Allocation optimale multicontraintes des workflows aux ressources d'un environnement Cloud Computing

Sonia Yassa

Université de Cergy-Pontoise
Ecole Doctorale Sciences et Ingénierie
ETIS (CNRS UMR 8051)
EISTI (Ecole Internationale des Sciences de Traitement de l'Information)

Thèse présentée par

Sonia YASSA

pour obtenir le grade de Docteur de l'Université de Cergy-Pontoise
Spécialité :
Sciences et Technologie de l'Information et de la Communication
STIC

Allocation optimale multi-contraintes des *workflows* aux ressources d'un environnement *cloud computing*

Soutenue le 10 juillet 2014 devant le jury composé de :

Pr. Dominique LAURENT	Université de Cergy Pontoise	Président du jury
Pr. Christophe CERIN	Université Paris 13	Rapporteur
Pr. Patrick SIARRY	Université de Paris-Est Créteil	Rapporteur
MdC. Alain PETROWSKI	Télécom SudParis	Examinateur
Pr. Bertrand GRANADO	Université Pierre et Marie Curie	Directeur de thèse
HDR. Rachid CHELOUAH	EISTI	Co-directeur de thèse
HDR. Hubert KADIMA	EISTI	Co-directeur de thèse

Résumé: Le *cloud computing* 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. Dans cette thèse, nous abordons le problème d'ordonnancement de *workflows* sur les infrastructures distribuées hétérogènes du *cloud computing*. Les approches d'ordonnancement de *workflows* existant dans le *cloud* se concentrent principalement sur l'optimisation biobjectif du *makespan* et du coût. Dans cette thèse, nous proposons des algorithmes d'ordonnancement de *workflows* basés sur des métaheuristiques. Nos algorithmes sont capables de gérer plus de deux métriques de QoS (*Quality of Service*), notamment, le *makespan*, le coût, la fiabilité, la disponibilité et l'énergie dans le cas de ressources physiques. En outre, ils traitent plusieurs contraintes selon les exigences spécifiées dans le SLA (*Service Level Agreement*). Nos algorithmes ont été évalués par simulation en utilisant (1) comme applications: des *workflows* synthétiques et des *workflows* scientifiques issus du monde réel et ayant des structures différentes; (2) et comme ressources *cloud*: les caractéristiques des services de Amazon EC2. Les résultats obtenus montrent l'efficacité de nos algorithmes pour le traitement de plusieurs QoS. Nos algorithmes génèrent une ou plusieurs solutions dont certaines surpassent la solution de l'heuristique de référence HEFT sur toutes les QoS considérées, y compris le *makespan* pour lequel HEFT est censé donner de bons résultats.

Mots clés: *cloud computing, workflow, ordonnancement, SLA, QoS, optimisation multiobjectif, métaheuristiques, algorithmes évolutionnaires, PSO.*

Abstract: *cloud computing* is increasingly recognized as a new way to use on-demand, computing, storage and network services in a transparent and efficient way. In this thesis, we address the problem of *workflows* scheduling on distributed heterogeneous infrastructure of *cloud computing* . The existing *workflows* scheduling approaches mainly focus on the bi-objective optimization of the *makespan* and the cost. In this thesis, we propose new *workflows* scheduling algorithms based on metaheuristics. Our algorithms are able to handle more than two QoS (Quality of Service) metrics, namely, *makespan*, cost, reliability, availability and energy in the case of physical resources. In addition, they address several constraints according to the specified requirements in the SLA (*Service Level Agreement*). Our algorithms have been evaluated by simulations. We used (1) synthetic *workflows* and real world scientific *workflows* having different structures, for our applications; and (2) the features of Amazon EC2 services for our *cloud*. The obtained results show the effectiveness of our algorithms when dealing with multiple QoS metrics. Our algorithms produce one or more solutions which some of them outperform the solution produced by reference HEFT heuristic over all the QoS considered, including the *makespan* for which HEFT is supposed to give good results.

Keywords: *cloud computing, workflow, scheduling, SLA, QoS, multiobjective optimization, metaheuristics, evolutionary algorithms, PSO.*

Table des matières

Introduction générale

Contexte et motivations

Le *cloud computing* 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. Généralement, il peut fournir quatre types de services: SaaS (*Software as a Service*), PaaS (*Platform as a Service*), IaaS (*Infrastructure as a Service*) et HaaS (*Hardware as a Service*). Ces services sont offerts avec différents niveaux de qualité de service afin de répondre aux besoins spécifiques de différents utilisateurs. Bien que de nombreux services de *cloud computing* ont des fonctionnalités similaires (par exemple des services de calculs, services de stockages, services réseaux, etc.), ils diffèrent les uns des autres par leurs QoS (Qualité de Service) non-fonctionnelles, telles que le temps, le coût, la disponibilité du service, la consommation d'énergie, etc.

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*. Pour les fournisseurs des services du *cloud*, appliquer de tels contrats SLA est impératif. Le manque de tels accords peut entraîner l'éloignement des utilisateurs d'un fournisseur *cloud* et compromettra la croissance de ce dernier.

Plusieurs applications scientifiques dans de nombreux domaines tels que la bioinformatique (Oinn, 2004), la physique (Ludascher, 2005), l'astronomie (Deelman, 2003) et bien d'autres 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, où chaque étape prend des données en entrée, effectue un traitement, et produit certaines données en sortie qui seront transmises aux étapes de traitement suivantes. Ces applications peuvent être modélisées par des graphes orientés acycliques, DAG (*Direct Acyclic Graph*), où les sommets dans le DAG représentent les tâches et les arcs représentent les dépendances des flux de données. Ces *workflows* impliquent souvent de longues simulations, des calculs à grande échelle et la

manipulation massive des données. Ils requièrent, pour leurs exécutions, des puissances de calcul élevées et la disponibilité de grandes infrastructures informatiques. Récemment, le *cloud computing* leur fournit, à la demande, ces infrastructures sous forme d'IaaS ou de HaaS avec différents niveaux de QoS. Une infrastructure *cloud* en tant que système distribué de ressources informatiques (serveurs, stockage, réseaux,…) hétérogènes affiche en général plus au moins le même comportement que celui d'une grille de calcul classique.

En raison de l'importance des applications de *workflows*, plusieurs projets de recherche ont été menés pour concevoir des systèmes de gestion de *workflows* et des algorithmes d'ordonnancement appropriés. Les systèmes de gestion de *workflows* peuvent être considérés comme un type de service facilitant l'automatisation des applications de e-business et e-science sur la grille et le *cloud*. Ils sont utilisés pour gérer ces applications de façon transparente en masquant l'orchestration et les détails d'intégration spécifiques lors de l'exécution des tâches sur les ressources distribuées de grille ou de *cloud*.

En vue d'ordonnancer efficacement les tâches de ces applications sur les environnements de *cloud computing*, les systèmes de gestion de *workflows* requièrent des stratégies d'ordonnancement plus élaborées pour répondre aux exigences de QoS (*makespan*, coût et autres) spécifiées dans le SLA, ainsi qu'aux relations de précédence entre les tâches du *workflow*. L'étude des stratégies d'ordonnancement de *workflows* devient un enjeu important dans le *cloud computing*.

Problèmes et objectifs

L'ordonnancement de *workflows* dans un *cloud* est un problème difficile. Ce problème est d'autant plus difficile lorsqu'il y a plusieurs facteurs à prendre en compte, à savoir: (1) les différentes contraintes et exigences de QoS des utilisateurs (par exemple le temps de réponse du service, le coût des services et autres); (2) l'hétérogénéité, la dynamique et l'élasticité des services de *cloud* disponibles; (3) les diverses possibilités de combiner ces services pour exécuter les tâches de *workflows*; (4) le transfert de gros volumes de données (les ressources peuvent être géographiquement distantes les unes des autres, ceci peut poser des problèmes aux *workflows* qui comportent une énorme quantité de données) et autres. Cependant, le problème d'ordonnancement de *workflows* est vu comme un problème d'optimisation

combinatoire, où il est impossible de trouver la solution globale optimale en utilisant des algorithmes ou des règles simples. Il est bien connu comme un problème NP-complet (Lenstra, 1978; Brucker, 2004) et dépend de la taille du problème à résoudre.

Le problème d'ordonnancement de *workflows* dans le *cloud* est largement étudié dans de nombreux travaux (Liu, 2010; Salehi, 2010; Guo, 2012; Verma, 2012). La plupart de ces travaux se sont concentrés uniquement sur l'optimisation de deux métriques de QoS, souvent, le *makespan* (temps d'exécution) et le coût. Dans cette thèse, notre objectif est de développer et d'évaluer de nouveaux algorithmes d'optimisation multiobjectif pour l'ordonnancement de *workflows* contraints par les SLA dans des environnements hétérogènes aux niveaux IaaS ou HaaS de l'infrastructure du *cloud computing*.

Dans tous les cas, les algorithmes doivent :

- respecter la *contrainte de précédence* entre les différentes tâches qui composent le *workflow* ;
- prendre en compte plusieurs métriques de QoS, telles que: le *makespan*, le coût, la fiabilité, la disponibilité, l'énergie dans le cas du HaaS et autres.

Selon les exigences spécifiées dans le SLA :

- respecter des *contraintes de QoS*, telles que, la *deadline* (échéance), le budget (coût maximal) et autres ;
- gérer d'autres *contraintes fortes*, par exemple, les restrictions d'exécuter certaines tâches sur certains types de ressources, ou pour des raisons de sécurité, des questions de puissance de calcul ou des exigences logicielles.

Contributions

L'ordonnancement de *workflows* est considéré comme un problème d'optimisation multiobjectif dont la complexité augmente avec l'augmentation du nombre de paramètres à considérer. Les métaheuristiques telles que les algorithmes évolutionnaires et l'optimisation par essaim particulaire, PSO (*Particle Swarm Optimization*), sont de bons candidats pour résoudre cette problématique. Les contributions de cette thèse portent, principalement, sur deux modèles de services de *cloud computing*, à savoir: le modèle *d'Infrastructure as a*

Service (IaaS) et le modèle du *Hardware as a Service* (HaaS). Elles peuvent êtres résumées comme suit :

- Au niveau d'IaaS virtuelles, nos principales contributions sont les suivantes :

- Développement d'un algorithme génétique (GA) pour l'ordonnancement de *workflows* basé sur les métriques de QoS (Yassa, 2011 ; Yassa, 2012 ; Yassa, 2013). Le but est d'optimiser plusieurs métriques de QoS, à savoir, le *makespan*, le coût, la fiabilité et la disponibilité lors de l'ordonnancement de *workflows* sur des infrastructures virtuelles de *cloud computing*.

- Développement d'un algorithme génétique basé Infection virale (GA*) pour l'ordonnancement de *workflows* tenant compte des contraintes de QoS et de contraintes fortes (Sublime, 2012 ; Yassa, 2013a). Le but est de développer un algorithme générique qui soit compatible avec un grand nombre de paramètres et qui prenne en compte les différentes contraintes citées précédemment, à savoir: les *contraintes QoS,* les *contraintes fortes* et la *contrainte de précédence.*
- Développement des algorithmes NSGA-II (*Non Dominated Sorting Genetic Algorithm-II*) et PAES (*Pareto Archive Evolution Strategy*) pour l'ordonnancement de *workflows* basé sur les QoS (Yassa, 2013a). Le but est de générer un ensemble de solutions de compromis selon les exigences des QoS spécifiées dans le SLA.

- Au niveau de l'infrastructure physique (HaaS), notre principale contribution concerne le développement d'une méthode basée sur l'optimisation multiobjectif par essaim particulaire combinée avec la technique d'adaptation dynamique de la tension et de la fréquence (*Dynamic Voltage and Frequency Scaling:* DVFS) nommée DVFS-MODPSO (*Multi-objective Discrete Particle Swarm Optimization*) (Yassa, 2012b ; Yassa, 2013b). Le but est d'optimiser simultanément plusieurs objectifs contradictoires dans un espace discret, notamment le *makespan*, le coût et l'énergie consommée, offrant ainsi plus de flexibilité à l'utilisateur pour évaluer ses préférences et choisir un ordonnancement qui réponde au mieux à ses exigences en termes de QoS.

Organisation du manuscrit

Cette thèse est divisée en quatre chapitres. Les chapitres 1 et 2 présentent le contexte et l'état de l'art des domaines sur lesquels nous travaillons, à savoir: le *cloud computing* et le *workflow* d'une part, l'optimisation multiobjectif et l'ordonnancement de *workflow* dans le *cloud*, d'autre part. Le chapitre 3 et le chapitre 4 détaillent nos contributions pour l'ordonnancement multiobjectif de *workflows*, respectivement, au niveau de l'offre des services *IaaS* et *HaaS* du *cloud*.

Le premier chapitre présente les concepts nécessaires à la compréhension générale de notre travail. Nous présentons, dans une première partie, le concept du *cloud computing*, ses modèles de services, ses modèles de déploiement et ses principaux acteurs. Nous explorons quelques challenges de recherche, dont l'ordonnancement de *workflows* dans le *cloud* fait partie. Ce défi attire notre attention et fait l'objet de cette thèse. Dans la deuxième partie de ce chapitre, nous présentons une introduction au *workflow* et aux systèmes de gestion de *workflows*, et nous mettons en lumière les intérêts du *cloud* pour les *workflows*.

Le chapitre 2 présente, d'abord, les principes de l'optimisation combinatoire multiobjectif qui nous sera utile pour modéliser et résoudre le problème d'ordonnancement de *workflows* dans le *cloud computing*. Nous présentons les concepts de base de l'optimisation multiobjectif, les méthodologies de résolution et les différentes familles de métaheuristiques. La deuxième partie de ce chapitre, présente un état de l'art des algorithmes d'ordonnancement de *workflows* dans le *cloud computing*. Nous analysons les problèmes des algorithmes existants et nous résumons les exigences à prendre en compte par les nouveaux algorithmes.

Dans le chapitre 3, nous traitons le problème d'ordonnancement de *workflows* sur l'*IaaS*. L'objectif ici est d'optimiser plusieurs métriques de QoS (*makespan*, coût, fiabilité, disponibilité), de satisfaire des contraintes sur les QoS (*deadline*, budget et autres …) et de gérer des contraintes fortes. Pour ce faire, après avoir présenté le modèle formel du problème d'ordonnancement de *workflows* sur l'infrastructure IaaS, nous présentons quatre métaheuristiques pour sa résolution. Nous adaptons les algorithmes évolutionnaires au problème d'ordonnancement de *workflows* dans le *cloud*. Nous présentons les algorithmes GA et GA* agrégeant les différentes métriques de QoS. Nous détaillons les algorithmes NSGA-II et PAES basés Pareto. Et enfin, nous présentons une évaluation expérimentale pour chaque algorithme et comparons leurs performances respectives.

Dans le chapitre 4, nous abordons le problème d'ordonnancement de *workflows* au niveau *HaaS*. Le but consiste à considérer, en plus des métriques de QoS du *makespan* et du coût, la consommation énergétique des ressources. Après avoir présenté un modèle formel du problème d'ordonnancement de *workflows* au niveau *HaaS*, nous détaillons l'algorithme d'ordonnancement, DVFS-MODPSO, basé Pareto. Une évaluation expérimentale montre l'efficacité de l'algorithme dans la réduction de l'énergie consommée.

Dans la dernière partie, nous présentons la conclusion et les perspectives de cette thèse.

Chapitre 1

Concepts fondamentaux de *cloud* et de *workflow*

1.1. Introduction

Le *cloud computing,* traduit le plus souvent en français par « *informatique dans les nuages* », « *informatique dématérialisée* » ou encore « *infonuagique* », est un domaine qui regroupe un ensemble de techniques et de pratiques consistant à accéder, en libre-service, à du matériel ou à des logiciels informatiques, à travers une infrastructure réseau (Internet). Ce concept rend possible la distribution des ressources informatiques sous forme de services pour lesquels l'utilisateur paie uniquement pour ce qu'il utilise. Ces services peuvent être utilisés pour exécuter des applications scientifiques et commerciales, souvent modélisées sous forme de *workflows*.

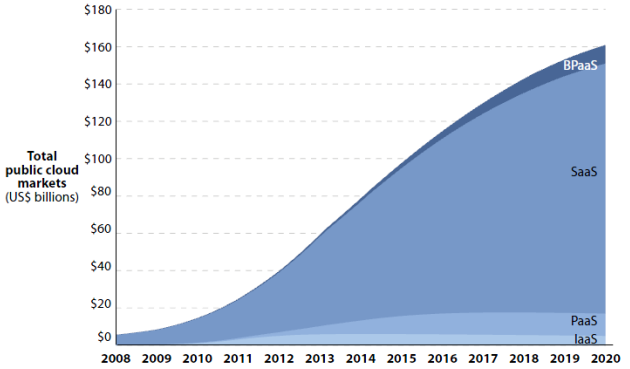
Ce chapitre présente une introduction au *cloud computing* et au *workflow*, nécessaire pour la compréhension générale de ce rapport.

Tout d'abord, nous présentons dans la section 1.2 une introduction au paradigme du *cloud computing.* Nous donnons un aperçu général du *cloud computing,* y compris sa définition, ses caractéristiques principales et une comparaison avec les technologies connexes. Nous présentons les différents modèles de service, les différents modèles de déploiement, ainsi que les différents acteurs du *cloud computing.* Nous résumons quelques challenges de recherche en *cloud computing.* Par la suite, nous présentons, dans la section 1.3, une introduction au *workflow* et systèmes de gestion de *workflow*. Nous donnons le concept du *workflow*, sa définition, et l'architecture de référence d'un système de gestion de *workflows*. Nous énumérons quelques systèmes de gestion de *workflows* existant dans les grilles et *cloud*s et, finalement, nous résumons l'intérêt du *cloud* pour les *workflows*.

1.2. Paradigme du *cloud computing*

1.2.1. Concept du *cloud computing*

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 (Garfinkel, 1999). C'est ensuite, vers la fin des années 90, que ce concept a pris de l'importance avec l'avènement du *grid computing* (Foster, 1999). Le terme *cloud* 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. Ici, les grandes centrales sont les *Datacenter*, le réseau est le plus souvent celui d'Internet et l'électricité correspond aux ressources informatiques. Le *cloud computing* n'est véritablement apparu qu'au cours de l'année 2006 (Vouk, 2008) avec l'apparition d'Amazon EC2 (*Elastic Compute 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*). D'après une étude menée par Forrester (Ried, 2011), le marché du *cloud computing* s'élevait à environ 5,5 milliards de dollars en 2008, il devrait atteindre plus de 150 milliards d'ici 2020, comme l'illustre la figure 1.1.



	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020
BPaaS ($)	0.15	0.23	0.35	0.53	0.80	1.26	1.95	2.93	4.28	6.00	7.66	9.08	10.02
SaaS ($)	5.56	8.09	13.40	21.21	33.09	47.22	63.19	78.43	92.75	105.49	116.39	125.52	132.57
PaaS ($)	0.05	0.12	0.31	0.82	2.08	4.38	7.39	9.80	11.26	11.94	12.15	12.10	11.91
IaaS ($)	0.06	0.24	1.02	2.94	4.99	5.75	5.89	5.82	5.65	5.45	5.23	5.01	4.78

Figure 1.1. Prévisions de la taille du marché du *cloud computing* public (Ried, 2011).

1.2.1.1. Vers une définition du *cloud computing*

Beaucoup de chercheurs ont tenté de définir le *cloud computing* (Geelan, 2008 ; McFedries, 2008 ; Buyya, 2009 ; Armbrust, 2010). La plupart des définitions attribuées à ce concept semblent se concentrer seulement sur certains aspects technologiques. L'absence d'une définition standard a généré non seulement des exagérations du marché, mais aussi des confusions. Pour cette raison, il y

a eu récemment des travaux sur la normalisation de la définition du *cloud computing,* à l'exemple de Vaquero et coll (Vaquero, 2009) qui ont comparé plus de 20 définitions différentes et ont proposé une définition globale. En guise de synthèse des différentes propositions données dans la littérature, nous introduisons une définition mixte, qui correspond aux différents types de *cloud* considérés dans les travaux réalisés dans cette thèse.

Nous définissons le *cloud* comme 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*).

1.2.1.2. Caractéristiques principales du *cloud computing*

Le *cloud computing* possède les caractéristiques suivantes :

- **Accès en libre-service à la demande.** Le *cloud computing* offre des ressources et services aux utilisateurs à la demande. Les services sont fournis de façon automatique, sans nécessiter d'interaction humaine (Mell, 2011).
- **Accès réseau universel.** Les services de *cloud computing* sont facilement accessibles au travers du réseau, par le biais de mécanismes standard, qui permettent une utilisation depuis de multiples types de terminaux (par exemple, les ordinateur portables, tablettes, smartphones) (Mell, 2011).
- **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 (Mell, 2011). En général, les utilisateurs n'ont aucun contrôle ou connaissance sur l'emplacement exact des ressources fournies. Toutefois, ils peuvent imposer de spécifier l'emplacement à un niveau d'abstraction plus haut.
- **Scalabilité et élasticité.** Des ressources supplémentaires peuvent être automatiquement mises à disposition des utilisateurs en cas d'accroissement de la demande (en réponse à l'augmentation des charges des applications) (Geelan, 2008), et peuvent être libérées lorsqu'elles ne sont plus nécessaires. 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*).
- **Autonome.** Le *cloud computing* est un système autonome et géré de façon transparente pour les utilisateurs. Le matériel, le logiciel et les données au sein du *cloud* peuvent être

automatiquement reconfigurés, orchestrés et consolidés en une seule image qui sera fournie à l'utilisateur (Wang, 2008).

- **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 (Armbrust, 2009).
- **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 (Buyya, 2008).
- **Garantie QoS.** 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 (Wang, 2008).
- **Basé-SLA.** Les *cloud*s sont gérés dynamiquement en fonction des contrats d'accord de niveau de service (SLA) (Buyya, 2008) 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.

Après avoir présenté les caractéristiques essentielles d'un service *cloud*, nous présentons, brièvement, dans la section suivante, quelques technologies connexes aux *clouds*.

1.2.1.3. Technologies connexes

Le *cloud computing* utilise des technologies telles que la virtualisation, l'architecture orientée services et les services web. Le *cloud* est souvent confondu avec plusieurs paradigmes informatiques, tels que *le Grid computing, l'Utility computing* et *l'Autonomic computing*, dont chacun partage certains aspects avec le *cloud computing.* La figure 1.2 montre la convergence des technologies qui ont contribué à l'avènement du *cloud computing.*

- ***Grid computing*** Le *grid computing* est un paradigme de calcul réparti qui coordonne des ressources autonomes et géographiquement distribuées pour atteindre un objectif de calcul commun. Le *grid* est basé sur le partage dynamique des ressources entre des participants, des organisations et des entreprises dans le but de pouvoir les mutualiser, et faire ainsi exécuter des applications scientifiques, qui sont généralement des calculs intensifs ou des traitements de très gros volumes de données. Le *cloud computing* est similaire au *grid computing* : les deux

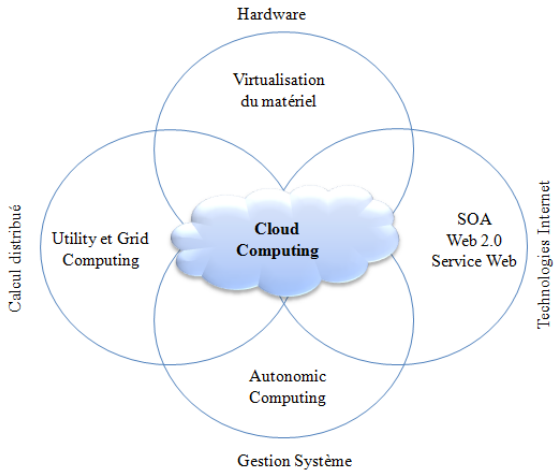


Figure 1.2. Convergence des différentes avancées menant à l'avènement du *cloud computing.*

adoptent le concept d'offrir les ressources sous forme de services. Toutefois, ils sont différents dans leur finalité : tandis que la technologie des grilles vise essentiellement à permettre à des groupes différents de partager l'accès en libre service à leurs ressources, le *cloud* a pour objectif de fournir aux utilisateurs des services « à la demande », sur le principe du « paiement à l'usage ». De plus le *cloud* exploite les technologies de virtualisation à plusieurs niveaux (matériel et plateforme d'application) pour réaliser le partage et l'approvisionnement dynamique des ressources.

- ***Utility computing*** (informatique utilitaire). *L'utility computing* est simplement une délocalisation d'un système de calcul ou de stockage. Il présente un modèle d'affectation des ressources à la demande et facturation des utilisateurs en fonction de leur usage. Le *cloud computing* peut être perçu comme une réalisation de l'informatique utilitaire. Il adopte un système de tarification basé sur l'utilité, entièrement pour des raisons économiques. Avec l'approvisionnement des ressources à la demande et le paiement à l'usage, les fournisseurs de services peuvent maximiser l'utilisation des ressources et minimiser leurs coûts d'exploitation.
- ***L'autonomic computing*** *L'autonomic computing* vise à construire des systèmes informatiques capables de s'auto-administrer en s'adaptant à des changements internes et externes sans intervention humaine. Le but de l'informatique autonome est de surmonter la complexité de la gestion des systèmes informatiques d'aujourd'hui. Bien que le *cloud computing* présente certaines caractéristiques autonomes, telles que l'approvisionnement automatique des ressources, son objectif est de diminuer le coût des ressources, plutôt que de réduire la complexité du système.
- **Virtualisation** La virtualisation est une technologie qui fait abstraction des détails du matériel physique et fournit des ressources virtualisées pour les applications de haut niveau. En effet, la

virtualisation regroupe l'ensemble des techniques matérielles ou logicielles permettant de faire fonctionner, sur une seule machine physique, plusieurs configurations informatiques (systèmes d'exploitation,…), de sorte à former plusieurs machines virtuelles, qui reproduisent le comportement des machines physiques. La virtualisation constitue le socle du *cloud computing*, car elle offre la possibilité de mettre en commun des ressources informatiques à partir de *clusters* de serveurs, et ainsi d'affecter ou réaffecter dynamiquement des machines virtuelles aux applications à la demande. La figure 1.3. montre l'architecture en couches de la technologie de virtualisation. Le moniteur de machine virtuelle (VMM), également appelé *hyperviseur*, partitionne la ressource physique du serveur physique sous-jacente en plusieurs machines virtuelles différentes, chacune fonctionnant sous un système d'exploitation distinct et une pile de logiciels utilisateur.

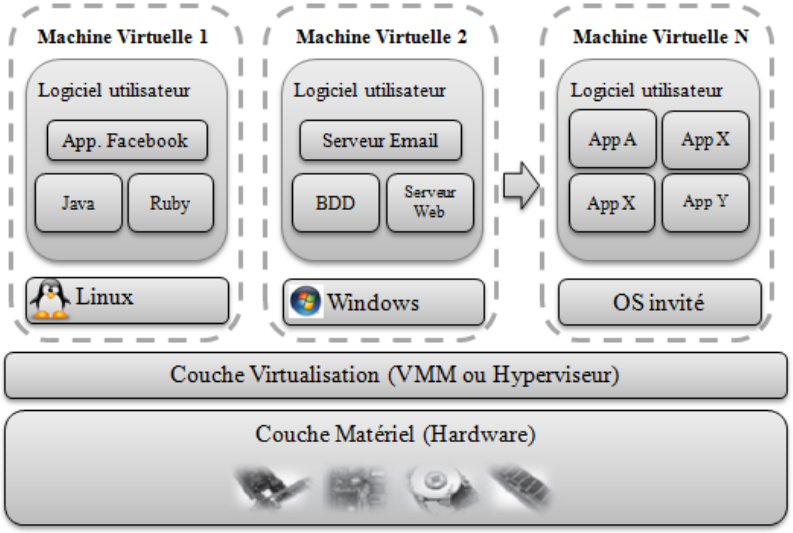


Figure 1.3. Une architecture en couches de la technologie de virtualisation.

1.2.2. Modèles du *cloud computing*

Cette section présente les modèles du *cloud*. Nous commençons par détailler les modèles de service, puis nous donnons les modèles de déploiement et enfin nous introduisons les acteurs du *cloud*.

1.2.2.1. Modèles de service du *cloud computing*

XaaS (*X as a Service*) représente la base du paradigme du *cloud computing*, où X représente un service tel qu'un logiciel, une plateforme, une infrastructure, un *Business Process*, etc. Nous présentons, dans cette section, quatre modèles de services (Rimal, 2009), à savoir: (1) Logiciel en tant que services SaaS (*Software as a Service*), où le matériel, l'hébergement, le *framework* d'application et le logiciel sont dématérialisés, (2) Plateforme en tant que service PaaS (*Platform as*

a Service), où le matériel, l'hébergement et le *framework* d'application sont dématérialisés, (3) Infrastructure en tant que service IaaS (*Infrastructure as a Service*) et (4) Matériel en tant que service HaaS (*Hardware as a Service*), où seul le matériel (serveurs) est dématérialisé dans ces deux derniers cas. La figure 1.4 montre le modèle classique et les différents modèles de service de *cloud*.

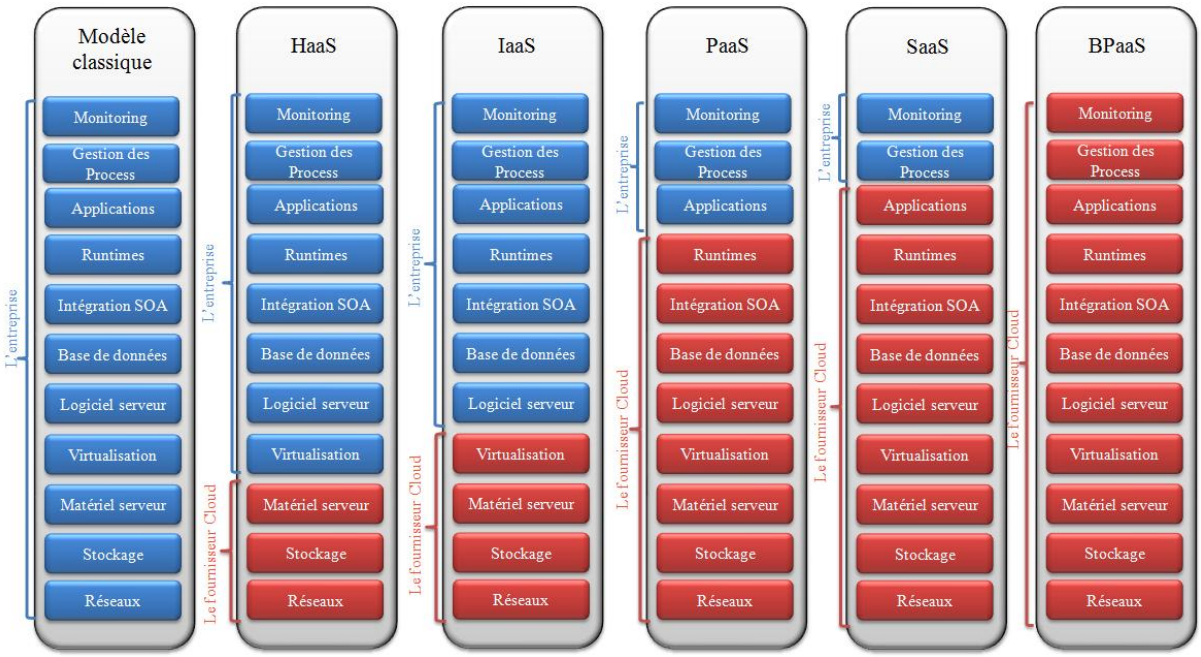


Figure 1.4. Les services XaaS du *cloud computing*

- ***Software as a Service* (SaaS)**. Le SaaS offre aux utilisateurs des applications sous forme de services en ligne déjà déployés dans le *cloud*. Cette couche est gérée par le fournisseur de SaaS de façon transparente pour les utilisateurs. Ces derniers ne savent ni où, ni comment ces applications sont déployées. Ils ne payent pas pour les posséder, mais plutôt pour les utiliser. Ils peuvent accéder à ces applications à tout moment via internet par le biais de n'importe quel dispositif connecté, tel que les ordinateur portables, tablettes, smartphones. Les applications sont utilisées soit directement via l'interface disponible, soit via des API fournies (souvent réalisées grâce aux *Web Services* ou à l'architecture REST (*REpresentational State Transfer*)). Les principales applications actuelles de ce modèle sont les applications de gestion de la relation client (CRM : *Customer Relationship Management*), 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).
- ***Business Process as a Service* (BPaaS)**. Contrairement au SaaS, qui concerne les applications logicielles, le BPaaS cible les processus métiers et les ressources sont mutualisées entre les différents utilisateurs. Il s'intègre au dessus des applications. Ce service est important pour les

entreprises, car il permet l'automatisation et l'orchestration des flux d'actions et la supervision de ces flux. En tant que service de *cloud*, le modèle BPaaS est accessible via les technologies Internet. Des exemples de BPaaS sont : la gestion des voyages, le paiement en ligne, la gestion financière, etc. Les fournisseurs de ce service sont : IBM, Atos, Wipro et autres.

