséance d'ordre d'exécution dans les tâches, une tâche ne peut pas commencer jusqu'à ce que tous ses parents sont déjà terminés.

La dépendance à un impact crucial pour la conception des algorithmes d'ordonnancement, donc dans cette subdivision les algorithmes sont discutés en suivant la même dichotomie comme fait illustrer dans la figure suivante :

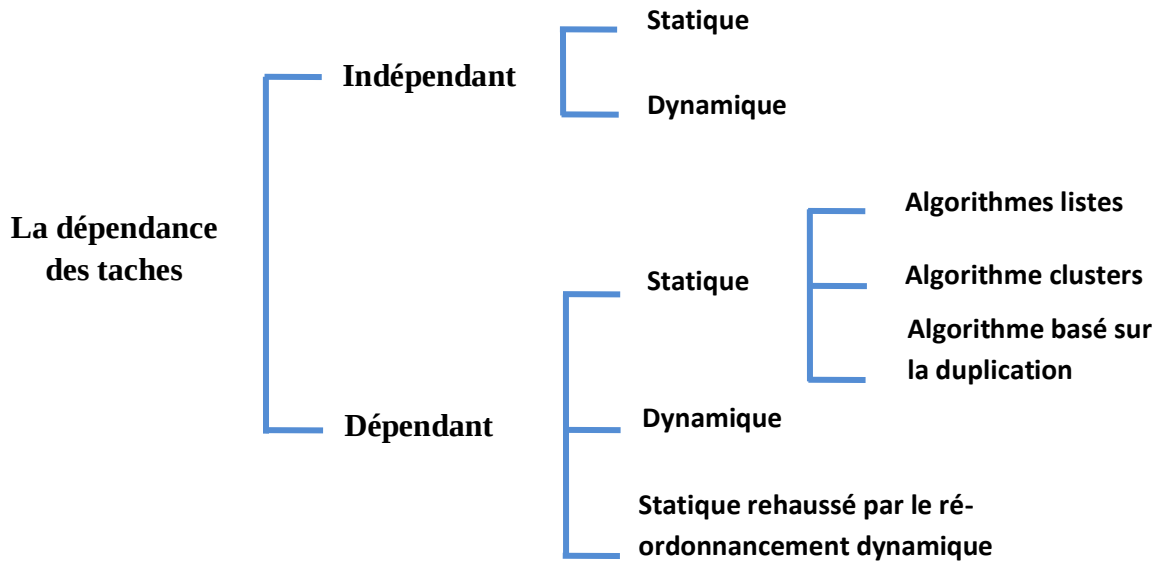


Figure I.5 : Les algorithmes d'ordonnancement de tâches [II.26].

I.5.2.1. Ordonnancement des tâches indépendantes

Quand un ensemble de tâches indépendantes arrivent, du point de vue du système, une stratégie commune est de les assigner selon la charge des ressources afin de réaliser un système performant. Du point de vue des applications, les algorithmes heuristiques statiques fondés sur l'exécution et l'estimation des coûts peuvent être appliqué.

I.5.2.2. Ordonnancement des tâches dépendantes

Quand un ensemble de tâches interconnectent qui s'exécutées dans un ordre de priorité, pour exécuter une certaine application. Ces tâches ont des dépendances entre elles **(workflow).**

I.5.2.2.1. Définition DAG (Graphe Acyclique Dirigé)

Un **DAG**, comme les trois mots le décrivent :

Graphe : un ensemble de nœuds **N** qui sont connectés par des arcs.

Dirigé : il n'y a aucun chemin dirigé dans le graphique des tâches qui peut commencer et finir à un nœud particulier **n.**

Acyclique : il n'y a pas de cycle dans n'importe quel chemin du graphe.

- **La figure I.6 ci-dessus représente la structure d'un DAG :**

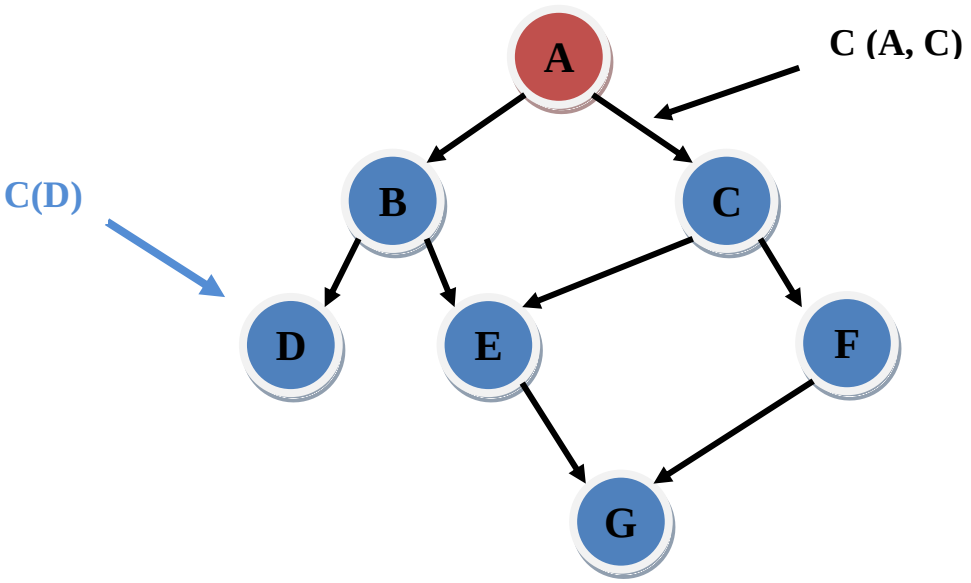


Figure I.6 : Structure d'un DAG.

- Chaque nœud a un poids nommé coût de calcul et noté par **C (ni).**
- Les arcs du **DAG** représentent les messages de communication entre un nœud et ses fils pour commencer l'exécution.
- Le poids d'un arc est appelé coût de communication et noté par **C (ni, nj).**

II.6. Méta-heuristiques d'ordonnancement de workflows dans le cloud computing

De nombreux chercheurs ont proposé différentes approches basées sur des heuristiques ou des méta-heuristiques pour résoudre le problème d'ordonnancement de workflows dans les systèmes distribués. Parmi les algorithmes proposés récemment et qui sont actuellement répandus dans le cloud computing :

- **Topcuoglu** et **al [I.5]**, ont proposé l'algorithme **HEFT (H**eterogeneous **E**arliest **F**inish **T**ime**),** qui détermine un ordonnancement d'un graphe **DAG (D**irect **A**cyclic **G**raph**)** de tâches sur un environnement de ressources hétérogène de manière à minimiser la durée totale d'exécution du workflow **(**makespan**)**. **HEFT** est l'une des solutions d'ordonnancements de listes les plus répandues et ayant un des meilleurs rapports performances-complexité. **HEFT** est composé de deux phases principales sont :

  1. **La détermination des priorités des tâches :** Consiste à attribuer des priorités aux tâches à ordonnancer. La priorité (rang) d'une tâche est son Botton-Level, elle correspond à la longueur du chemin critique de cette tâche au puits du graphe lorsque l'on considère la moyenne des durées des transferts et des calculs sur les ressources hétérogènes.

  2. **La sélection du processeur :** Lorsqu'une tâche est retirée de la liste triée des tâches, l'algorithme recherche la ressource qui minimise sa date de fin et la lui alloue, dans la mesure où toutes les dépendances de données sont satisfaites. L'algorithme prend en compte les temps d'inactivité des ressources, en utilisant un ordonnancement par insertion.

✓ **Ge** et **Wei [I.6],** ont développé un nouvel ordonnanceur, qui rend une décision d'ordonnancement, en évaluant tout le groupe de tâches dans la file d'attente des jobs et utilise l'algorithme génétique pour l'optimisation du makespan. Les résultats montrent que l'ordonnanceur peut obtenir un makespan plus court pour les jobs et un

meilleur équilibrage de charge des nœuds du cloud comparé aux résultats de la stratégie **FIFO** (**F**irst **I**n **F**irst **O**ut).

- ✓ **Lin** et **Lu [I.7],** ont proposé l'heuristique **SHEFT** (**S**calable **HEFT**) pour ordonnancer un workflow d'une façon élastique sur un environnement de cloud computing. Cet algorithme permet de minimiser le makespan du workflow et gère la scalabilité des ressources à l'exécution.

- ✓ **El-kenawy** et **al** [**I.8]**, ont proposé une version améliorée de l'algorithme **Max-Min**. L'algorithme utilise le temps d'exécution prévu, au lieu du temps complet, comme base de sélection. Ils ont utilisé les réseaux de Petri, qui sont bien adaptés pour la modélisation du comportement concurrentiel des systèmes distribués. Le résultat montre que cet algorithme réalise l'ordonnancement avec un makespan inférieur à celui de l'algorithme **Max-Min** original.

➔ **Les algorithmes présentés ci-dessus ont été conçus pour l'ordonnancement de workflows sur l'infrastructure distribuée et hétérogène de cloud qui se focalise sur l'optimisation du makespan.**

Les algorithmes d'ordonnancement doivent considérer d'autres métriques, telles que la disponibilité, ou le taux l'utilisation des ressources, ainsi que les aspects de sécurité. Parmi les algorithmes proposés :

- ✓ **Hakem** et **Butelle** [**I.9]**, ont présenté une heuristique bi-objectif gloutonne pour l'ordonnancement d'applications parallèles sur des systèmes informatiques distribués hétérogènes, nommée **BSA** (**B**i-objective **S**cheduling **A**lgorithm). **BSA** prend en compte le makespan et la probabilité d'échec de l'application. L'algorithme est basé sur une fonction de compromis, qui sélectionne les processeurs sur lesquels les tâches critiques doivent être "mappées" au cours du processus d'ordonnancement.

- ✓ **Wang** et **al** [**I.10]**, ont mis en œuvre la technique de réputation **RD (D**riven **R**eliability**)** pour évaluer la fiabilité des ressources dans les systèmes informatiques largement distribués. Ils ont proposé l'algorithme **LAGA (L**ook-**A**head **G**enetic **A**lgorithm**)**, qui utilise la réputation **RD** pour optimiser à la fois le makespan et la

fiabilité du workflow. Les résultats de simulations montrent que la réputation **RD** peut améliorer la fiabilité du workflow.

- ✓ **Jang** et **al** [**I.11]**, ont proposé un modèle d'ordonnancement des tâches dans lequel l'ordonnanceur invoque la fonction de l'algorithme génétique pour réaliser l'ordonnancement de tâches, en considérant la satisfaction des utilisateurs et les disponibilités des machines virtuelles **(VMs).**

Outre le makespan, le coût et les autres métriques de qualité de service, la consommation d'énergie est de plus en plus importante dans les environnements de cloud computing . En effet, avec le succès des services cloud, les fournisseurs déploient de plus en plus des Datacenters qui sont gourmands en énergie et qui émettent une quantité considérable de dioxyde de carbone. Par conséquent, des travaux plus récents ont porté sur le développement d'algorithmes d'ordonnancement prenant en compte la consommation énergétique

- ✓ **Guzek** et **al** [**II.34]**, examinent l'influence de la **DVFS**, en utilisant l'algorithme évolutionnaire multi-objectif **(NSGA-II)** pour l'ordonnancement bi-objectif du makespan et de la consommation énergétique d'un graphe **DAG (D**irect **A**cyclic **G**raph**)** de tâches sur une plateforme multiprocesseur hétérogène.

- ✓ **Chang-Tian** et **Jiong** [**II.35]**, ont utilisé la technique **DVS** et ont proposé deux algorithmes basés sur un algorithme génétique, à savoir: **ETU_GA (E**nergy consumption **T**ime **U**nify **G**enetic **A**lgorithm**)** et **ETDF_GA (E**nergy consumption **T**ime **D**ouble **F**itness **G**enetic **A**lgorithm**)**, pour l'ordonnancement de tâches indépendantes dans le cloud computing. Ces algorithmes visent à trouver l'équilibre entre le makespan et l'énergie.

- ✓ **Liu** et **al [II.36],** ont proposé un algorithme génétique multi-objectif **MOGA** pour l'ordonnancement de jobs sur les machines virtuelles du cloud. L'algorithme tente de minimiser la consommation d'énergie et de maximiser le profit du fournisseur, sous la contrainte de délai. Néanmoins, ils ne prennent pas en compte les contraintes de précédence entres les tâches.

➔ **Le tableau suivant montre la classification des algorithmes d'ordonnancement de workflows dans le cloud citée au dessus :**

Chercheur	Algorithme	Métaheuristique	Makespan	Coût	Energie	Contraintes QoS	Pareto
Topcuoglu	HEFT	X	✓	X	X	X	X
Ge	GA	✓	✓	X	X	X	X
Lin	SHEFT	X	✓	X	X	X	X
El-kenawy	Max-Min	X	✓	X	X	X	X
Hakem	BSA	X	✓	X	X	F	X
Wang	RD, LAGA	✓	✓	X	X	F	X
Jang	GA	✓	✓	✓	X	Di	X
Guzek	NSGA-II	✓	✓	X	✓	X	✓
Chang-Tian	ETU_GA, ETDF_GA	✓	✓	X	✓	X	X
Liu	MOGA	✓	X	X	✓	D	✓

Tableau I.3 : Classification des algorithmes d'ordonnancement de workflows dans le cloud [II.3].

✓ **: Oui.**

X : Non.

D : Deadline.

Di : Disponibilité.

F : Fiabilité.

II.7. Conclusion

Dans ce chapitre, nous avons présenté quelques définitions de base sur l'ordonnancement de workflow dans le cloud et les approches de réservation de ressources dans le cloud, qui nous sera utile pour modéliser et résoudre le problème d'ordonnancement de workflows.

Chapitre II :

Modélisation du problème multi-objectif

II.1. Introduction

L'optimisation multi-objective est une approche très importante du fait de la nature multi-objectif de la plupart des problèmes réels, tels que le problème d'ordonnancement de workflow dans le cloud.

Dans ce chapitre nous présentons d'abord un ensemble de définitions et de concepts de base de l'optimisation multi-objectif. Puis nous présentons le problème de l'optimisation multi-objectif et l'aide à la décision, par la suite on traite les méthodologies de résolution, ainsi nous présentons les concepts communs aux méta-heuristiques, suivis de quelques méthodes regroupées en deux catégories : les méta-heuristiques à base de solution unique, et les méthodes à base de population de solutions. Enfin nous présentons les algorithmes génétiques comme exemple de méta-heuristique.

II.2. L'optimisation multi-objectif

II.2.1. Définition

L'optimisation multi-objectif se réfère à la solution de problèmes avec deux ou plusieurs objectifs à satisfaire simultanément. Généralement, ces objectifs sont en conflit les uns avec les autres et sont exprimés dans différentes unités. En raison de leur nature, les problèmes d'optimisation multi-objectifs n'ont généralement pas un mais un ensemble de solutions, appelées solutions optimales de Pareto. Lorsque de telles solutions sont tracées dans un espace de fonction objectif, le graphique produit s'appelle le front de Pareto du problème. **[II.1]**

II.2.2. La multiplicité des solutions

Lorsque l'on cherche à obtenir une solution optimale à un problème d'optimisation multi-objectif donné, on s'attend souvent à trouver une solution et une seule. Mais La plupart du temps, on trouve une multitude de solutions, du fait que certains des objectifs sont contradictoires.

Donc, quand on résoudra un problème d'optimisation multi-objectif, on obtiendra une grande quantité de solutions. Ces solutions, comme on peut s'en douter, ne seront pas **optimales**, au sens où elles ne minimiseront pas tous les objectifs du problème.

Un concept intéressant, qui nous permettra de définir les solutions obtenues, est le **compromis**.

En effet, les solutions que l'on obtient lorsque l'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. **[II.2]**

II.2.3. Concepts et définitions

Nous présentons, dans cette section, les principaux concepts, définitions et notations liées au domaine de l'optimisation multi-objectif, notamment, la relation de dominance de Pareto, l'optimalité Pareto et le front de Pareto.

II.2.3.1. Formulation générale d'un problème d'optimisation multi-objectif

II.2.3.1.1. Définition

Un problème d'optimisation combinatoire multi-objectif, ***MOP*** (**M**ulti- Objectif combinatorial **O**ptimization **P**roblem) peut être défini par :

$$\textbf{(MOP)} = \begin{cases} \boldsymbol{Min\ F(x) = (f_1(x), \dots, f_n(x))} \\ \boldsymbol{s.c.\ x \in X} \end{cases} \qquad \textbf{(1).}$$

Où *n* est le nombre d'objectifs($\boldsymbol{n \geq 2}$), Le vecteur $x = (x_1, \dots, x_k) \in X$ représente le vecteur de k variables de décision, X représente l'ensemble de solutions réalisables dans l'espace décisionnel. Dans le cas combinatoire, X est un ensemble discret. À chaque solution $x \in X$ est associé un vecteur objectif $z \in Z$ sur la base d'un vecteur de fonctions $f : X \rightarrow Z$ tel que $z = (z_1, \dots, z_n) = f(x) = (f_1(x), \dots, f_n(x))$. L'ensemble $Z = f(X)$ représente les points réalisables dans l'espace objectif. **La figure II.1** présente la liaison entre l'espace décisionnel et l'espace objectif.

Sans perte de généralité, nous supposerons par la suite que $Z \subseteq R^n$ et que les fonctions objectif sont à minimiser. Contrairement à l'optimisation mono-objectif, la solution d'un problème multi-objectif n'est pas unique, mais c'est un ensemble de solutions non dominées, connu comme l'ensemble des solutions Pareto optimales. **[II.3]**

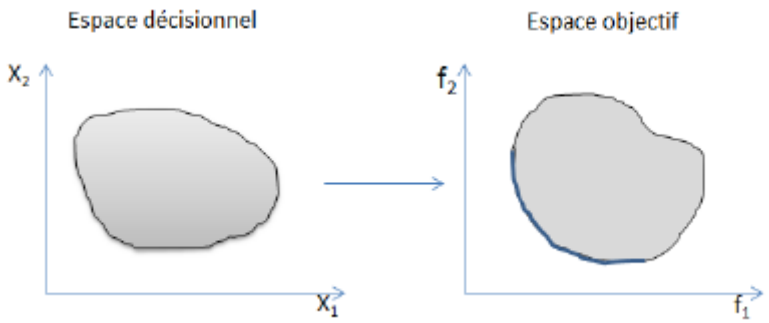


Figure II.1 : Espace décisionnel et espace objectif d'un MOP (cas de deux variables de décision X1 et X2 et de deux fonctions objectif f1 et f2).

II.2.3.2. Notions de dominance

Dans le cadre de l'optimisation multi-objectif, le décideur raisonne souvent en termes d'évaluation d'une solution sur chaque critère et se place dans l'espace objectif. Néanmoins, contrairement à l'optimisation mono-objectif où il existe une relation d'ordre total entre les solutions réalisables, il n'existe généralement pas une solution unique qui serait à la fois optimale pour chaque objectif, en raison de la nature conflictuelle des objectifs. Ainsi, une relation d'ordre partiel, appelée relation de dominance, est généralement définie. **[II.3]**

II.2.3.2.1. Définition (Dominance de Pareto)

Un vecteur objectif $z \in Z$ domine un vecteur objectif $z' \in Z$ si les deux conditions suivantes sont vérifiées :

$$\forall \mathbf{i} \in \{1, \dots, \mathbf{n}\}, \mathbf{z_i} \leq \mathbf{z'_i} \qquad \textbf{(1).}$$

$$\exists \mathbf{j} \in \{1, \dots, \mathbf{n}\}, \mathbf{z_j} \leq \mathbf{z'_j} \qquad \textbf{(2).}$$

Si **z** domine z′, cette relation sera notée $\mathbf{z} < \mathbf{z'}$**.** Par extension, une solution $\mathbf{x} \in \mathbf{X}$ domine une solution $\mathbf{x}' \in \mathbf{X}$, notée $x < x'$, ssi $f(x) < f(x')$. **La figure II.2** illustre la définition de la notion de dominance. Il existe des relations de dominance autres que celle de Pareto. **[II.3]**

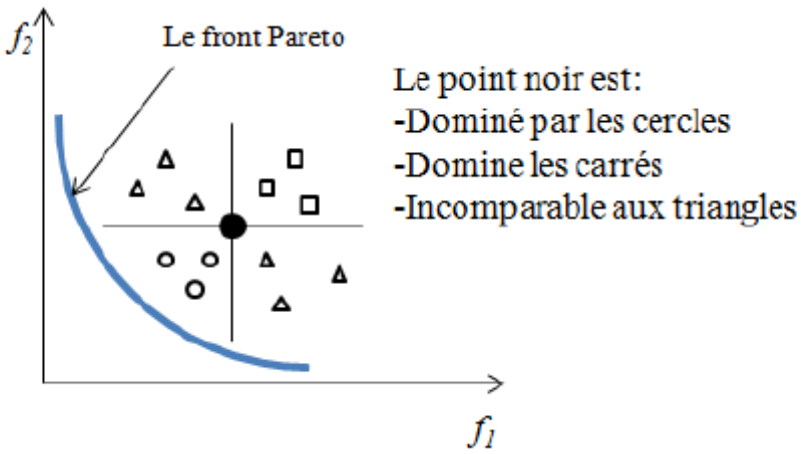


Figure II.2 : Relation de dominance (cas de deux objectifs à minimiser).

II.2.3.3. Optimalité de Pareto

La définition de solution Pareto optimale provient directement du concept de dominance.

Le concept a été proposé initialement par F.Y. Edgeworth **[II.4]** et généralisé par V. Pareto **[II.5]**

II.2.3.3.1. Définition (solution Pareto optimale)

Une solution $x \in X$ est Pareto optimale (ou non dominée) Ssi $\forall x' \in X, x' \prec x$.

Une solution Pareto optimale peut être considérée comme optimale, du fait qu'il n'est pas possible de trouver une solution permettant d'améliorer la valeur d'un objectif sans dégrader celle d'au moins un autre objectif. L'ensemble exact de toutes les solutions Pareto optimales sera noté X_E. Il constitue l'ensemble des solutions Pareto optimales (ensemble de solutions non-dominées). L'image de cet ensemble dans l'espace objectif représente le front de Pareto, et sera noté Z_N. **[II.3]**

II.2.3.3.2. Définition (Ensemble Pareto optimal)

Etant donné un MOP (f, X), l'ensemble Pareto optimal est défini comme suit:

$$X_E = \{x \in X \mid \nexists\, x' \in X, x' \prec x\}. \quad \textbf{[II.3]}$$

II.2.3.3.3. Définition (Front Pareto)

Etant donné un MOP (f, X) et son ensemble Pareto optimal X_E, le front Pareto est défini comme suit:

$$Z_N = \{f(x) \mid x \in X_E\}$$ **(Voir figure II.2). [II.3]**

II.2.4. Optimisation multi-objectif et aide à la décision

La résolution d'un problème multi-objectif consiste à déterminer un ensemble de solutions Pareto optimales. Il est nécessaire de faire intervenir l'humain à travers un décideur pour le choix final de la solution à garder. La première décision qui doit être prise lors du traitement d'un problème multi-objectif porte sur la façon de combiner la recherche et les processus décisionnels. Ceci peut être fait selon l'une des trois méthodes suivantes: **[II.6]**

1. **A priori :** Dans ce cas, le décideur intervient en aval du processus d'optimisation, pour fournir des préférences sur le problème à résoudre, afin d'aider la méthode de résolution dans sa recherche. En pratique, ceci revient à transformer le problème d'optimisation multi-objectif du départ en un problème mono-objectif. Ce dernier peut être résolu par une méthode dont une seule exécution permettra d'obtenir la solution recherchée. Cette approche nécessite une bonne connaissance a priori du problème. Elle est rapide, mais il faut cependant prendre en compte le temps de modélisation des préférences et la possibilité pour le décideur de ne pas être satisfait de la solution trouvée et de relancer la recherche avec une autre formulation des préférences.

2. **A posteriori :** La méthode de résolution cherche à fournir au décideur un ensemble de solutions Pareto optimales bien réparties. Il peut alors, au regard de cet ensemble de solutions, choisir celle qui lui semble la plus appropriée au vu de ses préférences. Ainsi il n'est plus nécessaire de modéliser les préférences du décideur, mais il faut désormais fournir un ensemble de solutions bien réparties, ce qui peut être difficile et coûteux en termes de temps de calcul.

3. **Interactive :** Dans cette approche, il existe une coopération directe et itérative entre le décideur et la méthode de résolution. Le décideur intervient pendant la recherche en ajustant ses préférences afin de guider la recherche. Cette approche permet de bien prendre en compte les préférences du décideur, mais nécessite sa présence tout au long du processus de recherche.

Les approches présentées ci-dessus ont chacune ses forces et ses faiblesses. Le choix d'une méthode ou d'une autre dépend des propriétés du problème à résoudre et des compétences du décideur.

➔ **Dans le cadre de notre problématique d'ordonnancement de workflows dans le cloud, nous nous plaçons dans le cas de l'approche a posteriori.**

II.2.5. Méthodologies de résolution

Les méthodes de résolution d'un problème d'optimisation sont nombreuses et variées. La **figure II.3** montre une éventuelle hiérarchie de celles-ci. Ces méthodes sont séparées tout d'abord en deux grandes familles : les méthodes exactes et les méthodes approchées.

1. **Les méthodes exactes** qui assurent de trouver l'optimum global, mais elles sont souvent très coûteuses en temps de calcul pour des problèmes NP-complets ou ayant plusieurs critères. Parmi ces méthodes, on peut citer : le Branch & Bound **[II.7]**, l'algorithme A* **[II.8]** et la programmation dynamique **[II.9]**.

2. **Les méthodes approchées** qui produisent des solutions proches de l'optimum en un temps raisonnable. Elles sont pratiques pour résoudre des instances de problèmes de grande taille, mais elles ne garantissent pas de trouver l'optimum global.

➔ **Dans notre problématique, nous avons tendance à utiliser les méthodes approchées.**

Les méthodes approchées sont réparties en deux sous-catégories :

- Les heuristiques.
- Les méthodes d'approximation.

Les heuristiques sont divisées en deux sous classes:

1. les heuristiques dédiées à un problème donné **[II.10].**
2. les méta-heuristiques qui sont des méthodes génériques non dédiées à un problème spécifique **[II.11].**

➔ **Dans le cadre de cette thèse, notre intérêt porte sur les méta-heuristiques pour la résolution de notre problème.**

➢ **La figure II.3 montre les méthodes de résolution d'un problème d'optimisation :**

II.2.6. Les métaheuristiques

II.2.6.1. Définition

Les méta-heuristiques sont une famille d'algorithmes stochastiques destinés 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**'. Leur capacité à résoudre un problème à partir d'un nombre minimale d'informations est contrebalancée par le fait qu'elles n'offre aucune garantie quant à l'optimalité de la meilleure solution trouvée. Seule une approximation de l'optimum global est donnée. Cependant ce constat n'est pas forcément un désavantage, étant donné qu'on préférera toujours une approximation de l'optimum global trouvé rapidement qu'une valeur exacte trouvé dans un temps rédhibitoire **[II.12].**

Les méta-heuristiques se caractérisant par leur capacité à résoudre des problèmes très divers, elles se prêtent naturellement à des extensions. Pour illustrer celles-ci, nous pouvons citer :

- **Les méta-heuristiques pour l'optimisation multi-objectif :** où il faut optimiser plusieurs objectifs contradictoires. Le but ne consiste pas ici à trouver un optimum global, mais à trouver un ensemble d'optima, qui forment une surface de compromis pour les différents objectifs du problème.
- **Les méta-heuristiques pour l'optimisation multimodale :** où l'on ne cherche plus l'optimum global, mais l'ensemble des meilleurs optima globaux et/ou locaux.
- **Les méta-heuristiques pour l'optimisation de problèmes bruités:** où il existe une incertitude sur le calcul de la fonction objectif, dont il faut tenir compte dans la recherche de l'optimum.
- **Les méta-heuristiques pour l'optimisation dynamique :** où la fonction objective varie dans le temps, ce qui nécessite d'approcher l'optimum à chaque pas de temps.
- **Les méta-heuristiques hybrides :** qui consistent à combiner différentes méta-heuristiques, afin de tirer profit des avantages respectifs.

Plusieurs classifications des méta-heuristiques ont été proposées, la plupart distinguent globalement deux familles comme illustré dans (**la figure II.4)**:

1. **Les méta-heuristiques à base de solution unique (S-méta-heuristiques) :** consistent à manipuler et améliorer une seule solution, tant que cela est possible (telles que les algorithmes de recherche locale **[II.13]**, de recherche tabou **[II.14]**, de recuit simulé **[II.15]**, etc).

2. **Les méta-heuristiques à base de population de solutions (P-méta-heuristiques) :** consistent à manipuler et améliorer un ensemble de solutions, nommé population, évolue en parallèle (telles que les algorithmes évolutionnaires, algorithmes à essaim de particules, etc).

- **Ces deux familles ont des caractéristiques complémentaires :**

➢ **Les S-méta-heuristiques** sont axées sur l'exploitation de l'espace de recherche, elles ont la capacité d'intensifier la recherche dans des régions locales prometteuses (afin de trouver une meilleure solution).

➢ **Les P-méta-heuristiques** sont orientées exploration, elles permettent une meilleure diversification de l'espace de recherche.

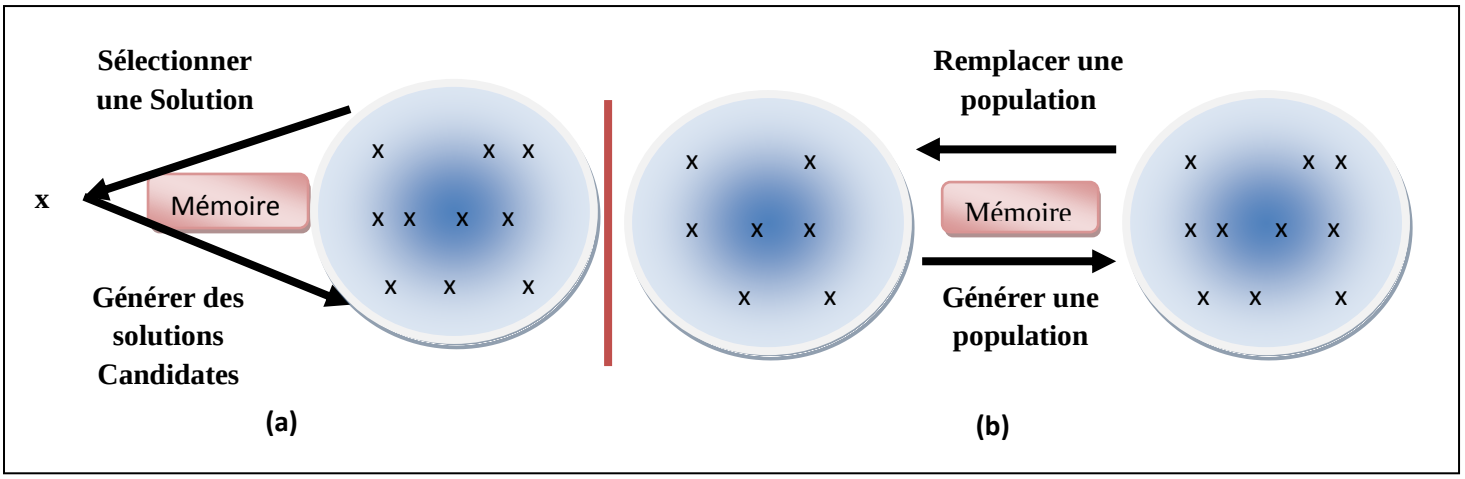


Figure II.4: Principes généraux d'une méta-heuristique à base de: (a) solution unique, (b) population. [II.10]

II.2.7. Concepts communs à tout type de méta-heuristique

Pour la conception de toute méta-heuristique, deux concepts fondamentaux doivent être considérés : la représentation des solutions manipulées par l'algorithme, et l'évaluation de ces solutions dans l'espace objectif. **[II.3].**

1. **Représentation de solution :** La conception de tout type de méta-heuristique nécessite une représentation (codage) d'une solution. La représentation joue un rôle majeur dans l'efficacité de toute méta-heuristique et constitue donc une étape de conception essentielle. La représentation doit être pertinente et adaptée au problème d'optimisation en question. En outre, le choix d'une représentation aura une influence considérable sur la façon dont les solutions seront manipulées par les opérateurs de recherche et lors de l'étape d'évaluation. On peut trouver plusieurs représentations pour un problème donné. Aussi, de nombreuses représentations peuvent être appliquées aux différents problèmes d'optimisation. Dans la littérature, quatre principales représentations (pour l'optimisation combinatoire) peuvent être distinguées :
   - ✓ Représentations basées sur des vecteurs de valeurs réelles.
   - ✓ Représentations discrètes.
   - ✓ Représentations binaires.
   - ✓ Représentations des permutations.

- **La figure II.5 illustre un exemple de chaque représentation.**

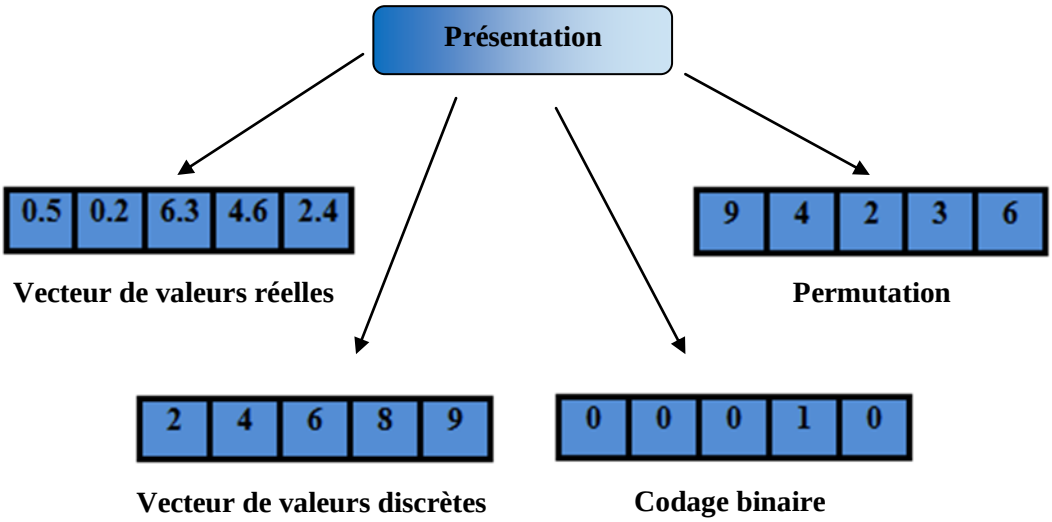


Figure II.5 : Principaux codages pour des problèmes d'optimisation.

2. **Evaluation :** L'étape d'évaluation correspond au calcul et l'affectation de valeurs objectifs pour une solution donnée. Dans le cas de l'optimisation mono-objectif, une fonction objective unique formule le but à atteindre. Elle associe à chaque solution une valeur scalaire, qui donne la qualité de la solution permettant ainsi d'avoir un ordre total de toutes les solutions de l'espace de recherche. Pour un problème d'optimisation multi-objectif, il existe plusieurs fonctions objectifs. De ce fait, l'évaluation consiste à associer, à chaque solution, un vecteur de valeurs dont la taille correspond au nombre d'objectifs considérés et chaque élément du vecteur quantifie la qualité de la solution par rapport à la fonction objectif correspondante. En conséquence, il n'existe plus d'ordre total entre les solutions, mais plutôt un ordre partiel, établi grâce à une relation de dominance. L'étape d'évaluation est un élément essentiel dans la conception d'une méta-heuristique. Elle permet d'orienter la méthode vers les bonnes solutions de l'espace de recherche. Notez qu'en pratique, l'évaluation est généralement l'étape la plus coûteuse, en termes de temps de calcul, de la méthode de résolution.

II.2.8. Concepts communs à tout type de méta-heuristique pour le cas multi-objectif

La résolution d'un MOP exige des contraintes supplémentaires, qui n'apparaissent pas dans l'optimisation mono-objectif. Toute méta-heuristique dédiée à la résolution d'un problème multi-objectif doit tenir compte de trois concepts, à savoir :

1. **Affectation de la qualité (« fitness ») d'une solution :** Le rôle de ce concept est de définir la façon d'évaluer la qualité d'une solution. Il existe quatre approches d'affectation de la qualité : les approches scalaires, les approches basées sur un critère, les approches basées sur la dominance entre solutions (ou approches Pareto) et les approches basées sur des indicateurs de performance **(voir figure II.3)** :

   1.1. Approches scalaires : Ces approches consistent à transformer le MOP initial en un problème mono-objectif ou à un ensemble de problèmes mono-objectif, dont il existe de nombreuses méthodes de résolution. Parmi ces méthodes, nous pouvons citer la méthode d'agrégation **[II.16]**, qui est une méthode très populaire, consistant à combiner les différentes fonctions du problème en une seule fonction objectif, généralement de façon linéaire, à l'aide d'une somme pondérée. La méthode ε-contraintes ou les méthodes basées sur un point de référence font partie aussi des approches scalaires **[II.17]**. Ces approches requièrent cependant une bonne connaissance du problème à résoudre pour être mises en place.

   1.2. Approches basées sur un critère : Ces approches ne transforment pas le MOP en un problème mono-objectif. Elles utilisent des opérateurs de recherche qui traitent séparément les différents objectifs. Deux groupes de méthodes existent dans la littérature :

   a. **La sélection lexicographique [II.18],** la sélection est réalisée suivant un ordre hiérarchique défini par le décideur entre les fonctions objectifs. Ces fonctions sont ensuite traitées les unes à la suite des autres, selon cet ordre

préétabli. Plusieurs méta-heuristiques ont été utilisées pour résoudre des MOP en utilisant la sélection lexicographique **[II.10]**.

b. **La sélection parallèle,** le premier travail a été publié par Schaffer **[II.19]**. Il est basé sur un algorithme génétique. L'algorithme développé **VEGA (V**ector **E**valuated **G**enetic **A**lgorithm**)** sélectionne les solutions courantes du front Pareto suivant chaque objectif, indépendamment des autres (sélection parallèle).

1.3. Approches basées sur la dominance entre solutions (Pareto) : Contrairement aux approches qui regroupent les différents objectifs ou les traitent séparément, les approches Pareto utilisent la notion de dominance dans leur processus de recherche, afin de classer les solutions de la population et leur affecter des « rangs» (et de sélectionner celles qui vont la faire converger vers un ensemble de solutions efficaces). Cette idée a été introduite pour la première fois par « Goldberg » dans les algorithmes génétiques **[II.20]**. Trois stratégies principales peuvent être distinguées **[II.21] (voir figure II.3) :**

a. Rang de dominance (dominance-rank) : Cette technique consiste à calculer le nombre de solutions qui dominent la solution considérée **(figure II.6.a).**

b. Compte de dominance (dominance-count) : La valeur de fitness d'une solution correspond au nombre de solutions dominées par la solution considérée **(figure II.6.b).**

c. Profondeur de dominance (dominance-depth) : Cette stratégie consiste à classer les solutions de la population en différents fronts. Une solution appartenant à un front ne domine aucune solution de ce même front. Le front Pareto correspond au front 1, le front 2 est le front obtenu en enlevant les solutions du front 1, et ainsi de suite... Ce principe est illustré dans la **(figure II.6.c).**

1.4. **Approches basées sur des indicateurs de performance :** L'affectation de la qualité d'une solution se fait en utilisant des indicateurs de performance qui sont généralement basés sur la notion de dominance. Parmi les algorithmes multi-objectifs basés sur des indicateurs nous trouvons l'algorithme **IBEA (I**ndicator **B**ased **E**volutionary **A**lgorithm**) [II.22]**.