- ***Platform as a Service* (PaaS)**. Ce type de *cloud* est plus orienté pour servir des développeurs d'applications. Il 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*).
- ***Hardware as a Service (HaaS).*** Dans ce cas, le fournisseur de *cloud* loue essentiellement du matériel physique (par exemple, serveur, stockage). Les utilisateurs accèdent au HaaS via Internet pour installer et configurer les serveurs loués. Ils ont le contrôle sur le système d'exploitation et la pile logiciel/matériel. Les communautés de recherche louent du HaaS pour leurs applications et les configurent selon leurs besoins. Par conséquent, les *workflows* ou applications orientées calculs-intensifs et/ou traitement intensifs de données (Rimal, 2009; Egwutuoha, 2012), qui étaient traditionnellement exécutés sur des systèmes HPC, peuvent maintenant être exécutés dans le *cloud*. Parmi les principaux fournisseurs qui offrent du HaaS, on peut citer Baremetal*cloud* (Baremetal, 2014), Softlayer (SoftLayer, 2014) et Grid5000 (Grid, 2014).
- ***Infrastructure as a Service (IaaS).*** L'IaaS permet la mise à disposition des ressources d'infrastructure telles que des capacités de calcul, des moyens de stockage, du réseau sous forme de services publics**.** L'IaaS est similaire au HaaS, mais les ressources offertes sont virtualisées**.** 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. Les utilisateurs

peuvent héberger et exécuter des logiciels, des applications quelconques, ou encore stocker des données sur l'infrastructure et ne paient que les ressources qu'ils consomment.

La particularité de l'IaaS est de fournir un approvisionnement en ressources informatiques, qui soit dynamique, flexible, extensible et qui possède de bonnes propriétés de passage à l'échelle pour répondre aux besoins spécifiques des utilisateurs. Le *cloud* est vu ainsi comme une source infinie de ressources de calcul, de stockage et de réseau accessibles en un modèle de paiement à l'usage. Internet devient une place de marché où l'infrastructure informatique est distribuée, et consommée en tant que marchandise.

Il existe de nombreux fournisseurs commerciaux et *open-source* de l'IaaS. Parmi les plateformes commerciales, nous pouvons citer: Amazon EC2 (Amaz, 2012), Rackspace (Racks, 2014), GoGrid (GoG, 2014), Microsoft Azure (Micros, 2014) et d'autres. Dans les communautés *open-sources*, plusieurs plateformes ont été développées, à savoir : *Eucalyptus* (Eucal, 2014), *Nimbus* (Nimb, 2014), *OpenNebula* (OpenN, 2014), *OpenStack* (OpenS, 2014) et d'autres.

Notre travail se concentre sur l'*Infrastructure as a Service* (IaaS) et le *Hardware as a Service* (HaaS), car ils fournissent tous les services de base nécessaires pour les applications de *workflows*, y compris les machines virtuelles, les processeurs, le stockage et l'accès aux services réseau. En outre, l'IaaS et le HaaS sont des technologies riches en fonctionnalités et matures actuellement.

Néanmoins, les services PaaS et SaaS peuvent être aussi utilisés pour les *workflows*. Le PaaS peut exposer un environnement de développement et d'exécution de haut niveau permettant de construire et de déployer les *workflows* sur l'infrastructure *cloud*. Les fournisseurs SaaS offrent aux utilisateurs finaux des solutions logicielles standardisées qui pourraient être intégrées dans leurs *workflows* existants.

Les avantages du *cloud computing* pour les *workflows* seront discutés dans la section 1.2.5.

1.2.2.2. Modèles de déploiement

Selon la définition du *cloud computing* donnée part le NIST (Mell, 2011), il existe quatre modèles de déploiement des services de *cloud*, à savoir : *cloud* privé, *cloud* communautaire, *cloud* public et *cloud* hybride, comme illustré dans la figure 1.5.

- ***cloud* privé.** L'ensemble des ressources d'un *cloud* privé est exclusivement mis à disposition d'une entreprise ou organisation unique. Le *cloud* privé peut être géré par l'entreprise elle-même (*cloud* privé interne) ou par une tierce partie (*cloud* privé externe). Les ressources d'un *cloud* privé se trouvent généralement dans les locaux de l'entreprise ou bien chez un fournisseur de services. Dans ce dernier cas, l'infrastructure est entièrement dédiée à l'entreprise et y est accessible via un réseau sécurisé (de type VPN). L'utilisation d'un *cloud* privé permet de

garantir, par exemple, que les ressources matérielles allouées ne seront jamais partagées par deux clients différents.

- ***cloud* communautaires.** L'infrastructure d'un *cloud* communautaire est partagée par plusieurs organisations indépendantes ayant des intérêts communs. L'infrastructure peut être gérée par les organisations membres ou par un tiers. L'infrastructure peut être située, soit au sein des dites organisations, soit chez un fournisseur de services.
- ***cloud* public.** L'infrastructure d'un *cloud* public est accessible à un large public et appartient à un fournisseur de services. Ce dernier facture les utilisateurs selon la consommation et garantit la disponibilité des services via des contrats SLA.
- ***cloud* hybride.** L'infrastructure d'un *cloud* hybride est une composition de plusieurs *cloud*s (privé, communautaire ou public). Les différents *cloud*s composant l'infrastructure restent des entités uniques, mais sont reliés par une technologie standard ou propriétaire permettant ainsi la portabilité des données ou des applications déployées sur les différents *cloud*s.

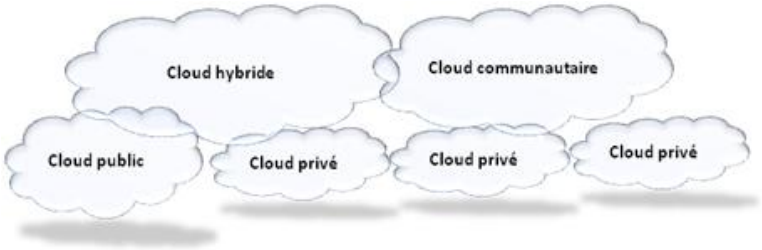


Figure 1.5. Modèles de déploiement du *cloud computing*

1.2.2.3. Acteurs du *cloud computing*

Selon l'architecture de référence du *cloud computing* décrite par le NIST (Fang, 2011), on distingue cinq acteurs principaux comme le montre la figure 1.6.

- *Consommateur (cloud Consumer) :* Une personne ou une organisation qui utilise un service fourni par un fournisseur. Dans cette thèse, nous utilisons aussi le terme utilisateur pour désigner un consommateur.
- *Fournisseur (cloud Provider) :* Une personne, une organisation ou une entité chargée de rendre un service à la disposition des parties intéressées.
- *Courtier (cloud broker) :* Une entité qui gère l'usage, la performance et l'approvisionnement de services, et qui négocie les relations entre les fournisseurs et les consommateurs.
- *Auditor (cloud auditor) :* Une entité qui peut procéder à une évaluation indépendante des services de *cloud* telle que la performance et la sécurité du *cloud*.
- *Carrier (cloud carrier) :* Un intermédiaire qui offre la connectivité et le transport des services des fournisseurs aux consommateurs.

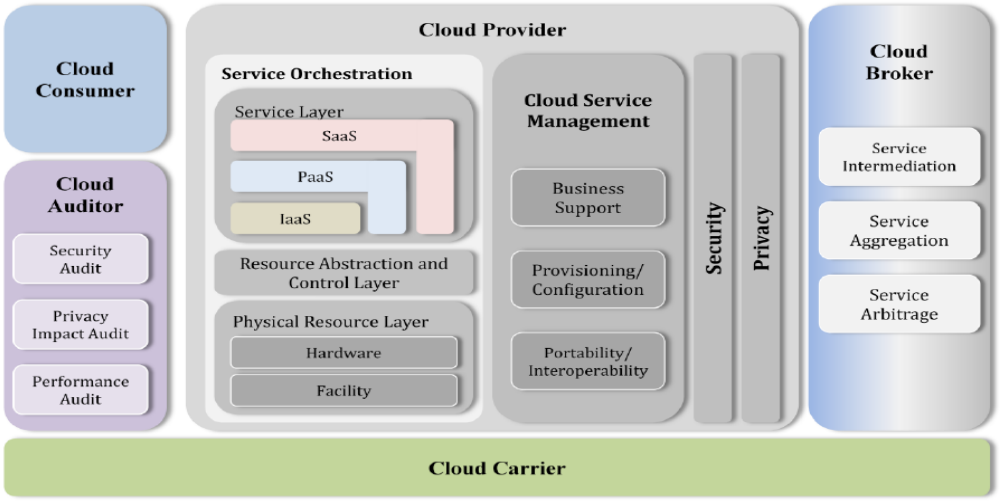


Figure 1.6. Modèle de référence conceptuel du *cloud* NIST 2011(Fang, 2011)

1.2.3. Challenges de recherche en environnement de *cloud computing*

Même si certaines des caractéristiques essentielles du *cloud computing* ont été réalisées par des efforts commerciaux et universitaires, de nombreux problèmes existants n'ont pas été pleinement pris en compte, et d'autres nouveaux défis continuent d'émerger (Zhang, 2010). Dans cette section, nous résumons quelques enjeux de recherche dans le *cloud computing.*

- **Migration des données entre les environnements non standards**

La plupart des fournisseurs de services de *cloud computing* utilisent des applications *cloud* propriétaires. Ces applications ne sont pas interopérables, ce qui rend très difficile pour les utilisateurs de faire passer leurs données à un autre fournisseur de *cloud* ou de revenir à leur machines.

- **Sécurité de données et confidentialité**

Dans le *cloud computing*, les données doivent êtres transférées entre les dispositifs de l'utilisateur et les *Datacenter* des fournisseurs de services de *cloud computing*, 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 computing* 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*. Les principaux avantages de la migration de VM est d'éviter les points chauds (*hot spots*); Toutefois, cela n'est pas simple à réaliser. Actuellement, la détection de points chauds et l'initiation d'une migration manque de souplesse pour répondre aux changements brusques de charge de travail (Zhang, 2010).

- **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 computing.* 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. Le problème de la consolidation optimale des serveurs est un problème d'optimisation NP-complet. Cependant, les activités de consolidation de serveurs ne devraient pas affecter les performances de l'application. Il est connu que l'utilisation des ressources de machines virtuelles individuelles peut varier au cours du temps (Wood, 2007). Pour les ressources du serveur qui sont partagées entre les machines virtuelles, comme la bande passante, la mémoire cache et les E/S disques, consolider au maximum un serveur peut entraîner la congestion des ressources quand une machine virtuelle change son utilisation de ressource sur le serveur. Par conséquent, il est parfois important d'observer les fluctuations des traces de VM et d'utiliser cette information pour une consolidation efficace de serveurs. Enfin, le système doit réagir rapidement aux congestions de ressources, quand elles se produisent (Wood, 2007).

- **Gestion de l'énergie**

Améliorer l'efficacité énergétique est un autre enjeu majeur dans le *cloud computing.* Il a été estimé que le coût de l'alimentation et du refroidissement compte pour 53% des dépenses de fonctionnement total des *Datacenter* (Hamilton, 2009). En 2006, les *Datacenter* aux États-Unis ont consommé plus de 1,5% de l'énergie totale produite dans l'année, et le pourcentage devrait croître de 18% par an (Li, 2009). Ainsi, les fournisseurs d'infrastructure doivent prendre des mesures pour réduire la consommation d'énergie. L'objectif est non seulement de réduire le coût de l'énergie dans les *Datacenter*, mais aussi pour répondre aux réglementations gouvernementales et aux normes environnementales.

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 (Brooks, 2000), est devenue très pertinente. L'ordonnancement des tâches intégrant une gestion de l'énergie (Vasic, 2009) et la consolidation des serveurs (Srikantaiah, 2009) sont deux autres façons de réduire la consommation d'énergie. Un défi majeur dans toutes les méthodes ci-dessus est de trouver un bon compromis entre l'économie d'énergie et les performances des applications.

- **Ordonnancement**

L'ordonnancement est un enjeu important qui influence considérablement les performances de l'environnement de *cloud computing.* Il existe plusieurs niveaux d'ordonnancement dans le *cloud*, notamment : l'ordonnancement au niveau application et l'ordonnancement au niveau infrastructure. Le premier consiste en l'ordonnancement (affectation) des tâches composant les applications des utilisateurs sur les services IaaS ou HaaS du *cloud*, et le deuxième niveau concerne l'affectation de machines virtuelles sur les infrastructures physiques (machines physiques) du *cloud*. Les deux niveaux d'ordonnancement sont des problèmes complexes.

Dans cette thèse, notre intérêt porte sur le premier niveau d'ordonnancement, où les applications de l'utilisateur sont souvent exprimées sous forme de *workflows*.

1.3. *Workflow* et systèmes de gestion de *workflow*s

1.3.1. Concepts de base et définitions de *workflows*

La notion de *workflow* (traduit en français par "flux de travail") est apparue dans l'industrie de l'image électronique et de la gestion de production assistée par ordinateur (GW, 1998). Ce concept a donc été créé dans le but d'automatiser les procédures de travail au sein des organisations. L'idée d'enchaîner différentes tâches pour réaliser un traitement complexe est pertinente. De plus, dans les infrastructures actuelles distribuées, gérant des ressources hétérogènes, telles que le *cloud computing*, bénéficier d'un environnement autorisant la définition et l'exécution des chaînes de traitement constitue une des fonctionnalités essentielles recherchée, à la fois par les scientifiques et au-delà par le grand public.

Deux grandes catégories d'usages utilisent la notion de *workflow* : les protocoles expérimentaux, dans des domaines tels que la biologie, l'astronomie, la physique, la neuroscience, la chimie, etc. (*workflow*s scientifiques) et les chaînes de traitement pratiquées dans des domaines commerciaux, financiers, pharmaceutiques (processus métiers). Elles donnent lieu à plusieurs pistes de recherche diverses, mais cependant connexes. Dans le cadre de cette thèse, nous traitons plus particulièrement les *workflows* scientifiques.

Définitions de *workflow*

La WfMC (*Workflow Management Coalition*) (WfMC, 1999) a donné une définition qui généralise la notion de *workflow* indépendamment des domaines spécifiques:

"*Workflow is the automation of business process, in whole or part during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.*"

Nous traduisons cette définition par: " Un *workflow* est l'automatisation d'un processus métier, en tout ou en partie, au cours de laquelle des documents, des informations ou des tâches sont passées d'un participant à un autre pour l'action, selon un ensemble de règles procédurales".

Cependant diverses définitions ont été proposées, selon les catégories d'usages: les définitions (Altintas, 2004 ; Best, 2007 ; Gil 2007a ; Adam, 2008 ; Lin, 2009 ; Ludäscher, 2009) concernent les *workflows* scientifiques et (Davenport, 1993 ; Weske, 2007) visent les *business process*.

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*" (Ludäscher, 2009). Pour les grands *workflows*, il est souhaitable de répartir les tâches entre plusieurs ressources, afin d'optimiser les temps d'exécution. En tant que tel, les *workflows* impliquent souvent des calculs répartis sur des *clusters*, des grilles, et d'autres infrastructures informatiques. Récemment, les *cloud*s *computing* sont évalués comme une plateforme d'exécution de *workflows* (Hoffa, 2008; Juve, 2008). L'exécution d'un *workflow* est gérée par un SGWf (Système de Gestion de *Workflow*) dont l'architecture de référence est décrite brièvement dans la section suivante.

1.3.2. Architecture des systèmes de gestion de *workflow*s

La gestion du *workflow* est une technologie en évolution rapide, qui est de plus en plus exploitée par les entreprises. 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 Workflow Management Coalition (WfMC, 95) en1995 est présentée dans la figure 1.7.

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

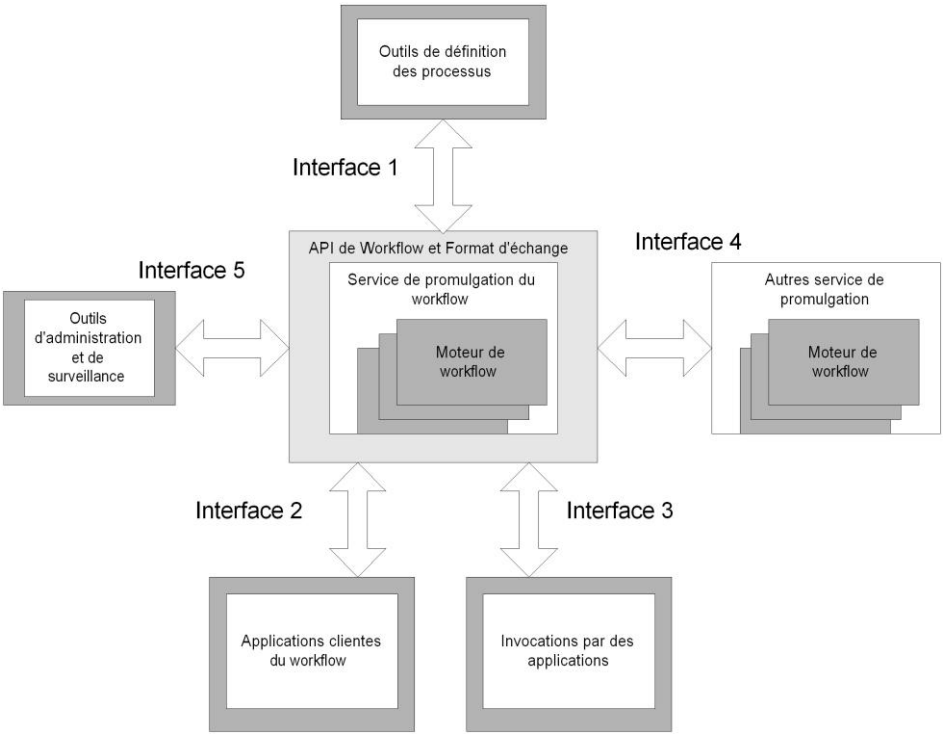


Figure 1.7. Modèle de référence des systèmes de gestion de *workflow* (WfMC, 95).

- *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*.
- *Interface3 :* 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*.

En raison de l'importance des applications de *workflows*, plusieurs projets de recherche ont été menés pour concevoir des systèmes de gestion de *workflow*. Ces systèmes permettent de définir, gérer et exécuter des *workflows* sur des ressources informatiques (Yu, 2005). L'objectif de la gestion de *workflow* est de s'assurer que des activités appropriées sont exécutées par les bons services, au bon moment (Cybok, 2008).

1.3.3. Systèmes de gestion de *workflow*s pour les grilles et *cloud*s

Actuellement, il existe plusieurs systèmes de gestion de *workflow* pour les grilles. Ces systèmes incluent GrADS (Berman, 2001), Pegasus (Deelman, 2003), GridFlow (Coa, 2003), Kepler (Altintas, 2004), Gridbus (Buyya, 2004), GridAnt (Laszewski, 2004), Taverna (Oinn, 2004), Askalon (Fahringer, 2005) et Triana (Churches, 2005). Pour une description détaillée de ces systèmes, nous renvoyons le lecteur à (Yu, 2005). En raison de l'aspect relativement nouveau du

cloud computing, on trouve très peu de publications à ce jour, sur les systèmes de gestion de *workflow* dans les *cloud*s. De nombreux chercheurs travaillent dans ce domaine pour le moment. Pandey et al. (Pandey, 2008) ont utilisé Aneka (Buyya, 2008) comme un middleware *cloud* sur lequel le système de gestion de *workflow* Gridbus (Yu, 2004) déploie et gère l'exécution des tâches. Hoffa et al. (Hoffa, 2008) ont exploré la différence entre l'exécution des *workflows* scientifiques sur les *cloud*s et sur les grilles, en utilisant *Montage* (Hoffa, 2008) au dessus du logiciel Pegasus-WMS (Deelman, 2005). Yang et al. (Yang, 2008) ont proposé un système de *workflow cloud* SwinDeW-C (Swinburne Decentralised *Workflow* for *cloud*). Ils utilisent la technologie de *cloud computing* pour faciliter l'exécution des *workflows*. Le *toolkit cloudbus* (Buyya, 2009) est une initiative vers la fourniture de solutions viables pour l'utilisation des infrastructures de *cloud computing*. Les auteurs proposent une vision plus large, qui intègre une architecture inter-*cloud* et un modèle informatique utilitaire orienté vers le marché. Le moteur de *workflow cloudbus* représente une étape vers la mise à l'échelle des applications de *workflow* sur des *cloud*s en utilisant l'informatique orientée vers le marché.

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

- **Ordonnancement de *workflow*s**

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.

Dans cette thèse, nous nous intéressons à l'ordonnancement de *workflows* dans les environnements distribués informatiques, tels que les *grids* et plus particulièrement le *cloud computing*. 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 multiobjectif.

1.3.4. Intérêts du *cloud* pour les *workflows*

Les *cloud*s offrent plusieurs avantages pour les applications à base de *workflow*s. Ces avantages facilitent:

- **L'approvisionnement de ressources**

Dans les grilles, l'ordonnancement est basé sur un modèle en *best-effort*, dans lequel l'utilisateur spécifie la quantité de temps nécessaire et délègue la responsabilité de l'allocation des ressources et d'ordonnancement de tâches à un ordonnanceur fonctionnant en mode *batch* utilisant des files d'attentes. 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**

Contrairement aux grilles, les *cloud*s 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* (Singh, 2005 ; Juve, 2008)

- **L'élasticité**

Outre l'approvisionnement des ressources à la demande, les *cloud*s permettent aussi aux utilisateurs de libérer des ressources à la demande. La nature élastique de *cloud*s 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 (i) de meilleures garanties de QoS aux utilisateurs, et (ii) 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**

Économiquement motivés, les fournisseurs de *cloud* commercial s'efforcent d'offrir de meilleures garanties de services par rapport aux fournisseurs de grille. 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. Cependant, l'effet à long terme de l'utilisation de ressources virtuelles dans les *clouds* qui partagent efficacement une "tranche" de la machine physique, plutôt que d'utiliser des ressources dédiées pour les *workflows* de calculs intensifs, est une question de recherche intéressante.

1.4. Conclusion

Dans ce chapitre, nous avons présenté une vue d'ensemble du *cloud computing*, du *workflow* et des intérêts du *cloud* pour les *workflows*. Nous avons exploré divers défis de recherche dans le *cloud*. Un de ces défis porte sur l'ordonnancement de *workflows*, qui présente un élément essentiel des systèmes de gestion de *workflows*. En général, l'ordonnancement de *workflows* est de nature multiobjectif. C'est un problème NP-complet qui nécessite d'utiliser des métaheuristiques pour sa résolution.

Dans le chapitre suivant, nous présentons les principes de base de l'optimisation multiobjectif suivie d'un état de l'art des différents algorithmes d'ordonnancement de workflows dans le cloud.

Chapitre 2

Optimisation multiobjectif et ordonnancement de *workflows* dans le *cloud computing* : état de l'art

2.1. Introduction

L'optimisation multiobjectif est une approche très importante du fait de la nature multiobjectif de la plupart des problèmes réels, tels que le problème d'ordonnancement de *workflow* dans le *cloud*. Les premiers travaux menés sur les problèmes multiobjectifs furent réalisés au 19ème siècle, sur des études en économie par Edgeworth (Edgeworth, 1881) et généralisés par Pareto (Pareto, 1896).

Ce chapitre est organisé en deux parties. Dans la première partie, nous donnons un état de l'art de l'optimisation multiobjectif. Nous présentons d'abord un ensemble de définitions et de concepts de base de l'optimisation multiobjectif dans la section 2.2.1. Puis nous présentons l'optimisation multiobjectif et l'aide à la décision, dans la section 2.2.2. La section 2.2.3 traite les méthodologies de résolution. Enfin, dans la section 2.2.4, nous présentons les concepts communs aux métaheuristiques, suivis de quelques méthodes regroupées en deux catégories : les métaheuristiques à base de solution unique, et les méthodes à base de population de solutions. Dans la deuxième partie de ce chapitre, nous donnons un panorama de méthodes d'ordonnancement de *workflow*s dans le *cloud computing.*

2.2. Optimisation multiobjectif : Etat de l'art

2.2.1. Concepts et définitions

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

2.2.1.1. Formulation générale d'un problème d'optimisation multiobjectif

Définition 2.1 Un problème d'optimisation combinatoire multiobjectif, *MOP* (M*ulti-Objective combinatorial Optimization Problem*) peut être défini par :

$$(MOP) = \begin{cases} Min\ F(x) = (f_1(x), \dots, f_n(x)) \\ s.c.\ \ x \in X \end{cases} \quad (2.1)$$

où *n* est le nombre d'objectifs ($n \geq 2$), *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 2.1 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 monoobjectif, la solution d'un problème multiobjectif n'est pas unique, mais c'est un ensemble de solutions non dominées, connu comme l'ensemble des solutions Pareto optimales.

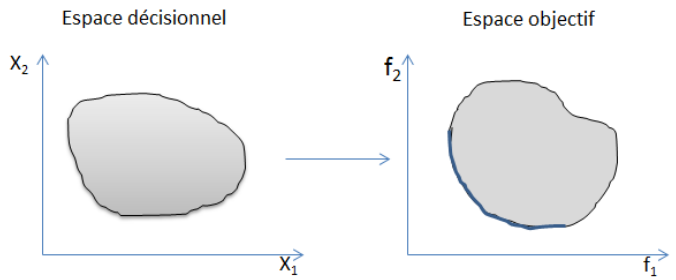


Figure 2.1. Espace décisionnel et espace objectif d'un *MOP* (cas de deux variables de décision X_1 et X_2 et de deux fonctions objectif f_1 et f_2).

2.2.1.2. Notions de dominance

Dans le cadre de l'optimisation multiobjectif, 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 monoobjectif 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.

Définition 2.2 (Dominance de Pareto). Un vecteur objectif $z \in Z$ domine un vecteur objectif $z' \in Z$ si les deux conditions suivantes sont vérifiées :

$$\forall i \in \{1, \ldots, n\}, z_i \leq z'_i \quad (2.2)$$

$$\exists j \in \{1, \ldots, n\}, z_j < z'_j \quad (2.3)$$

Si z domine z′, cette relation sera notée $z \prec z'$. Par extension, une solution $x \in X$ domine une solution $x' \in X$, notée $x \prec x'$, ssi $f(x) < f(x')$.

La figure 2.2 illustre la définition de la notion de dominance. Il existe des relations de dominance autres que celle de Pareto. Elles sont énoncées ci-dessous.

Définition 2.3 (Vecteur non-dominé) Un vecteur objectif $z \in Z$ *est non-dominé* ssi $\forall z' \in Z$, $z' \not\prec z$ (voir figure 2.2).

Définition 2.4 (Dominance faible) Un vecteur objectif $z \in Z$ domine *faiblement* un vecteur objectif $z' \in Z$ ssi $\forall i \in \{1, \ldots, n\}$, $z_i \leq z'_i$. Cette relation sera notée $z \preccurlyeq z'$.

Définition 2.5 (Dominance stricte) Un vecteur objectif $z \in Z$ domine *strictement* un vecteur objectif $z' \in Z$ ssi $\forall i \in \{1, \ldots, n\}$, $z_i < z'_i$. Cette relation sera notée $z \prec z'$.

Définition 2.6 (ε-dominance) Un vecteur objectif $z \in Z$ *ε-domine* un vecteur objectif $z' \in Z$ avec $\varepsilon > 1$, ssi $\forall i \in \{1, \ldots, n\}$, $z_i \leq \varepsilon . z'_i$. Cette relation sera notée $z \preccurlyeq_\varepsilon z'$.

2.2.1.3. 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 (Edgeworth, 1881) et généralisé par V. Pareto (Pareto, 1896).

Définition 2.7 (solution Pareto optimale) Une solution $x \in X$ est Pareto optimale (ou non-dominée) ssi $\forall x' \in X$, $x' \not\prec x$.

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.

Définition 2.8 (Ensemble Pareto optimal) Etant donné un MOP (f, X), l'ensemble Pareto optimal est défini comme suit: $X_E = \{x \in X | \nexists\, x' \in X,\ x' \prec x\}$.

Définition 2.9 (Front Pareto) Etant donné un MOP (f, X) et son ensemble Pareto optimal X_E, le front Pareto est défini comme suit: $Z_N = \{f(x) | \ x \in X_E\}$, (voir figure 2.2).

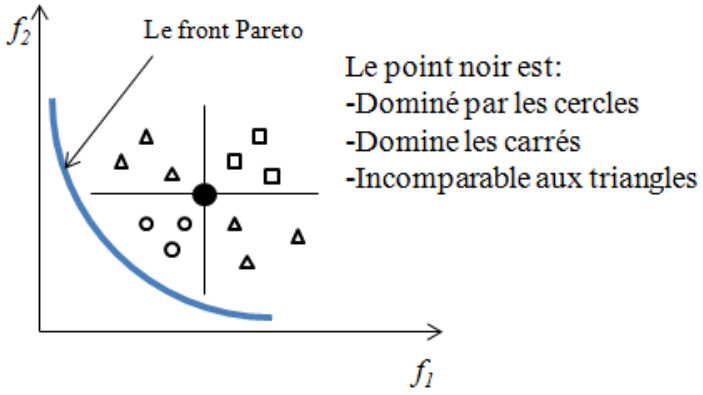


Figure 2.2. Relation de dominance (cas de deux objectifs à minimiser)

2.2.2. Optimisation multiobjectif et aide à la décision

La résolution d'un problème multiobjectif 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 multiobjectif porte sur la façon de combiner la recherche et les processus décisionnels. Ceci peut être fait selon l'une des trois méthodes suivantes (Collette, 2002) :

- ***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 multiobjectif du départ en un problème monoobjectif. 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.

- **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.
- **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 computing*, nous nous plaçons dans le cas des approches *a priori* et *a posteriori*. Dans ce dernier type d'approche, deux phases sont distinguées : la phase d'optimisation et la phase d'aide à la décision. Seule la première phase sera traitée dans notre travail, et sera considérée comme la résolution de notre problème.

2.2.3. Méthodologies de résolution

Les méthodes de résolution d'un problème d'optimisation sont nombreuses et variées. La figure 2.3 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.

Les méthodes exactes 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, comme le cas du problème d'ordonnancement de *workflows* dans le *cloud*. Parmi ces

méthodes, on peut citer : le *Branch & Bound* (Ulungu, 1995), l'algorithme A* (Stewart, 1991) et la programmation dynamique (Carraway, 1990).

Au contraire, les méthodes approchées produisent des solutions de haute qualité 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.

Pour des raisons pratiques, nous avons tendance à utiliser les méthodes approchées pour le problème d'ordonnancement de *workflows* dans le *cloud*. Nous nous contentons de trouver des solutions sous-optimales qui sont assez bonnes pour équilibrer la qualité et le temps de calcul.

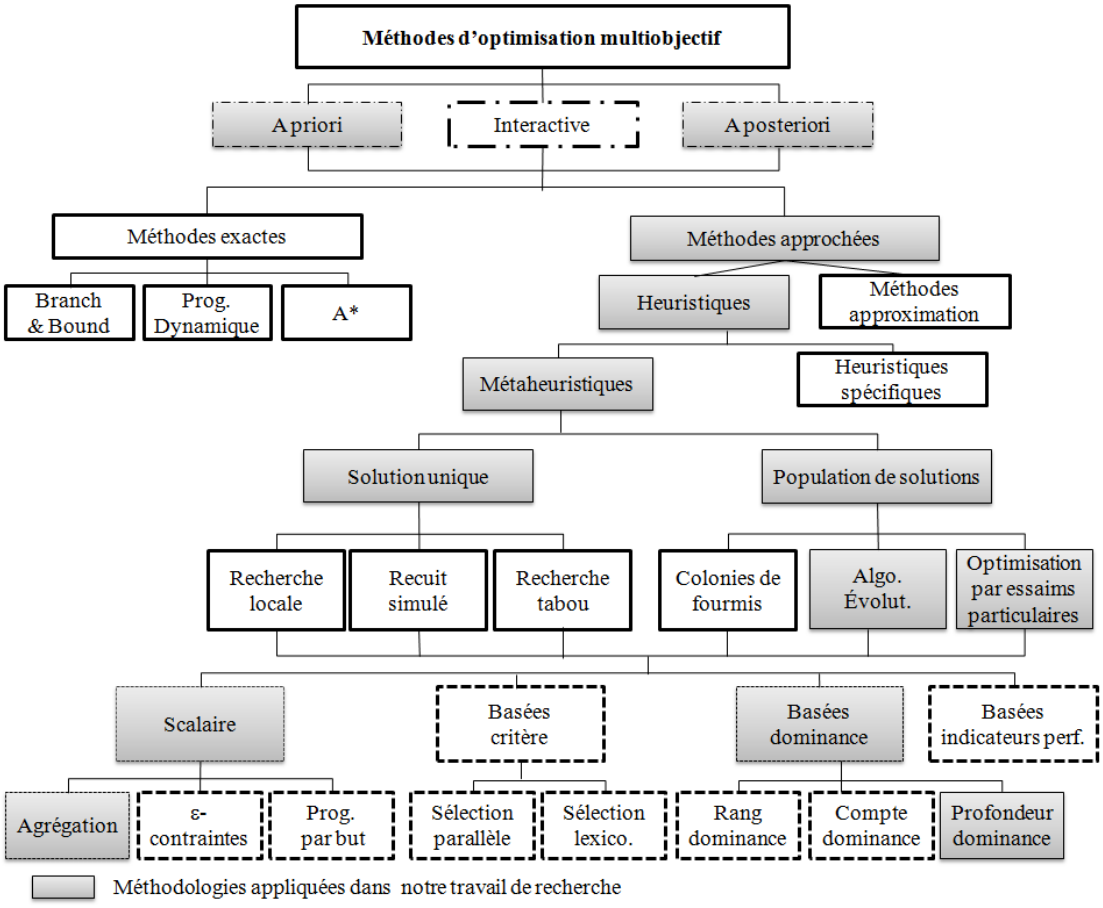


Figure 2.3. Classification des méthodes d'optimisation multiobjectif.