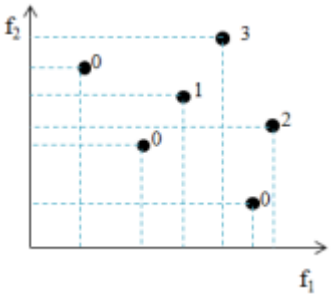


Figure II.6.a : Rang dominance.

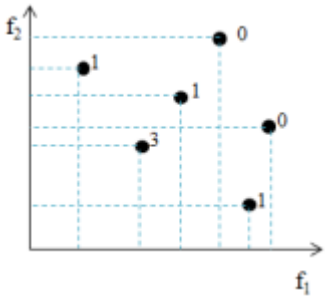


Figure II.6.b : Compte dominance.

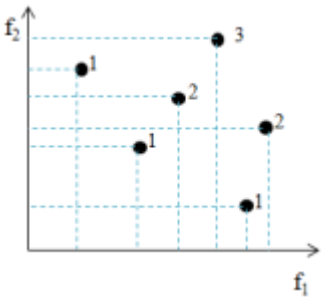


Figure. II.6.c : Profondeur dominance.

2. **La préservation de la diversité :** L'application de méta-heuristiques pour la résolution d'un problème d'optimisation multi-objectif implique de trouver une approximation de l'ensemble Pareto optimal. Cette approximation doit contenir un ensemble de solutions dont l'image dans l'espace objectif est à la fois proche du front Pareto (convergence) et suffisamment riche ou bien distribuée le long de la frontière (diversité). Ainsi, lorsque l'on parle de diversité en optimisation multi-objectif, on se réfère à l'espace objectif. Afin d'assurer cette diversité, la densité de solutions autour d'une solution donnée est calculée afin d'évaluer la répartition des solutions les unes par rapport aux autres. Il existe plusieurs méthodes pour définir cette notion de densité

(généralement basée sur la distance euclidienne), mais la finalité reste la même : pénaliser les zones de trop forte densité pour améliorer la répartition des solutions. La **figure II.7-(a)** montre une approximation de l'ensemble Pareto optimal ayant une bonne convergence et une bonne diversité. Dans la **figure II.7-(b),** les solutions sont très bien réparties mais sont loin du front de Pareto. Enfin la **figure II.7-(c)** montre que les solutions sont cette fois très proches du front de Pareto, mais ne couvrent pas certaines régions, ce qui provoque une perte d'information importante aux yeux du décideur.

3. **L'élitisme :** L'utilisation de l'élitisme doit permettre de préserver les meilleures solutions Pareto trouvées durant la recherche. Pour ce faire, une population de « bonnes » solutions est maintenue en parallèle de celles générées lors de la recherche. Cette population de solutions est appelée « archive ». L'archive peut ne pas être utilisée dans le processus de recherche, on parle alors d'archive « passive », ou au contraire être régulièrement sollicitée pour l'obtention de nouvelles solutions (archive « active »). Comme toute utilisation d'élitisme, il est important, lors de l'utilisation d'archive active, de prévenir la convergence prématurée de l'algorithme. La taille de l'archive peut être fixe ou non. La mise à jour des archives est principalement basée sur la notion de dominance et il est possible d'inclure un mécanisme de diversification, comme c'est le cas pour les solutions générées lors de la recherche.

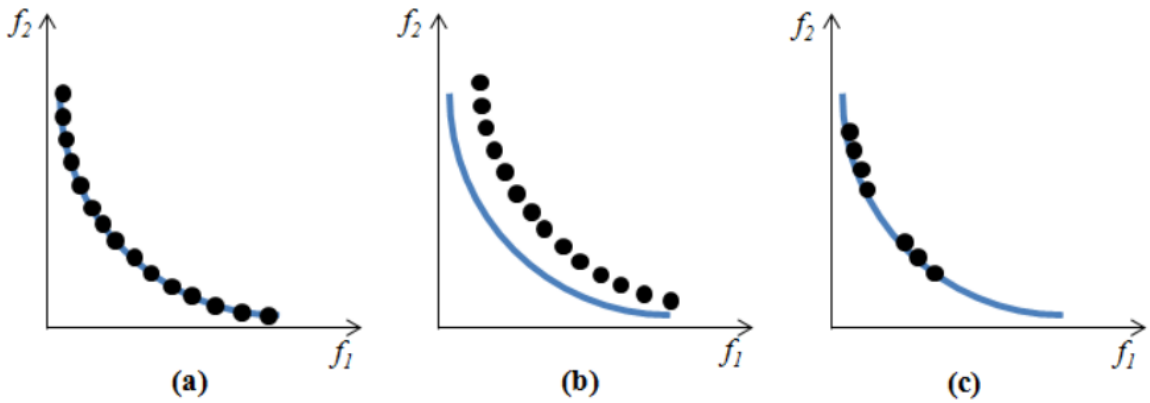


Figure II.7 : Illustration de la convergence et de la diversité en optimisation multi-objectif [II.3].

II.2.9. Méta-heuristiques à base de population de solutions

Contrairement aux S-méta-heuristiques, les P-méta-heuristiques travaillent sur un ensemble de points de l'espace de recherche, en commençant avec une population de solutions initiales puis elles s'efforcent de l'améliorer au fur et à mesure des itérations. L'intérêt de ces méthodes est d'explorer un très vaste espace de recherche et d'utiliser la population comme facteur de diversité ; de plus, elles sont très adaptées et très largement utilisées pour l'optimisation multi-objectif. Parmi ces méta-heuristiques, on distingue les algorithmes évolutionnaires, basés sur la théorie de l'évolution de Darwin **[II.23]**. L'intelligence en essaim basée sur le comportement des espèces vivant en colonies, comme les fourmis ou les abeilles, ou les systèmes immunitaires artificiels, qui miment les systèmes immunitaires biologiques.

Dans ce manuscrit, nous considérons principalement les algorithmes évolutionnaires, plus précisément les algorithmes génétiques.

II.2.9.1. Les algorithmes évolutionnaires

Les algorithmes évolutionnaires sont des algorithmes d'optimisation inspirés par l'évolution biologique des espèces et sont les algorithmes à base de population les plus utilisés. Les algorithmes évolutionnaires les plus populaires sont **les algorithmes génétiques [II.3]**.

II.2.9.1.1. Les algorithmes génétiques (GA)

II.2.9.1.1.1. Définition

Les Algorithmes Génétiques sont basés sur la théorie de l'évolution de Darwin. Ils consistent à faire évoluer une population de dispositifs à l'aide de différents opérateurs : sélection, croisements, mutations. Ils sont en particulier utilisés pour les problèmes d'optimisation comportant de nombreux paramètres et des objectifs multiples **[II.24]**.

II.2.9.1.1.2. La théorie de Darwin

En **(1859)**, Darwin montre que l'apparition d'espèces distinctes se fait par le biais de la sélection naturelle de variations individuelles **(voir figure II.9).** Cette sélection naturelle est fondée sur la lutte pour la vie, due à une population tendant naturellement à s'étendre mais disposant d'un espace et de ressources finis. Il en résulte que les individus les plus adaptés tendent à survivre plus longtemps et à se reproduire plus aisément **[II.24]**

II.2.9.1.1. 3. Définitions et notations

Soit à optimiser une fonction **F** à valeurs réelles définie sur un espace **Ω**, supposé muni d'une distance **d**. Le parallèle avec l'évolution naturelle a entrainé l'apparition d'un vocabulaire spécifique **[II.25]** :

- La fonction objectif **F** est appelée fonction performance, ou fonction d'adaptation (fitness).
- Les points de l'espace de recherche **W** sont appelés des individus.
- Les P-uplets d'individus sont appelés des populations.
- On parlera d'une génération pour la boucle principale de l'algorithme.

II.2.9.1.1.4. Le squelette

La pression de l'environnement est simulée à l'aide de la fonction d'adaptation **F**, et le principe darwinien, les plus adaptés survivent et se reproduisent, est implanté de la manière suivante **[II.25]**:

- **Initialisation** de la population P_0 en choisissant **P** individus dans **W**, généralement par tirage aléatoire avec une probabilité uniforme sur **W**.
- **Evaluation** des individus de P_0 (i.e. calcul des valeurs de **F** pour tous les individus).
- La génération **i** construit la population P_i à partir de la population P_{i-1}.
    - ✓ **Sélection** des individus les plus performants (au sens de **F**) de P_{i-1}.
    - ✓ Application (avec une probabilité donnée) des opérateurs génétiques aux parents sélectionnés, ce qui génère de nouveaux individus, les enfants : on parlera de mutation pour les opérateurs unaires, et de croisement pour les opérateurs binaires (ou n-aires).
    - ✓ **Evaluation** des enfants.
    - ✓ **Remplacement** des parents au moyen d'une sélection darwinienne parmi les enfants, avec participation éventuelle des parents.
- L'évolution stoppe quand le niveau de performance souhaité est atteint, ou qu'un nombre fixé de générations s'est écoulé sans améliorer l'individu le plus performant.

➢ **La figure suivante détaille squelette d'un algorithme évolutionnaire :**

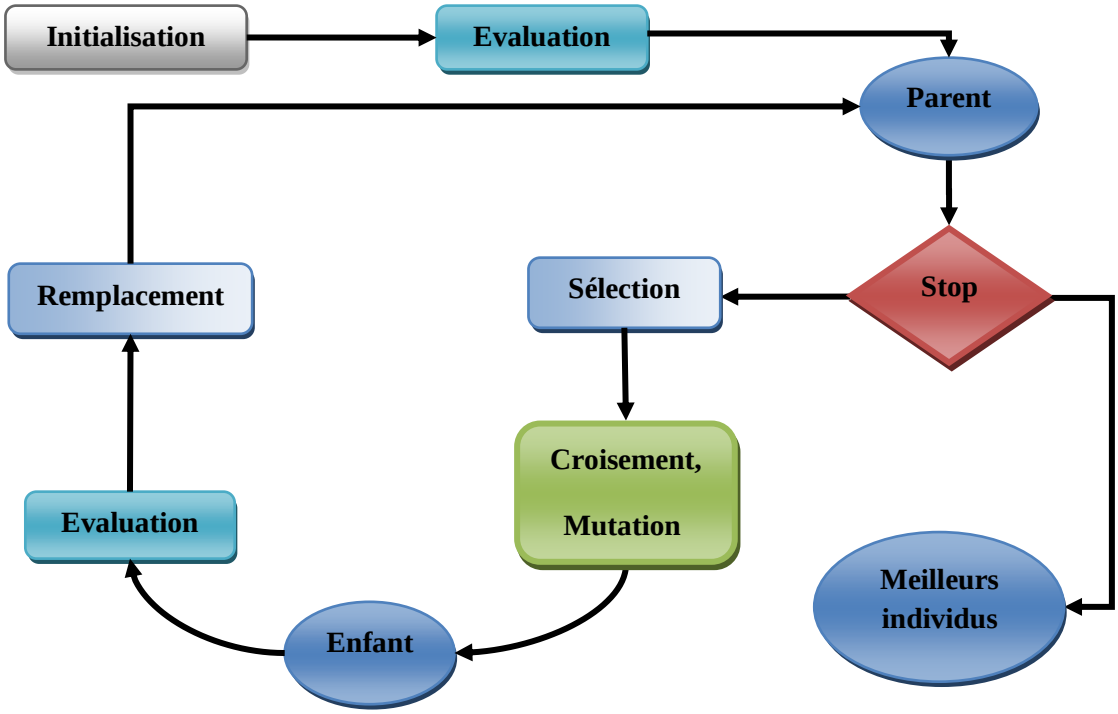


Figure II.8 : Squelette d'un algorithme évolutionnaire.

➔ **Nous allons maintenant détailler les principaux composants d'un algorithme évolutionnaire, en donnant des exemples**

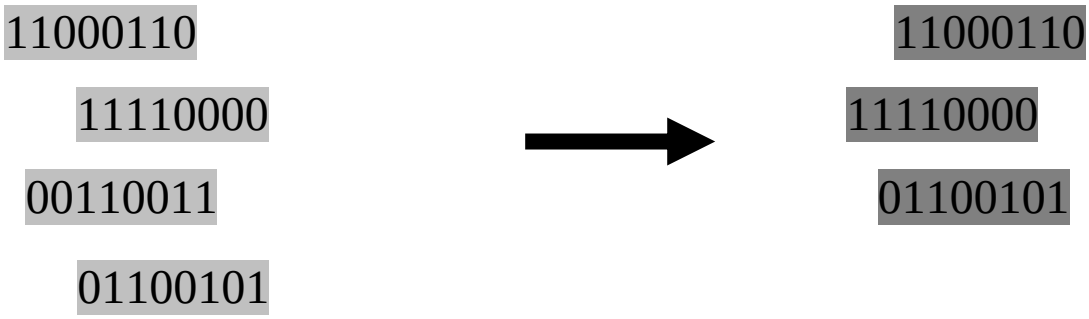
1. **L'initialisation et le Codage d'une population :** Les algorithmes génétiques agissent sur une population d'individus, et non pas sur un individu isolé donc Le premier pas dans l'implantation des algorithmes génétiques est de créer une population d'individus initiaux qui consiste à choisir un ensemble de solutions potentielles au problème d'optimisation posé, dont chaque solution potentielle va représenter un individu (dit aussi chromosome dans la terminologie de Holland). Tous ces individus vont être rassemblés dans ce que l'on nommera la population initiale. Par analogie avec la biologie, chaque individu de la population est codé par un chromosome ou génotype. Une population est donc un ensemble de chromosomes. Chaque chromosome code un point de l'espace de recherche. L'efficacité de l'algorithme génétique va donc dépendre du choix du codage d'un chromosome. Il existe trois principaux type de codage : binaire, gray ou réel.

2. **Sélection :** La sélection consiste à choisir les individus les mieux adaptés afin d'avoir une population de solution la plus proche de converger vers l'optimum global. Cet opérateur est l'application du principe d'adaptation de la théorie de Darwin.

   Il existe plusieurs techniques de sélection. Voici les principales utilisées :

- **Sélection par rang :** Cette technique de sélection choisit toujours les individus possédant les meilleurs scores d'adaptation.
- **Probabilité de sélection proportionnelle à l'adaptation :** Technique de la roulette ou roue de la fortune, pour chaque individu, la probabilité d'être sélectionné est proportionnelle à son adaptation au problème.
- **Sélection par tournoi :** Cette technique utilise la sélection proportionnelle sur des paires d'individus, puis choisit parmi ces paires l'individu qui a le meilleur score d'adaptation.
- **Sélection uniforme :** La sélection se fait aléatoirement, uniformément et sans intervention de la valeur d'adaptation.

➔ Voici un exemple avec des individus en représentation binaire une fois la sélection effectuée :

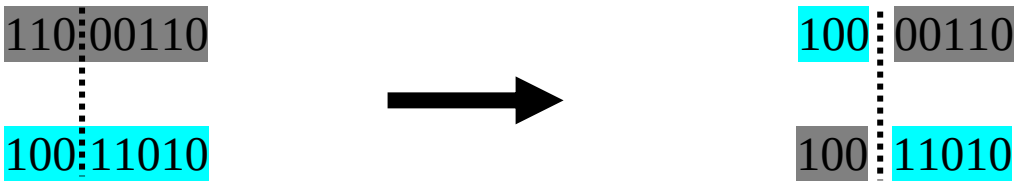


On réalise ensuite un croisement entre deux chromosomes parmi la population restante...

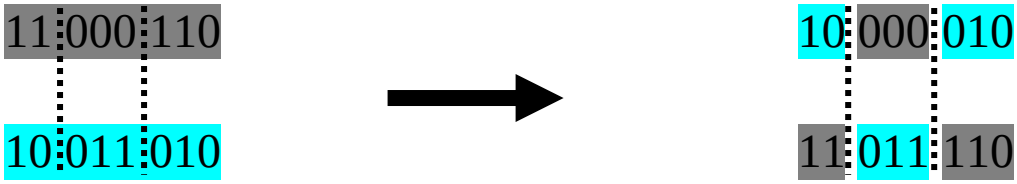
3. **Croisement :** Le croisement, ou enjambement, crossing-over, est le résultat obtenue lorsque deux chromosomes partage leurs particularités. Celui-ci permet le brassage génétique de la population et l'application du principe d'hérédité de la théorie de Darwin.

Il existe deux méthodes de croissement : simple ou double enjambement.

1. **Le simple enjambement** : consiste à fusionner les particularités de deux individus à partir d'un pivot, afin d'obtenir un ou deux enfants :



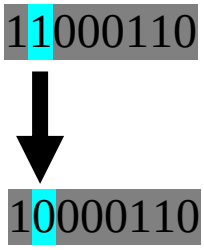
2. **Le double enjambement** : repose sur le même principe, sauf qu'il y a deux pivots :



On réalise ensuite une mutation sur les enfants obtenues lors du croisement...

1. **Mutation :** La mutation consiste à altérer un gène dans un chromosome selon un facteur de mutation. Ce facteur est la probabilité qu'une mutation soit effectuée sur un individu. Cet opérateur est l'application du principe de variation de la théorie de Darwin et permet, par la même occasion, d'éviter une convergence prématurée de l'algorithme vers un extremum local.

➔ **Voici un exemple de mutation sur un individu ayant un seul chromosome :**



4. **Fonction d'évaluation et fonction fitness :** Pour calculer le coût d'un point de l'espace de recherche, on utilise une fonction d'évaluation. L'évaluation d'un individu ne dépendant pas de celle des autres individus, le résultat fournit par la fonction d'évaluation va permettre de sélectionner ou de refuser un individu pour ne garder que les individus ayant le meilleur coût en fonction de la population courante : c'est le rôle de la fonction fitness.

II.3. Conclusion

Dans ce chapitre, nous avons présenté d'abord le cadre général de l'optimisation multi-objectif, qui nous sera utile pour modéliser et résoudre le problème d'ordonnancement de workflows. Puis, nous avons donné quelques concepts sur les méta-heuristiques et nous avons présenté les algorithmes génétiques utiles pour l'ordonnancement de workflows dans le cloud computing.

Chapitre III

Modélisation et tests

III.1. Introduction

Dans ce chapitre nous présentons d'abord l'outil de développement Eclipse et le framework **JMetal,** puis nous présentons l'algorithme l'ordonnancement de workflows HEFT. Par la suite, nous présentons une modélisation pour résoudre le problème d'ordonnancement bi-objectif par la méta-heuristique **NSGA-II**. Enfin nous terminons ce chapitre par quelques résultats obtenus après les tests.

III.2. Environnement de développement

Pour la réalisation de ce travail, nous avons eu recours aux environnements suivants:

III.2.1. Environnement Matériel

Pour développer l'application, nous avons utilisé comme environnement matériel un ordinateur COMPAC qui possède comme caractéristiques :

- Processeur : Intel(R) Celeron (R) CPU B800@1.50GHz.
- Mémoire RAM : 4 GO.
- Disque dur : 100 GO.

III.2.2. Environnement Logiciel

- Windows 7 comme Système d'exploitation.
- JAVA Eclipse version « Juno service release 1 ».
- Dev- C++.
- StarUML pour modéliser les diagrammes UML.
- Le simulateur JMetal.

III.2.2.1. Présentation de l'outil de développement

Figure III.1 : Logo Eclipse.