Les méthodes approchées sont réparties en deux sous-catégories: les heuristiques et les méthodes d'approximation (cf. Figure 2.3). Les heuristiques sont divisées en deux sous-classes: les heuristiques dédiées à un problème donné (Talbi, 1999) et les métaheuristiques

(Siarry, 2003), qui sont des méthodes génériques non dédiées à un problème spécifique. Dans le cadre de cette thèse, notre intérêt porte sur les métaheuristiques pour la résolution du problème d'ordonnancement de *workflows*.

2.2.4. Les métaheuristiques

Les métaheuristiques ont reçu une popularité importante durant les vingt dernières années. Leur utilisation dans de nombreuses applications montre leur efficacité pour résoudre des problèmes complexes et de grande taille (Talbi, 2009).

Plusieurs classifications des métaheuristiques ont été proposées, la plupart distinguent globalement deux familles: les métaheuristiques à base de solution unique (S-métaheuristiques) et celles à base de population de solutions (P-métaheuristiques), comme illustré dans la figure 2.4 (Talbi, 2009).

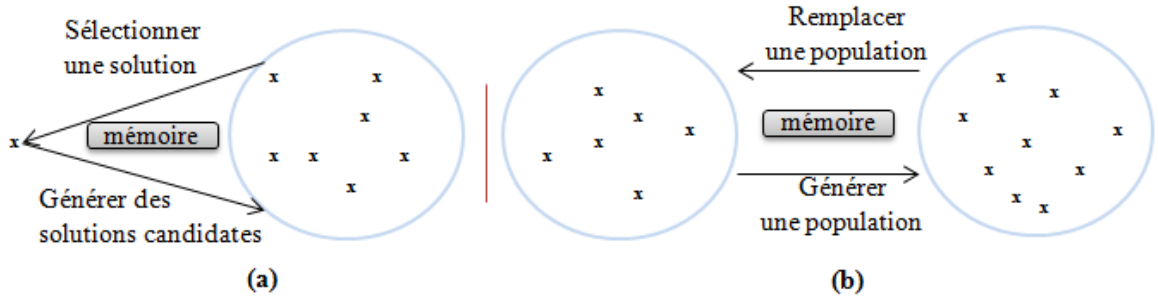


Figure 2.4. Principes généraux d'une métaheuristique à base de: (a) solution unique, (b) population.

Les méthodes de la première famille (telles que les algorithmes de recherche locale (Papadimitriou, 1982), de recherche tabou (Glover, 1986), de recuit simulé (Cerny, 1985), etc.) consistent à manipuler et améliorer une seule solution, tant que cela est possible. Par contre, dans les métaheuristiques à base de population (telles que les algorithmes évolutionnaires, algorithmes à essaim de particules, etc.), un ensemble de solutions, nommé *population*, évolue en parallèle. Ces deux familles ont des caractéristiques complémentaires : les S-métaheuristiques 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-métaheuristiques sont orientées exploration, elles permettent une meilleure diversification de l'espace de recherche. La figure 2.4 montre les principes généraux d'une S-métaheuristique et d'une P-métaheuristique (Talbi, 2009). Dans les chapitres

suivants de ce document, nous comptons utiliser les P-métaheuristiques pour résoudre le problème d'ordonnancement de *workflows*.

2.2.4.1. Concepts communs à tout type de métaheuristique

Pour la conception de toute métaheuristique, deux concepts fondamentaux doivent être considérés : la représentation des solutions manipulées par l'algorithme, et l'évaluation de ces solutions dans l'espace objectif.

- **Représentation de solution**

La conception de tout type de métaheuristique nécessite une représentation (codage) d'une solution. La représentation joue un rôle majeur dans l'efficacité de toute métaheuristique 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, discrètes, binaires, ou des permutations. La figure 2.5 illustre un exemple de chaque représentation. Le vecteur de valeurs discrètes est couramment utilisé pour modéliser l'ordonnancement de *workflows* dans le *cloud computing.*

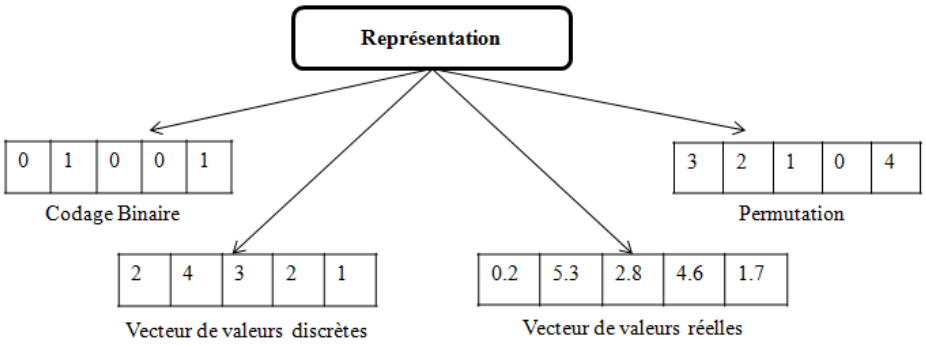


Figure 2.5. Principaux codages pour des problèmes d'optimisation.

- **Evaluation**

L'étape d'évaluation correspond au calcul et l'affectation de valeurs objectifs pour une solution donnée. Dans le cas de l'optimisation monoobjectif, une fonction objectif 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 multiobjectif, 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 métaheuristique. 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.

2.2.4.2. Concepts communs à tout type de métaheuristique pour le cas multiobjectif

La résolution d'un MOP exige des contraintes supplémentaires, qui n'apparaissent pas dans l'optimisation monoobjectif. Toute métaheuristique dédiée à la résolution d'un problème multiobjectif doit tenir compte de trois concepts, à savoir : l'affectation de la qualité (« *fitness* ») d'une solution, la préservation de la diversité et l'élitisme.

- **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 2.3) :

- **Approches scalaires.** Ces approches consistent à transformer le MOP initial en un problème monoobjectif ou à un ensemble de problèmes monoobjectif, dont il existe de nombreuses méthodes de résolution (Agrafiotis, 2001 ; Collette, 2002b). Parmi ces méthodes, nous pouvons citer la méthode d'agrégation (Sriniva, 1994), 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 (Miettinen, 1999). Ces approches requièrent cependant une bonne connaissance du problème à résoudre pour être mises en place.

- ***Approches basées sur un critère.*** *C*es approches ne transforment pas le MOP en un problème monoobjectif. Elles utilisent des opérateurs de recherche qui traitent séparément les différents objectifs. Deux groupes de méthodes existent dans la littérature : sélection lexicographique et sélection parallèle.

  Dans l'approche de sélection lexicographique (Fourman, 1985), 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 métaheuristiques ont été utilisées pour résoudre des MOP en utilisant la sélection lexicographique (Talbi, 1999).

  Dans l'approche par sélection parallèle, le premier travail a été publié par Schaffer (Schaffer, 1985). Il est basé sur un algorithme génétique. L'algorithme développé VEGA (*Vector Evaluated Genetic Algorithm*) sélectionne les solutions courantes du front Pareto suivant chaque objectif, indépendamment des autres (sélection parallèle). Nous invitons le lecteur à consulter (Collette, 2002; Berro, 2001) pour plus de détails.
- ***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 (Goldberg, 1989) dans les algorithmes génétiques.
  Trois stratégies principales peuvent être distinguées (voir figure 2.3) (Zitzler, 2004):
  (1) Rang de dominance (*dominance-rank*). Cette technique consiste à calculer le nombre de solutions qui dominent la solution considérée (figure 2.6).
  (2) Compte de dominance (*dominance-count*). La valeur de *fitness* d'une solution correspond au nombre de solutions dominées par la solution considérée (figure 2.7).
  (3) Profondeur de dominance (*dominance-depth*). Cette stratégie consiste à classer les solutions de la population en différents fronts (Goldberg, 1989). 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 2.8.
- ***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 multiobjectifs basés sur des indicateurs, nous trouvons l'algorithme IBEA (Zitzler, 2004b) (*Indicator Based Evolutionary Algorithm*).

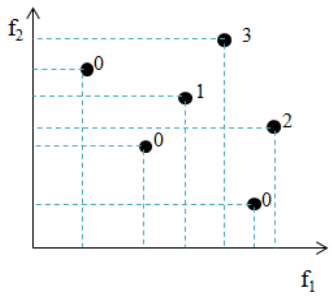


Figure 2.6. Rang dominance

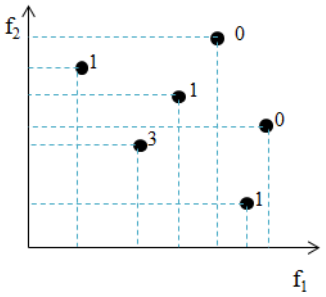


Figure 2.6. Compte dominance

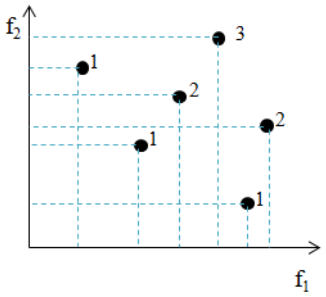


Figure. 2.8. Profondeur dominance

- **La préservation de la diversité**

L'application de métaheuristiques pour la résolution d'un problème d'optimisation multiobjectif 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 multiobjectif, 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 2.9-(a) montre une approximation de l'ensemble Pareto optimal ayant une bonne convergence et une bonne diversité. Dans la figure 2.9-(b), les solutions sont très bien réparties mais sont loin du front de Pareto. Enfin la figure 2.9-(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.

- **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.

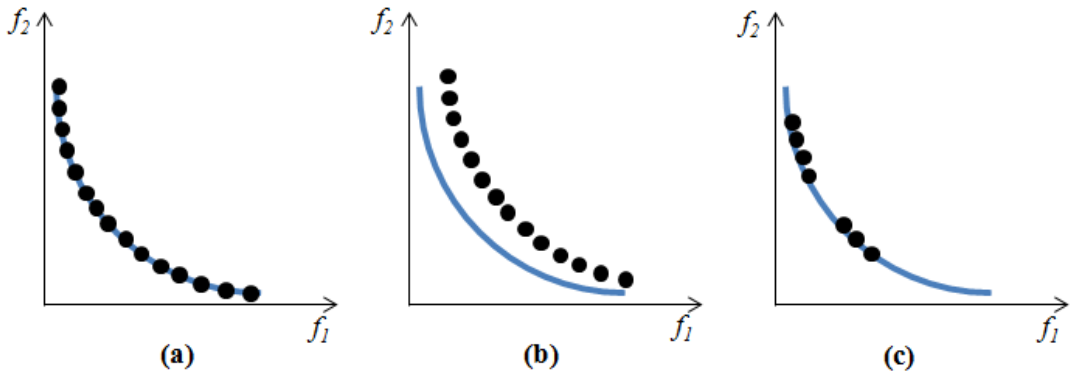


Figure 2.7. Illustration de la convergence et de la diversité en optimisation multiobjectif

2.2.4.3. Métaheuristiques à base de solution unique

Les S-métaheuristiques sont des méthodes itératives couramment utilisées dans le domaine de l'optimisation. Elles travaillent sur un seul point de l'espace de recherche à un instant donné, en commençant avec une solution initiale, puis elles s'efforcent de l'améliorer itérativement en choisissant une nouvelle solution dans son *voisinage*. En général, les opérateurs de recherche locale s'arrêtent quand une solution localement optimale est trouvée.

- **La recherche par descente**

La méthode de descente est une des métaheuristiques à base de solution unique, la plus simple et la plus utilisée. Elle est facile à mettre en place et donne de bons résultats (Papadimitriou, 1976). La figure 2.10 schématise son principe général, qui est aussi détaillé par l'algorithme 2.1. Partant d'une solution initiale donnée, à chaque itération, l'algorithme remplace la solution courante par une solution voisine (solution générée à partir de la

précédente en appliquant une modification locale) qui améliore la fonction objectif. La descente s'arrête lorsqu'il n'y a plus de solution voisine permettant d'améliorer la solution courante, ainsi un optimum local est atteint.

Il existe plusieurs stratégies pour la sélection d'une meilleure solution voisine : choisir la première solution voisine améliorante (*first improvement*), choisir la meilleure parmi toutes les solutions voisines (*best improvement*), ou encore en choisir une aléatoirement, parmi celles qui améliorent la solution courante (*random selection*).

Algorithme 2.1. Méthode de la descente

Sol = S_0 // *Génération d'une solution initiale*
Tant que ∃ solution voisine (sol) **faire**
- Génération des solutions voisines candidates V(Sol) // S′ *dans le voisinage*

S'il n'y a pas de meilleure solution voisine **alors**
Stop
Fin si
Sol= S′ //*Sélection d'une meilleure solution voisine S′ ∈ V(Sol)*
Fin tant que
Retour Solution finale trouvée (optimum local)

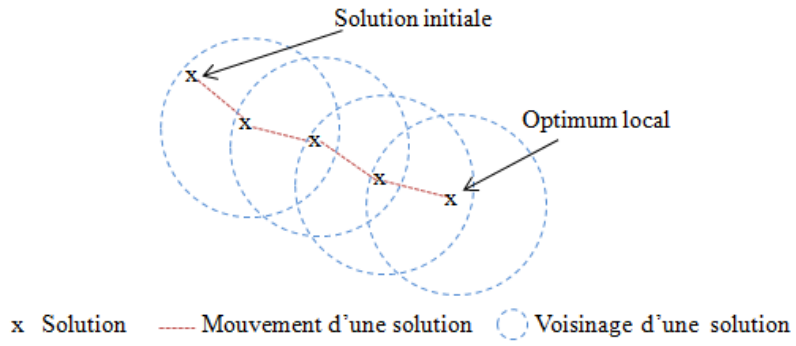


Figure 2.8. Processus d'une recherche par descente

Une des principales limites de la recherche locale par descente est qu'elle converge vers un optimum local. Pour éviter de rester bloqué sur un optimum local, des mécanismes supplémentaires sont proposés, menant à la création de nouveaux algorithmes (recuit simulé, recherche tabou), ou à l'intégration de la recherche locale dans un processus itératif (ILS). Plus de détails sur ces méthodes et d'autres types de S-métaheuristiques sont présentés dans (Talbi, 2009).

2.2.4.4. Métaheuristiques à base de population de solutions

Contrairement aux S-métaheuristiques, les P-métaheuristiques 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 multiobjectif. Parmi ces métaheuristiques, on distingue les algorithmes évolutionnaires, basés sur la théorie de l'évolution de Darwin (Darwin, 1859), 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. Plus de détails sur cette classe de métaheuristiques peuvent être trouvés dans (Talbi, 2009).
Dans ce manuscrit, nous considérons principalement les algorithmes évolutionnaires, plus précisément les algorithmes génétiques et l'optimisation par essaim particulaire (PSO, *Particle Swarm Optimization*) pour résoudre le problème d'ordonnancement de *workflows* dans le *cloud*.

2.2.4.4.1. 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**. Ces derniers ont été mis au point par John Holland dans les années 60 (Holland, 1962) puis raffinés par De Jong (DeJong, 1975), et popularisés par Goldberg (Goldberg, 1989).

- **Principes des algorithmes génétiques (GA)**

Le mécanisme des algorithmes génétiques consiste à faire évoluer une population de solutions de manière itérative vers l'optimum du problème, en appliquant des opérateurs génétiques. Leur principe s'inspire du principe d'évolution de Darwin (Darwin, 1959). Ce type d'algorithme permet d'imiter l'évolution d'une population dans laquelle l'information va pouvoir être partagée par les individus, pour en créer de nouveaux. Des modifications de l'information peuvent parfois avoir lieu de manière spontanée au sein de la population. Et enfin, le principe de sélection naturelle tend à garder les individus les mieux adaptés à leur

environnement. Les algorithmes génétiques reprennent ces principes à travers les opérateurs de croisement, de mutation et de sélection. Le principe général des algorithmes génétiques est illustré par l'organigramme de la figure 2.11.

L'algorithme est initialisé par une population d'individus choisie généralement de manière aléatoire, où chaque individu représente une version codée d'une solution potentielle. Cependant, d'autres mécanismes d'initialisation peuvent être utilisés suivant l'application (Bhanu, 1995). L'algorithme va procéder à l'évaluation des individus de la population, puis à la création de nouveaux individus. Pour ce faire, un certain nombre d'individus de la population sont sélectionnés selon une stratégie de sélection donnée, on les appelle les "parents". L'opérateur de croisement va permettre, à partir de ces parents, d'obtenir des individus dont l'information génétique consistera en un mélange de l'information présente dans les deux parents : les "enfants". Un opérateur de mutation sera éventuellement appliqué à ces enfants pour les diversifier. Finalement, aux termes de l'application des opérateurs de sélection, croisement et mutation, une stratégie de remplacement décide ensuite des solutions, parmi la population courante et les enfants, qui constitueront la nouvelle population. Cet enchaînement d'étapes représente une génération. L'algorithme s'arrête lorsqu'il atteint un critère d'arrêt donné. Différents opérateurs de sélection, croisement, mutation, remplacement, peuvent être utilisés. Nous allons détailler ceux que nous utilisons dans le cadre de la problématique d'ordonnancement de *workflows* dans le *cloud computing* au chapitre 3.

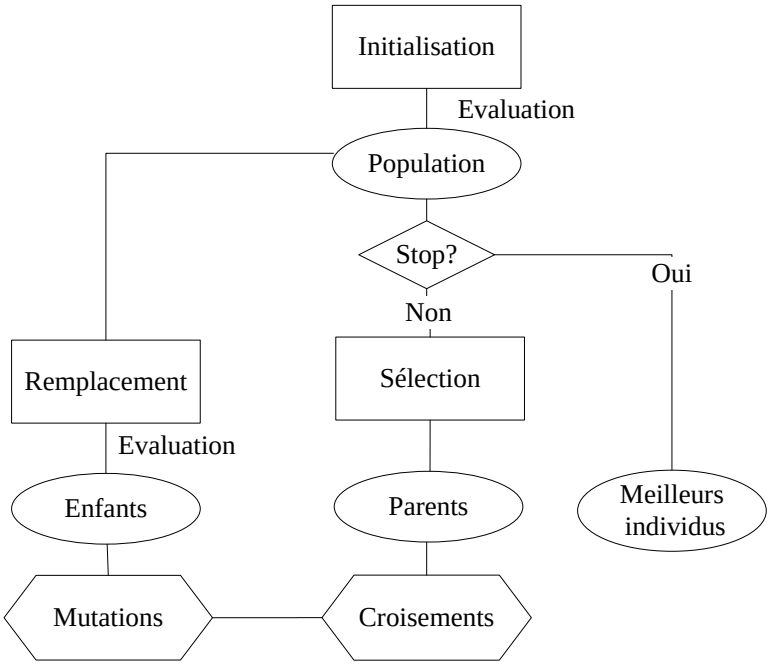


Figure 2.9. Principe général des algorithmes génétiques

- **Les algorithmes génétiques en optimisation multiobjectif**

Les algorithmes génétiques ont été très utilisés pour résoudre les problèmes multiobjectif, avec toutes les approches d'optimisation multiobjectif (Talbi, 1999). Une étude comparative de quelques algorithmes est disponible dans (Zitzler, 1999).

- ***Les GA pour l'approche scalaire***

Plusieurs GA ont été proposés pour l'approche transformant un problème multiobjectif en un problème monoobjectif, plus de détails sur ces algorithmes se trouvent dans (Talbi, 1999). (Collette, 2002) présente plusieurs GA pour résoudre les approches ε-contraintes.

- ***Les GA pour l'approche basée sur un critère***

L'algorithme génétique le plus répandu dans cette approche est l'algorithme VEGA proposé par Schaffer (Schaffer, 1985). La particularité de ce GA est de créer des sous-populations dont les individus sont dédiés à un objectif en particulier.

- ***Les GA pour l'approche Pareto***

Dans ce type d'approche, deux familles de méthodes se distinguent : les méthodes non élitistes et les méthodes élitistes.

Parmi les méthodes non élitistes, nous pouvons citer la méthode MOGA (*Multi-Objective Genetic Algorithm***)** proposée par Fonseca et al. (Fonseca, 1983), où chaque individu est rangé suivant la notion de *rang de dominance*, la méthode NSGA (*Non Dominated Sorting Genetic Algorithm*) proposée par Sriniva et al (Sriniva, 1994), où le calcul de la *fitness* s'effectue en séparant la population en plusieurs groupes, en utilisant la notion de la *profondeur de dominance*.

Les techniques élitistes ne sauvegardent pas les individus Pareto optimaux trouvés au cours des itérations (Elles se distinguent de deux manières : la difficulté de maintenir la diversité et leur lenteur de convergence vers la frontière Pareto).

Parmi les GA adoptant une stratégie élitiste, nous citons l'algorithme SPEA (*Strength Pareto Evolutionary Algorithm***),** proposé par Zitzler et al. (Zitzler, 1999), où le passage d'une génération à l'autre commence par la mise à jour des différentes sauvegardes. Tous les individus non dominés sont sauvegardés et les individus dominés, déjà présents, sont éliminés. Cet algorithme utilise la notion de *compte de dominance* pour calculer la *fitness* d'un individu. Une évolution de cet algorithme est disponible dans (Zitzler, 2002). Cette évolution a une plus grande complexité, mais présente de meilleures performances.

Enfin l'algorithme le plus employé et qui fait référence dans le domaine est NSGA-II, une évolution de NSGA (Deb, 2001; Deb, 2002). Dans cette version, l'auteur tente de résoudre toutes les difficultés de NSGA, en termes de complexité, non élitisme et utilisation du partage. Cet algorithme sera détaillé ultérieurement dans la section 3.5.1 de notre contribution au chapitre 3.

Pour avoir plus de détails et une vue complète des différentes méthodes, nous invitons le lecteur à consulter (Berro, 2001; Collette, 2002b).

2.2.4.4.2. Optimisation par Essaim Particulaire

- **Principe général**

L'Optimisation par Essaim Particulaire (OEP), ou *Particle Swarm Optimization* (PSO) en anglais, est une méthode d'optimisation stochastique basée sur une population de solutions. Elle a été développée par Kennedy et Eberhart en 1995 (Kennedy, 1985). Elle est inspirée du comportement social des colonies d'insectes, des nuées d'oiseaux, des bancs de poissons et bien d'autres sociétés animales évoluant en essaim. En effet, on peut observer chez ces animaux des dynamiques de déplacement relativement complexes, alors qu'individuellement chaque individu a une « intelligence » limitée, et ne dispose que d'une connaissance locale de sa situation dans l'essaim. Des règles simples, telles que « *aller à la même vitesse que les autres* », « *se déplacer dans la même direction* », ou encore « *rester proche de ses voisins* », suffisent pour maintenir la cohésion de l'essaim, permettant ainsi de mettre on œuvre des comportements collectifs complexes et adaptés. L'intelligence globale de l'essaim est la conséquence directe des interactions locales entre les différentes particules de l'essaim.

L'essaim de particules correspond à une population d'agents simples, appelés *particules*. Les particules représentent des solutions potentielles au problème d'optimisation. Elles survolent l'espace de recherche, afin de trouver l'optimum global. Chaque particule possède une *position* (le vecteur solution) et une *vitesse*. De plus, elle possède une mémoire lui permettant de se souvenir de sa meilleure performance (en position et en valeur) et de la meilleure performance atteinte par les particules « *voisines* » (informatrices).

Le déplacement d'une particule est influencé par une combinaison linéaire des trois composantes suivantes :

- Une composante *d'inertie* : la particule tend à suivre sa direction courante de déplacement;

- Une composante *cognitive* : la particule tend à se diriger vers le meilleur site par lequel elle est déjà passée ;
- Une composante *sociale* : la particule tend à se fier à l'expérience de ses congénères et, ainsi, à se diriger vers le meilleur site déjà atteint par ses voisins.

Dans le cas d'un problème d'optimisation, la qualité d'un site de l'espace de recherche est déterminée par la valeur de la *fonction objectif* en ce point. La stratégie de déplacement d'une particule est illustrée dans la figure 2.12.

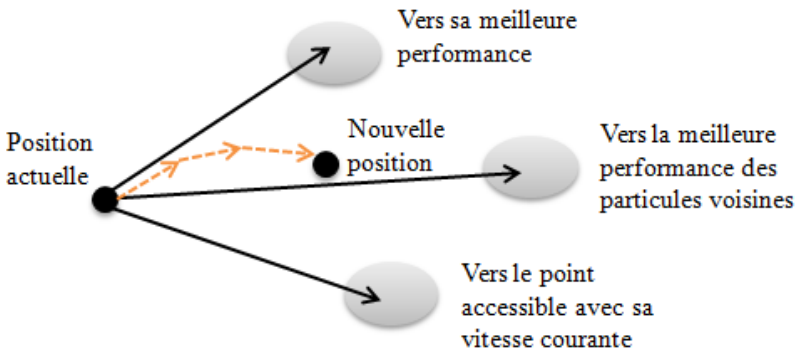


Figure 2.10. Stratégie de déplacement d'une particule

Dans un espace de recherche de dimension *D*, la particule *i* de l'essaim est modélisée par son vecteur position $X_i = (x_{i1}\,;\ x_{i2}; \ldots; x_{iD})$ et par son vecteur vitesse $V_i = (v_{i1}\,;\ v_{i2}; \ldots; v_{iD})$. Cette particule garde en mémoire la meilleure position par laquelle elle est déjà passée, que l'on note $Pbest_i = (Pbest_{i1}\,;\ Pbest_{i2}; \ldots; Pbest_{iD})$. La meilleure position atteinte par toutes les particules de l'essaim est $Gbest = (Gbest_1\,;\ Gbest_2; \ldots; Gbest_D)$.

A l'itération *k+1*, le vecteur vitesse et le vecteur position sont mis à jour à partir de l'équation (2.4) et de l'équation (2.5), respectivement.

$$V_i^{k+1} = \omega\, V_i^k + c_1 r_1 \left(Pbest_i^k - X_i^k \right) + c_2 r_2 \left(gGbest^k - X_i^k \right) \tag{2.4}$$

$$X_i^{k+1} = X_i^k + V_i^{k+1} \tag{2.5}$$

où ω est une constante, nommée coefficient d'inertie, $c_1$et c_2 sont deux constantes, appelées coefficients d'accélération, r_1 et r_2 sont deux nombres aléatoires tirés uniformément dans [0,1] à chaque itération et pour chaque dimension.

$\boldsymbol{\omega V_i^k}$: correspond à la composante d'inertie du déplacement, où le paramètre ω contrôle l'influence de la direction de déplacement sur le déplacement futur. Il est à noter que, dans certaines applications, le paramètre ω peut être variable ;

- $\boldsymbol{c_1 r_1}\left(\boldsymbol{Pbest_i^k - X_i^k}\right)$: correspond à la composante cognitive du déplacement, où le paramètre c_2 contrôle le comportement cognitif de la particule ;
- $\boldsymbol{c_2 r_2}\left(\boldsymbol{gGbest^k - X_i^k}\right)$: correspond à la composante sociale du déplacement, où le paramètre c_2 contrôle l'aptitude sociale de la particule.

La combinaison des paramètres ω, c_1 et c_2 permet de régler l'équilibre entre les phases de diversification et d'intensification du processus de recherche (Kennedy, 2001).

L'optimisation par essaim particulaire est un algorithme à population. Il commence par une initialisation aléatoire de l'essaim dans l'espace de recherche. A chaque itération de l'algorithme, chaque particule est déplacée suivant les équations 2.4 et 2.5. Une fois le déplacement des particules effectué, les nouvelles positions sont évaluées.

Les $Pbest_i$ ainsi que $Gbest$ sont alors mis à jour. Les différentes étapes du PSO sont présentées dans l'algorithme 2.2.

Concernant le critère d'arrêt, il est commun de fixer un nombre maximum d'évaluations de la fonction objectif ou un nombre maximum d'itérations. Cependant, selon le problème posé et les exigences de l'utilisateur, d'autres critères d'arrêt peuvent être utilisés.

Algorithme 2.2. Algorithme d'optimisation par essaim particulaire

---

Initialiser aléatoirement les particules : position et vitesse.
Evaluer les positions des particules
Pour chaque particule *i*, $Pbest_i = X_i$
Calculer *Gbest*
Tant que le critère d'arrêt n'est pas satisfait **faire**
Pour chaque particule **faire**
Déplacer les particules selon les équations 2.4 et 2.5
Evaluer les positions des particules
Mettre à jour $Pbest_i$
Fin pour
Mettre à jour *Gbest*
Fin

Le lecteur intéressé trouvera un état de l'art complet, entièrement dédié à l'optimisation par essaim particulaire et aux concepts qui lui sont associés, dans (Clerc, 2005).

- **L'Optimisation par Essaim Particulaire en optimisation multiobjectif**

La résolution d'un problème multiobjectif consiste à trouver un ensemble de solutions Pareto optimales. Vue la structure de la population de PSO, il est souhaitable de générer plusieurs solutions non-dominées en une seule itération. Ainsi, afin d'adapter le PSO à l'optimisation multiobjectif, il est nécessaire de répondre aux trois questions suivantes (Coello, 2002) :

- Comment sélectionner les particules afin de donner plus de préférence aux solutions non-dominées ?
- Comment maintenir les solutions non-dominées par rapport à celles déjà vues et rencontrées au cours du processus de la recherche ? Aussi, il est souhaitable que ces solutions soient bien réparties sur le front de Pareto.
- Comment maintenir la diversité dans l'essaim, afin d'éviter la convergence prématurée vers une seule solution ?

La première application du PSO en multiobjectif a été proposée par Moore et al. (Moore, 1999). Dans les paragraphes suivants, on dresse une liste non exhaustive des travaux existants en OEP multiobjectif.

- ***L'OEP pour l'approche scalaire***

Parsopoulos et Vrahatis (Parsopoulos, 2002) proposent d'agréger les objectifs suivant trois méthodes différentes : une agrégation linéaire classique, pour laquelle les poids sont fixés, une agrégation dynamique, pour laquelle les poids sont modifiés au cours du traitement et une agrégation dite « *bang-bang* » (Jin, 2001), pour laquelle les poids sont brutalement modifiés au cours du temps.

Baumgartner et al. (Baumgartner, 2004) utilisent une agrégation linéaire des objectifs. L'essaim est divisé en *n* sous-essaims de taille identique. Chaque sous-essaim utilise un vecteur de paramètres différents et évolue dans sa propre direction. Les solutions optimales au sens de Pareto sont déterminées à l'aide d'une méthode basée sur l'utilisation du gradient.

- ***L'OEP pour l'approche basée sur un critère***

Hu et al. (Hu, 2003) ont proposé un algorithme qui optimise un seul objectif à la fois, en utilisant un ordre lexicographique. Parsopoulos et al. (Parsopoulos, 2004) ont proposé une version parallèle de VEPSO (*Vector Evaluated Particle Swarm*) inspirée de VEGA (Schaffer, 1985). VEPSO est un algorithme multi-essaim, pour lequel chaque sous-essaim traite un des objectifs. Chaque sous-essaim communique avec les autres, à travers l'échange de leurs meilleures positions (Reyes-Sierra, 2006).

Dans (Chow, 2004), un algorithme multi-essaim est aussi proposé. Chaque essaim traite individuellement un des objectifs. L'échange des informations est effectué entre les sous-essaims voisins par l'intermédiaire de l'échange des vecteurs *Gbest*. A l'intérieur d'un sous-essaim, les particules sont complètement connectées. A l'opposé, une topologie locale de voisinage est utilisée pour connecter les différents sous-essaims.

- ***L'OEP pour l'approche Pareto***

Plusieurs algorithmes ont été développés dans cette approche. Ray et al. (Ray, 2002) ont proposé un algorithme utilisant la dominance de Pareto et qui combine l'OEP avec des techniques issues des algorithmes évolutionnaires. Les auteurs utilisent un opérateur de densité sur le voisinage pour promouvoir la diversité au sein de l'essaim.

Coello Coello et al. (Coello, 2002b; Coello, 2004) ont développé un algorithme basé sur l'utilisation d'une archive externe. La mise à jour de l'archive est réalisée suivant un critère géographique. L'espace de recherche est divisé en hyper-cubes auxquels on donne des *notes*, qui sont fonction du nombre de solutions non-dominées situées dans chacun d'eux. Les leaders sont choisis dans les hyper-cubes sélectionnés à l'aide d'un opérateur de sélection par «*roulette*» basé sur leurs notes. Un opérateur de mutation est aussi utilisé.

Dans (Li, 2003), les auteurs adaptent les principaux mécanismes de NSGA-II à l'OEP multiobjectif. Les leaders sont sélectionnés parmi les éléments de l'archive. Ils utilisent deux méthodes de sélection : une stratégie de « *niching* » et une stratégie de densité du voisinage. De même, dans (Raquel, 2005), la distance d'encombrement (*crowding* distance), déjà utilisée dans NSGA-II, sert de critère pour maintenir la diversité dans l'archive. La sélection des leaders se fait aussi à partir des éléments de l'archive.

De nombreuses autres méthodes sont référencées dans (Reyes-Sierra, 2006).

2.3. (Méta) heuristiques d'ordonnancement de workflows dans le cloud computing

Les services de *cloud computing* ne sont pas gratuits; un contrat de niveau de service SLA doit être établi entre les fournisseurs de services *cloud* et leurs clients. Ce contrat fixera le prix à payer pour une qualité de service des ressources à louer (Vaquero, 2008). Une fois le contrat SLA signé, les deux parties doivent respecter leurs engagements, afin d'éviter les pénalités. Du côté fournisseurs, ils doivent respecter leurs engagements en termes de puissance de calcul, de fiabilité de service, de sécurité et de disponibilité des ressources, pour satisfaire leurs clients, tout en s'efforçant de maximiser leurs profits. D'autre part, l'utilisateur cherche à accomplir son *workflow* en un temps minimum et à un prix réduit. La réalisation de ces objectifs exige une répartition efficace des tâches constituant les *workflows* utilisateurs sur le pool de ressources délivré par le fournisseur *cloud*.

Compte tenu de ce contexte et de la complexité croissante des *workflows*, des réseaux et des besoins de sécurité, les algorithmes d'ordonnancement deviennent de plus en plus compliqués, car ils doivent gérer un grand nombre de paramètres contradictoires et qui concernent différents acteurs, dont les intérêts ne sont pas les mêmes. Le problème d'ordonnancement de *workflow* dans ces circonstances est caractérisé par le fait qu'il comporte un groupe d'objectifs contradictoires et complexes. Par exemple, la réduction du prix d'exécution d'un *workflow* va augmenter généralement son temps d'exécution et peut également conduire à l'utilisation de machines qui sont moins fiables. En général, le problème d'ordonnancement de *workflows* est connu pour être NP-complet (Lenstra, 1978; Brucker, 2004). Il est impossible de trouver la solution optimale globale en utilisant des algorithmes simples. Par conséquent, ce problème a un enjeu majeur, qui est la nécessité de trouver une manière efficace pour évaluer tous ces différents objectifs à l'aide de (méta) heuristiques.

Quelques algorithmes ont été appliqués dans les systèmes distribués. Nous en présentons dans la section suivante un aperçu et leurs limitations.

2.3.1. Panorama de méthodes d'ordonnancement de *workflow*s dans le *cloud*

De nombreux chercheurs ont proposé différentes approches basées sur des heuristiques ou des métaheuristiques pour résoudre le problème d'ordonnancement de *workflows* dans les

systèmes distribués. Nous ne faisons pas ici un état de l'art exhaustif, mais nous nous intéressons principalement aux algorithmes proposés récemment et qui sont actuellement répandus dans le *cloud computing.*

Topcuoglu et al. (Topcuoglu, 2002) ont proposé l'algorithme HEFT (*Heterogeneous Earliest Finish Time*) qui détermine un ordonnancement d'un graphe (DAG, *Direct Acyclic Graph*) 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. Cela nous a poussés à détailler et utiliser cet algorithme dans nos contributions. HEFT est composé de deux phases principales: (1) détermination des priorités des tâches et (2) sélection du processeur. La première étape 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. La phase de sélection est effectuée comme suit : 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 (Ge, 2010) 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 (*First In First out*).

Lin et Lu (Lin, 2011) ont proposé l'heuristique SHEFT (*Scalable 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. (El-kenawy, 2012) 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.

Notons que la plupart de travaux cités ci-dessus se sont concentrés uniquement sur l'optimisation d'une seule métrique de qualité de service, qui est le *makespan* du *workflow*. Toutefois, avec l'avènement du *cloud* orienté sur le marché, les fournisseurs de *cloud*s commerciaux offrent, à la demande, des services et des ressources de calculs ayant des performances différentes, délivrées à des prix variés, selon les QoS négociées via le SLA. Les fournisseurs facturent les utilisateurs selon leurs consommations. Par exemple, pour *Amazon EC2*, qui est un fournisseur populaire d'IaaS, les utilisateurs sont facturés pour la consommation des ressources de calculs. Les utilisateurs peuvent décider de payer un peu plus cher pour réduire le temps d'exécution de leurs *workflows*, ou réduire les coûts d'exécution, en autorisant un temps d'exécution plus long, tant que la *deadline* est respectée. Par conséquent, les politiques d'ordonnancement doivent nécessairement considérer les prix des ressources, ainsi que le budget prévisionnel et le délai de l'utilisateur. Plusieurs travaux de recherche traitant ces aspects ont été menés. Nous présentons ci-après une liste non exhaustive des travaux les plus populaires.

Pandey et al. (Pandey, 2010) ont présenté un algorithme basé sur l'optimisation par essaim de particules (PSO) pour l'ordonnancement de *workflow* sur les ressources de *cloud*. Cet algorithme prend à la fois les coûts de calcul et les coûts de transmission de données et tente de minimiser le coût total d'exécution du *workflow*. Les résultats de l'évaluation montrent que le PSO peut réduire le coût total et donne un bon équilibrage de charge sur les ressources.

Salehi et al. (Salehi, 2010) ont proposé une stratégie d'ordonnancement hiérarchique axée sur le marché, qui comporte deux niveaux d'ordonnancement : le niveau service et le niveau tâche. L'ordonnancement au niveau service porte sur l'affectation des tâches aux différents services de *cloud* public, tandis que l'ordonnancement au niveau tâche porte sur l'optimisation de l'affectation des tâches aux différentes machines virtuelles dans les *Datacenter* locaux du *cloud*. Ils ont proposé deux politiques d'ordonnancement : une politique d'optimisation du temps (*Time-Opt*) et une optimisation des coûts (*Cost-Opt*). Leurs politiques d'ordonnancement visent à satisfaire la contrainte de coût ou le délai de l'application, en

étendant la capacité de calcul des ressources locales par la location de ressources externes chez des fournisseurs de *cloud* public.

Liu et al. (Liu, 2010) ont proposé l'algorithme d'ordonnancement des instances intensives de *workflows* contraints par le coût, nommé CTC (*Compromised-Time-Cost*), qui prend le coût et le temps comme facteurs essentiels. L'algorithme propose des stratégies permettant de trouver le compromis entre le temps d'exécution et le coût avec l'intervention de l'utilisateur à la volée. Les résultats de simulation montrent que cet algorithme peut atteindre un coût faible, tout en respectant la *deadline* désignée par l'utilisateur.

Netjinda et al. (Netjinda, 2012) ont utilisé la métaheuristique PSO pour l'optimisation du coût de *workflow*. L'algorithme utilise un schéma de décodage pour convertir les valeurs réelles d'une particule en des valeurs entières distinctes, représentant une solution. Les premiers résultats montrent une performance prometteuse, au point de vue du coût total et de la convergence de la fonction de *fitness*.

Guo et al. (Guo, 2012) ont formulé un modèle pour l'ordonnancement multiobjectif de tâches et ont décrit un algorithme de PSO qui optimise le temps et le coût dans le *cloud*.

Verma et al. (Verma, 2012) ont proposé un algorithme d'ordonnancement de *workflow* pour un environnement *cloud*, nommé DBD-CTO (*Deadline and Budget Distribution based Cost and Time Optimization*). L'algorithme tente de minimiser le coût et le temps d'exécution, tout en respectant la date limite et la contrainte budgétaire spécifiées par l'utilisateur.

Abrishami et al. (Abrishami, 2013) ont proposé deux algorithmes d'ordonnancement de *workflows* pour l'*IaaS cloud*s : IC-PCP (*IaaS cloud Partial Critical Path*) et IC-PCPD2 (*IaaS cloud Partial Critical Path with Deadline Distribution*). Ces algorithmes visent à minimiser le coût d'exécution de *workflow*, tout en respectant un délai. Le modèle de tarification des *cloud*s considéré est basé sur un intervalle de temps. Les résultats de simulations montrent que les deux algorithmes ont une performance prometteuse, avec IC-PCP qui est plus performant qu'IC-PCPD2, dans la plupart des cas considérés.

De manière générale, 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* et/ou du coût, avec ou sans contraintes de délais et/ou de budget. Toutefois, des pannes ou des échecs peuvent se produire sur les serveurs/services

de *cloud* de temps en temps ; ce qui provoque une grande menace pour les applications. Par conséquent, les algorithmes d'ordonnancement doivent tenir compte de ces situations pour maximiser la fiabilité des applications. De plus, ils doivent considérer d'autres métriques, telles que la disponibilité, ou le taux l'utilisation des ressources, ainsi que les aspects de sécurité. Peu de travaux de recherches ont été conduits dans ces directions dans le domaine du *cloud computing* :

Hakem et Butelle (Hakem, 2007) ont présenté une heuristique *biobjectif gloutonne* pour l'ordonnancement d'applications parallèles sur des systèmes informatiques distribués hétérogènes, nommée BSA (*Bi-objective Scheduling Algorithm*). 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. (Wang, 2011) ont mis en œuvre la technique de réputation RD (*Driven Reliability*) pour évaluer la fiabilité des ressources dans les systèmes informatiques largement distribués. Ils ont proposé l'algorithme LAGA (*Look-Ahead Genetic Algorithm*), 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. (Jang, 2012) 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 *Datacenter*s qui sont gourmands en énergie et qui émettent une quantité considérable de dioxyde de carbone. Cependant, les stratégies d'ordonnancement doivent adopter des mesures non seulement pour répondre aux exigences de QoS définies par l'utilisateur via les SLA, mais aussi pour veiller à ce que le profit des fournisseurs ne soit pas considérablement réduit, en raison de la forte consommation énergétique de leurs ressources. Par conséquent, des

travaux plus récents ont porté sur le développement d'algorithmes d'ordonnancement prenant en compte la consommation énergétique.

Il existe plusieurs stratégies de réduction de la consommation d'énergie, telles que la consolidation de charge de travail et la configuration des ressources physiques, via la technique d'adaptation dynamique de fréquence et de tension, DVFS (*Dynamic Voltage and Frequency Scaling*), qui est maintenant une technologie courante dans les processeurs modernes.

La consolidation consiste à placer les tâches de manière à consolider la charge de travail sur un nombre petit mais suffisant de ressources, permettant ainsi de maximiser l'utilisation de ces ressources et de mettre en veille celles qui ne sont pas utilisées (Khan, 2009 ; Lee, 2012). La consolidation est activée récemment par la technologie de virtualisation, qui fournit des mécanismes pour contrôler et adapter les actions de ressources physiques allouées aux instances de machines virtuelles (VMs), et migrer rapidement les instances de VMs entre ces ressources physiques (Eugen, 2011 ; Li, 2011). On parle alors, dans ce cas, de l'ordonnancement au niveau infrastructure ("mapping " de VMs sur les ressources physiques).

La technologie DVFS permet de changer (réduire ou augmenter) la fréquence et la tension du processeur, influençant ainsi la consommation énergétique, tout en ayant un impact sur le temps d'exécution. Plusieurs approches d'ordonnancement basées sur la technique DVFS ont été proposées, que ce soit pour les tâches indépendantes (Yao, 1995 ; Aydin, 2004 et Zhuo, 2005), ou pour les tâches ayant des contraintes de précédence (Zhu, 2003 ; Zhu, 2004 ; Ge, 2005 ; Mezmaz, 2010 ; Lee, 2011).

Wang et al. (Wang, 2010) présentent un algorithme d'ordonnancement tenant compte de l'énergie et proposent une discussion détaillée sur le calcul du *temps slack*. L'algorithme d'ordonnancement présenté permet de réduire les tensions pendant les phases de communication entre les tâches sur les processeurs parallèles homogènes. Lee et al. (Lee, 2009) proposent l'heuristique *ECS* (*Energy Conscious Scheduling*) pour résoudre le problème d'ordonnancement des tâches sur les systèmes informatiques hétérogènes. Cette heuristique prend en compte le *makespan* et la consommation d'énergie et permet de trouver un équilibre entre ces deux objectifs, en utilisant une nouvelle fonction objectif, nommée *RS* (*Relative Superiority*). Mezmaz et al. (Mezmaz, 2011) proposent un algorithme génétique hybride et

parallèle, qui prend en compte le *makespan* et la consommation d'énergie pour résoudre le problème d'ordonnancement d'applications parallèles ayant des relations de précédence sur les systèmes de calcul haute-performance, tels que les infrastructures de *cloud computing.* L'approche est un hybride entre l'algorithme génétique multiobjectif parallèle et l'heuristique *ECS* (Lee, 2009). Elle exploite la technique DVS pour la réduction de l'énergie consommée.

Guzek et al. (Guzek, 2012) examinent l'influence de la DVFS, en utilisant l'algorithme évolutionnaire multiobjectif (NSGA-II) pour l'ordonnancement biobjectif du *makespan* et de la consommation énergétique d'un graphe (DAG, *Direct Acyclic Graph*) de tâches sur une plateforme multiprocesseur hétérogène.

Chang-Tian et Jiong (Chang-Tian, 2012) ont utilisé la technique DVS et ont proposé deux algorithmes basés sur un algorithme génétique, à savoir: *ETU_GA* (*Energy consumption Time Unify Genetic Algorithm*) et *ETDF_GA* (*Energy consumption Time Double Fitness Genetic Algorithm*), 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. (Liu, 2013) ont proposé un algorithme génétique multiobjectif 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 2.1 recense les différents travaux cités précédemment.

Articles	Algorithme	Métaheuristique	*Makespan*	Coût	Fiabilité	Disponibilité	Énergie	Contrainte QoS	Pareto
Topcuoglu, 2002	HEFT	non	oui	non	non	non	non	non	non
Hakem, 2007	BSA	non	oui	non	oui	non	non	non	non
Lee, 2009	ECS	non	oui	non	non	non	oui	non	non
Ge, 2010	GA	oui	oui	non	non	non	non	non	non
Liu, 2010	CTC	non	oui	oui	non	non	non	D.	non
Pandey, 2010	PSO	oui	non	oui	non	non	non	non	non
Salehi, 2010	Time-Opt Cost-Opt	non	oui	oui	non	non	non	D. B.	non
Lin, 2011	SHEFT	non	oui	non	non	non	non	non	non
Mezmaz, 2011	hybrid AG	oui	oui	non	non	non	oui	non	oui
Wang , 2011	RD, LAGA	oui	oui	non	oui	non	non	non	non
Chang-Tian, 2012	ETU_GA, ETDF_GA	oui	oui	non	non	non	oui	non	non
El-kenawy, 2012	Max-Min	non	oui	non	non	non	non	non	non
Geo, 2012	PSO	oui	oui	oui	non	non	non	non	oui
Guzek, 2012	NSGA-II	oui	oui	non	non	non	oui	non	oui
Jang, 2012	GA	oui	oui	oui	non	oui	non	non	non
Netjinda, 2012	PSO	oui	non	oui	non	non	non	non	non
Verma, 2012	DBD-CTO	non	oui	oui	non	non	non	D. B.	non
Abrishami, 2013	IC-PCP; IC-PCPD2	non	non	oui	non	non	non	D.	non
Liu, 2013	MOGA	oui	non	non	non	non	oui	D.	oui
Notre travail	GA	oui	oui	oui	oui	oui	non	non	non
	GA*	oui	oui	oui	oui	oui	non	D., B, F., Di, CF.	non
	NSGA-II	oui	oui	oui	oui	oui	non	D., B, F., Di, CF	oui
	PAES	oui	oui	oui	oui	oui	non	non	oui
	MODPSO	oui	oui	oui	non	non	oui	non	oui

D. Deadline, B. Budget, F. Fiabilité, Di. Disponibilité, CF. Contraintes Fortes.

Tableau 2.1. Classification des algorithmes d'ordonnancement de *workflows* dans le *cloud*.

2.3.2. Discussion

Dans le tableau 2.1, nous constatons que :

1) La plupart de travaux présentés optimisent une seule ou au plus deux métriques de QoS à la fois. De plus, très peu de travaux considèrent la fiabilité et la disponibilité des services, ainsi que l'énergie consommée. Pour répondre à ce manque, nous développons un algorithme génétique (GA), qui prend en compte plusieurs métriques de QoS, à savoir le *makespan*, le coût, la fiabilité et la disponibilité des ressources pour l'ordonnancement de *workflow* sur l'*IaaS cloud*.
2) En outre, la plupart de ces travaux ne tiennent pas compte des contraintes sur les métriques de QoS (*deadline*, *Budget*, et autres) et d'autres contraintes fortes, qui obligent l'exécution de certaines tâches sur certains types de ressources, pour des raisons de sécurité, des exigences logicielles ou d'autres exigences fonctionnelles. Pour faire face à ces problèmes, nous proposons un algorithme génétique basé sur l'infection virale (GA*) pour l'ordonnancement de *workflow* sur l'*IaaS cloud*, prenant en compte ces différentes contraintes.
3) Un autre constat concerne l'approche multiobjectif utilisée. La plupart des travaux présentés sont encore monoobjectif ou basés sur une agrégation des objectifs. La plupart ne prêtent pas attention à la façon dont chacune des métriques de QoS peut affecter les autres. Dans notre travail, nous avons pris en compte cet aspect, en développant des approches d'ordonnancement basées Pareto. Nous proposons donc, à cette fin, les algorithmes suivants:
    - NSGA-II et PAES, pour l'ordonnancement de *workflows* sur l'*IaaS cloud*, qui permettent d'optimiser le coût, le *makespan*, la fiabilité et la disponibilité en même temps.
    - DVFS-MODPSO, pour l'ordonnancement de *workflows* sur le *HaaS cloud*. C'est une méthode basée sur l'optimisation multiobjectif par essaim particulaire combinée avec la technique DVFS, permettant d'optimiser simultanément le *makespan*, le coût et l'énergie consommée par les ressources du *HaaS cloud*.

2.4. Conclusion

Dans ce chapitre, nous avons présenté d'abord le cadre général de l'optimisation multiobjectif, qui nous sera utile pour modéliser et résoudre le problème d'ordonnancement de workflows. Puis, nous avons donné un état de l'art des méthodes d'ordonnancement de workflows dans le cloud. Nous avons mis en évidence les défis de cette thèse en décrivant les principaux manques des différents travaux actuels.

Le chapitre suivant présente nos contributions pour l'ordonnancement de workflow dans l'infrastructure IaaS.

Chapitre 3

Métaheuristiques pour l'ordonnancement de *workflow*s dans l'infrastructure IaaS

3.1. Introduction

En raison de la progression des techniques de virtualisation, les fournisseurs de services *cloud* mettent à la disposition de leurs clients une infrastructure IaaS composée d'un pool de ressources hétérogènes appelées machines virtuelles (VMs) qui peuvent être utilisées pour exécuter des *workflows* utilisateurs. Ces machines virtuelles sont hébergées sur plusieurs *Datacenter* et également liées entre elles via des réseaux privés ou via Internet.

L'ordonnancement de *workflows* est considéré comme un problème d'optimisation multiobjectif dont la complexité augmente avec l'augmentation du nombre de paramètres et de contraintes à considérer. Les algorithmes évolutionnaires sont des candidats parfaits pour résoudre ce type de problème.

Dans ce chapitre, nous adoptons deux approches communes pour la résolution du problème d'ordonnancement multiobjectif de *workflows* dans l'infrastructure IaaS à savoir, l'approche par agrégation des objectifs et l'approche Pareto. Dans la première approche, nous présentons deux métaheuristiques à base de population de solutions: un algorithme génétique (GA) prenant en compte plusieurs métriques de QoS et un algorithme génétique amélioré GA* permettant de considérer les différentes contraintes citées précédemment. Dans l'approche Pareto, nous présentons l'algorithme NSGA-II et l'algorithme PAES. Les raisons qui justifient nos choix de ces algorithmes peuvent être divisées en deux groupes. Tout d'abord, nous voulons considérer les deux approches communes de résolution, à savoir l'approche par agrégation et l'approche Pareto. En outre, nous volons considérer les deux classes de métaheuristiques, à savoir, les P-métaheuristiques (GA, GA*, NSGA-II) et les S-métaheuristiques (PAES).

Ce chapitre est organisé comme suit : dans la section 3.2, nous formalisons le problème d'ordonnancement de *workflows* dans l'*IaaS*, nous modélisons le *workflow*, *l'IaaS cloud*, les métriques de QoS considérées et nous finissons par la modélisation de l'ordonnancement sous forme d'un problème d'optimisation multiobjectif. Les sections 3.3 et 3.4 présentent des métaheuristiques basées sur l'approche d'agrégation des objectifs. Nous présentons d'abord dans la section 3.3, l'algorithme génétique (GA) suivi d'une évaluation expérimentale de l'algorithme (Yassa, 2012 ; Yassa, 2013). Ensuite, dans la section 3.4, nous détaillons l'algorithme génétique amélioré (GA*) et son évaluation expérimentale (sublime, 2012, Yassa, 2013a). La section 3.5 présente des métaheuristiques pour l'ordonnancement de *workflow* basées sur l'approche Pareto. Nous présentons les algorithmes NSGA-II et PAES suivis des expériences réalisées. La section 3.6 compare les différents algorithmes présentés et enfin, la section 3.7 conclut le chapitre.

3.2. Modélisation du problème d'ordonnancement de *workflow*s dans l'infrastructure IaaS

Dans cette section, nous décrivons notre modélisation du système d'une manière formelle. Dans ce contexte, l'objectif de l'ordonnancement consiste à répartir les tâches du *workflow* entre les services fournis par *l'IaaS* de façon à optimiser plusieurs métriques de QoS. Par conséquent, nous présentons d'abord le modèle de *cloud computing*, puis, nous décrivons notre modèle de *workflow* et les paramètres de QoS que nous traitons.

3.2.1. Modèle du *cloud computing*

Une infrastructure *IaaS* 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, $prixU_j$ et une fiabilité (taux d'échec, λ_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. Les poids des arrêtes représentent les vitesses de transferts ou les bandes passantes (en Mbps) des liens entre les VMs.

3.2.2. Modélisation du *workflow*

Nous modélisons une application de *workflow* sous forme d'un graphe orienté acyclique (DAG : Directed Acyclic Graph), noté G (V, E). L'ensemble de nœuds V = {T_1, ..., T_n} représente les tâches du *workflow*, l'ensemble des arcs E désigne les contraintes de précédence entre les tâches. Un arc est sous la forme d_{ij}= (T_i, T_j) ∈ E, où T_i est appelée la tâche mère de T_j, T_j est la tâche fille de T_i, d_{ij} sont les données produites par T_i et consommées par T_j. Nous supposons que la tâche fille ne peut pas être exécutée jusqu'à ce que toutes ses tâches mères soient achevées. Dans un graphe de tâches donné, une tâche sans prédécesseur est appelée une tâche d'entrée, et une tâche sans successeur est appelée une tâche de sortie.

La figure 3.1 illustre un exemple de *workflow* composé de 4 tâches et une IaaS composée de 3 VM.

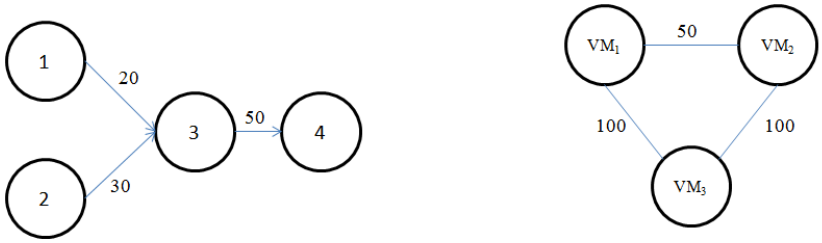


Figure 3.1. Un exemple de *workflow* et des ressources d'une IaaS.

3.2.3. Caractérisation des métriques de QoS

3.2.3.1. *Makespan*

L'une des mesures les plus utilisées dans l'évaluation des algorithmes d'ordonnancement de *workflows* est le temps de complétion ou *makespan*. Le *makespan* est la différence entre la date de soumission du *workflow* et la date de réception des résultats. Autrement dit, c'est la date de fin d'exécution de la dernière tâche du *workflow*. Le *makespan* est donné dans l'équation (3.1).

$$Makespan = \max_{T_i \in V}\{DF\ (T_i)\} \qquad (3.1)$$

où, $DF\ (T_i)$ est la Date de Fin d'exécution de la tâche T_i.

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.

3.2.3.2. 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 C_{total} d'exécution d'un *workflow* dans l'IaaS est défini dans l'équation 3.2. Il se compose du coût d'exécution des différentes tâches C_{ex} et du coût de transferts des données C_{tr}.

$$\text{Coût} = C_{ex} + C_{tr} \tag{3.2}$$

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

$$C_{ex} = \sum_{i=1}^{n} DF(T_i) * PRIXU_j \tag{3.3}$$

Le coût de transfert de données (C_{tr}) est décrit comme suit:

$$C_{tr} = \sum_{i=1}^{n} \sum_{j=1}^{n} D_{ij} * TRC_{ij} \tag{3.4}$$

où,

- D_{ij} : caractérise la quantité de données à transférer de la tâche T_i à la tâche T_j.
- TRC_{ij} : représente le coût de communication entre la machine où T_i est mappée et une autre machine où T_j est affectée. Le coût de communication est ajouté au coût total lorsque deux tâches ont des dépendances de données entre elles, (c'est à dire $D_{ij} > 0$). Pour deux ou plusieurs tâches s'exécutant sur la même machine, le coût de transfert est négligé.

En général, le coût d'exécution et le coût de transmission sont inversement proportionnels, respectivement, au temps de calcul et au temps de transfert, comme indiqué dans la figure 3.2.

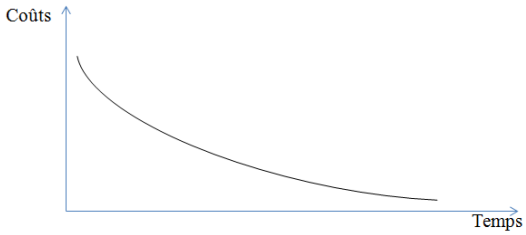


Figure 3.2. Compromis entre les temps et coût d'execution des tâches

3.2.3.3. Fiabilité

La fiabilité représente la probabilité que le *workflow* soit entièrement exécuté avec succès, sans aucune faute de ressources. Notre modèle de fiabilité est inspiré de celui décrit dans (Wang, 2011). Il est basé sur un taux d'échec λ qui est une propriété intrinsèque de la ressource. La fiabilité est calculée dans l'équation 3.5.

$$\text{Fiabilité} = \exp^{-\sum_{i=1}^{n} DF(T_i) * \lambda_j} \quad (3.5)$$

où, λ_j est le taux d'échec de la machine qui exécute la tâche T_i.

3.2.3.4. Disponibilité

La disponibilité peut être utilisée pour déterminer dans quelle mesure l'ensemble de machines virtuelles louées sont utilisées. Elle est calculée dans la formule 3.6.

$$\text{Disponibilité} = \frac{1}{m} * \left(1 - \sum_{j=1}^{m} \frac{TimeStart_j - TimeEnd_j}{MakeSpan}\right) \quad (3.6)$$

où,

- *m* est le nombre total de ressources au niveau de l'infrastructure IaaS ;
- $TimeStart_j$ est la date où VM_j est lancée et $TimeEnd_j$ est la date où cette instance de ressource est arrêtée.

3.2.4. Modèle d'ordonnancement

Étant donné: (1) un fournisseur d'une infrastructure IaaS offrant un ensemble de machines virtuelles, (2) un *workflow* d'un utilisateur composé d'un ensemble *T* de *n* tâches qui doivent être exécutées sur ces VM. Le problème d'ordonnancement du *workflow* sur l'IaaS revient à construire un mapping *M* des tâches aux VMs (sans violer les contraintes de précédence), qui minimise les métriques de QoS suivantes: le *makespan*, le coût, la fiabilité et la disponibilité.

Ces QoS sont souvent conflictuelles, par conséquent, le problème d'ordonnancement de *workflow* peut être formulé comme le problème d'optimisation multiobjectif suivant:

$$Minimiser\ Makespan\ (M)$$
$$Minimiser\ Coût\ \ (M)$$
$$Maximiser\ Fiabilité\ (M)$$
$$Minimiser\ Disponibilité\ (M)$$

3.3. Un algorithme génétique pour l'ordonnancement de *workflow* sur l'IaaS

Étant donné que l'ordonnancement de *workflows* est un problème NP-complet, il devient difficile de le résoudre en utilisant des heuristiques classiques. L'utilisation d'algorithmes génétiques (GA) est généralement efficace pour résoudre un tel problème.

Afin d'adapter l'algorithme génétique à notre problématique, nous devons définir (1) une structure adéquate représentant une solution; (2) un modèle de fonction de *fitness* qui permet de simplifier le problème autant que possible tout en étant suffisamment générique pour être compatible avec un grand nombre de métriques de QoS; et (3) des opérateurs génétiques adéquats pour l'évolution des solutions.

3.3.1. Etapes de l'algorithme GA

Les différentes étapes de l'algorithme GA, que nous proposons sont illustrées dans l'organigramme de la figure 3.3 et seront détaillées dans les sous-sections suivantes.

L'algorithme commence par l'initialisation de la génération initiale de manière aléatoire. Après, le processus de planification (*Scheduling Process*) est effectué sur chaque individu afin de satisfaire les relations de précédence entre les différentes tâches. Chaque individu sera évalué en utilisant une fonction *fitness* (*Fitness computation*) qui sera calculée en fonction des préférences de QoS des utilisateurs. Le processus d'évolution de la population est assuré par différents opérateurs génétiques, notamment, la sélection, le croisement, la mutation et le remplacement. L'algorithme itère le processus jusqu'à atteindre un nombre maximum d'itérations ou jusqu'à ce que la valeur globale de la *fitness* ne s'améliore plus.

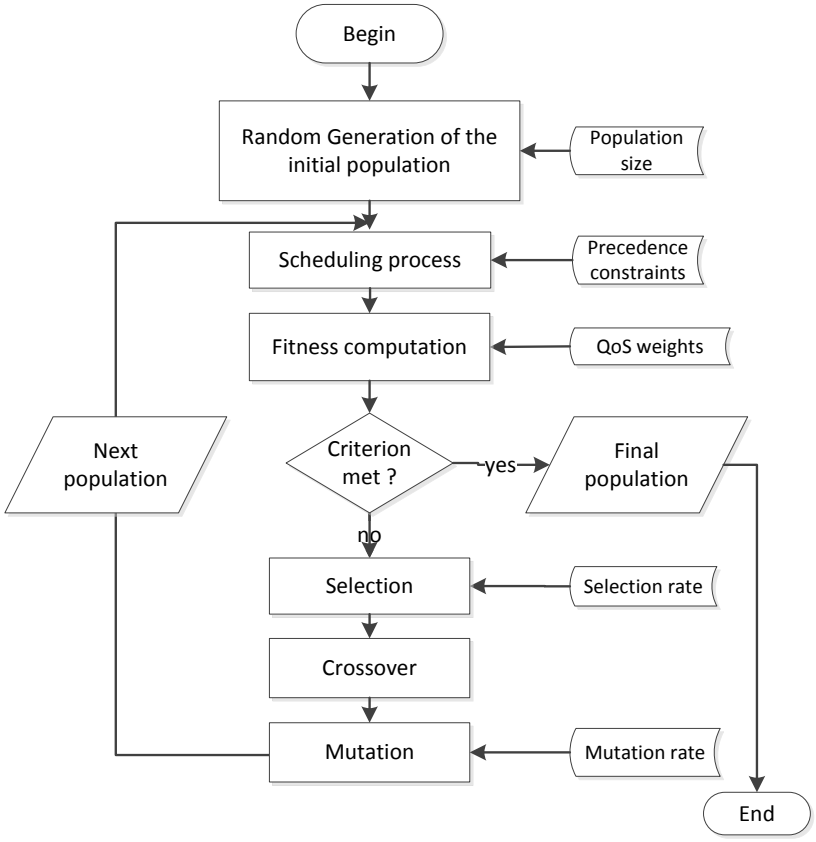


Figure 3.3. Schéma général de l'algorithme GA

3.3.1.1. Représentation d'une solution

Pour le problème d'ordonnancement de *workflows*, une solution valable doit respecter les conditions suivantes:

- Une tâche ne peut démarrer que lorsque tous ses prédécesseurs ont été exécutés et ont transmis leurs données.
- Une machine virtuelle peut exécuter une seule tâche à la fois mais peut en gérer plusieurs dans le temps, par conséquent les tâches doivent être planifiées sur une tranche de temps disponible.
- Chaque tâche est exécutée une seule fois sur une seule machine.

Notre codage d'une solution est le suivant: chaque tâche T_i du *workflow* a une valeur de rang et est affectée à une machine virtuelle VM_j de l'*IaaS cloud*. Un exemple de solution est représenté dans la figure 3.4. La taille du tableau correspond au nombre de tâches qui composent le *workflow*.

Tâche	T_1	T_2	T_3	T_4	T_5	…	T_n
Rang Tâche	55	41	32	27	78	…	1
Identifiant VM	VM_3	VM_2	VM_1	VM_1	VM_2	…	VM_3

Figure 3.4. Représentation d'une solution pour l'ordonnancement

La valeur du rang dans cette représentation est indépendante de la relation de précédence entres les tâches. Elle permet d'indiquer l'ordre d'exécution des tâches dans le cas où il existe deux ou plusieurs tâches qui pourraient être lancées en même temps sur une même VM. Plus de détails à ce sujet seront donnés dans le paragraphe relatif au *Processus de Scheduling*.