➔ **Au lancement du logiciel l'écran a l'aspect suivant :**

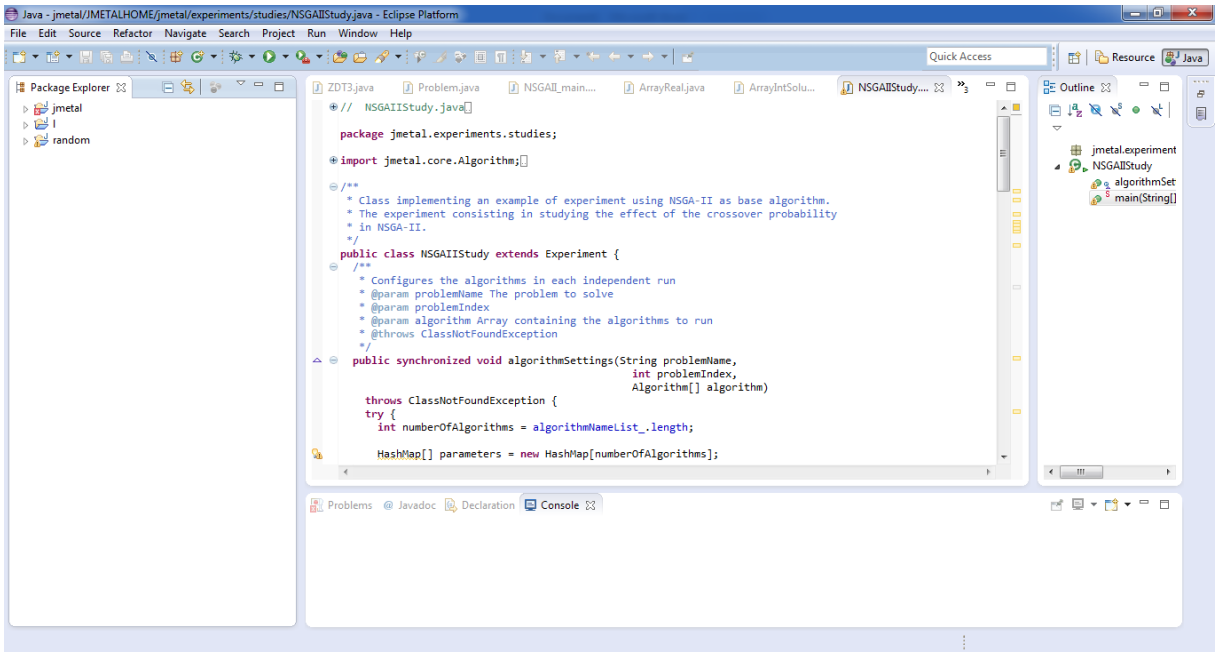


Figure III.2: L'interface principale d'Eclipse.

➔ La partie supérieure de l'écran donne l'accès au système de menu et la barre d'outils. Sur la première ligne :

File **:** Permet d'ouvrir un nouveau fichier, un nouveau projet, enregistrer votre travail et imprimer.

Edit **:** Donne l'accès aux fonctions copier coller classiques ainsi qu'à des outils de présentations.

Run **:** Permet de lancer l'exécution d'un programme.

III.2.2.2. Présentation du langage de programmation java

Java est un langage de programmation orienté objet, développé par Sun Microsystems et destiné à fonctionner dans une machine virtuelle, il permet de c réer des logiciels compatibles avec des nombreux systèmes d'exploitation tels que UNIX, Windows, Mac OS, avec peu ou pas de modifications.

Java et non seulement un langage de programmation puissant conçu pour être sûr, inter

plateformes et international, mais aussi un environnement de développement qui est continuellement étendu pour fournir des nouvelles caractéristiques et des bibliothèques permettant de gérer de manière élégante des problèmes traditionnellement complexes dans les langages de programmation classiques, tels que le multithreading, les accès aux bases des données, la programmation réseau, l'informatique répartie. **[III.1]**

III.2.2.3. Présentation du langage de programmation C++

C++ est un des langages les plus célèbres au monde. Très utilisé, notamment dans le secteur des jeux vidéo qui apprécie ses performances et ses possibilités, le C++ est désormais incontournable pour les développeurs.

Le C++ est le descendant du langage C. Ces deux langages, bien que semblables au premier abord, sont néanmoins différents. Le C++ propose de nouvelles fonctionnalités, comme la programmation orientée objet (POO). Elles en font un langage très puissant qui permet de programmer avec une approche différente du langage C. **[III.2]**

III.2.2.4. Présentation du simulateur JMetal

JMetal (Java Meta-heuristique Algorithme) signifie Algorithmes méta-heuristiques en Java, et il s'agit d'un framework orienté objet Java pour une optimisation multi-objectif avec des techniques méta-heuristiques.

JMetal fournit un ensemble riche de classes qui peuvent être utilisées comme éléments constitutifs de techniques multi-objectifs, de cette façon, en profitant de la réutilisation des codes, les algorithmes partagent les mêmes composants de base, tels que l'implémentation d'opérateurs génétiques et les estimateurs de densité, facilitant ainsi non seulement le développement de nouvelles techniques multi-objectifs, mais aussi la réalisation de différents types d'expériences. **[II.1]**

III.2.2.4.1. Installation

JMetal est écrit en Java, ne nécessitant aucun autre logiciel supplémentaire. L'exigence est d'utiliser Java **JDK** 1,5 ou plus récent. Le code source est regroupé dans un paquet tar.gz qui peut être téléchargé à partir du site http://jmetal.sourceforge.net voir la « **figure III.3** ». Un fichier **« jar »** est également disponible si vous êtes intéressé uniquement par l'exécution des algorithmes fournis. Il existe plusieurs façons de travailler avec les programmes Java. Nous

décrivons brièvement comment installer et Développés avec JMetal en utilisant la ligne de commande dans un terminal texte et dans un Environnement de développement (IDE) Eclipse.

1. **Les étapes de l'installation :**

1.1.Téléchargement du paquet tar.gz :

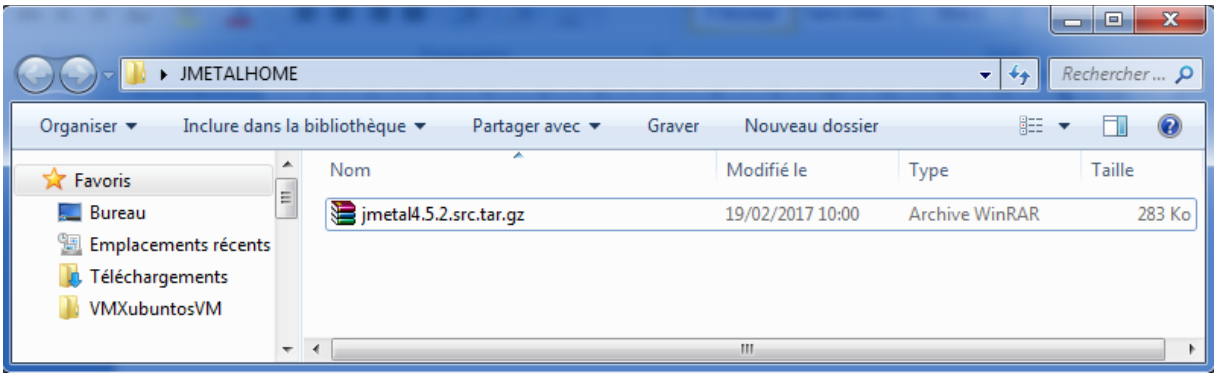


Figure III.3 : Téléchargement du paquet tar.gz.

1.2. Décompression du paquet tar.gz :

Indépendamment de note façon préférée de travailler avec Java, nous devons décompresser le paquet tar.gz, et décompressez le paquet « tar » résultant. En utilisant la ligne de commande, cela peut être fait en tapant:

1. **gzip -d jmetal.tar.gz**
2. **tar xf jmetal.tar**

Ou bien:

1. **tar zxf jmetal.tar.gz**

Remarque : vu que ces commandes sont des commandes « Unix » on utilise « Cygwin » comme interpréteur de commande qui permet de simuler un environnement « Unix » sous « Windows ».

➔ On obtiendra le code source de JMetal dans le répertoire nommé « **JMETALHOME** » dans lequel l'archive est décompressé, qui a la structure décrite dans la « **figure III.3 » :**

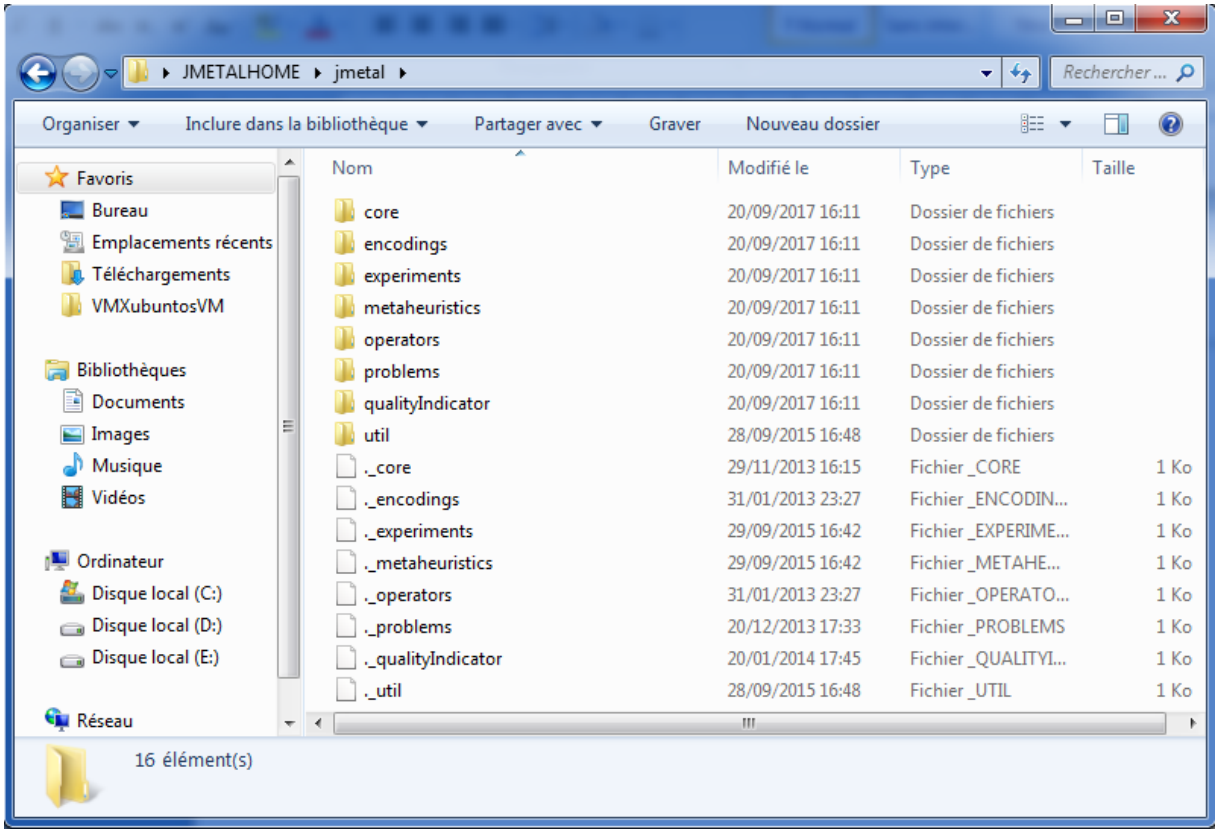


Figure III.4 : La structure du code source de JMetal.

1.3. Définition de la variable d'environnement CLASSPATH

Pour ajouter le répertoire **JMETALHOME** à la variable d'environnement **CLASSPATH**, on tape :

1. **export CLASSPATH=$CLASSPATH:$JMETALHOME**

1.4. Utilisation de JMetal avec eclipce :

1. Sélectionner **File** -> **New** -> **Java Project**.
2. Nommer le projet (par exemple **jMetal**) puis cliqué sur le button «**Next** ».
3. Selectionner «**Link additional source** » et définir **JMETALHOME** comme emplacement de dossier.
4. Cliquer sur «**Finish**».

1.5. Configuration et exécution d'un algorithme :

➔ Nous utilisons **NSGA-II** comme exemple d'algorithme fournis par JMetal:

✓ Pour configurer l'algorithme :

1. Ouvrir le fichier **« NSGAII main.java »** on le sélectionnant dans le paquet « **jmetal.metaheuristics.nsgaII ».**

✓ Pour exécuter l'algorithme :

1. On clique avec le bouton droit de la souris sur **« NSGAII main.java »** dans l'arborescence du projet ou dans la partie vide des fenêtres contenant le fichier.

2. Sélectionnez Exécuter en tant que! Application Java. En conséquence, vous obtenez deux fichiers contenant les solutions optimales de Pareto et le front de Pareto trouvé par l'algorithme. Par défaut, ces fichiers sont nommés « **VAR »** et « **FUN »**, respectivement. Ils se trouvent dans le répertoire où l'espace de travail du projet est stocké.

III.2.2.4.2. Structure d'emballage JMetal

JMetal est composé de huit composants, qui sont représentés à la **figure III.4**. Ils peuvent être des noyaux, des problèmes, des méta-euristiques, des operateurs, des encodages, des indicateurs de qualité, des utilitaires et des expériences. Nous les décrivons brièvement :

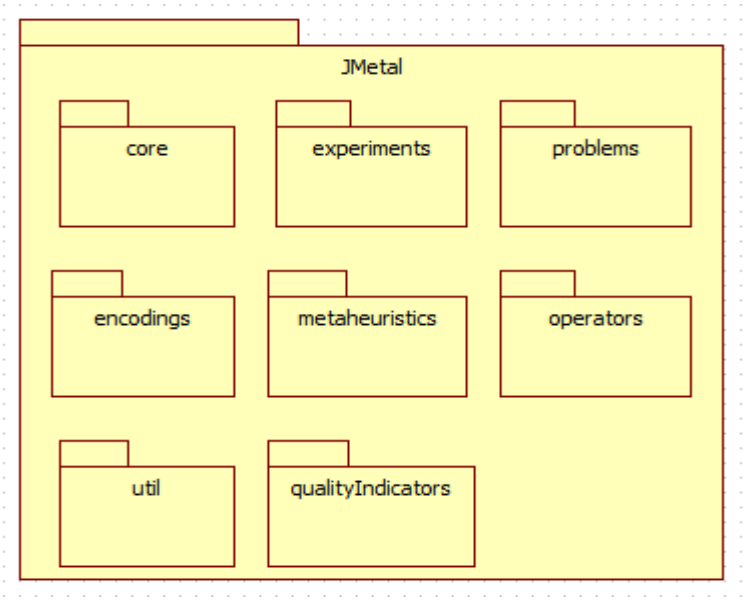


Figure III.5 : La structure de packages JMetal. [II.1]

1. **Noyau (Core) :** Ce paquet contient les ingrédients de base à utiliser par les métaheuristiques développées sous JMetal. Les principes classes dans ce paquet sont: Algorithme, Opérateur, Problème, Variables, Solution, SolutionSet, SolutionType.

2. **Problèmes (Problems) :** Tous les problèmes disponibles dans JMetal sont inclus dans ce package. Nous pouvons trouver des problèmes bien connus (ZDT, DTLZ, WFG) plus d'autres familles de problèmes plus récents (LZ07, CEC2009Competition).En outre nous pouvons trouver d'autres problèmes (Fonseca, Kursawe, Schaffer, OKA2, etc…).

3. **Métaheuristiques (Metaheuristics) :** Ce paquet contient les métaheuristiques mises en œuvre dans JMetal. On site (NSGAII, SPEA2, PAES, PESA-II, GDE3, FastPGA, etc…).

4. **Opérateurs (Operators) :** Ce package contient différents types d'objets operateurs, y compris les operateurs de croisement, de mutation, de sélection et de recherche locale.

5. **Encodages (Encodings) :** Ce paquet contient les représentations des variables de base et les types de solutions, les classes inclus dans ce package sont:

- **Variables (Variable):** Binary, BinaryReal, Int, Real, Permutation, ArrayInt, ArrayReal.

- **Types de solution (solutionType):** ArrayIntSolutionType, ArrayRealSolutionType, ArrayRealAndBinarySolutionType, BinaryRealSolutionType, BinarySolutionType, IntRealSolutionType IntSolutionType, PermutationSolutionType, RealSolutionType.

6. **Utilitaires (Util) :** Ce paquet contient un certain nombre de classes utilitaires tel que la classe « neighborhood » utilisé dans algorithmes évolutifs cellulaires etc…

7. **Indicateurs de qualités (QualityIndicators) :** Contient un certain nombre d'indicateurs de qualité pour évaluer la performance des métaheuristiques multi-objectifs.

8. **Expériences (Expriments) :** Ce paquet contient un ensemble de classes destinées à réaliser des études typiques en optimisation multi-objectifs

III.2.2.4.3. L'architecture de JMetal

Un diagramme de classe UML représentant les principaux composants et leurs relations représenté à la **figure III.5 :**

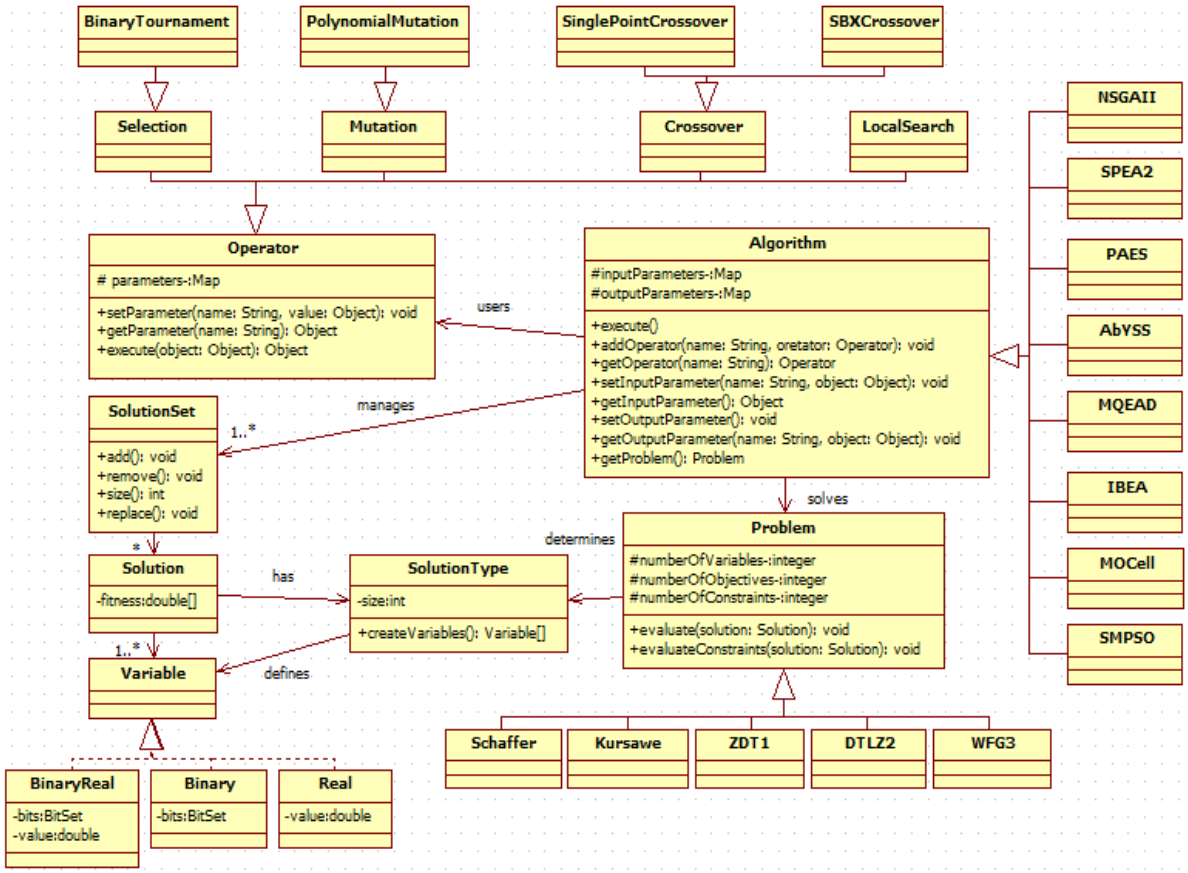


Figure III.6 : Le diagramme de classe de JMetal. [II.1]

III.3. Présentation de l'algorithme d'ordonnancement de workflow HEFT

III.3.1. Définition

HEFT (**H**eterogeneous **E**arliest **F**inish **T**ime) est parmi les ordonnancements de tâches les plus répandus, ayant un des meilleurs rapports performances-complexité (performances élevées et complexité temporelle inférieure). Cette heuristique, proposée par Topcuoglu et al, détermine un ordonnancement total d'un **DAG** sur un environnement hétérogène de manière à minimiser la durée totale d'exécution de l'application (makespan). **[II.3]**

Le principe général de l'algorithme est d'associer chaque tâche au processus le plus approprié (qui minimise sa première heure de fin).