Une représentation simplifiée d'une solution est donnée dans la figure 3.5. Ceci se traduit littéralement par: *la tâche T_1 a un Rang 55 et est affectée à la machine virtuelle 3. La tâche T_2 a un rang 41 et est affectée à la machine 2, etc.*

T_1	T_2	T_3	T_4	T_5		T_n
(3, 55)	(2, 41)	(3, 32)	(2, 27)	(3, 20)	…	(2, 1)

Figure 3.5. Représentation simplifiée d'une solution

3.3.1.2. Processus d'ordonnancement (*Scheduling process*)

Le processus de génération et d'évaluation d'une solution est composé de deux étapes. La première étape consiste en la génération d'une solution à l'aide de l'algorithme génétique: l'algorithme GA attribue à chaque tâche du *workflow* un rang et l'affecte aléatoirement à une VM. La deuxième étape consiste à interpréter et évaluer la solution générée afin de calculer sa fonction de *fitness*. Pour ce faire, un processus de *Scheduling* déploie la solution générée ou modifiée. Il ordonnance les tâches dans le temps et calcule les différentes métriques de QoS.

Dans notre algorithme, le Processus de *Scheduling* est similaire à celui utilisé dans l'algorithme HEFT (Topcuoglu, 2002). L'algorithme HEFT ordonnance les tâches sur les différentes ressources dans un ordre donné basé sur une valeur d'un attribut rang (similaire à celui que nous avons dans notre modèle de solution (figure 3.4)). Les rangs dans HEFT sont calculés en fonction des temps de calcul et de communication estimés des tâches.

Le principe du processus de *Scheduling* est le suivant (Algorithme 3.1): Pour toute solution S générée par l'algorithme GA, chercher toutes les tâches qui sont prêtes à être

exécutées (tâches dont les prédécesseurs ont terminés leurs exécutions), puis choisir parmi toutes les tâches prêtes trouvées, celle ayant le rang le plus élevé afin de l'affecter à la VM associée. Le processus est répété de manière itérative jusqu'à ce que toutes les tâches aient été ordonnancées. Une fois le processus terminé, les métriques de QoS de l'ordonnancement sont calculées.

Algorithme 3.1. Pseudo-code du processus de *Scheduling*

```
Soit S une solution générée par GA
 i=0
Tantque ( i< |S|) faire
Trouver les tâches prêtes dans S
   Ready = S.GetTasksReady() ;   // tâches dont les parents ont été ordonnancés
Trouver la tâche ayant le meilleur rang dans Ready.
    P = ready.GetBestRankedPair () ;
 Schedule (P.Task, P.VM) ;
 i=i+1
Fin Tantque
Retourner le plan d'ordonnancement
Calculer les métriques de QoS
ComputeQoS () ;
```

La figure 3.6, illustre un exemple du processus de *Scheduling* d'un *workflow* composé de 4 tâches sur 2 VM.

La principale différence entre notre modèle et l'algorithme de HEFT est que, contrairement à l'algorithme HEFT où les valeurs de rang sont déterminées et fixées par l'heuristique dès le départ, nos attributs de rang sont d'abord générés aléatoirement et seront ensuite modifiés au cours du processus évolutif dans la phase de mutation de l'algorithme GA.

Notre justification pour le choix de ce type de *Scheduling* est essentiellement dû au fait qu'il assure le respect des contraintes de précédence. En outre, l'intégration du paramètre de rang dans le codage de la solution permet toujours de résoudre le cas où il existe deux ou plusieurs tâches prêtes, qui pourraient être planifiées simultanément. A titre d'illustration, la commutation des rangs des tâches T_1 et T_2 de la figure 4.6 changera le plan d'ordonnancement.

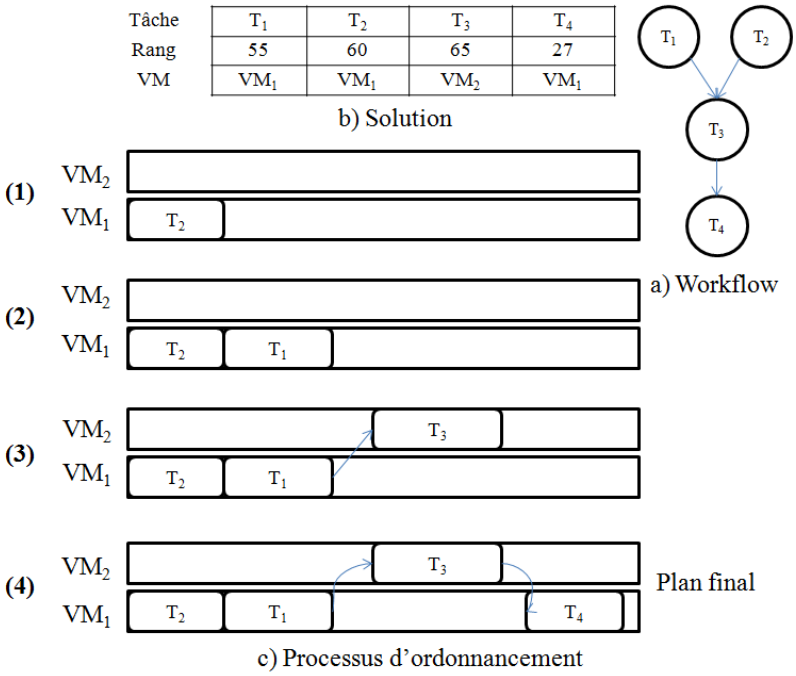


Figure 3.6. Illustration du processus de Scheduling d'un *workflow* simple sur deux VM.

3.3.1.3. Fonction de *fitness*

Les algorithmes génétiques sont généralement efficaces pour résoudre des problèmes d'optimisation multiobjectifs. Cependant, ce n'est pas un processus facile en raison de la difficulté de concevoir une fonction de *fitness* appropriée au problème étudié. Dans cette section, nous introduisons un modèle de fonction de *fitness* qui est générique et qui peut être facilement adapté pour optimiser tout type d'objectif numérique.

Supposons que nous ayons *k* métriques de QoS à optimiser; ces métriques peuvent être classées en deux catégories: les métriques de QoS à maximiser (fiabilité) et celles à minimiser (*makespan*, coût, etc.). Pour les deux cas de figures, les QoS ont des échelles et des valeurs très différentes. Par conséquent, la normalisation est nécessaire pour que la fonction de *fitness* ne soit pas biaisée par une des métriques ayant une valeur élevée. La normalisation est donnée par l'équation 3.7.

$$QoS_{iNormalized} = \begin{cases} \frac{QoS_i}{MaxQoS_i}, \ Si\ QoS_i \ \text{à}\ minimiser \\ 1 - \frac{QoS_i}{MaxQoS_i}, \ Si\ QoS_i \ \text{à}\ maximiser \end{cases} \quad (3.7)$$

Comme on peut le voir, l'équation 3.7 calcule la valeur normalisée pour une métrique de qualité de service donnée QoS_i à une itération donnée. Cependant, à l'exception des variables telles que la fiabilité dont le maximum théorique est de 100%, une telle valeur est la plupart du temps inconnue. Dans notre algorithme, la valeur $MaxQoS_i$ représente la valeur maximale de la métrique de QoS_i trouvée au cours de l'itération précédente. Compte tenu de cette définition, la normalisation d'une même métrique peut varier d'une itération à l'autre.

Dans cette section, nous avons utilisé la méthode d'agrégation pour définir la valeur de *fitness globale* d'une solution. Ainsi, la deuxième étape lors de la conception de la fonction de *fitness* est de combiner linéairement les différents objectifs normalisés en utilisant la méthode de la somme pondérée, comme indiqué dans l'équation 3.8. Dans ce modèle de *fitness*, la valeur de 0 correspond à la meilleure valeur et la valeur de 1 correspond à la pire valeur.

$$Fitness^* = \sum(\omega_i \times QoS_{iNormalized}) \tag{3.8}$$

où,

ω_i est le poids affecté à la variable de qualité de service QoS_i, $\sum \omega_i = 1$. L'objectif est de donner plus ou moins d'importance à la métrique de QoS correspondante. Les valeurs par défaut des poids sont de 0.25 pour les variables de qualité de service qui seront optimisées, et de 0 pour celles qui ne seront pas étudiées.

3.3.1.4. Opérateurs génétiques

Les opérateurs génétiques représentent le cœur d'un algorithme génétique. L'utilisation de tels opérateurs permet de générer de nouvelles solutions à partir de celles déjà existantes.

- **Opérateurs de sélection et de croisement**

L'opérateur de croisement est utilisé pour générer une ou deux nouvelles solutions à partir de deux solutions parents. Toutefois, avant d'effectuer le croisement, nous utilisons un processus de sélection afin de choisir les solutions parents à faire reproduire.

Plusieurs algorithmes génétiques pour l'ordonnancement de *workflow* utilisent la méthode de sélection par roulette *« roulette wheel »*, où chaque solution est attribuée plus ou moins de chances pour participer à la génération de la population suivante, selon la valeur de sa *fitness*. Dans notre algorithme, nous procédons en deux étapes pour sélectionner les solutions parents:

- D’abord, le pool de solutions est réduit en ne gardant que α% des meilleures solutions, en fonction de leur valeur de *fitness*. α étant un paramètre de taux de sélection renseigné en entrée de l'algorithme génétique. Généralement, il est fixé à 50%.
- Une fois cette étape effectuée, des paires de solutions parents sont choisies avec une probabilité égale dans le pool de solutions sélectionnées afin de les croiser.

L'objectif d'un opérateur de croisement est de combiner les caractéristiques de deux bonnes solutions afin de générer des solutions encore meilleures. Il existe plusieurs types de croisements tels que: le croisement aléatoire (*Random Crossover*), le croisement en un point (*One-point Crossover*), le croisement à deux points (*Two-point Crossover*), et autres. Dans cette thèse, nous avons implémenté ces différents types de croisements et nous avons proposé un operateur de croisement élitiste (*Elitist Crossover*).

- ***One-point Crossover* (*Cut Crossover*)**

Cet opérateur consiste à choisir aléatoirement un point de coupure pour partager chaque parent en deux parties. Le premier enfant est construit en utilisant la première partie du premier parent et la deuxième partie du deuxième parent. A l'inverse, le deuxième enfant est une combinaison de la seconde partie du premier parent et de la première partie du second parent. La figure 3.7 illustre le principe du C*ut Crossover* pour l’ordonnancement d’un *workflow* composé de 6 tâches sur une *IaaS* composée de 3 VM. On remarquera que les rangs des tâches ne sont pas présentés, car ils ne seront pas affectés par l’opérateur de croisement.

- ***Two-point Crossover***

Nous avons implémenté le croisement décrit dans (Yu, 2007), où deux points aléatoires sont choisis pour former une fenêtre de croisement. Puis les segments de VM se trouvant dans la fenêtre de croisement sont échangés pour former deux solutions enfants, comme illustré sur la figure 3.8.

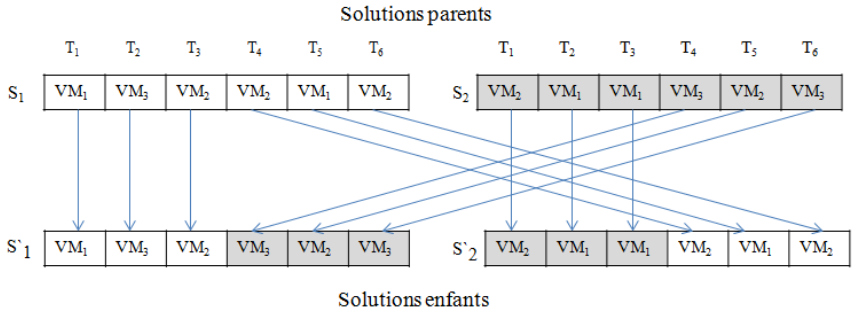


Figure 3.7. Exemple du « *Cut Crossover* » pour l'ordonnancement de *workflow*.

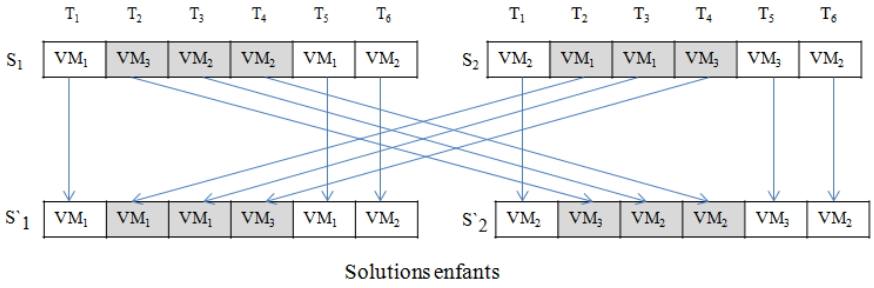


Figure 3.8. Exemple du « *Two-point Crossover* » pour l'ordonnancement de *workflows*.

- ***Random Crossover***

La solution enfant hérite de façon aléatoire des gènes de ses parents, la figure 3.9 illustre ce type de croisement.

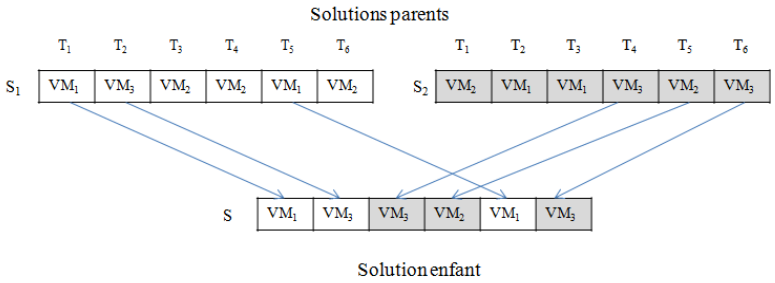


Figure 3.9. Exemple du « *Rondom Crossover* » pour l'ordonnancement de *workflows*.

- ***Crossover Elitist.***

Nous proposons ce type de croisement qui est basé sur une version modifiée du *Random Crossover*. L'idée principale consiste à sélectionner, de manière aléatoire, des gènes des deux parents S_i et S_j sur la base d'un ratio *r* calculé en fonction de leurs valeurs de *fitness*,

respectivement, F_i et F_j. Le calcul du ration est illustré dans l'équation 3.9. La solution enfants héritera de r% des caractéristiques du meilleur parent et de 1-r% du deuxième parent. Cela compense le fait que nous n'utilisons pas un processus de sélection par « roulette », en donnant plus de chance à la meilleure solution de reproduire ses caractéristiques. La figure 3.10 montre un exemple du croisement élitiste, où la *fitness* de S_1 est supposée meilleure que celle de S_2. Ainsi S_1 transmettra plus de caractéristiques à la solution enfant.

$$r = \frac{Max\{F_i, F_j\}}{F_i + F_j} * 100 \tag{3.9}$$

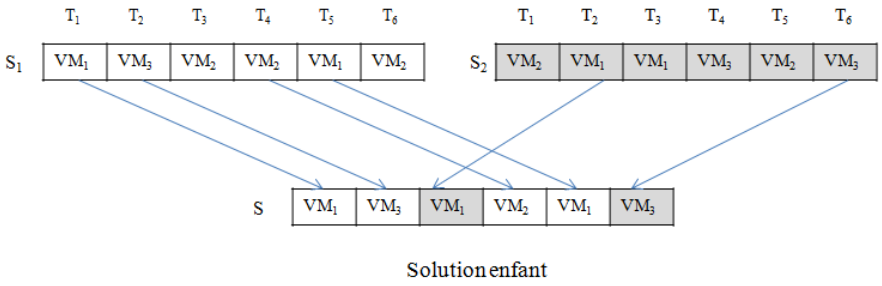


Figure 3.10. Exemple du « *Crossover Elitist* » pour l'ordonnancement de *workflows*.

- **Opérateur de mutation**

L'objectif de l'opérateur de mutation est d'éviter une convergence vers une solution optimale locale, en réintroduisant des caractéristiques aléatoires qui n'appartiennent à aucune des solutions parents. Dans notre modèle, nous introduisons trois types de mutations qui ont des probabilités égales de se produire. Pour toute solution sélectionnée pour le processus de mutation, une des modifications suivantes pourra se faire:

(1) Changer la VM attribuée à une tâche sélectionnée aléatoirement;
(2) Changer le rang d'une tâche sélectionnée aléatoirement;
(3) Permuter les rangs de deux tâches (quand il est possible).

Ces mutations peuvent présenter des changements au processus de *Scheduling*.

- **Remplacement**

Nous utilisons une stratégie de remplacement élitiste pour construire la nouvelle population. Cette stratégie consiste à conserver les meilleures solutions (selon leurs valeurs de *fitness*) provenant des deux populations, parent et enfant.

3.3.2. Expériences et résultats

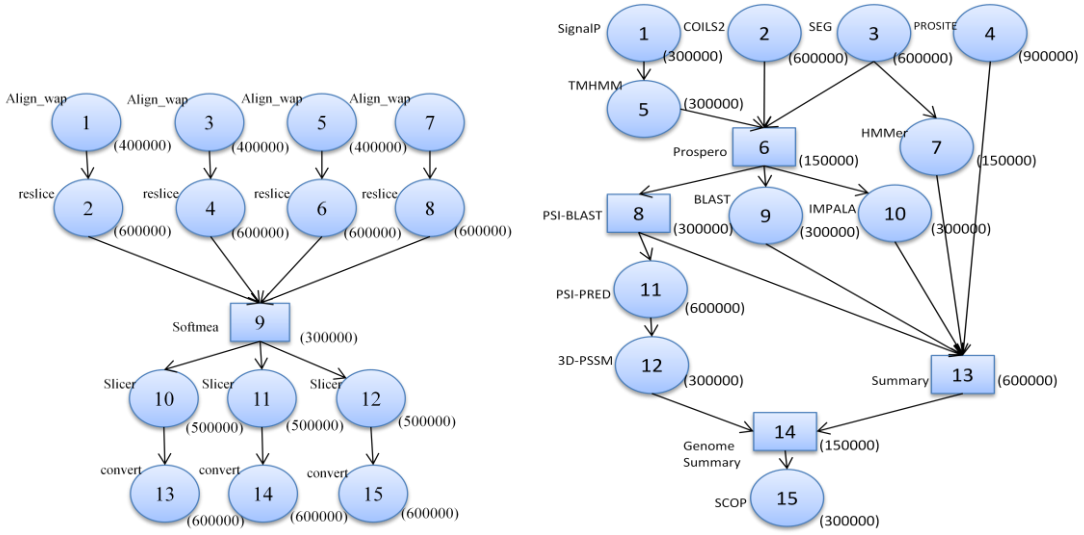
Dans cette section, nous discutons une évaluation expérimentale de l'algorithme génétique proposé.

3.3.2.1. Paramètres expérimentaux

Les paramètres expérimentaux concernent à la fois le *workflow* utilisateurs, le modèle d'infrastructure IaaS du fournisseur, ainsi que les paramètres de l'algorithme GA.

- **Paramètres du *workflow***

Pour nos expériences, nous utilisons deux structures de *workflows* qui sont courantes dans les applications de *workflow* scientifiques: une structure parallèle et une structure hybride. Une application parallèle se compose de plusieurs *pipelines* à être exécutés en parallèle (voir figure 3.11a). Un *pipeline* exécute un certain nombre de tâches dans un ordre séquentiel. Par exemple, dans la figure 3.11a, il y a quatre *pipelines* (1-2, 3-4, 5-6 et 7-8) avant la tâche 9. Une application ayant une structure hybride est une combinaison complexe d'applications parallèles et séquentielles (voir figure 3.11b). Dans nos expériences, nous avons choisi deux applications bien connues issues du monde réel; déjà étudiées dans d'autres travaux de recherches (Yu, 2006; Fatemipour, 2012). Nous avons choisi un *workflow* de la neuroscience (Zhao, 2004) pour notre application parallèle et un *workflow* d'annotation de protéines (O'Brien, 2004) développé par London-Centre-e-science, comme application hybride. La figure 3.11 montre leurs représentations simplifiées sous forme de graphe. Le nombre indiqué entre parenthèses à côté de chaque nœud représente la longueur de la tâche (le nombre d'instructions à exécuter) en Millions d'Instructions (MI). Chaque tâche dans le *workflow* génère des données de sorties qui seront utilisées par ses tâches filles comme des données d'entrée. Les données d'E/S des différentes tâches varient de 10 MB à 1024 MB. Dans nos expériences, nous avons utilisé aussi le *workflow* synthétique décrit dans (Topcuoglu, 2002) et d'autres *workflows* générés aléatoirement.



a. Application parallèle (Workflow fMRI (Zhao, 2004))

b. Application hybride (Workflow annotation protein (O'Brien, 2004))

Figure 3.11. Représentation de deux *workflows* scientifiques issus du monde reel.

- **Paramètres de l'infrastructure IaaS**

Nos expériences sont menées en utilisant les caractéristiques des instances de machines virtuelles fournies par Amazon EC2 (Amaz, 2013; Sarda, 2011). Chaque VM a sa propre vitesse de traitement, un prix et un taux d'échec. Les caractéristiques de ces VM sont indiquées dans le tableau 3.1. Comme les taux d'échecs des instances ne sont pas fournis, nous les avons générés, en nous basant sur des ressources ayant des performances similaires. Nous supposons que les vitesses de transferts moyens entre les instances sont fixées à 100 Mbps. Nous utilisons le modèle de facturation similaire à celui d'*Amazon cloudFront* (http://aws.amazon.com/*cloud*front/) pour le coût de transfert unitaire de données entre les VM.

Instances VMs	Vitesse (MIPS)	Prix $/h	Taux d'échec
m1.xlarge	5400	0.92	10^{-7}
m2.xlarge	4800	0.57	10^{-7}
m2_2xlarge	8500	1.14	10^{-6}

Tableau 3.1. Caractéristiques des instances de VM

- **Paramètres de l'algorithme génétique GA.**

Les différents paramètres de l'algorithme GA sont décrits dans le tableau 3.2. Le choix des valeurs des paramètres est effectué en expérimentant l'algorithme sur les deux *workflows*

et trois VM, pour différentes valeurs de ces paramètres. Nous choisissons les valeurs permettant à l'algorithme d'obtenir les meilleurs résultats d'ensemble. Cette méthode est utilisée pour la détermination de tous les paramètres choisis dans cette thèse.

Paramètre	Valeur
Taille de la population	100
Nombre de générations	100
Taux de croisement	0.8
Taux de mutation	0.05
Taux de sélection	0.5

Tableau 3.2. Paramètres de l'algorithme GA

3.3.2.2. Evaluation des performances

Pour évaluer les performances de l'algorithme GA, nous avons réalisé plusieurs expériences afin de (1) comparer les différentes méthodes de croisements présentées et (2) tester la fonction de *fitness* multiobjectif agrégée lorsque plusieurs métriques de QoS sont requises.

- **Analyse des méthodes de croisement**

Afin d'évaluer et de comparer les performances des quatre méthodes de croisements: *Cut Crossover*, *Two-point Crossover*, *Random et l'Elitiste,* nous avons appliqué l'algorithme GA pour optimiser le *makespan* d'un *workflow* synthétique déjà étudié dans de travaux antérieurs (Topcuoglu, 2002). (Dans ces conditions, ce *workflow* est connu avoir un meilleur *makespan* de 76 secondes). Les convergences de l'algorithme GA utilisant à chaque fois une des méthodes de croisement ci-dessus sont illustrées dans la figure 3.12.

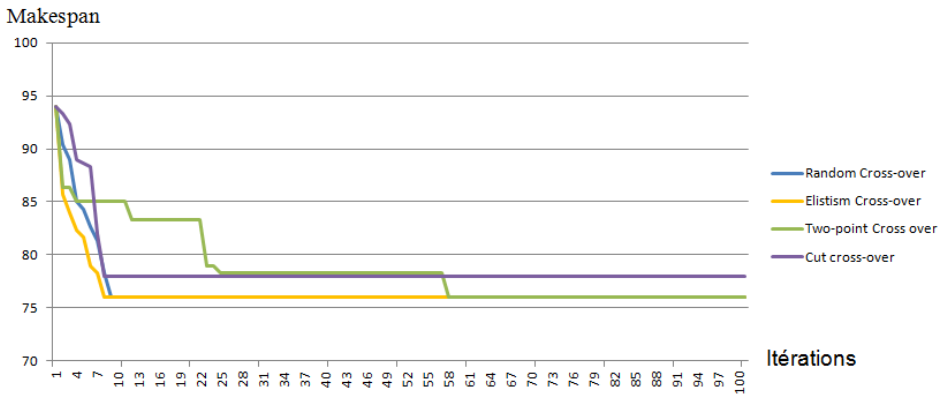


Figure 3.12. Comparaison des méthodes de croisements sur un *workflow* synthétique

La figure 3.12 montre que toutes les méthodes de croisements ont tendance à donner le même résultat final en termes de *makespan*. Néanmoins, la vitesse de convergence de l'algorithme GA utilisant le croisement élitiste est plus rapide que les autres. Afin de confirmer ces résultats, nous avons effectué des expériences en utilisant les *workflows* issues du monde réel (le *workflow* hybride et le *workflow* parallèle) et les 3 VM de l'infrastructure IaaS décrites précédemment.

Les figures 3.13 et 3.14 montrent les résultats obtenus, respectivement, pour le *workflow* parallèle et le *workflow* hybride. Ces résultats montrent que, dans les deux cas de figures, l'algorithme GA utilisant l'*Elitist Corssover* a une meilleure convergence comparée au *Two-point Crossover*.

Nous avons effectué, cette fois, des tests en faisant varier la taille de la population (5, 10, 15, 20, 25, 50, 75, 100, 200). Les résultats obtenus pour le *workflow* parallèle et le *workflow* hybride sont illustrés respectivement, dans les figures 3.15 et 3.16. Nous avons réalisé 20 simulations par taille de population. Les résultats montrent la supériorité de l'opérateur de croisement *Elitist* en termes de convergence et de qualité de la solution. Nous choisissons ainsi d'utiliser cette méthode comme opérateur de croisement, pour l'ordonnancement de *workflows* dans l'infrastructure IaaS.

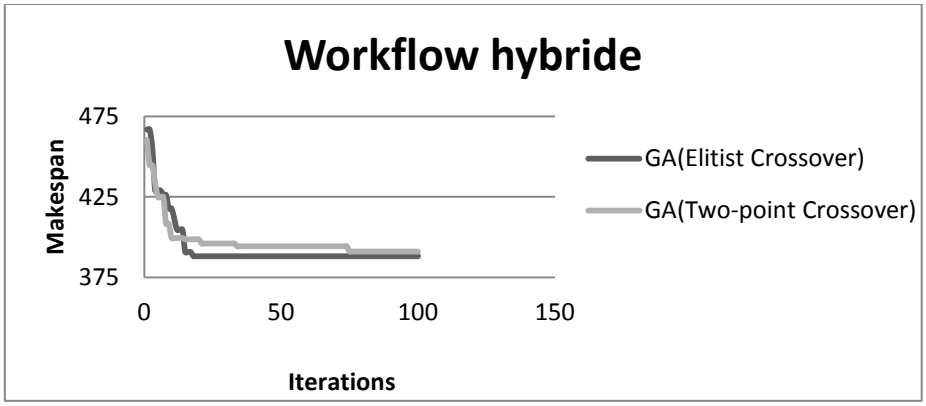


Figure 3.13. Comparaison des méthodes de croisements sur le *workflow* hybride.

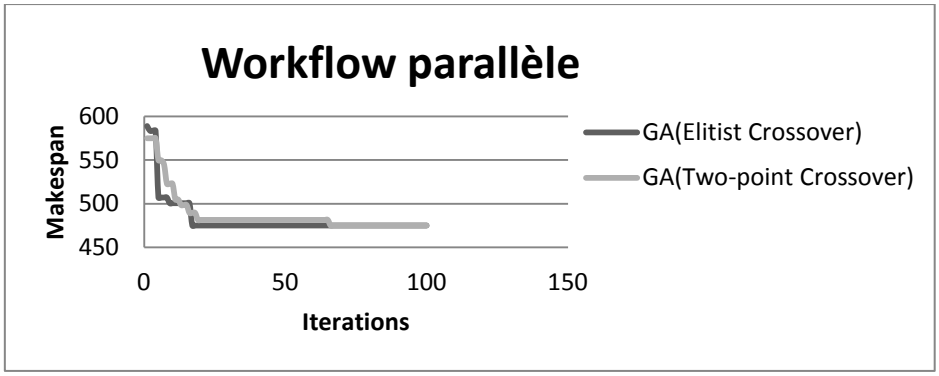


Figure 3.14. Comparaison des méthodes de croisements sur le *workflow* parallèle.

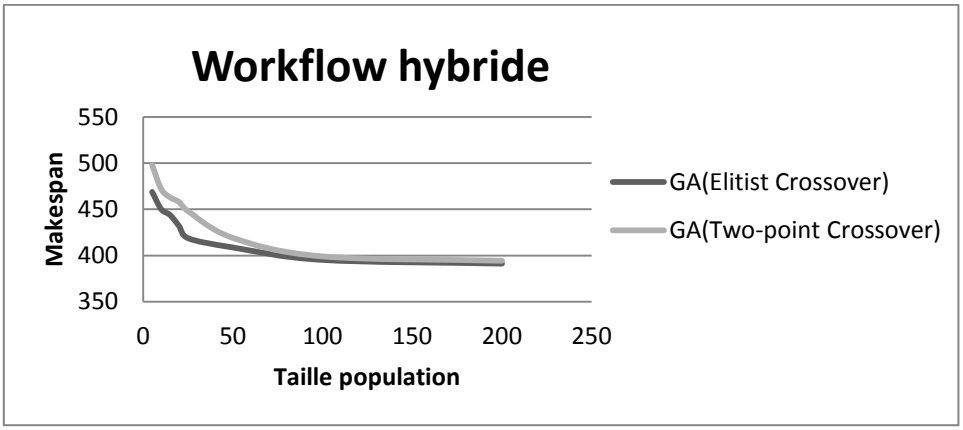


Figure 3.15. Comparaison des croisements en fonction de la taille de la population (*workflow* hybride).

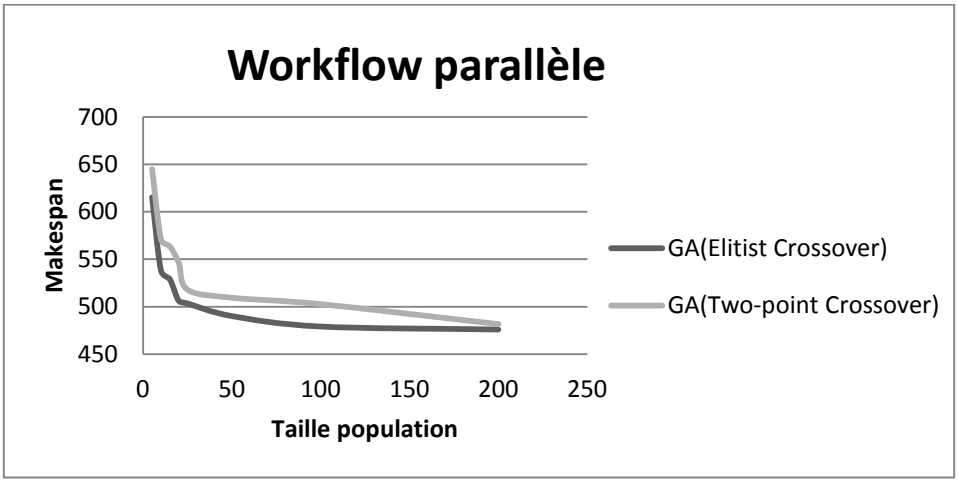


Figure 3.16. Comparaison des croisements en fonction de la taille de la population (*workflow* parallèle).

- **Intérêt de la fonction de *fitness* multiobjectif agrégeant plusieurs métriques de QoS**

A notre connaissance, en raison de l'absence actuellement de travaux de recherche traitant l'ordonnancement de *workflows,* qui prennent en compte, à la fois, les quatre métriques de qualités de services (*makespan*, coûts, fiabilité et disponibilité), et la configuration hétérogène de l'*IaaS* considérée dans ce chapitre, il devient très difficile de comparer nos algorithmes sans résultats de référence. Ainsi, nous proposons le processus d'évaluation suivant:

- Nous effectuons une optimisation monoobjectif de chacune des métriques de QoS considérées individuellement afin d'obtenir sa meilleure valeur approximative possible.
- Nous appliquons l'algorithme GA, qui utilise la fonction *fitness* agrégeant les différentes métriques de QoS et nous extrayons la valeur de chaque métrique de QoS dans la solution trouvée.
- Enfin, nous comparons les résultats obtenus pour chaque métrique de QoS par l'algorithme GA avec son meilleur résultat obtenu par l'optimisation monoobjectif.
- Nous effectuons également plusieurs simulations, en faisant varier les poids des différentes QoS.

Ces expériences sont réalisées sur les deux types de *workflows*, à savoir le *workflow* hybride et le *workflow* parallèle, en utilisant les 3 types de VM de l'IaaS. Les différents résultats sont illustrés dans les tableaux et figures suivants.