L'algorithme prend le workflow en entrée ainsi que l'ensemble des données qui concerne la métrique de temps, Il fournit comme résultat les ressources choisies par l'algorithme pour chaque tâche, ainsi que le temps total du service pour ces associations tâches-ressources.

III.3.2. Etapes de la méthode HEFT

L'algorithme HEFT se divise en plusieurs étapes, présentées ci-dessous :

Algorithm HEFT
1 : Calculer la priorité (le rang) de chaque tâche ;
2 : Trier les tâches dans une liste d'ordonnancement par ordre décroissant de priorité de tâches ;
3 : while la liste d'ordonnancement est non vide ; **do**
4 : Sélectionner la première tâche i de la liste pour l'ordonnancement ;
5 : f**or** chaque processeur **k do**
6 : Calculer la valeur de EFT (i, k) utilisant la politique d'ordonnancement à base d'insertion ;
7 : end for
8 : Attribuer la tâche i au processeur j qui minimise l'EFT de la tâche i ;
9 : end while +

- **Les phases de l'algorithme HEFT :**

L'algorithme HEFT a deux phases d'exécution

1. **La phase de prioritisation :** les priorités de toutes les tâches sont calculées en utilisant ranku. Ensuite une liste de tâches est générée en triant les tâches par leurs ranku (priorités) décroissants.

2. **La phase de sélection de machine :** dans cette phase, les tâches sont assignées aux machines qui minimisent leurs **EFT (E**arliest **F**inish **T**ime**)**.

Dans notre cas on s'intéresse à la phase de prioritisation pour générer la liste des tâches ordonnées.

➔ **Présentation de la formule qui calcul le rang des tâches :** Les tâches sont triées par leurs priorités d'ordonnancement qui sont basées sur le rang upward ($rank_u$). $rank_u$ est la distance maximale d'une tâche par rapport à tâche de sortie du DAG. Il est défini pour la tâche n_i récursivement comme suit **[III.3]**:

$$\mathbf{rank_u(n_i) = \overline{coût\_calcul_i} + \max_{n_j \in succ(n_i)} (\overline{c_{i,j}} + rank_u(n_j))}$$

où :

- $succ(n_i)$ est l'ensemble des successeurs immédiats de la tâche n_i.
- $\overline{coût\_calcul_i}$ le coût moyen de l'exécution de la tâche n_i.
- $\overline{c_{i,j}}$ est le coût moyen des communications de l'arête (i, j).

Il est calculé pour toutes les tâches du DAG en commençant par la tâche de sortie qui a comme rang :

$$\mathbf{rank}_u(\mathbf{n}_{\mathbf{sortie}}) = \mathbf{coût\_calcul}_{sortie}$$

Remarque : Dans le cas où plusieurs tâches ont le même rang une seule tâche est choisie aléatoirement.

➔ **La gestion de la dépendance des tâches :**

Une tâche prête est définie comme la tâche dont les tâches parentes ont terminé leurs exécutions. Au début une file d'attente de l'ordonnanceur se compose des tâches d'entrée car elles n'ont pas de tâches parentes. Lorsqu'une tâches ti est terminée, la file d'attente est mis à jour en supprimant ti est en insérant la nouvelles tâches prêtes.

➔ **Voici un exemple d'exécution de l'algorithme HEFT (figure III.8) en prenant un fichier d'entrée représenté dans la (figure III.7) :**

Nombre de Tâches

Nombre de Machines

Matrice de coût de calcul

Matrice de vitesse de transfert de données entre les machines

Matrice de données à transférer entre les tâches

app_10xplat_5.txt - Bloc-n...

Fichier Edition Format Affichage ?

```
10
5

14  16 9  10 17
13  19 18 14 20
110 13 19 50 99
13  8  17 7  12
12  13 10 9  14
13  16 9  8  17
7   15 11 12 14
5   11 14 7  13
18  12 20 22 11
21  7  16 19 5

0 1 1 1 1
1 0 1 1 1
1 1 0 1 1
1 1 1 0 1
1 1 1 1 0

-1 18 12 9 11 14 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 19 16 -1
-1 -1 -1 -1 -1 -1 23 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 27 23 -1
-1 -1 -1 -1 -1 -1 -1 -1 13 -1
-1 -1 -1 -1 -1 -1 -1 15 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 17
-1 -1 -1 -1 -1 -1 -1 -1 -1 11
-1 -1 -1 -1 -1 -1 -1 -1 -1 13
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

Figure III.7 : Exemple d'un fichier d'entrée.

➔ **Apres l'exécution de l'algorithme HEFT on obtient les résultats suivants :**

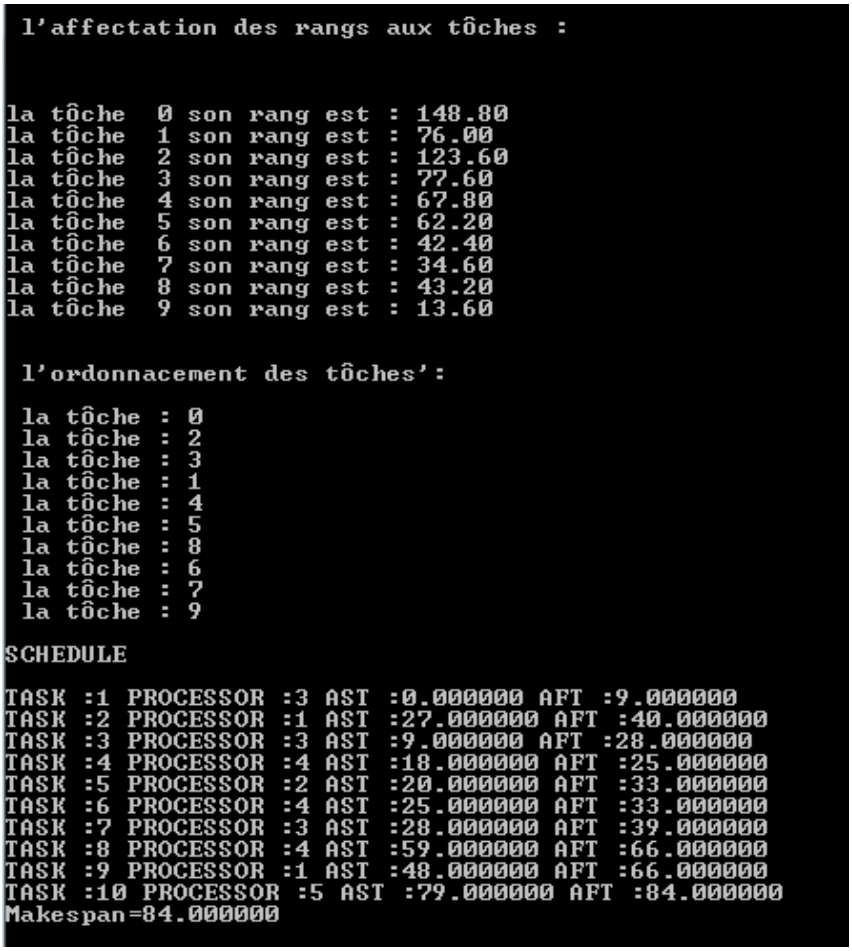


Figure III.8 : Exemple d'exécution de l'algorithme HEFT.

Dans cette section, nous décrivons le modèle de notre système par une simple formalisation des différents paramètres qui doivent être pris en considération lors de l'élaboration d'un algorithme d'ordonnancement des workflows. Notre objectif est de répartir les tâches de workflows entre les services du cloud de manière à optimiser les critères de QoS (le coût et le makespan) des utilisateurs.

III.3.3. Modèle du cloud

Une infrastructure est composée d'un pool de machines virtuelles **U= {VM_1, ..., VM_m}** hétérogènes qui sont interconnectées. Ces machines sont mises à la disposition des utilisateurs sous forme de services à la demande via un modèle de paiement à l'usage. Chaque instance de machine virtuelle, **VM_j,** peut être décrite par une vitesse de calcul

(MIPS$_j$) correspondant au nombre d'instructions que la ressource peut traiter par seconde, un coût monétaire d'utilisation par unité de temps **prixU$_j$**. Le comportement de l'infrastructure **IaaS** peut être modélisé sous forme d'un graphe pondéré non-orienté. Les nœuds du graphe représentent les machines virtuelles et les arêtes correspondent aux liens d'interconnexions entre elles.La figure **III.9(b)** montre un exemple de telle infrastructure.

III.3.4. Modélisation du workflow

D'après le fichier d'entrée illustré dans la **figure III.7** au dessus on génère le workflow suivant (**figure III.9(a)**) :

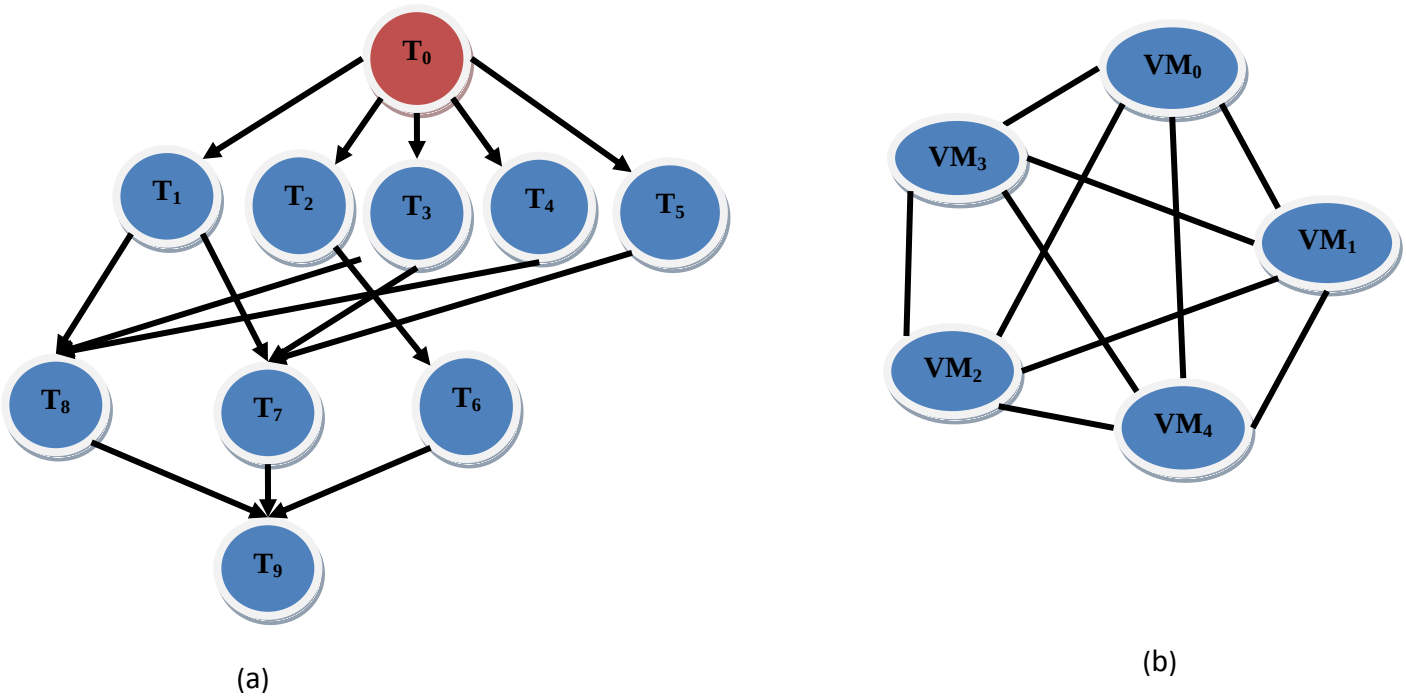


Figure III.9 : Un exemple de workflow et des ressources.

III.3.5. Les métriques QoS mesurées

Les objectifs que nous voulons tenter optimiser sont :

1. **Makespan**

Le makespan est l'une des mesures les plus utilisées dans l'évaluation des algorithmes d'ordonnancement de workflows. En général, le makespan est composé du temps d'exécution des tâches et du temps de transferts 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.

Pour calculer le makespan, il est nécessaire de définir les attributs suivants :

1. **EST**$(\mathbf{n_i}, \mathbf{m_j})$ **'Earliest execution Start Time'** de la tâche n_i sur la machine m_j.

   – Pour la tâche d'entrée $n_{entrée}$:

   $$\mathbf{EST}(\mathbf{n_{entrée}}, \mathbf{m_j}) = \mathbf{0}.$$

   – Pour les autres tâches:

   $$\mathbf{EST}(\mathbf{n_i}, \mathbf{m_j}) = \mathbf{max}\{\mathbf{avail[j]}, \mathbf{max}_{\mathbf{n_m} \in \mathbf{pred}\ \mathbf{n_i}}(\mathbf{AFT}\ \mathbf{nm} + \mathbf{cm}, \mathbf{i})$$

Où : **pred(n_i)** est un ensemble des prédécesseurs immédiats de la tâche $\mathbf{n_i}$. ***avail*[j]** est le prochain temps où la machine **mj** sera prête pour l'exécution des tâches (la machine sera disponible). Le **'max'** interne de l'équation de EST donne le temps où toutes les données nécessaires à l'exécution de ni sont arrivées à la machine **mj**.

2. **EFT (ni, mj) 'Earliest execution Finish Time'** de la tâche ni sur la machine$\mathbf{m_j}$.

– Pour toute les tâches : $\boldsymbol{EFT}(\mathbf{n_j},\mathbf{m_j}) = \boldsymbol{Coût\_Calcul_{i,j}} + \boldsymbol{EFT}(\mathbf{n_j}, \mathbf{m_j})$.

Après qu'une tâche **ni** soit ordonnancée sur une machine $\mathbf{m_j}$ **alors :**

– **'Actual Start Time'** de la tâche **ni** est : $\mathbf{AST(n_j) = EST(n_j, m_j)}$
et
– **'Actual Finish Time'** de la tâche **ni** est : $\mathbf{AFT(n_j) = EFT(n_j, m_j)}$.

Après l'ordonnancement de toutes les tâches du DAG, le temps de complétition de toutes les tâches (makespan) est l'Actual Finish Time de la tâche de sortie **(3.1).**

$$\textbf{makespan= AFT } (n_{sortie}). \quad (3.1)$$

L'objectif du problème d'ordonnancement de tâches est de déterminer un assignement des tâches d'une application donnée sur les machines de la plateforme de manière à minimiser le makespan.

3. Coût

Suite à la caractéristique orientée marché des services actuels, 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 données (par exemple, par **MB**) entre deux services et le prix pour le traitement par unité de temps (par exemple, par heure). Le coût total d'exécution d'un *workflow* dan $\mathbf{Coût}_{total}$ est défini dans l'équation **(3.2).** Il se compose du coût d'exécution des différentes tâches $\mathbf{Coût}_{ex}$et du coût de transferts des données $\boldsymbol{Coût}_{tr}$.

$$\mathbf{Coût}_{total} = \mathbf{Coût}_{ex} + \boldsymbol{Coût}_{tr} \textbf{ (3.2)}$$

Le coût d'exécution d'une tâche donnée dépend, à la fois, de la date de fin d'exécution sur la machine $\mathbf{VM_j}$ qui lui est allouée et du prix unitaire de cette machine $\mathbf{prixU_j}$. Ainsi, est donné par l'équation suivante:

$$\mathbf{Coût}_{ex} = \sum_{i=1}^{n} \boldsymbol{DF(Ti)} * \boldsymbol{PrixUj} \qquad \textbf{(3.3)}$$

Le coût de transfert de données ($\boldsymbol{Coût_{tr}}$) est décrit comme suit:

$$\boldsymbol{Coût_{tr}} = \sum_{i=1}^{n} \sum_{j=1}^{n} \boldsymbol{D_{ij}} * \boldsymbol{TRC_{ij}} \qquad \textbf{(3.4)}$$

où,

- $\boldsymbol{D_{ij}}$: caractérise la quantité de données à transférer de la tâche $\boldsymbol{T_i}$ à la tâche $\boldsymbol{T_j}$.
- $\boldsymbol{TRC_{ij}}$: représente le coût de communication entre la machine où $\boldsymbol{T_i}$ est mappée et une autre machine où $\boldsymbol{T_j}$ est affectée.

Après avoir choisi comment représenter le workflows, l'environnement cible (Cloud), les métriques de QoS, la fonction objective, nous allons résoudre le problème d'optimisation multi-objectif en utilisant l'algorithme **NSGA-II,** On Ajoute une classe **« monpro »** qui hérite de la classe Problem de JMetal qui calcule les deux métriques de QoS « coût, makespan » « voir le diagramme de classe de la **figure III.10** ».

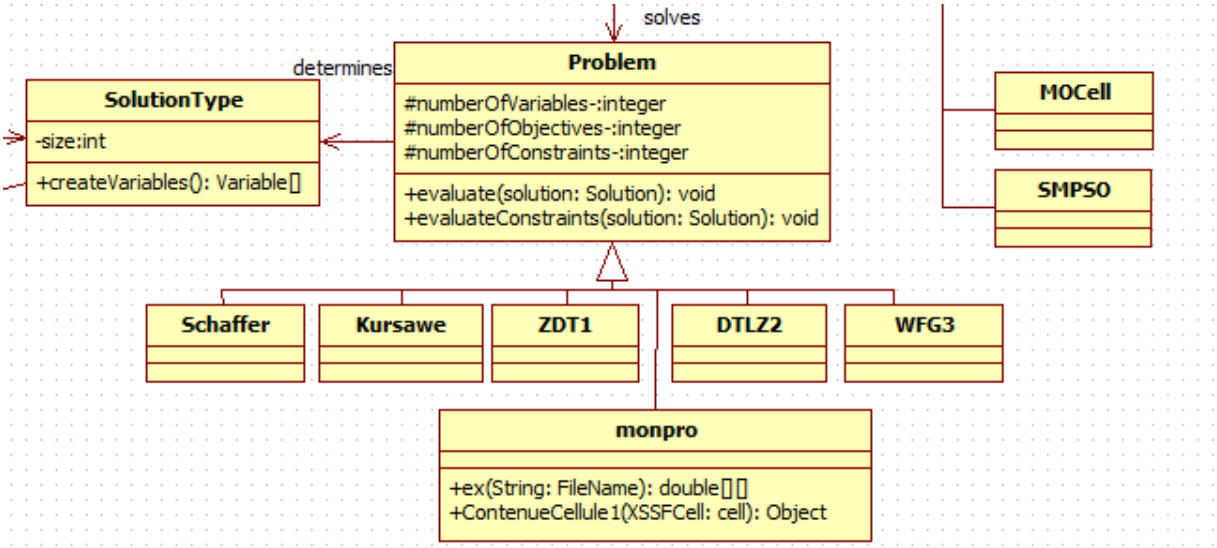


Figure III.10 : Diagramme de classe.

III.4. NSGA-II pour l'ordonnancement de *workflow*s

NSGA-II a longtemps été l'algorithme phare dans le domaine de l'optimisation évolutionnaire multi-objectif. Il apparaît comme l'un des algorithmes de référence pour trouver l'ensemble de solutions Pareto optimal, avec une excellente distribution de solutions, du fait qu'il intègre un opérateur de sélection basé sur la distance de "Crowding". Le fonctionnement général de l'algorithme NSGA-II est décrit dans la section suivante.

III.4.1. Etapes de l'algorithme NSGA-II

NSGA-II est un algorithme élitiste, n'utilisant pas d'archive externe pour stocker les solutions non dominées. Pour gérer l'élitisme, **NSGA-II** assure qu'à chaque nouvelle génération, les meilleurs individus rencontrés soient conservés.

L'algorithme **NSGA-II** commence par une génération aléatoire d'une population initiale « **P0 »** de « **N »** individus parents. A la génération **t**, une population **Qt** de **N** enfants est créée à partir de la population parent **Pt**, en utilisant les opérateurs génétiques de **sélection,** de **croisement** et de **mutation**. Ensuite, les deux populations sont fusionnées pour former une nouvelle population **Rt** de taille **2N**. La recherche des solutions non-dominées permet de classer les individus de **Rt**, en plusieurs fronts de rangs différents. L'identification des différents fronts **Fi** est effectuée, en utilisant la profondeur de dominance décrite dans la section **II.2.8** du chapitre **II**. La nouvelle population parent **Pt+1** est alors construite en incluant les meilleurs individus de **Rt** appartenant aux fronts de rang les plus faibles, tant que le nombre d'individus présents dans **Pt+1** est inferieur à **N**. Pour le front suivant, il y'a **N-|Pt+1|** places restantes dans la nouvelle population **Pt+1**. Par conséquent, les individus sont triés selon leurs distances de **Crowding**, les **N-|Pt+1|** meilleurs individus au sens de cette distance seront insérés à **Pt+1**. Ce choix permet de favoriser les solutions dans les régions les moins denses, dans le but d'améliorer la diversité des solutions. L'algorithme suivant résume toutes les étapes décrites ci-dessus.

➔ **Pseudo-code de l'algorithme NSGA-II :**

1 :	Générer les populations **P0** et **Q0** de taille **N**
2 :	**Répéter**
3 :	Création de **Rt = Pt∪ Qt**
4 :	Calcul des différents fronts **Fi** de la population **Rt** par la profondeur de dominance
5 :	Mettre **Pt+1 =∅** et **i = 0**
6 :	**TantQue | Pt+1 | + |Fi| < N Faire**
7 :	**Pt+1 = Pt ∪ Fi**
8 :	**i = i + 1 ;**
9 :	**FinTantQue**
10 :	Inclure dans **Pt+1** les **(N- |Pt+1|)** individus de **Fi** les mieux répartis au sens de la distance de Crowding
11 :	Sélection dans **Pt+1** et création de **Qt+1** par application des opérateurs de croisement, mutation et infection virale (dans le cas de contrainte forte)
12 :	**Jusqu'à** atteindre un critère d'arrêt.

III.4.2. Calcul de la distance de "Crowding"

La distance de Crowding di d'un point particulier i se calcule en fonction du périmètre de l'hypercube formé par les plus proches voisins de *i* sur chaque objectif. Les étapes suivantes sont utilisées pour calculer la distance de *Crowding* de chaque point d'un ensemble**.**

1. Soit ***l*=|*Z*|** le nombre d'individus de ***Z***. Pour tout ***i=1,.., l,*** initialiser ***d_i= 0.***

2. Pour chaque objectif (QoS_m), ***m =1,..., M***, Trier cet ensemble Z suivant l'ordre décroissant de la fonction f_m en construisant le vecteur I^m: I^m = **trier (*Z*, f_m,>).**

3. Attribuer une grande valeur numérique à la distance de *Crowding* des solutions extrêmes (correspondant à une valeur minimale ou maximale de la QoS_m) : et pour les autres individus, c'est-à-dire pour **tout *j =2 à (l-1)*** calculer:

$$d_j = \sum_{m=1}^{M} d_{I_i^m} + \frac{f_m^{I_{j+1}^m} - f_m^{I_{j-1}^m}}{f_m^{max} - f_m^{min}}$$

Où,

f_m^{max} et f_m^{min} sont respectivement la valeur maximale et minimale de la fonction f_m , *et* $f_m^{I_j^m}$ est la valeur de la fonction f_m du vecteur I^m de l'individu ***j.***

III.4.3. Operateurs génétiques

1. **Opérateur de sélection :** Nous utilisons, dans NSGA-II, la sélection par tournoi (BinaryTournament).

2. **Opérateur de croisement :** Nous choisissons le **simple enjambement** (Singlepoint crossover).

3. **Opérateur de mutation:** Nous choisissons (PolynomialMutation).

➔ Pour plus de détaille voir le **chapitre II** séction **II.2.9.1.1**

III.4.4. Paramètres de l'algorithme génétique NSGA-II

Les différents paramètres de l'algorithme NSGA-II sont décrits dans le **Tableau III.1**. Le choix des valeurs des paramètres est effectué en expérimentant l'algorithme sur un workflows de dix tâches (10) sur cinq (5) VM, pour différentes valeurs de ces paramètres.

Nous avons utilisé plusieurs configurations de platforme :

Avec : 5, 10, 20, 40, 60 VMs. Et des workflows de 5, 20, 60 tâches.

Paramètre	Valeur
Taille de la population	20
Nombre de générations	20
Taux de croisement	0.9
Taux de mutation	1/nombre de tâches

Tableau III.1 : Paramètres de l'algorithme génétique NSGA-II.

III.5. Test et résultats

III.5.1. Exécution du programme NSGA-II :

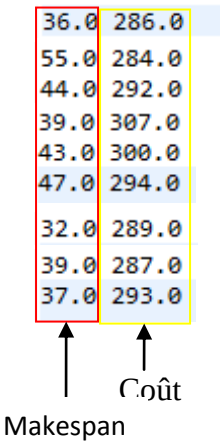
Pour le programme **NSGA-II** et les résultats des exécutions voir l'annexe.

III.5.2. Makespan et coût en fonction de nombre de tâches :

1. **Test 1 : Calcul de makespan et du coût en fonction de taille des workflows(10, 20, 40) sur une configuration de ressources égal à 5.**

✓ **Pour un nombre d'exécution =5 :**

- Pour un exemple de 10 taches sur 5 VMs :



➔ **Après 5 exécutions :**

1. Le makespan moyen est :

(55+32)/2=43,5

2. Le coût moyen est :

(307+284) /2=295,5

- Pour un exemple de 20 taches sur 5 VMs :

```
92.0 949.0
90.0 997.0
82.0 955.0
93.0 941.0
67.0 915.0
111.0 940.0
98.0 957.0
72.0 967.0
82.0 961.0
```

➔ **Après 5 exécutions :**

1. Le makespan moyen est :

(111+67)/2=89

2. Le coût moyen est :

(997+915) /2=956

- Pour un exemple de 40 taches sur 5 VMs :

```
154.0 3301.0
141.0 3380.0
152.0 3316.0
148.0 3318.0
172.0 3341.0
141.0 3324.0
157.0 3324.0
146.0 3236.0
```

➔ **Après 5 exécutions :**

1. Le makespan moyen est :

(172+141)/2=156.5

2. Le coût moyen est :

(3380+3301) /2=3340.5

➔ Ce graphe représente le makespans par apport au nombre de tâches d'après les résultats obtenue en exécution de NSGA-II :

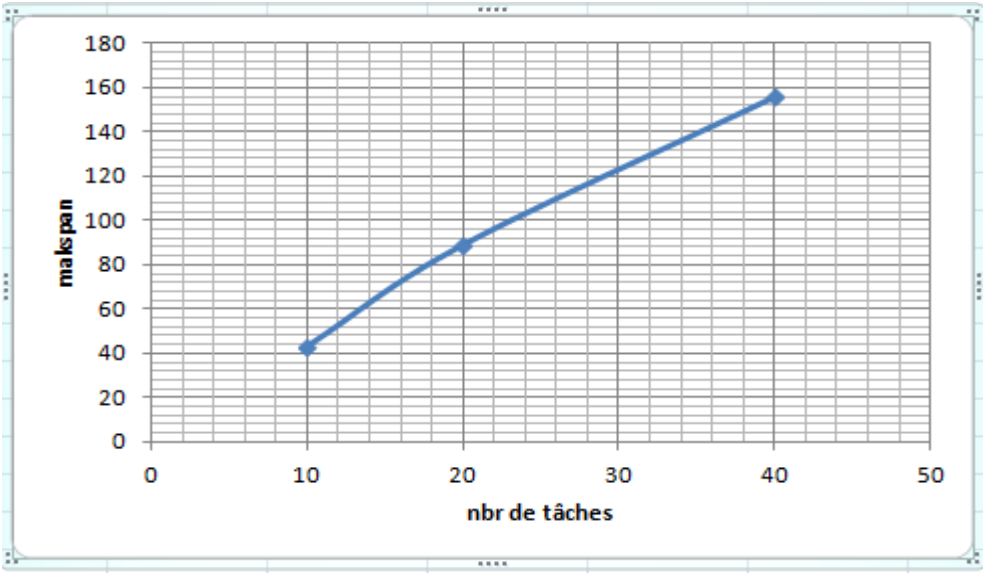


Figure III.11 : Le graphe des makespans par rapport aux nombres de tâches.

Analyse : Dans ce graphe on remarque que le makespan augmente avec le nombre de tâches.

➔ Ce graphe représente les coûts par apport au nombre de tâches d'après les résultats obtenue en exécution de NSGA-II :

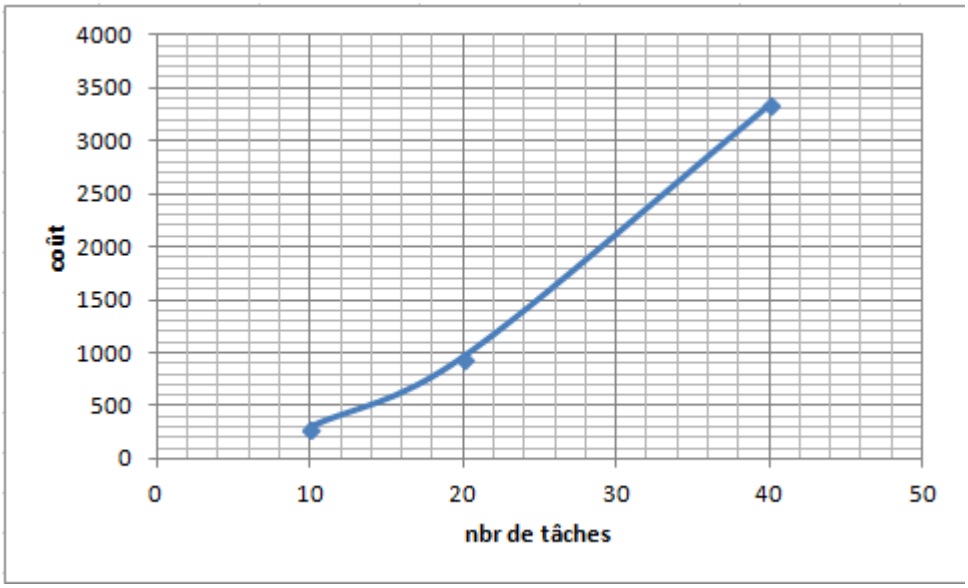


Figure III.12 : Le graphe des coûts par rapport aux nombres de tâches.

Analyse : Dans ce graphe on remarque que le coût augmente avec le nombre de tâches.

2. Test 2 : Calcul de makespan et de coût on utilisant la même taille de workflows (10 tâches) exécuté sur plusieurs configurations de ressources.

- Pour un exemple de 10 taches sur 5 VMs :

119.0	550.0
106.0	605.0
129.0	526.0

makespan coût

1. Le makespan moyen est :

(129+119)/2=124

2. Le coût moyen est :

(526+605) /2=565,5

- Pour un exemple de 10 taches sur 10 VMs :