Le tableau 3.3 montre les résultats obtenus en optimisation monoobjectif de chaque QoS individuellement pour le *workflow* hybride. Le tableau 3.4 présente les résultats pour le *workflow* parallèle.

Optimisation monoobjectif	*Makespan* (s)	Fiabilité (%)	Coût ($)	Disponibilité (%)
Makespan	**441,176**	99,9995172	2,3829	35,185
Coût	688,235	99,9993118	**1,91176**	66,6667
Fiabilité	616,349	**99,999624**	2,50272	51,2732
Disponibilité	456,738	99,9995763	2,50511	**34,1826**
Meilleurs valeurs	**441,176**	**99,999624**	**1,91176**	**34,1826**

Tableau 3.3. Meilleures valeurs des QoS pour le *workflow* hybride

Optimisation monoobjectif	*Makespan* (s)	Fiabilité (%)	Coût ($)	Disponibilité (%)
Makespan	**481,612**	99,9994593	3,25549	18,8851
Coût	894,118	99,9991059	**2,48366**	66,6667
Fiabilité	962,963	**99,9996214**	3,45921	56,8929
Disponibilité	481,612	99,9994593	3,25549	**18,8851**
Meilleurs valeurs	**481,612**	**99,9996214**	**2,48366**	**18,8851**

Tableau 3.4. Meilleures valeurs des QoS pour le *workflow* parallèle

Les résultats obtenus par l'algorithme GA agrégeant les différentes QoS avec différents poids, sont affichés respectivement, dans le tableau 3.5 et sur la figure 3.17 pour le *workflow* hybride ; et dans le tableau 3.6 et sur la figure 3.18 pour le *workflow* parallèle.

On note que, dans les figures 3.17 et 3.18, les résultats multiobjectifs sont évalués entre 0 et 100 ; 0 étant la pire valeur d'une QoS donnée, et 100 représente la meilleure valeur obtenue en optimisation monoobjectif de cette QoS. Aussi, la configuration GA (w1 w2 w3 w4) dans ces figures signifie que nous avons agrégé les différentes métriques de QoS, à savoir le *makespan*, la fiabilité, le coût et la disponibilité dans la fonction de *fitness* de l'algorithme GA en utilisant, respectivement, les poids w1, w2, w3, w4.

Optimisation Multiobjectif	*Makespan* (s)	Fiabilité (%)	Coût ($)	Disponibilité (%)
GA (0.25 0.25 0.25 0.25)	441,176	99,999342	2,32503	36,7593
GA (0.14 0.58 0.14 0.14)	652,941	99,999342	1,96805	64,3083
GA (0.22 0.56 0.11 0.11)	476,471	99,999490	2,24946	44,76
Meilleures valeurs	**441,176**	**99,999624**	**1,91176**	**34,1826**

Tableau 3.5. Résultats pour le *workflow* hybride

Optimisation Multiobjectif	*Makespan* (s)	Fiabilité (%)	Coût ($)	Disponibilité (%)
GA (0.25 0.25 0.25 0.25)	481,612	99,999459	3,25549	18,8851
GA (0.14 0.58 0.14 0.14)	600	99,999354	2,95267	40,9465
GA (0.22 0.56 0.11 0.11)	509,807	99,999441	3,17939	25,1625
Meilleures valeurs	**481,612**	**99,9996214**	**2,48366**	**18,8851**

Tableau 3.6. Résultats pour le *workflow* parallèle

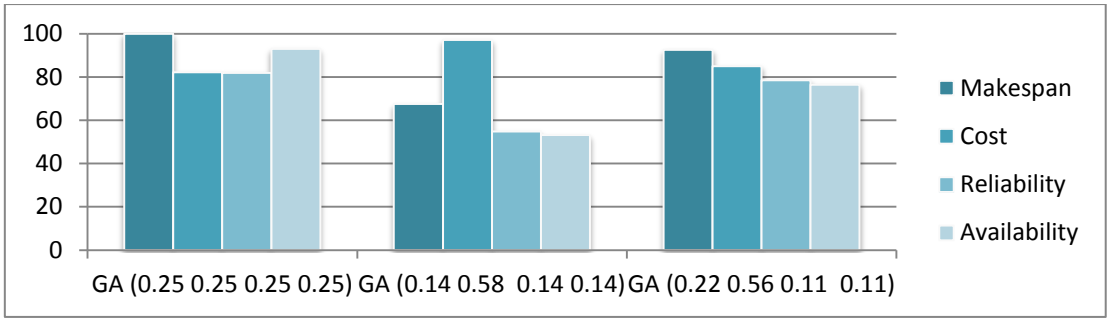


Figure 3.17. Résultats pour le *workflow* hybride

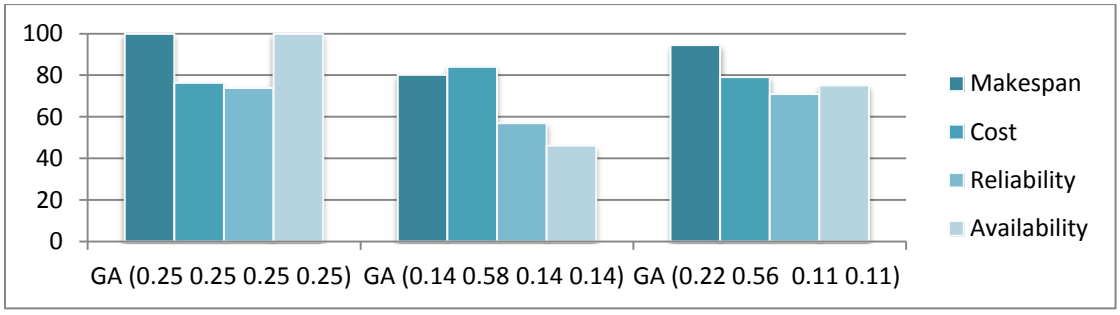


Figure 3.18. Résultats pour le *workflow* parallèle

D'après la figure 3.17, nous pouvons constater que l'algorithme GA pour le *workflow* hybride donne de bons résultats d'ensemble en utilisant les valeurs de poids de 0,25 pour chaque métrique de QoS (ceci est représenté par la configuration GA (0.25 0.25 0.25 0.25)). Nous pouvons voir que toutes les métriques atteignent plus de 80% de leur meilleure valeur. Le *makespan* atteint la meilleure valeur obtenue en optimisation monoobjectif. Dans l'expérience GA (0.14 0.58 0.14 0.14), un poids important est donné au coût afin de voir son impact sur les autres métriques de QoS. Les résultats montrent que la réalisation d'un bon coût, conduit à une dégradation significative des autres métriques de QoS. Renforcer le poids du *makespan* dans la configuration (GA (0.22 0.56 0.11 0.11)) donne à nouveau de bons résultats d'ensemble pour toutes les métriques de QoS.

Concernant le *workflow* parallèle, les résultats sont pratiquement similaires à ceux du *workflow* hybride, comme le montre la figure 3.18. Les performances obtenues avec les valeurs de poids par défaut GA (0.25 0.25 0.25 0.25) sont supérieures à 70% pour toutes les métriques de QoS. L'amélioration du coût impacte aussi les autres métriques.

Pour résumer, les résultats des expériences (avec les valeurs des poids par défaut pour toutes les variables) donnent de bons résultats, un score supérieur ou égal à 70% pour l'ensemble des métriques de QoS, pour les deux types de *workflows* hybride et parallèle.

3.3.3. Discussion

Dans cette section, nous avons présenté un algorithme génétique GA pour traiter le problème d'ordonnancement d'un *workflow* utilisateur sur une infrastructure virtualisée d'un fournisseur de services IaaS. Dans cet algorithme, nous avons proposé un nouveau modèle de fonction de *fitness,* qui est générique et qui prend en compte un grand nombre de métriques de QoS, négociées entre l'utilisateur et le fournisseur et qui sont spécifiées dans le SLA. Nous avons mis l'accent sur l'optimisation de multiples métriques de QoS, notamment, le *makespan*, le coût, la fiabilité et la disponibilité des ressources, lors de l'ordonnancement du *workflow* utilisateur sur l'infrastructure IaaS.

Les résultats expérimentaux montrent que l'algorithme GA donne des bons résultats d'ensemble pour toutes les métriques de QoS. Toutefois, l'utilisateur peut exiger de respecter certaines contraintes complémentaire sur les QoS telles que le délai, le *budget* et autre. En outre, il peut exiger de respecter des contraintes fortes liées à l'affectation de certaines tâches sur certaines VM. Ces contraintes ajoutent un niveau de difficultés au problème d'ordonnancement de *workflows*.

3.4. Un algorithme génétique basé sur l'infection virale pour l'ordonnancement de *workflows* dans l'IaaS

Nous avons vu que l'algorithme d'ordonnancement basé GA, présenté dans la section précédente est efficace pour traiter plusieurs métriques de QoS à savoir: le *makespan*, le coût, la fiabilité et la disponibilité. Néanmoins, il ne prend pas en compte des contraintes sur ces métriques (telles que les délais, le budget), ainsi que des contraintes fortes (qui obligent d'exécuter certaines tâches sur certaines VM). Dans cette section, nous proposons d'améliorer l'algorithme GA, en ajoutant une méthode de pénalité à la fonction objectif pour satisfaire les contraintes de QoS, et un operateur d'infection virale pour traiter les contraintes fortes spécifiées dans le SLA. Nous nommons cette algorithme GA*.

3.4.1. Etapes de l'algorithme GA*

Le schéma général de l'algorithme génétique amélioré est présenté sur la figure 3.19. Nous avons adopté les mêmes étapes que l'algorithme GA présenté dans la section précédente, et nous avons ajouté l'operateur de l'infection Virale et une méthode de pénalité

dans la fonction objectif. Nous décrivons, dans cette partie, uniquement les composants ajoutés.

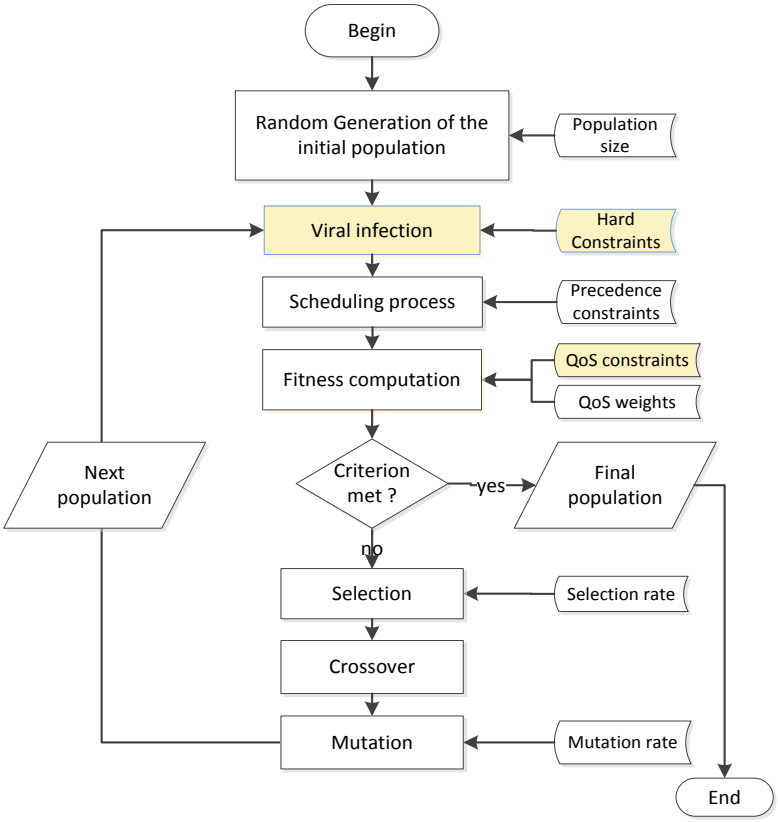


Figure 3.19. Schéma général de l'algorithme GA*

3.4.1.1. Fonction de *fitness* et pénalité

Afin de prendre en considération les contraintes de QoS comme le délai, le budget et autres, nous avons intégré une fonction de pénalité à la fonction de *fitness*, $\boldsymbol{Fitness}^*$ calculée précédemment dans l'équation 3.8 de la section 3.3.1.3: à chaque fois que l'un des composants de *fitness* n'était pas conforme aux exigences requises, une pénalité quadratique est appliquée à la composante concernée, selon l'équation 3.10.

$$Penalty_i = \begin{cases} \text{Max}\left(1, \left(\frac{QoS_i}{Limit_i}\right)^2\right), & \text{si } QoS_i \text{ à minimiser} \\ \text{Max}\left(1, \left(\frac{Limit_i}{QoS_i}\right)^2\right), & \text{si } QoS_i \text{ à maximiser} \end{cases} \quad (3.10)$$

où,

- QoS_i représente une variable de qualité de service calculée après l'ordonnancement.

- $Limit_i$ représente la contrainte de la métrique de QoS_i à respecter.

La formule finale de la fonction de *fitness* est donnée par l'équation 3.11.

$$Fitness = Fitness^* \times \prod Penalty_i \quad (3.11)$$

3.4.1.2. Opérateur d'Infection Virale

L'une des faiblesses des algorithmes génétiques est qu'ils ne peuvent pas gérer facilement les problèmes de contraintes fortes. Dans le cas du problème d'ordonnancement de *workflows*, les contraintes telles que "*une tâche T_a doit être exécutée sur les services VM_b ou VM_c*" sont assez fréquentes, pour des raisons de sécurité, de puissance de calcul, ou autres. Le processus d'évolution aléatoire des algorithmes génétiques n'est pas efficace dans le traitement de ce genre de contraintes.

En effet, la plupart des algorithmes génétiques et stratégies d'évolution attribuent simplement des faibles valeurs de *fitness,* aux solutions qui ne satisfassent pas ces contraintes fortes. Les algorithmes passent alors la plupart du temps à chercher des solutions aléatoires qui sont réalisables, avant de commencer réellement l'optimisation des métriques de QoS. Par conséquent, le processus d'optimisation sera considérablement ralenti, si le nombre de contraintes augmente.

Afin de garantir que toutes les solutions générées respectent les contraintes fortes, nous utilisons un processus *d'infection virale* inspiré du processus décrit dans (Kanoh, 1997) au lieu de supprimer les solutions non valides. Une contrainte forte peut être codée comme un gène, ainsi une solution au problème d'ordonnancement qui exige de respecter, par exemple la contrainte "*tâche T_a doit être exécutée sur une machine VM_b*", nécessite de contenir une paire *(T_a, VM_b)* afin de respecter cette contrainte.

L'idée derrière le processus d'infection virale est de créer une population, de soi-disant de virus, sous forme de chromosomes *(T_x, VM_y),* représentant les contraintes fortes. Durant la phase d'infection virale, toutes les solutions de la population vont êtres vérifiées, et celles qui ne respectent pas les contraintes fortes exigées seront infectées aléatoirement par les virus concernés (afin de les modifier). Un exemple de ce processus est montré dans la figure 3.20.

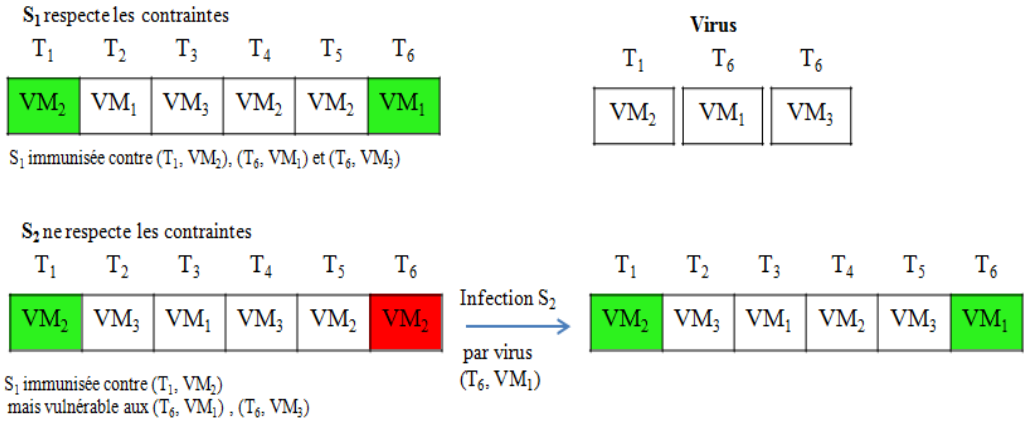


Figure 3.20. Exemple d'infection virale sur une solution d'ordonnancement

Dans l'exemple de la figure 3.20, nous avons deux contraintes fortes, à savoir: l*a tâche T_1 doit être ordonnancée sur la machine VM_2 et la tâche T_6 doit être affectée soit à la machine VM_1, soit à la machine VM_3*. Cela nous donne la population de virus {(T_1, VM_2), (T_6, VM_1) et (T_6, VM_3)}. On peut remarquer que les deux solutions de cet exemple, S_1 et S_2 sont immunisées contre (T_1, VM_2), car les deux solutions respectent la contrainte de la tâche T_1. Cependant, S_2 ne respecte pas les contraintes de la tâche T_6, ce qui signifie que cette solution est vulnérable, à la fois, aux virus (T_6, VM_1) et (T_6, VM_3). Dans cet exemple, S_2 est infectée par le virus (T_6, VM_1). Par conséquent, le gène de S_2 dont l'index est T_6 est modifié. On notera que les attributs rang des solutions ne sont pas représentés sur cette figure, car ils ne sont pas affectés par le processus d'infection virale.

3.4.2. Expériences et résultats

Cette section présente une évaluation expérimentale de l'algorithme d'ordonnancement de *workflows* basée GA*. Les expériences visent à démontrer l'intérêt de la méthode de pénalité et l'intérêt de l'operateur de l'infection virale, lorsque l'utilisateur exige de respecter respectivement, des contraintes sur les QoS (budget, deadline, etc.) et des contraintes fortes ($tâche_i$ sur VM_j). Nous avons repris les mêmes paramètres expérimentaux concernant les *workflows*, le modèle de l'infrastructure IaaS et les paramètres de l'algorithme génétique, décrits précédemment, dans la section 3.3.2.1.

- **Intérêt de la méthode de pénalité proposée lors de l'ajout de contraintes sur les métriques de QoS.**

Nous avons effectué des simulations, afin d'évaluer l'efficacité de la méthode de pénalité, lors de l'ajout de contraintes sur les différentes métriques de QoS, comme les contraintes de délais, de budgets, etc. Nous avons implémenté l'algorithme HEFT (Topcuoglu, 2002) afin d'obtenir quelques résultats de référence.

Comme nous l'avons indiqué au chapitre 2, l'algorithme HEFT est parmi les algorithmes les plus largement utilisés, pour l'ordonnancement de *workflows* optimisant le *makespan*. Les résultats de l'application de cette heuristique sur le *workflow* parallèle et le *workflow* hybride, sont présentés dans le tableau 3.7.

Nous avons utilisé ces résultats afin de fixer certaines contraintes sur les métriques de QoS telles que le *deadline*. Cependant, pour les autres paramètres de QoS tels que le *budget* et la fiabilité minimale, nous avons utilisé les résultats obtenus par l'optimisation monoobjectif décrite précédemment. Nous avons dégradé légèrement les meilleures valeurs obtenues pour chacune de ces métriques, afin de définir leurs contraintes respectives.

Les tableaux 3.8 et 3.9 présentent respectivement les résultats de l'ordonnancement du *workflow* parallèle et du *workflow* hybride sur 3 VM, en utilisant l'algorithme GA*. Les métriques de QoS considérées sont le *makespan*, le coût et la fiabilité, sous une ou plusieurs contraintes sur ces métriques.

Workflow	Algo.	*Makespan* (s)	Coût ($)	Fiabilité (%)
Parallèle	HEFT	500,575	3,40664	99,9995
Hybride	HEFT	405,882	2,41024	99,9995

Tableau 3.7. Résultats de HEFT sur le *workflow* parallèle et le *workflow* hybride

Algo.	Contraintes	*Makespan* (s)	Coût ($)	Fiabilité (%)
GA*	*Makespan*<=500	**481,612**	3,25549	99,9995
		479,532	3,28068	99,9995
GA*	Coût <=2.5	894,118	**2,48366**	99,9992
GA*	Fiabilité>=99.9997	666,667	3,788806	**99,9997**
GA*	*Makespan* <=550, Coût <=3.2	**529,412**	**3,12954**	99,9994

Tableau 3.8. Résultats de l'ordonnancement basé contraintes QoS du *workflow* parallèle

Algo	Contraintes	*Makespan* (s)	Coût ($)	Fiabilité (%)
GA*	*Makespan* <=400	**394,002**	2,48582	99,9996
		397,9	2,485817	99,9996
GA*	Coût <=2.25	688,235	**1,91176**	99,9994
GA*	Fiabilité >=99.9997	480,122	2,78731	**99,9997**
GA*	*Makespan* <=450, Coût <=2.5	**441,176**	**2,33467**	99,9995

Tableau 3.9. Résultats de l'ordonnancement basé contraintes QoS du *workflow* hybride

Dans les deux premières lignes de ces deux tableaux, l'utilisateur fixe des contraintes de délais afin d'obtenir des meilleurs *makespan*. Il spécifie des délais de 500 et de 400 pour, respectivement, le *workflow* parallèle et le *workflow* hybrides. Nous avons lancé plusieurs simulations, à partir desquelles nous n'avons gardé que les solutions qui sont meilleures que la solution de HEFT. L'analyse de ces résultats confirme que ces solutions respectent bien les contraintes exigées.

Les deuxièmes et les troisièmes lignes des deux tableaux montrent les résultats lorsque l'utilisateur exige des restrictions, respectivement, sur le budget (prix maximal autorisé) et la fiabilité minimale requise. Nous pouvons voir que les solutions trouvées et la solution HEFT sont non dominées, au sens de Pareto. Cela est dû au fait d'avoir relâché la contrainte du *makespan*.

Enfin, dans les dernières lignes des deux tableaux, les résultats obtenus lors de l'ajout de deux contraintes en même temps, sur les deux objectifs conflictuels du *makespan* et du coût sont montrés. Les solutions de cette expérience restent conformes aux restrictions requises, et encore non-dominées par la solution HEFT.

En résumé, l'algorithme GA* donne toujours des résultats qui sont non-dominées ou même meilleurs sur toutes les métriques de qualité de service, comparés aux solutions de HEFT. Ces résultats restent valables même après avoir appliqué une ou plusieurs contraintes sur les métriques de QoS. À partir de ces expériences, nous pouvons confirmer l'efficacité de la méthode de pénalité proposée, mettant en œuvre des contraintes sur les métriques de qualité de service.

- **Intérêt de l'opérateur de l'infection virale (VI) lors de l'ajout de contraintes fortes sur les tâches.**

Nous effectuons une autre expérience afin de montrer l'efficacité de l'algorithme GA* utilisant l'opérateur de l'infection virale (VI), quand l'utilisateur exige de respecter une ou plusieurs contraintes fortes (de type T_i doit être exécuté sur VM_j), dans l'ordonnancement du *workflow* sur les services IaaS. Pour cela, nous utilisons l'algorithme génétique simple GA décrit précédemment. Nous faisons varier le nombre de contraintes fortes (NbCstr.) et calculons le pourcentage de solutions qui violent ces contraintes, après chaque itération. Les résultats de ces expériences sont présentés dans la figure 3.21.

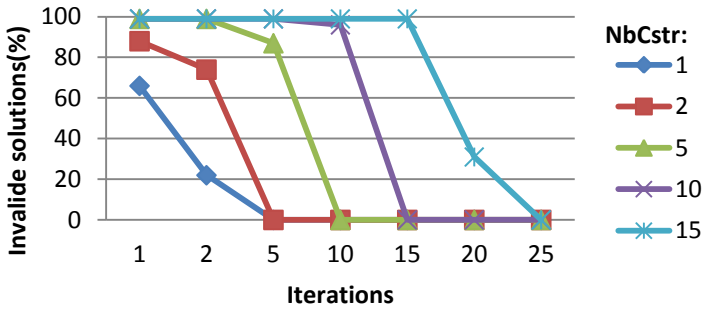


Figure 3.21. Pourcentage de solutions invalides dans le GA classique

Comme on peut le remarquer sur cette figure, les résultats mettent en évidence le fait que, sans aucun mécanisme de résolution de contraintes fortes, l'algorithme GA classique passe un certain nombre d'itérations à essayer de les résoudre. Cela est dû à son processus aléatoire. On remarque également que l'efficacité de l'algorithme GA se dégrade avec l'augmentation du nombre de contraintes fortes. Cependant, lors de l'utilisation de l'opérateur de l'infection virale dans l'algorithme GA*, ces pourcentages sont réduits à 0% à la fin de chaque itération. En effet, toutes les solutions non réalisables sont infectées par les virus correspondants afin de les modifier.

Cette expérience souligne que l'utilisation de l'opérateur de l'infection virale est un moyen efficace, pour faire face aux contraintes fortes, dans le problème d'optimisation d'ordonnancement de *workflows* dans l'infrastructure IaaS.

3.4.3. Discussion

Dans cette partie, nous avons présenté l'algorithme GA* appliqué à l'ordonnancement de *workflows* dans l'infrastructure IaaS. Les résultats des expériences montrent que l'algorithme GA* est capable de traiter efficacement les contraintes de QoS, et les contraintes fortes spécifiées dans le SLA négocié entre les utilisateurs et les fournisseurs des services IaaS dans un *cloud computing* .

Les algorithmes GA et GA* sont basés sur la transformation du problème d'ordonnancement multiobjectif de *workflows* en un problème monoobjectif. La section suivante traite cette problématique, en utilisant l'approche Pareto, afin de proposer à l'utilisateur un ensemble de solutions non-dominées.

3.5. Algorithmes d'ordonnancement de *workflow*s basés Pareto

Dans cette section, nous nous intéressons à la résolution du problème d'ordonnancement multiobjectif de *workflow* dans l'infrastructure IaaS, en faisant appel à l'optimisation multiobjectif basée sur l'approche Pareto. Le but est d'offrir à l'utilisateur un ensemble de solutions de compromis, au lieu d'une solution unique, qui est le cas des algorithmes GA et GA*. Pour ce faire, nous adaptons les algorithmes NSGA-II et PAES à cette problématique, dans le but de trouver des approximations du front de Pareto optimal.

3.5.1. NSGA-II pour l'ordonnancement de *workflow*s dans l'infrastructure IaaS

NSGA-II a longtemps été l'algorithme phare dans le domaine de l'optimisation évolutionnaire multiobjectif (Justesen, 2009). 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.

3.5.1.1. 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 P_0 de N individus parents. A la génération *t*, (voir figure 3.22) une population Q_t de N enfants est créée à partir de la population parent P_t, en utilisant les opérateurs génétiques de *sélection*, de *croisement* et de *mutation*. Ensuite, les deux populations sont fusionnées pour former une nouvelle population R_t de taille 2N. La recherche des solutions non-dominées permet de classer les individus de R_t, en plusieurs fronts de rangs différents. L'identification des différents fronts F_i est effectuée, en utilisant la profondeur de dominance décrite dans la section 2.2.4.2 du chapitre 2. La nouvelle population parent P_{t+1} est alors construite en incluant les meilleurs individus de R_t appartenant aux fronts de rang les plus faibles, tant que le nombre d'individus présents dans P_{t+1} est inferieur à N. Pour le front suivant, il y'a $N-|P_{t+1}|$ places restantes dans la nouvelle population P_{t+1}. Par conséquent, les individus sont triés selon leurs distances de *Crowding* (Deb, 2002), les $N-|P_{t+1}|$ meilleurs individus au sens de cette distance seront insérés à P_{t+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 3.2 résume toutes les étapes décrites ci-dessus.

Algorithme 3.2. Pseudo-code de l'algorithme NSGA-II

Générer les populations P_0 et Q_0 de taille N
Répéter
  Création de $R_t = P_t \cup Q_t$
  Calcul des différents fronts F_i de la population R_t par la profondeur de dominance
  Mettre $P_{t+1} = \varnothing$ et i = 0
  TantQue $|P_{t+1}| + |F_i| < N$ **Faire**
    $P_{t+1} = P_t \cup F_i$
    i = i + 1
  FinTantQue
  Inclure dans P_{t+1} les $(N- |P_{t+1}|)$ individus de F_i les mieux répartis au sens de la distance de *Crowding*
  Sélection dans P_{t+1} et création de Q_{t+1} par application des opérateurs de croisement, mutation et infection virale (dans le cas de contrainte forte)
Jusqu'à atteindre un critère d'arrêt.

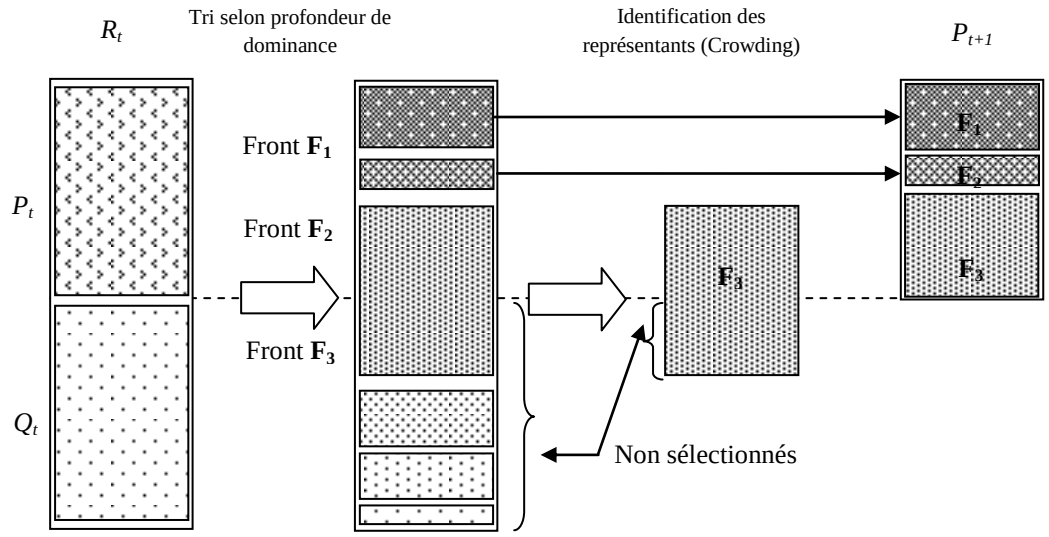


Figure 3.22. Schéma de fonctionnement de NSGA-II (Deb, 2001)

3.5.1.2. Calcul de la distance de "*Crowding*"

La distance de *Crowding* d_i 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 *Z* (deb, 2001).

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) : $d_{I_1^m} = d_{I_l^m} = \infty$, et pour les autres individus, c'est-à-dire pour tout $j =2$ à $(l-1)$ calculer:

$$d_j = \sum_{m=1}^{M} d_{I_l^m} + \frac{f_m^{I_{j+1}^m} - f_m^{I_{j-1}^m}}{f_m^{max} - f_m^{min}} \qquad (3.12)$$

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*.

3.5.1.3. Operateurs génétiques

Le calcul de la distance de *Crowding* pour NSGA-II ne sert pas uniquement pour la sélection des individus à inclure dans P_{t+1} (la population contenant les élites) mais il intervient

aussi dans le processus de sélection des individus pour les opérateurs de croisement et de mutation.

- **Processus de sélection**

Nous utilisons, dans NSGA-II, la sélection par tournoi. Elle consiste à tirer aléatoirement deux solutions de la population parent et de garder la solution ayant le rang inférieur (solution dominante). Si les deux solutions ont le même rang (les solutions sont non dominées), la solution ayant la plus grande distance de *Crowding* est sélectionnée.

- **Gestion des contraintes**

Dans la phase de sélection des solutions, lorsque deux individus sont comparés, leurs contraintes de QoS sont d'abord examinées. Si les deux solutions satisfassent les contraintes exigées, dans ce cas, la notion de dominance et de distance de *Crowding* sont appliquée afin de déterminer la solution potentielle à choisir. Si les deux sont des solutions non réalisables, alors, elles ne seront pas qualifiées. Si l'une est réalisable et l'autre ne l'est pas, celle qui est réalisable sera sélectionnée.

- **Opérateur de croisement**

Tous les opérateurs de croisement décrits dans la section 3.3.1.4 peuvent être utilisés dans NSGA-II, à l'exception de l'opérateur *Elitist Crossover* qui doit être modifié du fait qu'il se base sur la *fitness* globale des solutions. Cependant, dans ce qui suit, nous choisissons le *two-point crossover* comme opérateur de croisement par défaut de NSGA-II.

Dans NSGA-II, le codage d'une solution, le processus de *Scheduling*, l'infection viral et l'opérateur de mutation sont identiques à ceux présentés dans l'algorithme GA*.

3.5.2. PAES pour l'ordonnancement de *workflow*s dans l'infrastructure IaaS

PAES (*Pareto Archived Evolution Strategy*) proposé initialement par Knowles et Corne (Knowles, 1999) est une métaheuristique multiobjectif simple, qui applique une stratégie d'évolution par recherche locale, afin de trouver et de maintenir des solutions non-dominées, dans une archive de solutions. L'algorithme n'est pas basé sur une population. Il utilise un seul individu à la fois pour la recherche des solutions.

3.5.2.1. Etapes de l'algorithme PAES

L'algorithme PAES commence par générer aléatoirement une solution initiale et l'insère dans l'archive. La procédure, à l'itération *t*, est poursuivie par la génération d'une nouvelle

solution candidate par mutation de la solution courante et compare les deux solutions. Si l'une des solutions domine l'autre, celle qui domine est acceptée. D'autre part, si aucune des deux solutions ne domine l'autre, la solution candidate est comparée aux solutions déjà archivées. Les solutions dominées dans l'archive sont retirées et la nouvelle solution est ajoutée à l'archive. Dans le cas où l'un des membres de l'archive domine la nouvelle solution, cette dernière est rejetée. Enfin, si aucun des membres de l'archive ne la domine et que l'archive n'est pas pleine, la nouvelle solution est ajoutée à l'archive, et si l'archive est pleine, la solution dans la région la plus peuplée de l'espace est supprimée, laissant la place à la nouvelle solution. L'algorithme 3.3 présente les étapes de PAES. Nous utilisons pour cet algorithme, les mêmes modèles et processus de résolution: représentation d'une solution, processus de *Scheduling* et opérateur de mutation présentés précédemment dans l'algorithme génétique.

Algorithme 3.3. Algorithme PAES

Générer aléatoirement une solution *c* et l'ajouter à l'archive
Tant Que le nombre d'itération maximum n'est pas atteint **faire**
  Générer une solution *n* par mutation de *c* et évaluer *n*
  Si *n* domine c
    Alors remplacer *c* par *n*
      mettre à jour l'archive
      Sinon si *c* domine *n*
        Alors écarter *n*
      Sinon *n* et *c* sont nom dominées
        Mettre à jour l'archive en utilisant *n.*
        Sélectionner la solution courante *c* dans la région la moins encombrée
FinTantQue

3.5.3. Expériences et résultats

Cette section présente une évaluation expérimentale des algorithmes NSGA-II et PAES pour l'ordonnancement multiobjectif de *workflow* dans l'*IaaS cloud*. Les expériences visent à démontrer l'efficacité des algorithmes dans la génération d'un ensemble de solutions de compromis entre les différentes métriques de QoS, permettant ainsi à l'utilisateur de choisir celle qui lui convient, selon ses besoins.

3.5.3.1. Paramètres expérimentaux

Nous avons repris les mêmes paramètres expérimentaux de *workflows* (hybride et parallèle) et des caractéristiques du *cloud*, décrits précédemment, dans la section 3.3.2.1. Les paramètres de NSGA-II et PAES sont récapitulés dans le tableau 3.10.

Algorithme	**Paramètre**	**Valeur**
NSGA-II	Taille de la population Nombre de générations Taux de croisement Taux de mutation	100 100 0.8 0.05
PAES	Nombre d'itérations Taille de l'archive	5000 100

Tableau 3.10. Paramètres des algorithmes NSGA-II et PAES

3.5.3.2. Evaluation des performances

Dans ce qui suit, nous présentons les résultats expérimentaux des deux algorithmes NSGA-II et PAES. Ces algorithmes on l'avantage de s'adapter à n'importe quel nombre de métriques de QoS. Nous nous concentrons principalement sur l'optimisation de trois métriques car les résultats peuvent être visualisés graphiquement.

- **Résultats obtenus pour NSGA-II**

Les figures 3.22 et 3.23 montrent respectivement les résultats obtenus, en appliquant l'algorithme NSGA-II au *workflow* hybride et au *workflow* parallèle sur trois VM. Les métriques de QoS considérées sont le *makespan*, le coût et la fiabilité.

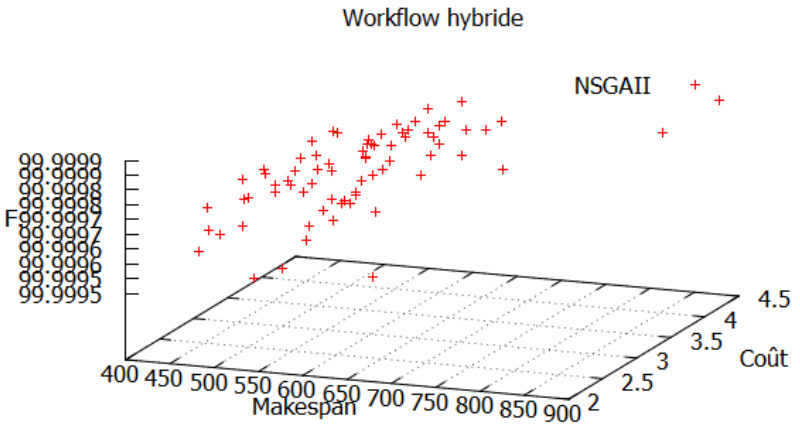


Figure 3.23. Solutions non dominées obtenues par NSGA-II pour le *workflow* hybride

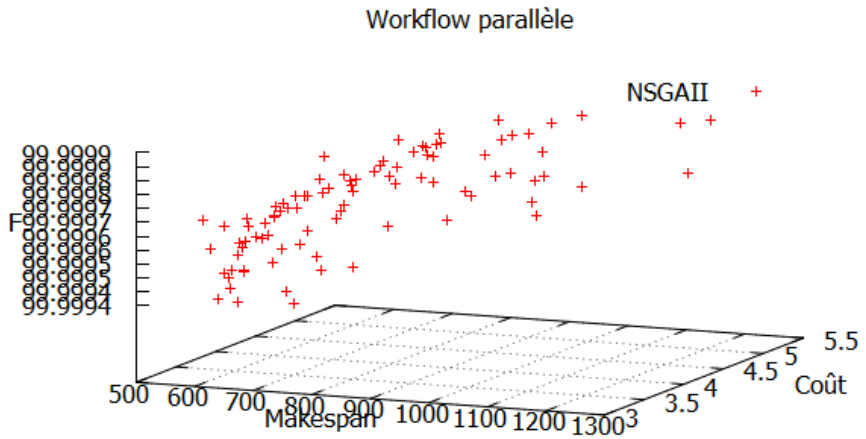


Figure 3.24. Solutions non dominées obtenues par NSGA-II pour le *workflow* parallèle

Nous pouvons constater, sur ces deux figures, que les solutions non dominées générées par NSGA-II sont bien distribuées pour les deux types de *workflows*.

- **Gestion de contraintes de QoS par NSGA-II**

Afin de tester l'efficacité de l'algorithme NSGA-II, pour le traitement des contraintes sur les métriques de QoS, exigées par l'utilisateur, nous avons effectué des expériences, en fixant les valeurs de quelques métriques de QoS à ne pas dégrader. Dans une première expérience, nous avons essayé d'optimiser le *Makespan*, le coût, la fiabilité du *workflow* hybride sous la contrainte de délai (*Makespan* < 450). Dans une deuxième expérience, nous avons optimisé les mêmes métriques de QoS sous la contrainte de budget (Coût < 3). Dans une dernière expérience, nous avons optimisé le *makespan*, le coût, la disponibilité du *workflow* hybride et nous avons fixé une contrainte sur la disponibilité minimale des VM (Disponibilité < 50%). Les différents résultats obtenus sont illustrés sur la figure 3.25 pour les deux premières expériences et sur la figure 3.26 pour la troisième expérience.

Comme on peut le voir sur les figures 3.25 et 3.26, l'algorithme NSGA-II est capable de trouver, à chaque fois, un ensemble de solutions non dominées respectant les contraintes exigées par l'utilisateur. L'algorithme trouve pour chaque expérience peu de solutions non dominées, vue que l'espace de recherche est réduit par la contrainte exigée.

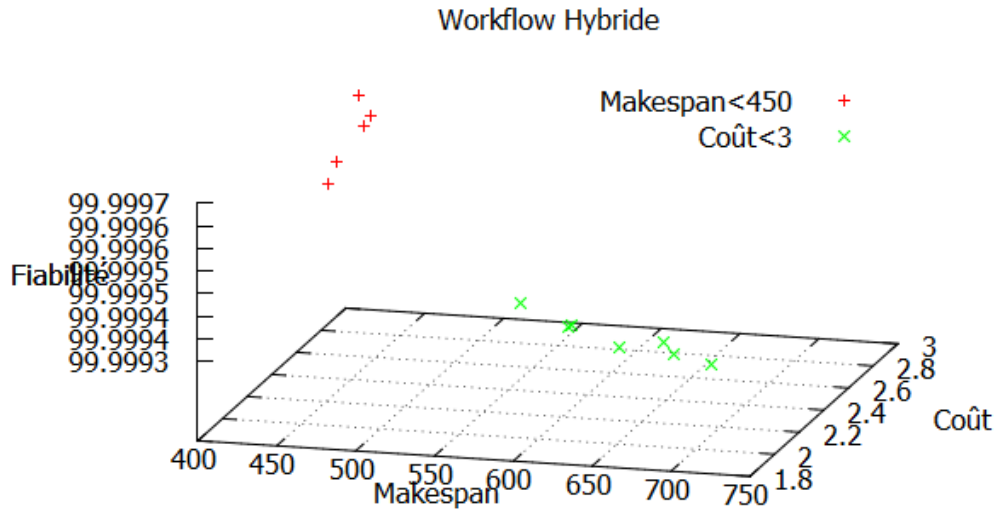


Figure 3.25. Solutions non dominées obtenues par NSGA-II pour le *workflow* hybride sous contrainte de budget ou de délai.

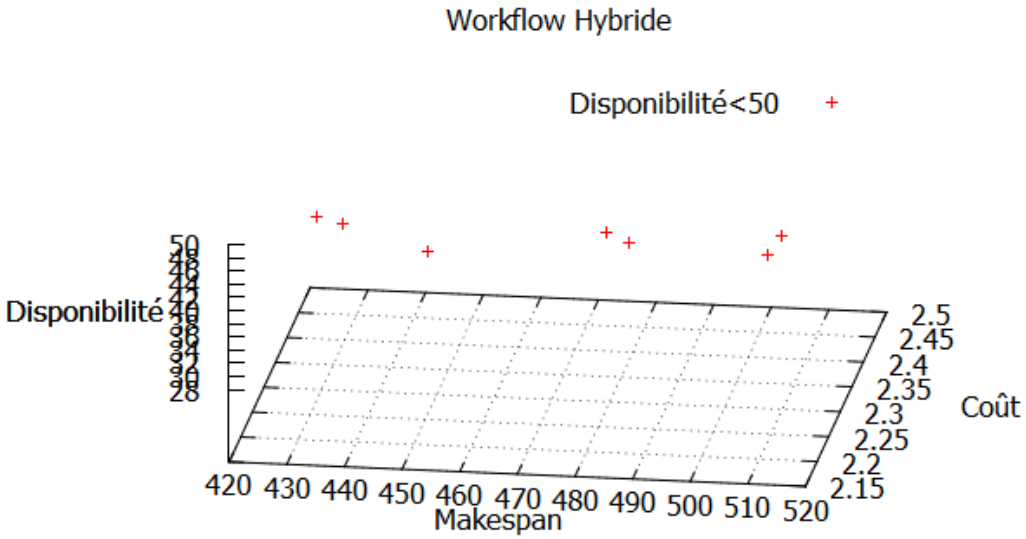


Figure 3.26. Solutions non dominées obtenues par NSGA-II pour le *workflow* hybride sous contrainte de disponibilité

- **Résultats obtenus par PAES**

Les résultats obtenus en appliquant l'algorithme PAES, respectivement, au *workflow* hybride et au *workflow* parallèle sur trois VM sont montrés dans la figure 3.27 et la figure

3.28. Les métriques de QoS considérées sont le *makespan*, la fiabilité et le coût. Ces figurent illustrent aussi les résultats obtenus par NSGA-II (représenté par le signe d'addition en couleur rouge), pour avoir une idée sur la qualité de deux ensembles trouvés de solutions.

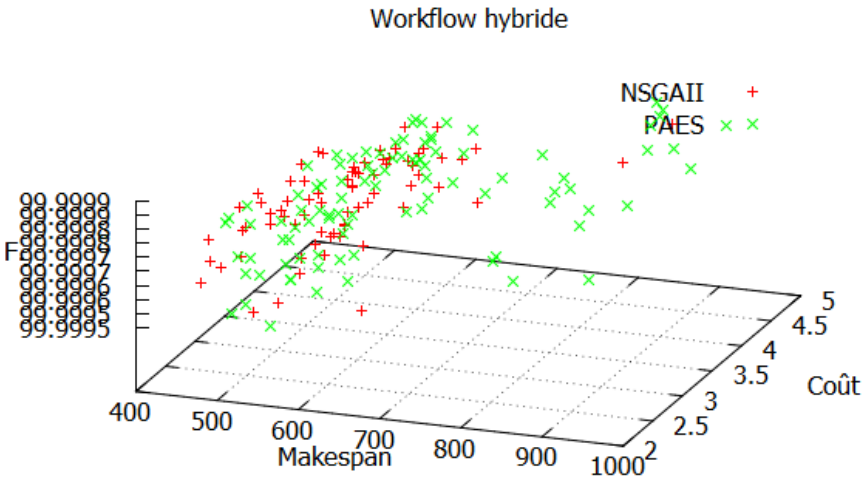


Figure 3.27. Solutions non dominées obtenues par PAES et NSGA-II pour le *workflow* hybride

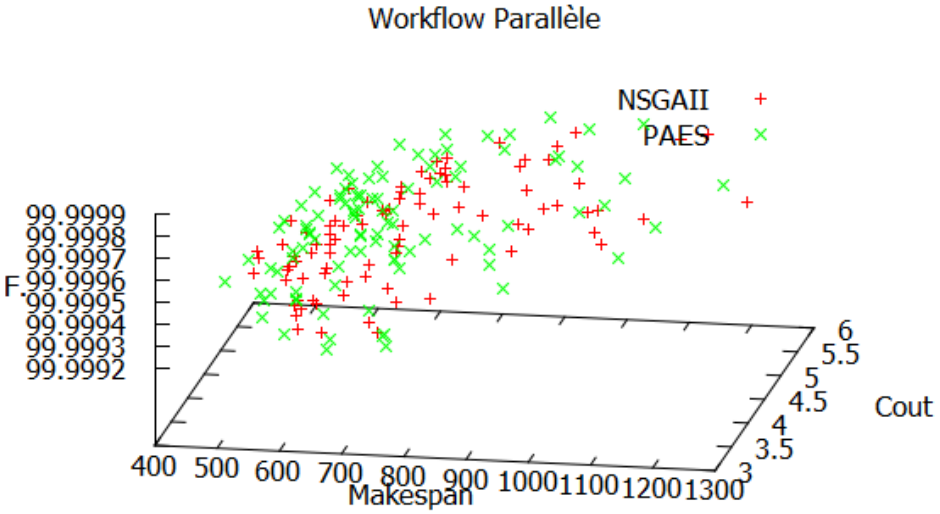


Figure 3.28. Solutions non dominées obtenues par PAES et NSGA-II pour le *workflow* parallèle

Sur la figure 3.27, nous observons que NSGA-II donne un ensemble de solutions non dominées qui est meilleur que celui généré par PAES pour le *workflow* hybride. Sur la figure 3.28, NSGA-II semble aussi meilleur que PAES pour le *workflow* parallèle. Afin de confirmer ces constats, nous avons utilisé la métrique C introduite par Zitzler (Zitzler, 1999) pour comparer, à chaque fois, les deux ensembles.

- **La métrique C**

La métrique $C(A,B)$est une métrique relative comparant deux surfaces de compromis A et B. La valeur $C(A,B)$ correspond au pourcentage d'éléments de B faiblement dominés par au moins un des éléments de A. Le ratio se calcule comme indiqué à l'équation 3.13.

Soient A, B deux ensembles de vecteurs de décisions. La fonction C associe à chaque paire ordonnée (A, B) une valeur de l'intervalle [0; 1]:

$$C(A,B) := \frac{|b \in B | \exists a \in A : a \geq b\}|}{|B|} \tag{3.13}$$

La valeur $C(A,B) = 1$ signifie que tous les vecteurs de décisions de B sont faiblement dominés par ceux de A. A l'inverse, $C(A,B) = 1$ signifie qu'aucun des points de B n'est faiblement dominé par un point de A.

Ainsi, plus la valeur $C(A,B)$ se rapproche de 1, meilleure la surface de compromis A est considérée par rapport à B. La métrique C n'étant pas symétrique: $C(A,B) \neq C(B,A)$, ainsi, il serait nécessaire de calculer les deux valeurs $C(A,B)$ et $C(B,A)$ afin d'obtenir une mesure plus fiable des deux surfaces de compromis A et B à comparer.

Les résultats de la métrique C des ensembles de solutions non dominées trouvés par NSGA-II est PAES pour le *workflow* hybride et le *workflow* parallèle sont présentés dans le tableau 3.11. Le résultat représenté dans chaque case du tableau correspond à la comparaison de l'algorithme en ligne avec l'algorithme en colonne.

Workflow	*Workflow* Hybride		*Workflow* Parallèle	
Algorithme	NSGA-II	PAES	NSGA-II	PAES
NSGA-II	-	0,83	-	0,7
PAES	0,29	-	0.19	-

Tableau 3.11. Résultats obtenus par NSGA-II et PAES évalué par la mesure *C*

En effet, les résultats du tableau 3.11 affirment la supériorité de NSGA-II par rapport à PAES pour l'ordonnancement des deux types de *workflows*.

3.6. Discussion

Dans cette section, nous avons présenté deux algorithmes d'optimisation par l'approche Pareto, à savoir: NSGA-II et PAES pour l'ordonnancement multiobjectif de *workflows* dans l'infrastructure IaaS. Ces algorithmes sont facilement adaptables au problème d'ordonnancement de *workflows* vu le faible nombre de paramètres à régler.

Les résultats des expériences montrent que les deux algorithmes permettent de générer de bons ensembles de solutions de compromis entre les différentes métriques de QoS considérées. Toutefois, les résultats montrent que NSGA-II est meilleur que PAES pour les deux types de *workflows* étudiés. Les résultats montrent aussi que NSGA-II est efficace dans la génération des solutions, respectant les contraintes de QoS exigées par l'utilisateur (délai, budget et autres). La difficulté de ces algorithmes est le choix final de la solution Pareto optimale, parmi toutes les solutions présentées.

3.7. Comparaison des différents algorithmes

Pour terminer, nous tentons, dans cette section, de comparer tous les algorithmes d'ordonnancement de *workflow* dans l'infrastructure IaaS présentés dans ce chapitre.

Plusieurs expériences ont été effectuées afin d'évaluer les vitesses de convergence des différents algorithmes présentés dans ce chapitre, notamment: l'algorithme génétique simple (GA), l'algorithme génétique amélioré (GA*), l'algorithme NSGA-II, l'algorithme PAES et l'algorithme HEFT (Topcuoglu, 2002).

Nous avons appliqué ces cinq algorithmes pour résoudre le problème d'ordonnancement du *workflow* hybride et du *workflow* parallèle sur 3 VM. Nous avons considéré quatre métriques de QoS (*makespan*, coût, fiabilité et disponibilité) et quatre contraintes fortes (contraintes de type $tâche_i/VM_j$). Les sorties des algorithmes ont été évaluées entre 0 et 1, par le calcul de la valeur moyenne des métriques de QoS normalisées, pour chaque algorithme appliqué aux deux instances de *workflows*. Une valeur de 1 représente la meilleure valeur possible (c.-à-d. toutes les QoS sont proches de leurs meilleures valeur obtenues, en les

optimisant individuellement) et 0 représente la pire valeur. Les résultats ont été évalués après chaque itération des algorithmes. La figure 3.29 montre la progression de la valeur de *fitness* de chaque algorithme, en fonction du nombre d'itérations, et la figure 3.30 illustre les résultats en fonction du temps.

Comme NSGA-II et PAES génèrent plusieurs solutions d'ordonnancement non dominées, nous devons choisir une seule solution pour les comparaisons. A cet effet, nous sélectionnons, pour chaque algorithme, la solution la plus proche, au sens de distance euclidienne, à la solution de GA*. La figure 3.31 montre une représentation graphique de la ou des solution(s) trouvée(s) par chaque métaheuristique d'ordonnancement du *workflow* hybride sur 3 VMs, optimisant trois métriques de QoS (*makespan*, coût, et fiabilité). La figure 3.32 montre les résultats obtenus pour le *workflow* parallèle.

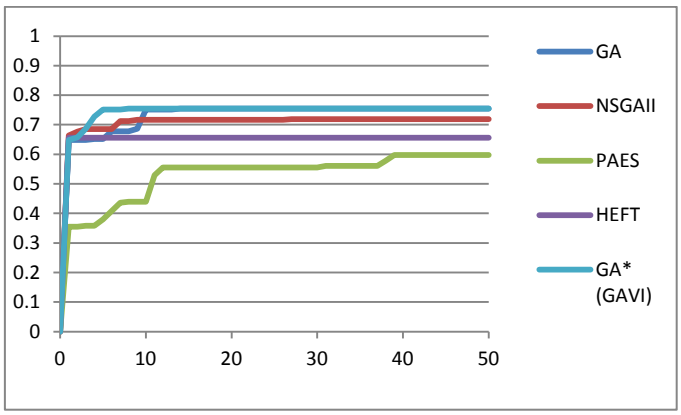


Figure 3.29. Progression de la fonction de *fitness* en fonction du nombre d'itérations

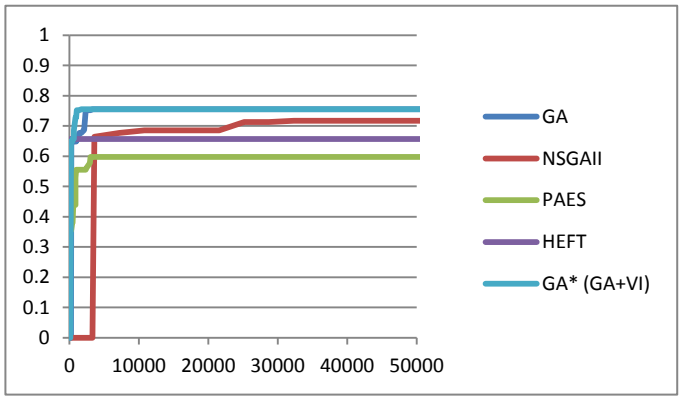


Figure 3.30. Progression de la fonction de *fitness* en fonction du temps

Nous constatons, sur ces figures , que les deux algorithmes GA et GA * sont meilleurs que les autres algorithmes. De plus, GA* est plus rapide à converger que GA.

Même s'il est vrai que HEFT donne un résultat dans les plus brefs délais et que PAES itère rapidement, le résultat en termes de valeur de *fitness* est en faveur de GA*, GA et NSGA-II.

Le cas de NSGA-II est différent. Bien que cet algorithme donne des solutions souvent non dominées avec celles de GA et GA*, cet algorithme reste lent. Le temps nécessaire pour calculer le front Pareto après chaque itération est extrêmement long, en particulier lorsque le nombre de métriques de QoS est grand ou que la taille de la population est importante.

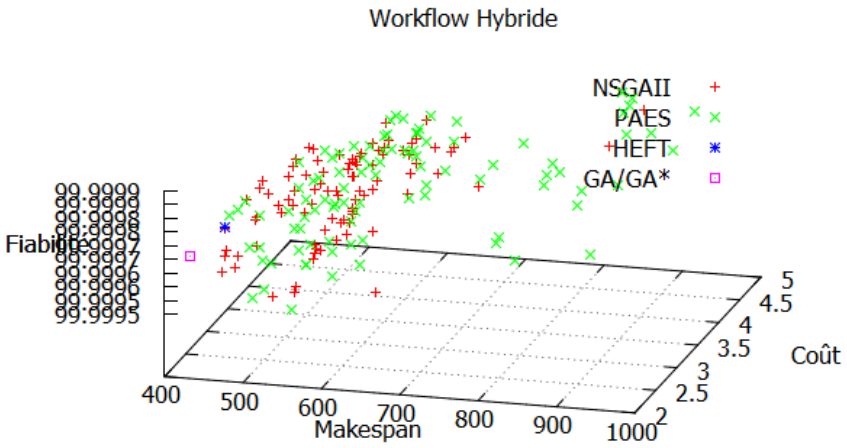


Figure 3.31. Résultats obtenus pour le *workflow* hybride

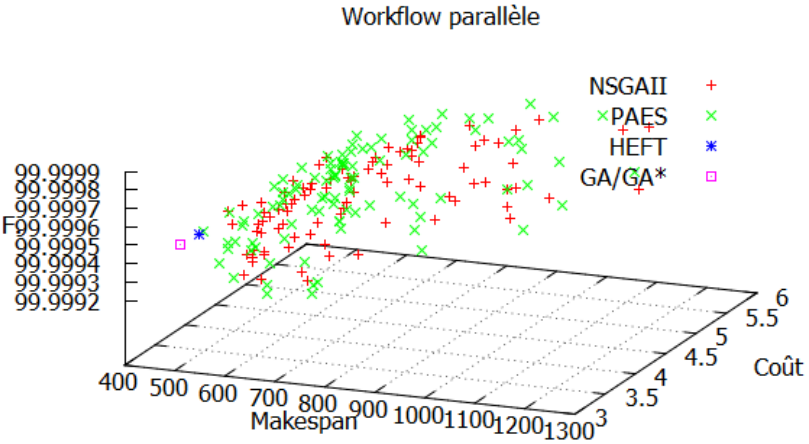


Figure 3.32. Résultats obtenus pour le *workflow* parallèle

- **Comparaison des différents algorithmes lors du redimensionnement du nombre de VM.**

Dans cette dernière expérience, nous avons essayé d'optimiser le *makespan*, le coût, la fiabilité et la disponibilité, lors du redimensionnement du nombre de ressources.

Le tableau 3.12 présente les résultats obtenus, en appliquant HEFT, GA* et NSGA-II sur un *workflow* composé de 194 tâches, respectant la structure de la FFT (*Fast Fourier Transform*) dont certaines données sont générées aléatoirement. Le nombre de VM utilisé est de 10, 20 et 30.

Nbre. VM	Algo.	*Makespan* (s)	Coût ($)	Fiabilité	Disponibilité (%)
10	HEFT	194,978	11,0826	99,999402	19,63
	GA*	189,005	4,34939	99,999404	17,41
	NSGA-II	257,593	9,38684	99,999496	36,00
20	HEFT	127,186	4,25881	99,999354	39,72
	GA*	116,191	4,15135	99,999321	35,68
	NSGA-II	169,324	7,70379	99,999534	50,11
30	HEFT	96,3181	6,69343	99,999383	45,67
	GA*	95,4597	3,9005	99,999458	50,96
	NSGA-II	145,61	4,52974	99,999465	62,66

Tableau 3.12. Résultats des différents algorithmes pour une grande instance de *workflow* et une IaaS scalable

D’après le tableau 3.12, les résultats tendent à confirmer ce que nous avions déjà observé dans les figures 3.29 et 3.30, et cela reste vrai même avec la scalabilité des ressources dans l’infrastructure IaaS: l'algorithme NSGA-II est significativement plus lent à donner des solutions qui sont de mêmes qualités que celle de GA*. On constate aussi que GA* donne une meilleure solution que l'algorithme HEFT, non seulement en terme du coût, de la fiabilité et de la disponibilité, mais aussi en terme du *makespan* pour lequel l'algorithme HEFT est censé donner un bon résultat.

3.8. Conclusion

Dans ce chapitre, nous avons traité le problème d'optimisation multiobjectif appliqué à l'ordonnancement de *workflows* basé QoS, dans les infrastructures virtuelles (IaaS) de *cloud computing.* Nous avons commencé par présenter notre modélisation du problème d'ordonnancement de *workflows*, avant de présenter les métaheuristiques développées pour sa résolution.

Concernant la partie modélisation, nous avons présenté notre modélisation du *cloud computing* sous forme de graphe non orienté, et le *workflow* sous forme d'un graphe orienté acyclique. Nous avons donné les formules mathématiques des différentes métriques de QoS considérées, notamment, le *makespan*, le coût, la fiabilité et la disponibilité. Ces métriques de QoS sont conflictuelles, ce qui nous a amené à formuler le problème d'ordonnancement de *workflows* comme un problème d'optimisation multiobjectif, nécessitant l'utilisation des métaheuristiques pour sa résolution.

Dans la partie résolution, nous avons opté pour deux approches de résolutions, à savoir l'approche par agrégation et l'approche Pareto.

Dans l'approche par agrégation, nous avons transformé le problème d'ordonnancement multiobjectif en un problème simple objectif, en agrégeant les différentes métriques de QoS avec des poids (préférences) renseignés à priori par l'utilisateur (ou un expert). Dans cette approche, nous avons présenté :

- L'algorithme génétique (GA) pour optimiser uniquement les différentes métriques de QoS considérées. Nous avons présenté notre codage d'une solution au problème d'ordonnancement de *workflow*, plusieurs types de croisements et une nouvelle fonction de *fitness générique* capable de traiter plusieurs QoS.
- L'algorithme génétique basé Infection virale (GA*) qui vise à optimiser les différentes métriques de QoS et traiter les différentes contraintes exigées, à savoir : les *contraintes QoS* telles que le deadline, le budget et autres contraintes négociées entre le fournisseur et le client; les *contraintes fortes* qui forcent d'exécuter certaines tâches sur certaines ressources spécifiques et enfin les *contraintes de précédences entres* les tâches. Dans cet algorithme, nous avons ajouté une nouvelle méthode de pénalité à la fonction de *fitness*, afin de respecter les différentes *contraintes QoS*. En outre, nous avons adapté l'opérateur

d’infection virale afin de résoudre efficacement les contraintes fortes liées aux exigences tâche/VM.

Dans l’approche Pareto, nous avons traité le problème d’ordonnancement multiobjectif, en utilisant des méthodes fondées sur la dominance au sens de Pareto, privilégiant ainsi une recherche qui satisfait au mieux toutes les métriques de QoS. Nous avons présenté dans cette approche :

- L’algorithme NSGA-II qui est une métaheuristiques basée population et qui vise à générer un ensemble de solutions de compromis selon les exigences de QoS de l’utilisateur.

- L’algorithme PAES qui est une métaheuristique basée sur une solution unique, utilisant une archive externe pour maintenir l’ensemble des solutions nom dominées.

Les différents algorithmes ont été évalués par simulation, en utilisant des *workflows* synthétiques et des *workflows* scientifiques issus du monde réel et aussi les caractéristiques de ressources du *cloud* Amazon EC2. Les résultats montrent que nos métaheuristiques sont efficaces dans la résolution du problème d’ordonnancement de *workflows*, en prenant en compte plusieurs métriques de QoS. Elles sont toutes meilleures que l’heuristique HEFT et que GA* surpasse les autres métaheuristiques en terme du temps CPU.

Chapitre 4

Métaheuristique à base de PSO pour l'ordonnancement de *workflow* au niveau HaaS

4.1. Introduction

Depuis quelques années, le critère énergétique tend à devenir une métrique importante dans l'exploitation du *cloud computing.* Afin de maitriser la consommation énergétique, nous abordons cette problématique via l'étude du problème d'ordonnancement de *workflows* au niveau HaaS (Hardware as a Service) du *cloud*. L'ordonnancement vise, d'un côté à satisfaire les exigences de QoS de l'utilisateur, en minimisant le coût et le temps total d'exécution du *workflow*, selon les spécifications du SLA; et de l'autre côté, à maximiser le profit du fournisseur, tout en minimisant la consommation énergétique des ressources. Ces objectifs sont conflictuels, il n'est pas possible de réduire tous les objectifs simultanément. Ainsi trouver un bon compromis entre l'économie d'énergie et les QoS d'exécution de *workflows* est un défi majeur.

Pour répondre à ce défi, nous proposons un nouvel algorithme multiobjectif basé sur l'optimisation par essaim particulaire, nommé MODPSO (*Multi-objective Discrete Particle Swarm Optimization*), combiné avec la technique d'adaptation dynamique de la tension et de la fréquence, DVFS (Yassa, 2012b ; Yassa, 2013b). L'algorithme permet de générer un ensemble de solutions de compromis entre les performances du système et la consommation d'énergie. L'approche proposée est évaluée par simulation, sur un ensemble d'applications

synthétiques et réelles. Les résultats de simulations ont montré que cet algorithme améliore considérablement les performances de quelques approches existantes.

Ce chapitre est organisé comme suit: Dans la section 4.2, nous reformalisons, via des modèle mieux adaptés à ce contexte, le problème d'ordonnancement de *workflow* (modélisation du *cloud*, du *workflow*, des métriques de QoS et enfin de l'ordonnancement multiobjectif). Dans la section 4.3, nous détaillons notre métaheuristique d'ordonnancement, nommée DVFS-MODPSO. La section 4.4, présente une évaluation expérimentale des performances de l'approche proposée. Nous concluons ce chapitre, dans la section 4.5.