```
124.0 407.0
94.0 571.0
```

1. Le makespan moyen est :

$$(124+94)/2=109$$

2. Le coût moyen est :

$$(571+407)\ /2=489$$

➔ Ce graphe représente les makespans par apport au nombre de VMs d'après les résultats obtenues en exécution de NSGA-II :

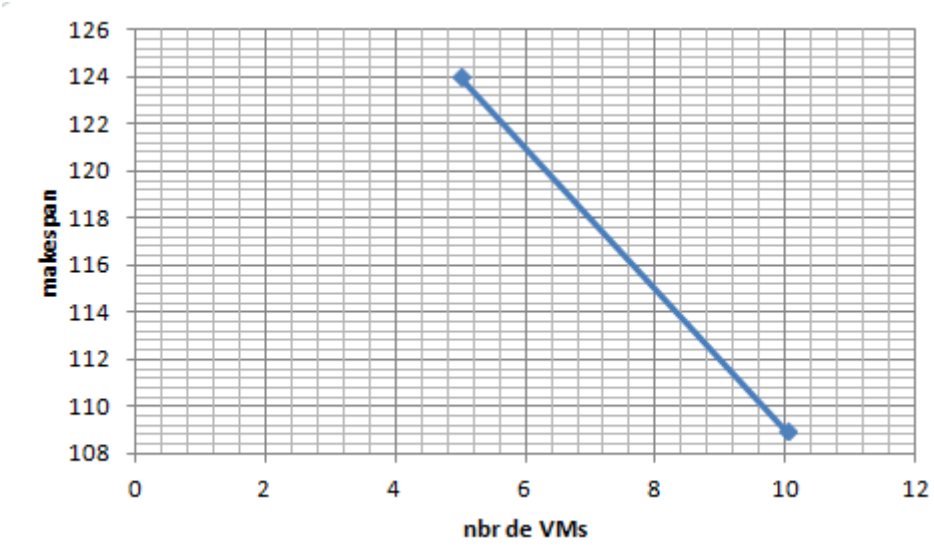


Figure III.13 : Le graphe des makespans par rapport aux nombres de VMs.

Analyse : Dans ce graphe on remarque que le makespan diminue on augmentant le nombre de machines.

➔ Ce graphe représente les coûts par apport au nombre de VMs d'après les résultats obtenues en exécution de NSGA-II :

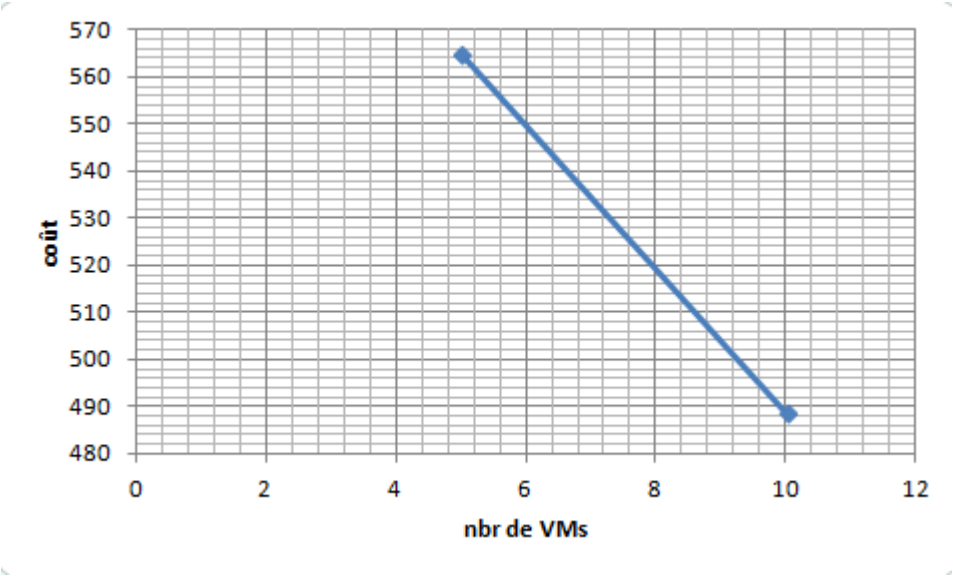


Figure III.14 : Le graphe des coûts par rapport aux nombres de VMs.

Analyse : Dans ce graphe on remarque que le coût diminue on augmentant le nombre de machines.

III.5.3. Comparaison de makespan

1. L'exécution on utilisant l'algorithme **HEFT** : voici les résultats

➔ Pour 10 tâches :

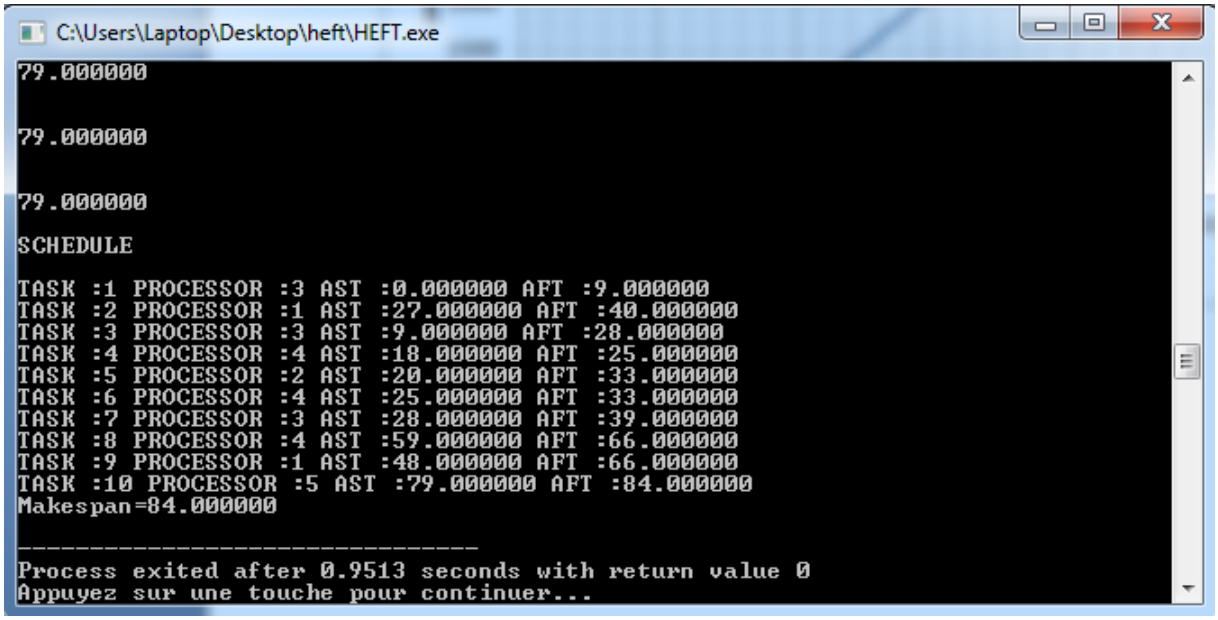


Figure III.15 : Résultats d'exécution de l'algorithme HEFT pour 10 tâches.

➔ Pour 20 tâches :

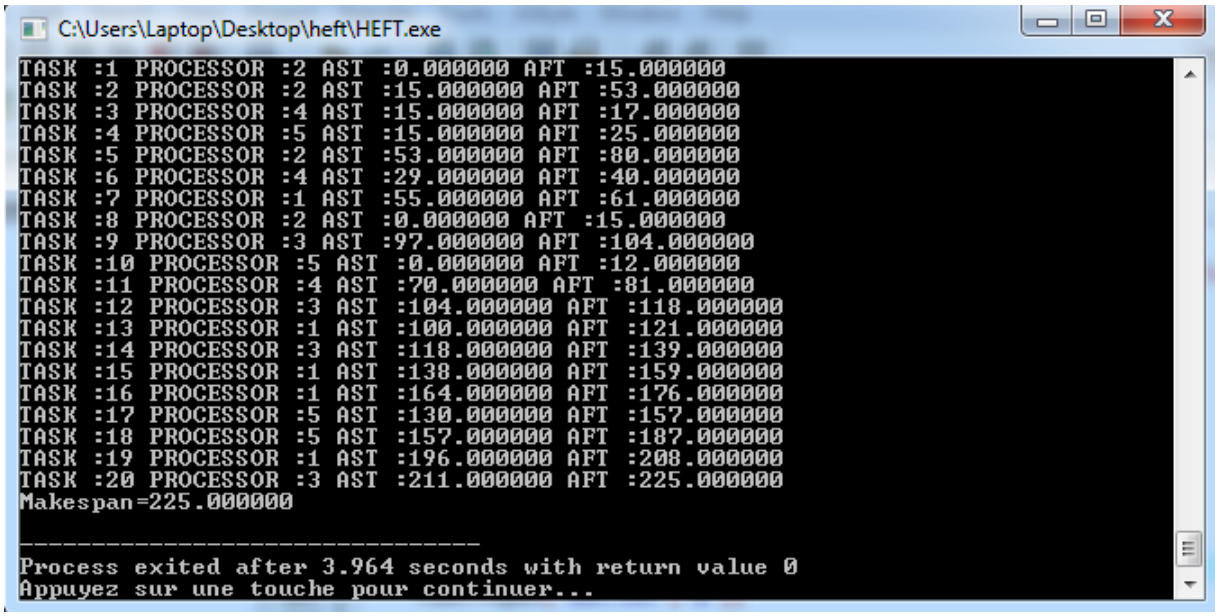


Figure III.16 : Résultats d'exécution de l'algorithme HEFT pour 20 tâches.

➔ Pour 40 tâches :

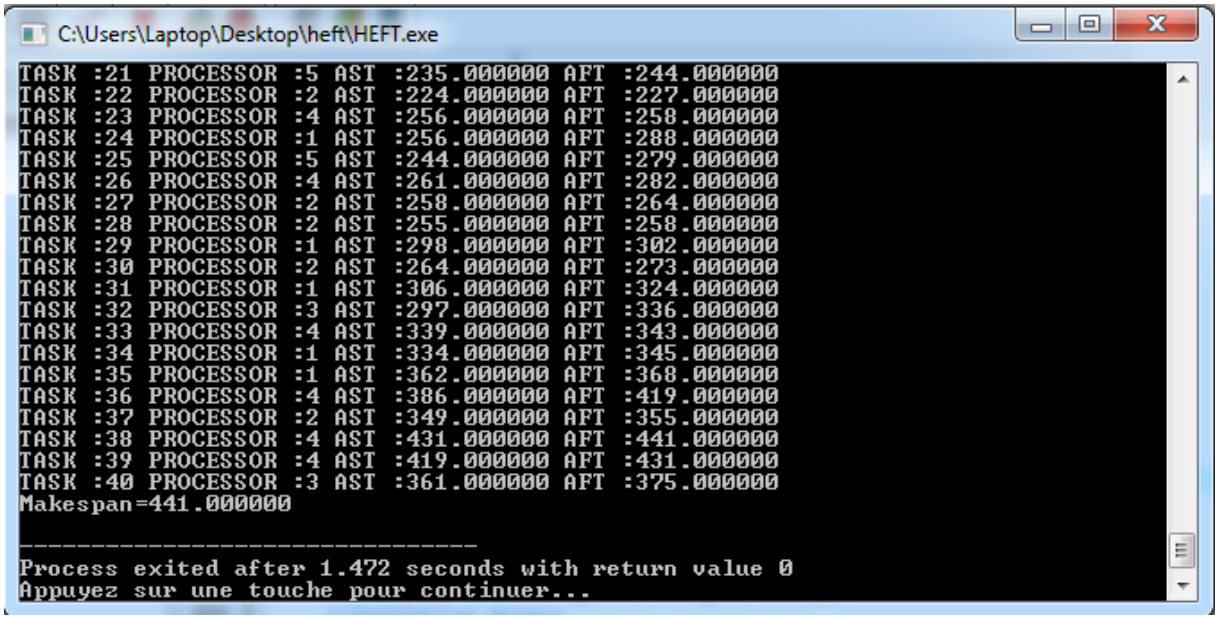


Figure III.17 : Résultats d'exécution de l'algorithme HEFT pour 40 tâches.

➔ Pour60 tâches :

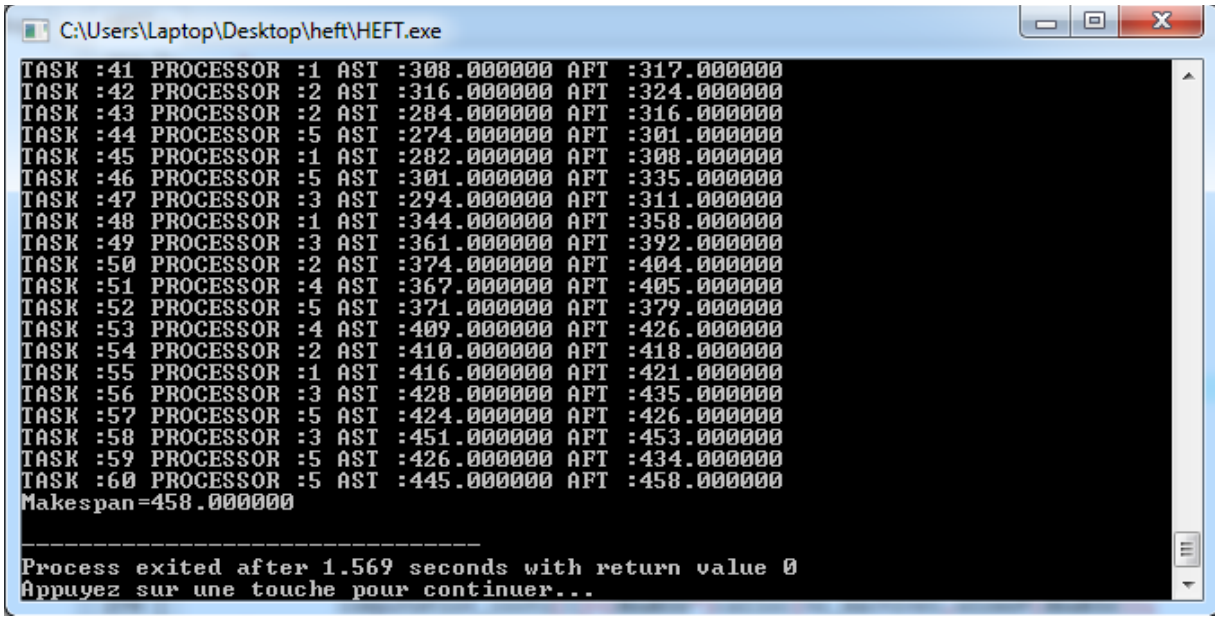
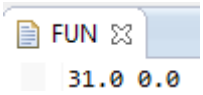


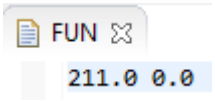
Figure III.18 : Résultats d'exécution de l'algorithme HEFT pour 60 tâches.

2. L'exécution on utilisant l'algorithme **NSGA-II** : voici les résultats

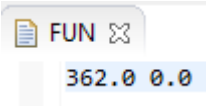
➔ Pour 10 tâches :



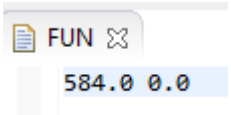
➔ Pour 20 tâches :



➔ Pour 40 tâches :



➔ Pour 60 tâches :



➔ **Le graphe suivant représente une comparaison de makespan par apport aux nombres de tâches avec HEFT et NSGA-II :**

tâche	HEFT	NSGA
10	84	31
20	225	211
40	441	362
60	458	584

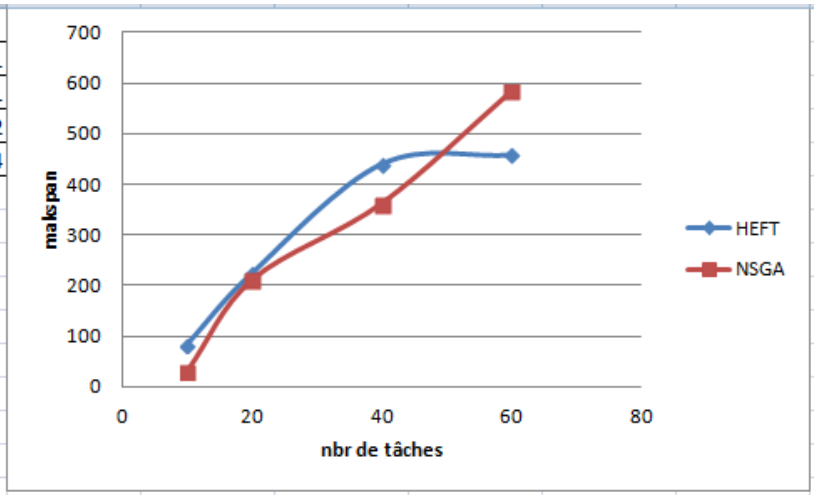


Figure III.19 : Graphe pour la Comparaison de makespan.

Analyse : Pour un nombre petit de tâches NSGA-II présente de meilleur résultats, mais à partir de 50 tâches NSGA-II donne des temps d'exécution (makespan) plus élevés que ceux obtenus par

III.6. Conclusion

Dans ce chapitre, nous avons présenté les étapes suivies pour la résolution du problème d'ordonnancement du workflows dans le cloud, nous avons testé un algorithme bi-objectif qui est l'algorithme NSGA-II avec comme métriques QoS le makespan et le coût.

Conclusion générale

Le Cloud computing fournit un modèle informatique, qui permet d'accéder d'une façon transparente et à la demande, à un pool de ressources hétérogènes physiques ou virtualisées (serveurs, stockage, applications et services), à travers le réseau. Ces ressources sont délivrées sous forme de services reconfigurables et élastiques, à base d'un modèle de paiement à l'usage dont les garanties sont offertes par le fournisseur, via des contrats de niveau de service SLA (*Service Level Agreement*). L'exploitation efficace de ces services sous contraintes souvent contradictoires pour les fournisseurs et les utilisateurs soulève plusieurs défis scientifiques.

Ce mémoire s'intéresse particulièrement à la problématique d'ordonnancement des workflows dans le cloud. Ce problème est considéré comme étant un problème d'optimisation multi-objectif, en raison de la nature conflictuelle des métriques de QoS à prendre en compte. Nous avons utilisé une méthode de type méta-heuristique pour résoudre cette problématique. En effet, nous avons opté pour une approche Pareto via un algorithme génétique (GA) pour résoudre le problème d'ordonnancement avec comme objectifs optimiser le temps de complétion (makespan) et le coût.

Nous avons modélisé le problème d'ordonnancement de workflow sous le Framework JMetal, et nous avons choisi comme méta-heuristique l'algorithme NSGAII.

Nous avons réalisé plusieurs tests en utilisant plusieurs tailles de workflows et une simple comparaison des résultats a été réalisée avec des résultats que nous avons obtenu en utilisant l'algorithme d'ordonnancement de workflows dit HEFT.

Dans beaucoup de cas, les résultats obtenus par NSGAII sont meilleurs par rapport à ceux réalisés par HEFT.

Comme perspective à ce travail, une extension pour la prise en compte de plusieurs autres métriques de QoS (objectifs tels que la fiabilité et la disponibilité des ressources, l'énergie consommée) peut-être réalisée.

Aussi, d'autres algorithmes codés dans JMetal peuvent être utilisé pour résoudre le problème d'ordonnancement de workflows dans un environnement hétérogène et distribué tel le Cloud.

Bibliogaraphie

I.1	cloud computing (Décider, concevoir, piloter, Améliorer). Auteurs : Romain Heunion,Hubert Tournier, Eric Bourgeois.
I.2	cloud computing (Guillaume Plouni 2013).
I.3	Une approche basée agents pour l'allocation des ressources dans le Cloud Computing Thèse Présenté par : FAREH Med el-kabir. Année Universitaire: 2014-2015
I.4	Mémoire, ingénieur 2010 : Ordonnancement d'applications de type Workflow sous un système distribué de type Grille de calcul. Thème réalisée par M^{r} Amirouche KHELFA et M^{r} Ahmed AITAHMED.
I.5	Topcuoglu, H., Hariri, S., Wu, M.Y. Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans Parallel Distrib Syst 13(2002)260-274.
I.6	Ge, Y. and Wei, G. GA-Based Task Scheduler for the Cloud Computing Systems. In Proceedings of the IEEE Int. Conf. on Web Information Systems and Mining, page 181-186, 2010
I.7	Li., J., Peng, J., Lei, Z. and Zhang, W. An Energy Efficient Scheduling Approach Based on Private clouds. Journal of Information and Computational Science, 8(2011)716-724.
I.8	El-kenawy, E-S.T., El-Desoky, A.I., Al-rahamawy, M.F. Extended Max-Min Scheduling Using Petri Net and Load Balancing. International Journal of Soft Computing and Engineering (IJSCE), 2(2012) 2231-2307.
I.9	Hakem, M. and Butelle, F. Reliability and Scheduling on Systems Subject to Failures. nInt. Conf. on Parallel Processing (ICPP), Sept. 2007.
I.10	Wang, X., Yeo, C.S., Buyya, R.K. and Su, J. Optimizing the Makespan and Reliability for Workflow Applications with Reputation and a Look-ahead Genetic Algorithm. Journal Future Generation Computer Systems, 27(2011)1124-1134.
I.11	Jang, S.H., Kim, T.Y., Kim, J.K. and Lee, J.S. The Study of Genetic Algorithm-based Task Scheduling for Cloud Computing. International Journal of Control and Automation, 5(2012)157-162.
I.12	Guzek, M., Diaz, C., Pecero, J., Bouvry, P. and Zomaya, A.Y. Impact of Voltage Levels Number for Energy-aware Bi-objective DAG Scheduling for Multi-processors Systems, 5th Int. Conf. on Advances in Information Technology, IAIT 2012, pages 6- 7, Bangkok, Thailand, 2012.
I.13	Chang-Tian, Y. and Jiong, Y. Energy-aware Genetic Algorithms for Task Scheduling in cloud computing. In Proceedings of Seventh IEEE ChinaGrid Annual Conference, 43-48, 2012.
I.14	Liu, J., Luo, X.G., Zhang, X.M., Zhang, F. and Li, B.N. Job Scheduling Model for Cloud Computing Based on Multi-Objective Genetic Algorithm, IJCSI International Journal of Computer Science Issues, 10(2013)134-139.
II.1	http://jmetal.sourceforge.net/
II.2	Optimisation multi-objectif (Yann Collette – Patrick Siarry).
II.3	Allocation optimale multi-contraintes des workflows aux ressources environnement Cloud Computing thèse présentée par Sonia Yassa.
II.4	(Edge worth, 1881) Edgeworth. F.Y. Mathematical Physics. P. Keagan, London, 1881.
II.5	(Pareto, 1996) Pareto, V. Cours d'économie politique. Rouge, Lausanne, 1996.
II.6	(Collette, 2002) Collette, Y. Contribution à l'évaluation et au perfectionnement des méthodes d'optimisation multiobjectif : application à l'optimisation des plans de rechargement de combustible nucléaire, Thèse de doctorat, Université de Paris 12, 2002.
II.7	(Ulungu, 1995) Ulungu, E.L. and Teghem, J. The two phases method: An efficient procedure to solve biobjective combinatorial optimization problems. Foundation of computing and decision science, 20(1995)49-156.
II.8	(Stewart, 1991) Stewart B.S. and White, C.C. Multiobjective A*. Journal of the ACM, 38(1991)775-814.

Bibliogaraphie

II.9	(Carraway, 1990) Carraway, R.L., Morin, T.L and Moskowitz, H. Generalized dynamic programming for multicriteria optimization. European Journal of Operational Research, 44(1990)95-104
II.10	Talbi, E. G. Metaheuristics: from design to implementation. Wiley, 2009.
II.11	(Siarry, 2003) Siarry, P. Dréo, J. Pétrowski, A. Taillard, E. Métaheuristiques pour l'optimisation difficile, Eyrolles, 2003.
II.12	https://fr.wikipedia.org/wiki/M%C3%A9taheuristique
II.13	Papadimitriou, C. and Steiglitz, K. Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall, 1982.
II.14	Glover, F. Tabu Search - part i. ORSA Journal on Computing, 1(1986)190-206.
II.15	Cerny, V.A. Thermodynamical approch to the traveling salesman problem: An efficient simulation algorithm. Journal of Optimization Theory and Applications, 45(1985)41-51.
II.16	Sriniva, N. and Kalyanmoy, D. Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. Evolutionary Computation, 2(1994)221–248.
II.17	Miettinen, K. Nonlinear Multiobjective Optimization. Kluwer Academic Publishers, 1999.
II.18	Fourman, M. P. Compaction of Symbolic Layout using Genetic Algorithms. In Genetic,Algorithms and their Applications: Proceedings of the First Int. Conf. on Genetic Algorithm, pages141-153, 1985.
II.19	Schaffer, J. D. Multiple Objective Optimisation with Vector Evaluated Genetic Alggorithm. In genetic Algorithm and their Applications: Proceedings of the First Int. Conf. on Genetic Algorithm, pages 93-100, 1985.
II.20	Goldberg, D E. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, 1989.
II.21	Zitzler, E. Laumanns, M. and Bleuler, S. A Tutorial on Evolutionary Multiobjective Optimization, In Metaheuristics for Multiobjective Optimisation, Vol. 535 de Lecture Notes in Economics and Mathematical Systems, pages 3-37, Springer. Berlin, 2004.
II.22	Zitzler, E. and Kunzli. S. Indicator-based selection in multiobjective search. Parallel Problem Solving from Nature, PPSN VIII, 3242(2004)832-842.
II.23	Darwin, C. On the origin of the species by means of natural selection: Or, the preservation of n favored races in the struggle for life. John Murray, 1859.
II.24	http://magnin.plil.net/spip.php?article45
II.25	http://www.enseignement.polytechnique.fr/profs/informatique/Eric.Goubault/poly/cours009.html
III.1	Mémoire de fin d'étude licence en informatique (Analyse, conception et réalisation d'une application client/ serveur pour la gestion des stocks)
III.2	www.depannetonpc.net/lexique/lire_74_langage-c.html
III.3	Mémoire (Hybridation d'heuristiques pour le problème d'ordonnancement dans les grilles de calcul) réalisé par Mr BOUALI Lyes et Melle BELKADI Halima

Annexe

Le programme NSGA-II.

```
//  NSGAII_main.java

package jmetal.metaheuristics.nsgaII;

import java.io.File;

/**
 * Class to configure and execute the NSGA-II algorithm.
 *
 * Besides the classic NSGA-II, a steady-state version (ssNSGAII) is also
 * included (See: J.J. Durillo, A.J. Nebro, F. Luna and E. Alba
 *                 "On the Effect of the Steady-State Selection Scheme in
 *                 Multi-Objective Genetic Algorithms"
 *                 5th International Conference, EMO 2009, pp: 183-197.
 *                 April 2009)
 */

public class NSGAII_main {
  public static Logger      logger_ ;      // Logger object
  public static FileHandler fileHandler_ ; // FileHandler object

  /**
   * @param <HSSFWorkbook>
  * @param args Command line arguments.
   * @throws JMException
   * @throws IOException
   * @throws SecurityException


   * Usage: three options
   *      - jmetal.metaheuristics.nsgaII.NSGAII_main
   *      - jmetal.metaheuristics.nsgaII.NSGAII_main problemName
   *      - jmetal.metaheuristics.nsgaII.NSGAII_main problemName paretoFrontFile
   */
 public static <HSSFWorkbook> void main(String [] args) throws
                               JMException,
                               SecurityException,
                               IOException,
                               ClassNotFoundException {
    Problem   problem   ; // The problem to solve
    Algorithm algorithm ; // The algorithm to use
    Operator  crossover ; // Crossover operator
    Operator  mutation  ; // Mutation operator
    Operator  selection ; // Selection operator

    HashMap  parameters ; // Operator parameters

    QualityIndicator indicators ; // Object to get quality indicators



    // Logger object and file to store log messages
    logger_      = Configuration.logger_ ;
    fileHandler_ = new FileHandler("C:\\Users\\Laptop\\Desktop\\messages\\message.xls");
    logger_.addHandler(fileHandler_) ;


     indicators = null ;
     if (args.length == 1) {
       Object [] params = {"Int"};
       problem = (new ProblemFactory()).getProblem(args[0],params);
     } // if
     else if (args.length == 2) {
       Object [] params = {"Real"};
       problem = (new ProblemFactory()).getProblem(args[0],params);
       indicators = new QualityIndicator(problem, args[1]) ;
     } // if
     else { // Default problem
    //  problem = new Kursawe("ArrayReal", 6);
       //problem = new Kursawe("BinaryReal", 3);
      // problem = new ZDT3("ArrayReal", 3);
      problem = new monpro("ArrayReal",10);
     // problem = new Water("Real");
    //   problem = new DTLZ1("Real");
       //problem = new OKA2("Real") ;

     } // else
```

```
 algorithm = new NSGAII(problem);
 // algorithm = new gGA(problem);
   //algorithm = new ssNSGAII(problem);
//monpro  x =new monpro("ArrayReal", 3);
//x.ex( "C:\\Users\\Laptop\\Desktop\\d\\DONNEES.xlsx");
   // Algorithm parameters
   algorithm.setInputParameter("populationSize",100);
   algorithm.setInputParameter("maxEvaluations",100);

   // Mutation and Crossover for Real codification
   parameters = new HashMap() ;
   parameters.put("probability", 0.9) ;
   parameters.put("distributionIndex", 2.0) ;
   crossover = CrossoverFactory.getCrossoverOperator("SBXCrossover", parameters);

   parameters = new HashMap() ;
   parameters.put("probability", 1.0/problem.getNumberOfVariables()) ;
   parameters.put("distributionIndex", 3.0) ;
   mutation = MutationFactory.getMutationOperator("PolynomialMutation", parameters);

   // Selection Operator
   parameters = null ;
   selection = SelectionFactory.getSelectionOperator("BinaryTournament", parameters) ;


 // Add the operators to the algorithm
 algorithm.addOperator("crossover",crossover);
 algorithm.addOperator("mutation",mutation);
 algorithm.addOperator("selection",selection);

 // Add the indicator object to the algorithm
 algorithm.setInputParameter("indicators", indicators) ;

 // Execute the Algorithm
 long initTime = System.currentTimeMillis();
 SolutionSet population = algorithm.execute();
 long estimatedTime = System.currentTimeMillis() - initTime;

 // Result messages
logger_.info("Total execution time: "+estimatedTime + "ms");
logger_.info("Variables values have been writen to file VAR");
population.printVariablesToFile("VAR");
logger_.info("Objectives values have been writen to file FUN");
 population.printObjectivesToFile("FUN");
 if (indicators != null) {
   logger_.info("Quality indicators") ;
   logger_.info("Hypervolume: " + indicators.getHypervolume(population)) ;
   logger_.info("GD         : " + indicators.getGD(population)) ;
   logger_.info("IGD        : " + indicators.getIGD(population)) ;
   logger_.info("Spread     : " + indicators.getSpread(population)) ;
   logger_.info("Epsilon    : " + indicators.getEpsilon(population)) ;


     int evaluations = ((Integer)algorithm.getOutputParameter("evaluations")).intValue();
     logger_.info("Speed      : " + evaluations + " evaluations") ;


   } // if
  } //main
} // NSGAII_main
```

Figure III.12 : programme NSGA-II.

Les résultats :

Annexe

```
la population initiale :
3
2
3
3
1
0
2
3
2
4
l'ordre d'execution des taches :

le nombre de lignes est :  10
tache: 0
tache: 2
tache: 3
tache: 1
tache: 4
tache: 5
tache: 8
tache: 6
tache: 7
tache: 9

l'affectation des taches aux machines :
la tache n° 0 s'execute sur la machine n° 3
la tache n° 2 s'execute sur la machine n° 2
la tache n° 3 s'execute sur la machine n° 3
la tache n° 1 s'execute sur la machine n° 3
la tache n° 4 s'execute sur la machine n° 1
la tache n° 5 s'execute sur la machine n° 0
la tache n° 8 s'execute sur la machine n° 2
la tache n° 6 s'execute sur la machine n° 3
la tache n° 7 s'execute sur la machine n° 2
la tache n° 9 s'execute sur la machine n° 4

la population initiale :
0
4
2
0
4
2
4
0
1
4
l'ordre d'execution des taches :

le nombre de lignes est :  10
tache: 0
tache: 2
tache: 3
tache: 1
tache: 4
tache: 5
tache: 8
tache: 6
tache: 7
tache: 9

l'affectation des taches aux machines :
la tache n° 0 s'execute sur la machine n° 0
la tache n° 2 s'execute sur la machine n° 4
la tache n° 3 s'execute sur la machine n° 2
la tache n° 1 s'execute sur la machine n° 0
la tache n° 4 s'execute sur la machine n° 4
la tache n° 5 s'execute sur la machine n° 2
la tache n° 8 s'execute sur la machine n° 4
la tache n° 6 s'execute sur la machine n° 0
la tache n° 7 s'execute sur la machine n° 1
la tache n° 9 s'execute sur la machine n° 4
```

Annexe

```
la population initiale :
0
4
2
4
3
1
0
3
2
0
l'ordre d'execution des taches :

le nombre de lignes est :  10
tache: 0
tache: 2
tache: 3
tache: 1
tache: 4
tache: 5
tache: 8
tache: 6
tache: 7
tache: 9

l'affectation des taches aux machines :
la tache n° 0 s'execute sur la machine n° 0
la tache n° 2 s'execute sur la machine n° 4
la tache n° 3 s'execute sur la machine n° 2
la tache n° 1 s'execute sur la machine n° 4
la tache n° 4 s'execute sur la machine n° 3
la tache n° 5 s'execute sur la machine n° 1
la tache n° 8 s'execute sur la machine n° 0
la tache n° 6 s'execute sur la machine n° 3
la tache n° 7 s'execute sur la machine n° 2
la tache n° 9 s'execute sur la machine n° 0
```

```
la population initiale :
1
1
0
4
0
3
2
2
4
3
l'ordre d'execution des taches :

le nombre de lignes est :  10
tache: 0
tache: 2
tache: 3
tache: 1
tache: 4
tache: 5
tache: 8
tache: 6
tache: 7
tache: 9

l'affectation des taches aux machines :
la tache n° 0 s'execute sur la machine n° 1
la tache n° 2 s'execute sur la machine n° 1
la tache n° 3 s'execute sur la machine n° 0
la tache n° 1 s'execute sur la machine n° 4
la tache n° 4 s'execute sur la machine n° 0
la tache n° 5 s'execute sur la machine n° 3
la tache n° 8 s'execute sur la machine n° 2
la tache n° 6 s'execute sur la machine n° 2
la tache n° 7 s'execute sur la machine n° 4
la tache n° 9 s'execute sur la machine n° 3
```

Annexe

```
la population initiale :
1
0
4
3
4
2
3
4
3
0
l'ordre d'execution des taches :

le nombre de lignes est :  10
tache: 0
tache: 2
tache: 3
tache: 1
tache: 4
tache: 5
tache: 8
tache: 6
tache: 7
tache: 9

l'affectation des taches aux machines :
la tache n° 0 s'execute sur la machine n° 1
la tache n° 2 s'execute sur la machine n° 0
la tache n° 3 s'execute sur la machine n° 4
la tache n° 1 s'execute sur la machine n° 3
la tache n° 4 s'execute sur la machine n° 4
la tache n° 5 s'execute sur la machine n° 2
la tache n° 8 s'execute sur la machine n° 3
la tache n° 6 s'execute sur la machine n° 4
la tache n° 7 s'execute sur la machine n° 3
la tache n° 9 s'execute sur la machine n° 0


la population initiale :
3
0
0
2
1
1
4
2
3
2
l'ordre d'execution des taches :

le nombre de lignes est :  10
tache: 0
tache: 2
tache: 3
tache: 1
tache: 4
tache: 5
tache: 8
tache: 6
tache: 7
tache: 9

l'affectation des taches aux machines :
la tache n° 0 s'execute sur la machine n° 3
la tache n° 2 s'execute sur la machine n° 0
la tache n° 3 s'execute sur la machine n° 0
la tache n° 1 s'execute sur la machine n° 2
la tache n° 4 s'execute sur la machine n° 1
la tache n° 5 s'execute sur la machine n° 1
la tache n° 8 s'execute sur la machine n° 4
la tache n° 6 s'execute sur la machine n° 2
la tache n° 7 s'execute sur la machine n° 3
la tache n° 9 s'execute sur la machine n° 2
```

Annexe

```
la population initiale :
4
4
3
3
0
4
3
0
2
3
l'ordre d'execution des taches :

le nombre de lignes est :  10
tache: 0
tache: 2
tache: 3
tache: 1
tache: 4
tache: 5
tache: 8
tache: 6
tache: 7
tache: 9

l'affectation des taches aux machines :
la tache n° 0 s'execute sur la machine n° 4
la tache n° 2 s'execute sur la machine n° 4
la tache n° 3 s'execute sur la machine n° 3
la tache n° 1 s'execute sur la machine n° 3
la tache n° 4 s'execute sur la machine n° 0
la tache n° 5 s'execute sur la machine n° 4
la tache n° 8 s'execute sur la machine n° 3
la tache n° 6 s'execute sur la machine n° 0
la tache n° 7 s'execute sur la machine n° 2
la tache n° 9 s'execute sur la machine n° 3

la population initiale :
4
1
0
0
4
3
0
4
1
4
l'ordre d'execution des taches :

le nombre de lignes est :  10
tache: 0
tache: 2
tache: 3
tache: 1
tache: 4
tache: 5
tache: 8
tache: 6
tache: 7
tache: 9

l'affectation des taches aux machines :
la tache n° 0 s'execute sur la machine n° 4
la tache n° 2 s'execute sur la machine n° 1
la tache n° 3 s'execute sur la machine n° 0
la tache n° 1 s'execute sur la machine n° 0
la tache n° 4 s'execute sur la machine n° 4
la tache n° 5 s'execute sur la machine n° 3
la tache n° 8 s'execute sur la machine n° 0
la tache n° 6 s'execute sur la machine n° 4
la tache n° 7 s'execute sur la machine n° 1
la tache n° 9 s'execute sur la machine n° 4
```

Annexe

```
la population initiale :
1
0
3
2
1
1
2
1
0
3
l'ordre d'execution des taches :

le nombre de lignes est :  10
tache: 0
tache: 2
tache: 3
tache: 1
tache: 4
tache: 5
tache: 8
tache: 6
tache: 7
tache: 9

l'affectation des taches aux machines :
la tache n° 0 s'execute sur la machine n° 1
la tache n° 2 s'execute sur la machine n° 0
la tache n° 3 s'execute sur la machine n° 3
la tache n° 1 s'execute sur la machine n° 2
la tache n° 4 s'execute sur la machine n° 1
la tache n° 5 s'execute sur la machine n° 1
la tache n° 8 s'execute sur la machine n° 2
la tache n° 6 s'execute sur la machine n° 1
la tache n° 7 s'execute sur la machine n° 0
la tache n° 9 s'execute sur la machine n° 3

la population initiale :
4
2
1
4
1
4
3
2
0
3
l'ordre d'execution des taches :

le nombre de lignes est :  10
tache: 0
tache: 2
tache: 3
tache: 1
tache: 4
tache: 5
tache: 8
tache: 6
tache: 7
tache: 9

l'affectation des taches aux machines :
la tache n° 0 s'execute sur la machine n° 4
la tache n° 2 s'execute sur la machine n° 2
la tache n° 3 s'execute sur la machine n° 1
la tache n° 1 s'execute sur la machine n° 4
la tache n° 4 s'execute sur la machine n° 1
la tache n° 5 s'execute sur la machine n° 4
la tache n° 8 s'execute sur la machine n° 3
la tache n° 6 s'execute sur la machine n° 2
la tache n° 7 s'execute sur la machine n° 0
la tache n° 9 s'execute sur la machine n° 3
```

Annexe

```
la population initiale :
3
0
2
1
2
0
1
4
0
4
l'ordre d'execution des taches :

le nombre de lignes est :  10
tache: 0
tache: 2
tache: 3
tache: 1
tache: 4
tache: 5
tache: 8
tache: 6
tache: 7
tache: 9

l'affectation des taches aux machines :
la tache n° 0 s'execute sur la machine n° 3
la tache n° 2 s'execute sur la machine n° 0
la tache n° 3 s'execute sur la machine n° 2
la tache n° 1 s'execute sur la machine n° 1
la tache n° 4 s'execute sur la machine n° 2
la tache n° 5 s'execute sur la machine n° 0
la tache n° 8 s'execute sur la machine n° 1
la tache n° 6 s'execute sur la machine n° 4
la tache n° 7 s'execute sur la machine n° 0
la tache n° 9 s'execute sur la machine n° 4

la population initiale :
2
2
4
3
2
2
4
1
1
1
l'ordre d'execution des taches :

le nombre de lignes est :  10
tache: 0
tache: 2
tache: 3
tache: 1
tache: 4
tache: 5
tache: 8
tache: 6
tache: 7
tache: 9

l'affectation des taches aux machines :
la tache n° 0 s'execute sur la machine n° 2
la tache n° 2 s'execute sur la machine n° 2
la tache n° 3 s'execute sur la machine n° 4
la tache n° 1 s'execute sur la machine n° 3
la tache n° 4 s'execute sur la machine n° 2
la tache n° 5 s'execute sur la machine n° 2
la tache n° 8 s'execute sur la machine n° 4
la tache n° 6 s'execute sur la machine n° 1
la tache n° 7 s'execute sur la machine n° 1
la tache n° 9 s'execute sur la machine n° 1
```

Annexe

```
la population initiale :
0
0
4
0
4
4
4
3
4
1
l'ordre d'execution des taches :

le nombre de lignes est :  10
tache: 0
tache: 2
tache: 3
tache: 1
tache: 4
tache: 5
tache: 8
tache: 6
tache: 7
tache: 9

l'affectation des taches aux machines :
la tache n° 0 s'execute sur la machine n° 0
la tache n° 2 s'execute sur la machine n° 0
la tache n° 3 s'execute sur la machine n° 4
la tache n° 1 s'execute sur la machine n° 0
la tache n° 4 s'execute sur la machine n° 4
la tache n° 5 s'execute sur la machine n° 4
la tache n° 8 s'execute sur la machine n° 4
la tache n° 6 s'execute sur la machine n° 3
la tache n° 7 s'execute sur la machine n° 4
la tache n° 9 s'execute sur la machine n° 1


la population initiale :
0
0
1
4
3
2
4
3
3
4
l'ordre d'execution des taches :

le nombre de lignes est :  10
tache: 0
tache: 2
tache: 3
tache: 1
tache: 4
tache: 5
tache: 8
tache: 6
tache: 7
tache: 9

l'affectation des taches aux machines :
la tache n° 0 s'execute sur la machine n° 0
la tache n° 2 s'execute sur la machine n° 0
la tache n° 3 s'execute sur la machine n° 1
la tache n° 1 s'execute sur la machine n° 4
la tache n° 4 s'execute sur la machine n° 3
la tache n° 5 s'execute sur la machine n° 2
la tache n° 8 s'execute sur la machine n° 4
la tache n° 6 s'execute sur la machine n° 3
la tache n° 7 s'execute sur la machine n° 3
la tache n° 9 s'execute sur la machine n° 4
```

Annexe

```
la population initiale :
3
4
4
1
3
1
2
4
1
4
l'ordre d'execution des taches :

le nombre de lignes est :  10
tache: 0
tache: 2
tache: 3
tache: 1
tache: 4
tache: 5
tache: 8
tache: 6
tache: 7
tache: 9

l'affectation des taches aux machines :
la tache n° 0 s'execute sur la machine n° 3
la tache n° 2 s'execute sur la machine n° 4
la tache n° 3 s'execute sur la machine n° 4
la tache n° 1 s'execute sur la machine n° 1
la tache n° 4 s'execute sur la machine n° 3
la tache n° 5 s'execute sur la machine n° 1
la tache n° 8 s'execute sur la machine n° 2
la tache n° 6 s'execute sur la machine n° 4
la tache n° 7 s'execute sur la machine n° 1
la tache n° 9 s'execute sur la machine n° 4

la population initiale :
4
4
0
1
2
4
0
1
3
2
l'ordre d'execution des taches :

le nombre de lignes est :  10
tache: 0
tache: 2
tache: 3
tache: 1
tache: 4
tache: 5
tache: 8
tache: 6
tache: 7
tache: 9

l'affectation des taches aux machines :
la tache n° 0 s'execute sur la machine n° 4
la tache n° 2 s'execute sur la machine n° 4
la tache n° 3 s'execute sur la machine n° 0
la tache n° 1 s'execute sur la machine n° 1
la tache n° 4 s'execute sur la machine n° 2
la tache n° 5 s'execute sur la machine n° 4
la tache n° 8 s'execute sur la machine n° 0
la tache n° 6 s'execute sur la machine n° 1
la tache n° 7 s'execute sur la machine n° 3
la tache n° 9 s'execute sur la machine n° 2
```

Annexe

```
la population initiale :
2
0
0
1
2
3
2
0
1
4
l'ordre d'execution des taches :

le nombre de lignes est :  10
tache: 0
tache: 2
tache: 3
tache: 1
tache: 4
tache: 5
tache: 8
tache: 6
tache: 7
tache: 9

l'affectation des taches aux machines :
la tache n° 0 s'execute sur la machine n° 2
la tache n° 2 s'execute sur la machine n° 0
la tache n° 3 s'execute sur la machine n° 0
la tache n° 1 s'execute sur la machine n° 1
la tache n° 4 s'execute sur la machine n° 2
la tache n° 5 s'execute sur la machine n° 3
la tache n° 8 s'execute sur la machine n° 2
la tache n° 6 s'execute sur la machine n° 0
la tache n° 7 s'execute sur la machine n° 1
la tache n° 9 s'execute sur la machine n° 4

la population initiale :
4
0
1
2
1
0
2
2
0
4
l'ordre d'execution des taches :

le nombre de lignes est :  10
tache: 0
tache: 2
tache: 3
tache: 1
tache: 4
tache: 5
tache: 8
tache: 6
tache: 7
tache: 9

l'affectation des taches aux machines :
la tache n° 0 s'execute sur la machine n° 4
la tache n° 2 s'execute sur la machine n° 0
la tache n° 3 s'execute sur la machine n° 1
la tache n° 1 s'execute sur la machine n° 2
la tache n° 4 s'execute sur la machine n° 1
la tache n° 5 s'execute sur la machine n° 0
la tache n° 8 s'execute sur la machine n° 2
la tache n° 6 s'execute sur la machine n° 2
la tache n° 7 s'execute sur la machine n° 0
la tache n° 9 s'execute sur la machine n° 4
```

Annexe

```
la population initiale :
2
4
1
4
4
1
1
2
0
4
l'ordre d'execution des taches :

le nombre de lignes est :  10
tache: 0
tache: 2
tache: 3
tache: 1
tache: 4
tache: 5
tache: 8
tache: 6
tache: 7
tache: 9

l'affectation des taches aux machines :
la tache n° 0 s'execute sur la machine n° 2
la tache n° 2 s'execute sur la machine n° 4
la tache n° 3 s'execute sur la machine n° 1
la tache n° 1 s'execute sur la machine n° 4
la tache n° 4 s'execute sur la machine n° 4
la tache n° 5 s'execute sur la machine n° 1
la tache n° 8 s'execute sur la machine n° 1
la tache n° 6 s'execute sur la machine n° 2
la tache n° 7 s'execute sur la machine n° 0
la tache n° 9 s'execute sur la machine n° 4

la population initiale :
1
2
2
4
3
4
3
4
1
1
l'ordre d'execution des taches :

le nombre de lignes est :  10
tache: 0
tache: 2
tache: 3
tache: 1
tache: 4
tache: 5
tache: 8
tache: 6
tache: 7
tache: 9

l'affectation des taches aux machines :
la tache n° 0 s'execute sur la machine n° 1
la tache n° 2 s'execute sur la machine n° 2
la tache n° 3 s'execute sur la machine n° 2
la tache n° 1 s'execute sur la machine n° 4
la tache n° 4 s'execute sur la machine n° 3
la tache n° 5 s'execute sur la machine n° 4
la tache n° 8 s'execute sur la machine n° 3
la tache n° 6 s'execute sur la machine n° 4
la tache n° 7 s'execute sur la machine n° 1
la tache n° 9 s'execute sur la machine n° 1
```

Et dans VAR on trouvera les meilleurs individus parmi les 20 crées:

Annexe

VAR

```
1 1 0 4 0 3 2 2 4 3
2 2 4 3 2 2 4 1 1 1
```

Et dans FUN le temps d'exécution et coût des meilleurs individus :

FUN

```
33.0 317.0
47.0 289.0
```