4.2. Modélisation du problème d'ordonnancement de *workflow*s au niveau HaaS

4.2.1. Modèle du *cloud computing*

Le *cloud computing* utilisé dans ce chapitre, est considéré comme un ensemble de ressources offert par un fournisseur de *cloud*, pour exécuter des applications clientes. Notre modèle de *cloud* est inspiré du modèle décrit dans (Mezmaz, 2010). Nous supposons que le *cloud* est hébergé dans des *Datacenter*s composés par des machines hétérogènes. Ces *Datacenters* délivrent un ensemble de services hébergés sur plusieurs serveurs informatiques, qui sont rendus disponibles à la souscription, via un modèle de paiement à l'usage. Dans notre modèle, le niveau HaaS de l'infrastructure du *cloud computing* comporte un ensemble de $P= \{p_1, p_2, ..., p_m\}$ processeurs hétérogènes, qui sont totalement interconnectés. Les processeurs ont des capacités de traitement variées et sont fournis à des prix de traitement différents (voir *ec.* du tableau 4.1). Chaque processeur $p_j \in P$ est basé DVFS, c'est-à-dire, il peut fonctionner avec différents VSL (c.-à-d. des fréquences d'horloge différentes). Pour chaque processeur $p_j \in P$, on lui attribue aléatoirement un ensemble V_j de VSL, parmi les trois différentes séries de VSL présentées dans le tableau 4.1. On considère que les processeurs consomment de l'énergie même s'ils sont inactifs, et on suppose que la tension la plus faible est utilisée dans ce cas (Lee, 2011).

En outre, chaque processeur $p_j \in P$ dispose d'un ensemble de liens $Lp_j= \{ljp_1, ljp_2, ..., ljp_k\}$, $1 \leq k \leq m$; où $ljp_k \in R^+$ est la bande passante disponible (en Mbps) dans le lien entre les processeurs p_j et p_k, avec $ljp_j = 1$. On suppose qu'un message peut être transmis d'un

processeur à l'autre lorsqu'une tâche s'exécute sur le processeur destinataire; ce qui est possible dans de nombreux systèmes. Enfin, les communications entre les tâches s'exécutant sur le même processeur sont négligées. Le tableau 4.1 montre les niveaux de VSLs pour les trois catégories de processeurs : TURION MT-34, OMAP et PENTIUM M.

VSL.	P1: TURION MT-34			P2: OMAP			P3: PENTIUM M		
	Volt. (V)	Freq. (Ghz)	R.Speed (%)	Volt. (V)	Freq. (Ghz)	R.Speed (%)	Volt. (V)	Freq. (Ghz)	R.Speed (%)
1	1,2	1,8	100	1,35	0,6	100	1,48	1,4	100
2	1,15	1,6	88,88	1,27	0,55	91,66	1,44	1,2	96,76
3	1,1	1,4	77,77	1,2	0,5	83,33	1,31	1	88,14
4	1,05	1,2	66,66	1	0,25	41,66	1,18	0,8	79,51
5	1	1	55,555	0,9	0,13	20,83	0,96	0,6	64,42
6	0,9	0,8	44,44						
Prix	*ec*= 1,14			*ec*= 0,57			*ec*= 0,76		

Tableau 4.1. Différents niveaux de tensions, fréquences et vitesses relatives des processeurs

4.2.2. Modèle du *workflow*

Nous reprenons la modélisation décrite dans la section 3.2.2 du chapitre précédent, où l'on modélise un *workflow* sous forme d'un DAG noté, G (V, E).

Comme notre algorithme nécessite une seule tâche d'entrée et une tâche de sortie, nous ajoutons deux tâches fictives T_{entry} et T_{exit} respectivement, au début et à la fin du *workflow*. Le temps d'exécution de ces tâches fictives est nul et elles sont reliées aux tâches d'entrée et de sortie avec des arcs ayant des poids nuls.

Nous supposons que, chaque tâche $T_i \in T$ a un temps d'exécution de base, qui est une valeur indépendante pour chaque machine. On note w_{ij}, le temps de calcul basique d'une tâche T_i sur une ressource de calcul p_j, lorsque cette dernière tourne à une vitesse maximale (ça correspond au niveau 1 dans le tableau 4.1). Le temps d'exécution moyen de la tâche T_i est défini comme suit:

$$w_i = \sum_{j=1}^{m} \frac{w_{ij}}{m} \tag{4.1}$$

Le temps de calcul réel wr_{ijk} de la tâche T_i sur la machine p_j, utilisant la vitesse d'exécution relative s_k est défini comme suit:

$$wr_{ijk} = \frac{w_{ij}}{s_k} \tag{4.2}$$

Nous supposons également qu'on associe à chaque arc (T_i, T_j) $\in$E une valeur tr_{ij} représentant le temps nécessaire pour transférer des données de T_i vers T_j. Le temps de transfert peut être calculé, en fonction de la bande passante l_{mn} entre les ressources respectivement p_m et p_n exécutant ces tâches de la manière suivante :

$$tr_{ij} = \frac{d_{ij}}{l_{mn}} \tag{4.3}$$

Cependant, le temps de transfert n'est nécessaire que lorsque deux tâches sont affectées à des processeurs différents. Ceci dit, le temps de transfert lorsque les tâches sont affectées à un même processeur peut être considéré égal à zéro.

Nous définissons *pred(T_i)* comme l'ensemble de tous les prédécesseurs de T_i et *succ(T_i)* l'ensemble de tous les successeurs de T_i.

La date de début d'exécution au plus tôt (*Earliest Start Time*) et la date de fin d'exécution (*Earliest finish time*) d'une tâche T_i sur un processeur p_j sont, respectivement, représentées par *EST(T_i, p_j)* et *EFT(T_i, p_j)* (équations 4.4 et 4.5). *EST(*T_i*)* et *EFT*(T_i*)* représentent, respectivement, les dates de début et de fin sur n'importe quel processeur.

$$EST(T_i, p_j) = \begin{cases} 0\,, & si\ T_i = T_{entry} \\ \max\left\{P_{av\,(T_i, p_j)}\,,\ \max_{T_j \in pred(T_i)}\left(EFT(T_j, p_k) + tr_{ji}\right)\right\}, & \text{sinon} \end{cases} \tag{4.4}$$

Tel que,

$P_{av\,(T_i, p_j)}$ définit la date de début au plus tôt où le processeur p_j sera disponible pour commencer l'exécution de la tâche T_i.

$$EFT(T_i, p_j) = w_{ij} + EST(T_i, p_j) \tag{4.5}$$

On note que la date de début d'exécution actuelle, *AST(T_i, p_j)* et la date de fin d'exécution actuelle *AFT(T_i, p_j)* de la tâche T_i sur un processeur p_j, peuvent être, respectivement, différentes des dates de début *EST(T_i, p_j)* et de fin *EFT(T_i, p_j)*, si la date *EFT(T_k, p_j)*, d'une autre tâche T_k planifiée sur le même processeur est plus lente que sa date *EST(T_k, p_j)* (Lee, 2011).

La figure 4.1 illustre une application sous forme d'un *workflow* composé de 10 tâches présentée dans (Topcuoglu, 2002). Les numéros de tâches T_i sont à l'intérieur des nœuds et les valeurs de d_{ij} sont à côté des arêtes correspondantes. Le tableau 4.2 présente les détails de ce *workflow* (Topcuoglu, 2002). Les valeurs présentées dans la dernière colonne du tableau représentent les priorités des tâches. La priorité d'une tâche T_i, représentée par $Pr(T_i)$, est

calculée par récursivité, en traversant le DAG à partir de la tâche de sortie T_{exit} vers le haut comme suit (équation 4.6):

$$Pr(T_i) = \begin{cases} \overline{w}_{T_{exit}}, & si\ T_i = T_{exit} \\ \overline{w}_i + \max\limits_{T_j \in succ(T_i)} \{\overline{Tr}_{ij} + Pr(T_j)\}, & \text{sinon} \end{cases} \quad (4.6)$$

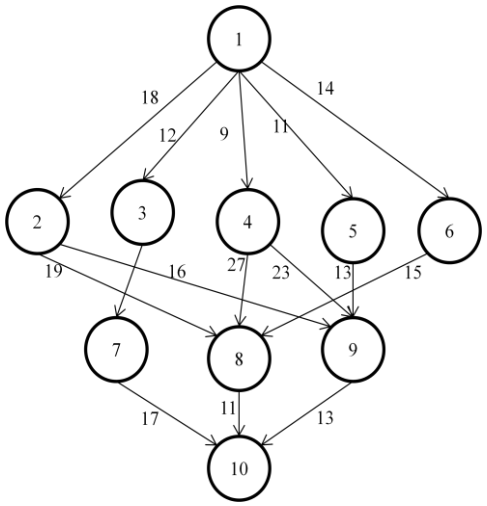


Figure 4.1. Un exemple de *workflow*

Tâche(T_i)	p_1	p_2	p_3	$\overline{\omega}_i$	Priorité($Pr(T_i)$)
1	14	16	9	13	108,000
2	13	19	18	16.67	77,000
3	11	13	19	14.33	80,000
4	13	8	17	12.67	80,000
5	12	13	10	11.67	69,000
6	13	16	9	12.67	63,333
7	07	15	11	11	42,667
8	5	11	14	10	35,667
9	18	12	20	16.67	44,333
10	21	7	16	14.67	14,667

Tableau 4.2. Les temps d'exécution des tâches et leurs priorités

4.2.3. Caractérisation des métriques de QoS

4.2.3.1. Modèle énergétique

Parmi les principales techniques d'économie de l'énergie au niveau système, nous trouvons la technique d'adaptation dynamique de la tension et fréquence, nommée DVFS (*Dynamic Voltage and Frequency Scaling).* La technique DVFS permet de changer la fréquence et la tension d'alimentation de la CPU, influençant ainsi la consommation énergétique, tout en ayant un impact sur le temps d'exécution.

Dans cette thèse, nous utilisons un modèle d'énergie dérivé du modèle de consommation dans un circuit CMOS (*Complementary Metal-Oxide Semi-conductor*) (Lee, 2011). Il existe deux sources principales de consommation dans un circuit CMOS. La puissance dynamique (une puissance est dissipée à chaque fois que le circuit est sollicité), et la puissance statique (une puissance est dissipée lorsque le circuit est dans un état stable). La puissance totale dissipée par un processeur est dominée par la puissance dynamique, soit environ 90%. La puissance dynamique $P_{dynamic}$ est donnée par l'équation (4.7).

$$P_{dynamic} = AC_{ef}v^2f \tag{4.7}$$

Avec,

A : le facteur d'activité défini comme le nombre moyen de transition par cycle d'horloge.
C_{ef}: la capacité effective
v : la tension d'alimentation
f : la fréquence d'horloge

L'équation 4.7 montre que la tension d'alimentation est le facteur dominant, par conséquent sa réduction serait le plus influent à la réduction de la consommation énergétique.

Nous définissons l'énergie consommée E_c durant l'exécution des différentes tâches du *workflow* comme suit:

$$E_c = \sum_{i=1}^{n} AC_{ef}\ v_i^2\ f_i\ \ wr_i^* \ = \ \sum_{i=1}^{n} \gamma\ v_i^2\ f_i\ \ wr_i^* \tag{4.8}$$

tel que:

$\gamma = AC_{ef}$: constante pour une machine donnée.

v_i , f_i : tension d'alimentation et fréquence du processeur sur lequel la tâches T_i est exécutée, respectivement.

wr_i^*: le temps de calcul réel de la tâche T_i sur le processeur qui lui est affecté.

En périodes d'inactivités du processeur, ce dernier passe en mode de veille. La tension d'alimentation et la fréquence relative sont au niveau le plus bas. Ainsi, nous définissons l'énergie consommée pendant les périodes d'inactivités des processeurs E_i comme suit:

$$E_i = \sum_{j=1}^{m} \ \sum_{idle_{jk} \in IDLE_j} \gamma\ v_{minj}^2\ f_{minj}\ \ L_{jk} \tag{4.9}$$

où**:**

$IDLE_j$: l'ensemble de toutes les périodes d'inactivités sur la machine p_j.

v_{minj} (f_{minj}) : la plus basse tension d'alimentation (fréquence) de la machine p_j.

L_{jk}: la durée d'une période d'inactivité $idle_{jk}$ sur la machine p_j.

Au final, notre modèle d'énergie consommée par les ressources du *cloud* pour l'achèvement du *workflow* est donné par l'équation 4.10 :

$$E_{total} = E_c + E_i \tag{4.10}$$

4.2.3.2. *Makespan*

Comme précisé dans la section 3.2.3.1 du chapitre précédent, le *makespan* est la différence entre la date de soumission du *workflow* et la date de réception des résultats. Dans, ce chapitre, le *makespan* correspond à la date de fin d'exécution actuelle de la tâche de sortie T_{exit} du *workflow,* comme indiqué dans l'équation 4.11.

$$T_{total} = AFT(T_{exit}) \tag{4.11}$$

4.2.3.3. Coût

Le coût d'exécution du *workflow* dans HaaS est calculé de la même façon que dans le chapitre précédent. Nous tenons à préciser que, dans ce cas, le coût d'exécution d'une tâche donnée T_i dépend de la date de fin d'exécution réelle wr_i^* de T_i sur la machine qui lui est allouée et du prix de cette machine.

$$C_{total} = \sum_{i=1}^{n} wr_i^* * ec_i + \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} * trc_{ij} \tag{4.12}$$

4.2.4. Modèle d'ordonnancement

Étant donné (1) un fournisseur d'une infrastructure HaaS, offrant un ensemble P de m processeurs hétérogènes, et (2) un *workflow* d'un utilisateur composé d'un ensemble T de n tâches qui doivent être exécutées sur ces processeurs. Le problème d'ordonnancement du *workflow* sur le HaaS est de construire un *mapping* M des tâches aux processeurs (sans violer les contraintes de précédence), qui minimise les objectifs conflictuels suivants: le *makespan*, les coûts et la consommation d'énergie. Par conséquent, le problème d'ordonnancement de *workflows* peut être formulé comme le problème d'optimisation multiobjectif suivant:

Makespan: $Minimiser\ T_{total}(M)$ (4.13)

Coût: $Minimiser\ C_{total}(M)$ (4.14)

Energie: $Minimiser\ E_{total}(M)$ (4.15)

4.3. DVFS-MODPSO pour l'ordonnancement de *workflow*s basé-QoS

La conception originale du PSO est appropriée pour trouver des solutions aux problèmes d'optimisation à variables continues. Cependant, comme l'ordonnancement de *workflows* décrit dans ce document est un problème qui est, à la fois, de nature discret et multiobjectif, nous proposons une approche efficace pour résoudre ce problème. Nous présentons une version discrète et multiobjectif du PSO standard (voir section 2.2.4.4.2 du chapitre 2) combinée avec la technique DVFS, nommée DVFS-MODPSO. Les enjeux clés de notre approche consistent en:

- La définition de la position et de la vitesse des particules, en fonction des caractéristiques des variables discrètes du problème d'ordonnancement de *workflows*.
- La résolution de l'aspect multiobjectif du problème, en modifiant le PSO standard afin de générer un ensemble de solutions non-dominées répondant, ainsi aux différents objectifs considérés au lieu de générer une seule solution.

4.3.1. Représentation d'une position de particule

Une position d'une particule représente une solution réalisable du problème d'ordonnancement de *workflows*. Nous représentons une position par un ensemble ordonné de n triplets $M= [<\text{tâche } (T_i), \text{ressource } (P_j), VSL(L_k)>]$ $(i \in [1,n], j \in [1,m], k \in [1,L(j)])$, où *n* est le nombre de tâches composant le *workflow*, *m* est le nombre de processeurs disponibles dans l'environnement du *cloud* et *L(j)* représente le nombre de points de fonctionnement (VSL, Voltage Scaling Level) du processeur P_j. Chaque triplet signifie qu'une tâche T_i est affectée à un processeur P_j, qui fonctionne au niveau de tension L_k. La position d'une particule satisfait la contrainte de précédence entre les tâches du *workflow*. La figure 4.2 illustre un exemple d'une position sous forme d'un tableau, représentant une solution du problème d'ordonnancement d'un *workflow* composé de 5 tâches sur un *cloud* composé de 3 ressources, ayant chacune plusieurs niveaux de VSL.

T_1	T_2	T_3	T_4	T_5
P_1	P_2	P_1	P_3	P_3
L_1	L_2	L_3	L_1	L_1

Figure 4.2. Exemple de position d'une particule.

4.3.2. Mise à jour de la vitesse et de la position

Le processus de génération d'une nouvelle position pour une particule donnée, dans l'essaim est décrit dans les formules suivantes:

$$V_i^{k+1} = V_i^k \oplus \left(\left(c_1 R_1 \otimes \left(\, pBest_i \ominus X_i^k\right)\right) \oplus \left(c_2 R_2 \otimes \left(gBest \ominus X_i^k\right)\right)\right) \quad (4.16)$$

$$X_i^{k+1} = X_i^k \oplus V_i^{k+1} \quad (4.17)$$

Nous avons redéfini les différents opérateurs utilisés dans ces deux formules comme suit:

- **L'opérateur de soustraction (⊖):** la différence entre deux positions de particules, désignée par X1 ⊖ X2, est définie comme étant un ensemble de triplets, dans lequel chaque triplet <tâche, service, VSL> indique si les contenus des éléments correspondants dans X1 sont différents de ceux de X2. Si c'est le cas, ce triplet obtient ses valeurs (services et VSL) à partir de la position qui a la plus faible valeur du VSL. Pour les triplets qui ont le même contenu en X1 et X2, leurs VSL correspondants seront diminués. On notera que la variation du VSL permet des fluctuations sur l'énergie, le *makespan* et le coût. La figure 4.3 montre un exemple de fonctionnement de l'opérateur de soustraction de 2 positions X1 et X2.

X1

T_1	T_2	T_3	T_4	T_5
P_1	P_2	P_1	P_3	P_3
L_1	L_2	L_3	L_1	L_1

X2

T_1	T_2	T_3	T_4	T_5
P_1	P_2	P_2	P_1	P_2
L_1	L_2	L_1	L_1	L_3

X1⊖ X2

T_1	T_2	T_3	T_4	T_5
P_1	P_2	P_1	P_1	P_2
L_2	L_3	L_3	L_1	L_3

Figure 4.3. Exemple de fonctionnement de l'opérateur de soustraction

- **L'opérateur de multiplication (⊗):** la multiplication entre un nombre et une position est définie comme un ensemble de triplets, où: un seuil $\alpha \in [0, 1]$ est défini, un nombre aléatoire *R* est généré pour chaque triplet <tâche, service, VSL >; comparer $c*R$ et α: si $c*R \geq \alpha$, *alors* diminuer le VSL du triplet, sinon l'augmenter. Cet opérateur ajoute une capacité d'exploration à l'algorithme. La figure 4.4 illustre un exemple de fonctionnement de l'opérateur de multiplication.

$c = 2,\ \alpha = 0.5$

R

0.35	0.25	0.45	0.2	0.1

X

T_1	T_2	T_3	T_4	T_5
P_1	P_2	P_1	P_1	P_2
L_2	L_3	L_3	L_1	L_3

c*R⊗X

T_1	T_2	T_3	T_4	T_5
P_1	P_2	P_1	P_1	P_2
L_3	L_4	L_4	L_1	L_2

Figure 4.4. Exemple de fonctionnement de l'opérateur de multiplication.

- **L'opérateur d'addition (⊕):** l'addition de deux positions consiste à sélectionner la position dominante.

4.3.3. Etapes de l'algorithme DVFS-MODPSO

Le pseudo-code suivant décrit les étapes générales de l'algorithme DVFS-MODPSO.

L'algorithme commence par initialiser les positions et les vitesses des particules. Pour obtenir la position d'une particule, le VSL (tension et fréquence) de chaque ressource est initialisée de façon aléatoire, puis l'algorithme HEFT est appliqué pour générer une solution faisable, minimisant le temps d'exécution total. Ce processus est répété plusieurs fois afin d'initialiser les positions de toutes les particules de l'essaim.

Apres l'initialisation de l'essaim, les solutions sont évaluées et classées selon la profondeur de dominance, en utilisant les objectifs du *makespan*, du coût et de l'énergie. L'algorithme maintient une *Archive Externe EA*, permettant de préserver les meilleures solutions non-dominées. Après toutes ces initialisations, dans la boucle principale de l'algorithme, la nouvelle vitesse et la position de chaque particule sont calculées, après avoir sélectionné la meilleure position globale *gBest* dans l'archive externe, et éventuellement, après avoir effectué une mutation. La particule est évaluée et sa *pBest* correspondante est mise à jour. Après chaque itération, l'archive externe est mise à jour. Une fois la condition de terminaison atteinte, l'archive externe contenant le front de Pareto est retournée comme résultat final.

Algorithme 4.1 Ordonnancement *workflow* : DVFS-MODPSO

// Initialisation du Swarm avec HEFT

1. **Pour** *i = 1* à *SNum* (*SNum* est la taille de l'essaim de particules)
   a. **Pour** *j = 1* à *m* (*m* est le nombre de processeurs)
   - Initialiser aléatoirement *VSL(j)* (choisir aléatoirement la tension /fréquence du processeur à partir de l'ensemble de ses points de fonctionnement).
   b. **Fin pour**
   c. Initialiser *S[i]* en utilisant l'heuristique avec HEFT (*S* est l'essaim de particules)
   d. Initialiser la vitesse *V* de chaque particule
      i. *V [i] = S[i]*
   e. Initialiser la meilleure position personnelle (*pBest*) de chaque particule
      ii. *pBest [i] = S[i]*
   f. Évaluer les objectifs de chaque particule :
      iii. *Évaluer S[i]*
   g. Initialiser la meilleure particule globale (*gBest*) avec la meilleure parmi les *SNum* particules:
      iv. *gBest = Meilleure particule trouvée dans S*
2. **Fin Pour**
3. Ajouter les solutions non dominés trouvées dans *EA* (*EA* est l'archive externe stockant le front de Pareto)
4. Initialiser le nombre d'itération *t = 0*
5. *Répéter jusqu'à t > G* (*G* est le nombre maximum d'itérations)
   a. ***Pour*** *i = 1 à SNum* (la taille de l'essaim)
      i. Choisir aléatoirement la meilleure particule globale pour *S* dans Archives *EA* et stocker sa position dans *gBest*.
      ii. Calculer la nouvelle vitesse *V[i]* selon l'équation (4.16).
      iii. Calculer la nouvelle position de *S[i]* selon l'équation (4.17).
      iv. Si (*t < G * PMUT*) alors (PMUT est la probabilité de mutation)
      Effectuer une mutation sur *S[i]*.
      v. Évaluer *S[i]*
      vi. Mettre à jour la meilleure position de chaque particule *S[i]*:
      $\boldsymbol{Si}\ S[i] \preccurlyeq pBests[i] \vee (S[i] \sim pBests[i])\ Alors\ pBests[i] = S[i]$
   b. **Fin Pour**
   c. Mettre à jour l'archive externe *EA*:
   - Ajouter toute nouvelle solution non dominée de *S* dans *EA*
     $\boldsymbol{Si}\ S[i] \preccurlyeq x, \forall\ x \in EA\ \ Alors\ \ EA = EA \cup \{S[i]\}$
   - Retirez toutes les particules dominées par S[i] dans *EA*
     $EA = EA - \{y \in EA | y \prec S[i]\}$
   - ***Si*** l'archive est pleine alors choisir aléatoirement la particule à remplacer dans *EA*.
6. **Retourner** le front Pareto (l'ensemble des solutions non-dominées de *EA*)

4.4. Evaluation expérimentale

Dans cette section, nous présentons les expériences réalisées pour valider l'approche proposée et les résultats obtenus.

4.4.1. Paramètres expérimentaux

Les paramètres expérimentaux concernent, à la fois, les *workflows* utilisateurs et les ressources de *cloud*.

4.4.1.1. Paramètres du *workflow*

Pour nos expériences, nous utilisons les deux *workflows* présentés dans le chapitre précédent, à savoir: le *workflow* fMRI (Zhao, 2004) de la neuroscience, ayant une structure parallèle et le *workflow* d'annotation de protéines (O'Brien, 2004) développé par London-Centre-e-science, ayant une structure hybride. Nous utilisons aussi le *workflow* synthétique décrit dans (Topcuoglu, 2002) et qui est représenté dans la figure 4.1.

4.4.1.2. Paramètres du *cloud*

Dans nos expériences, nous avons utilisé trois catégories de ressources dont les caractéristiques sont décrites dans le tableau 4.1. Nous avons choisi les processeurs TURION MT-34, OMAP et PENTIUM M qui sont équipés de la technologie DVFS et qui ont des capacités de traitement variées. La topologie du système est telle que tous les services sont connectés les uns aux autres. Les bandes passantes réseaux disponibles entre ces services sont 50 Mbps et 100 Mbps. Le coût total que l'utilisateur doit payer pour l'exécution de son *workflow* est composé de deux parties : le coût de traitement et le coût de transmission de données. Nous avons choisi le modèle de prix donné par Amazon EC2 pour les coûts de traitements des différentes catégories de services, et le modèle de tarification donné par *Amazon cloudFront* pour les coûts de transferts des données entre les services. Nous avons effectué des expériences, en utilisant successivement 3, 4, 5 et 12 services.

4.4.1.3. Paramètres de l'algorithme DVFS-MODPSO

Les paramètres de notre algorithme DVFS-MODPSO sont résumés dans le tableau 4.3.

Paramètre	Valeur
Taille de l'essaim	50
Nombre d'itérations	100
Taille de l'archive	50
probabilité de mutation	0.05

Tableau 4.3. Paramètres de l'algorithme DVFS-MODPSO

4.4.2. Evaluation des performances

Nous évaluons les performances de notre algorithme, DVFS-MODPSO, sur toutes les instances de *workflows* décrites précédemment. En raison de l'absence de travaux relatifs qui considèrent, à la fois, la configuration hétérogène du *HaaS cloud* et les trois métriques de qualité de service (*makespan*, coût, énergie) en même temps, nous avons comparé nos résultats avec ceux de l'heuristique HEFT, que nous avons implémentée. HEFT est une heuristique qui optimise le *makespan* et qui a un bon rapport performances-complexité.

- **Résultats et discussions**

Les figures 4.5, 4.6 et 4.7 montrent des exemples de fronts de Pareto obtenus, par application de l'algorithme DVFS-MODPSO sur, respectivement, le *workflow* synthétique, le *workflow* hybride et le *workflow* parallèle. Elles montrent aussi la solution trouvée par HEFT sur chacun de ces *workflows*.

Comme on peut le remarquer, à partir de ces figures, contrairement à l'algorithme HEFT qui donne une seule solution, notre approche propose un ensemble de solutions non dominées.

Ces résultats illustrent les définitions de base de l'optimisation multiobjectif données dans la section 2.2.1 du chapitre 2.

Maintenant, pour comparer les deux approches, et afin d'analyser l'efficacité de notre proposition, en termes de valeurs de *makespan*, du coût et de la consommation d'énergie, nous avons été inspirés par la méthode décrite dans (Wang, 2010), où l'on compare la solution générée par HEFT à une seule solution du front de Pareto, trouvée par notre algorithme multiobjectif DVFS-MODPSO.

Le processus d'évaluation suit les étapes suivantes:

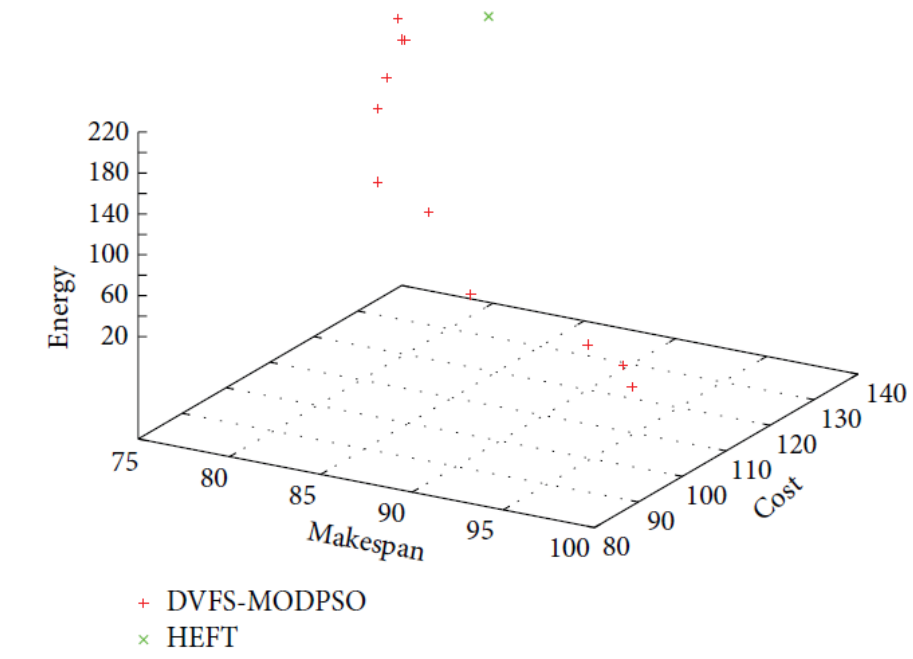


Figure 4.5. Solutions non dominées obtenues pour le *workflow* synthétique.

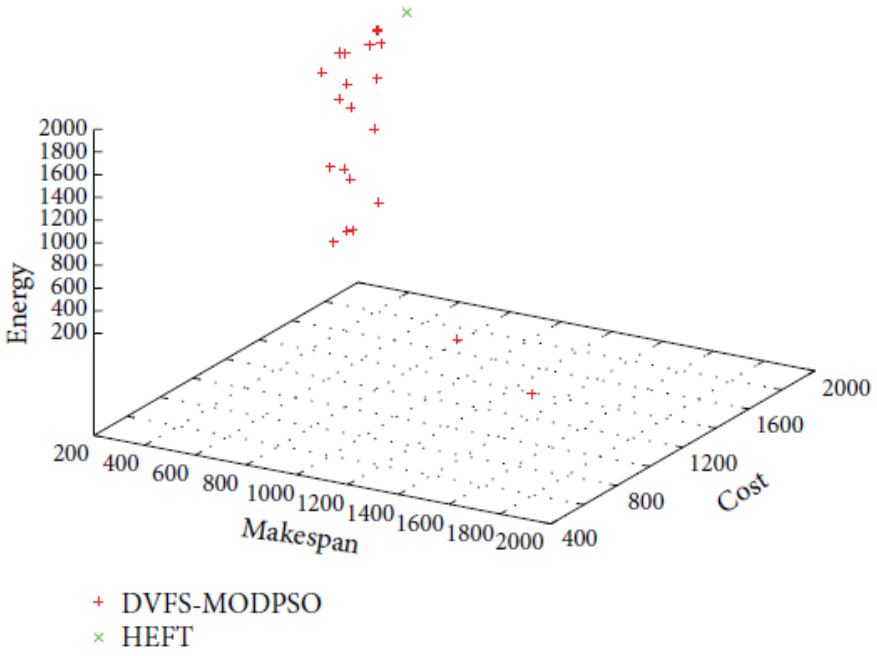


Figure 4.6. Solutions non dominées obtenues pour le *workflow* hybride.

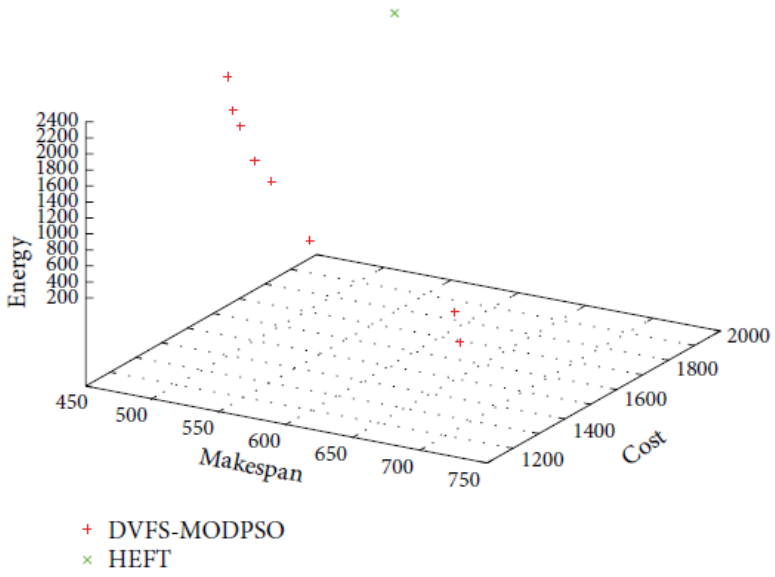


Figure 4.7. Solutions non dominées obtenues pour le *workflow* parallèle.

Pour chaque instance de *workflow* :

- Appliquer l'algorithme HEFT afin d'obtenir une solution *S*.
- Appliquer l'algorithme DVFS-MODPSO pour obtenir l'ensemble de solutions non dominées *EA*.
- Sélectionner une solution *S'* de l'ensemble de solutions du front de Pareto *EA*. Cette solution est la plus proche de *S* au sens de la distance euclidienne.
- Comparer S et S'.

Les différents résultats sont illustrés dans les tableaux et les figures suivantes. Les résultats montrent l'amélioration obtenue par notre algorithme par rapport à HEFT en fonction de la structure des applications de *workflow* et en fonction du nombre de processeurs utilisés.

Nbr.	Gain sur HEFT (%)		
de proc.	*Makespan*	Coût	Énergie
3	0	21,26	40,08
4	0,22	4,64	15,35
5	0	7,56	17,26
6	0,03	6,52	26,75
12	0	9,69	32,55
moyenne	**0,05**	**9,93**	**26,40**

Tableau 4.4. Améliorations pour le *workflow* synthétique.

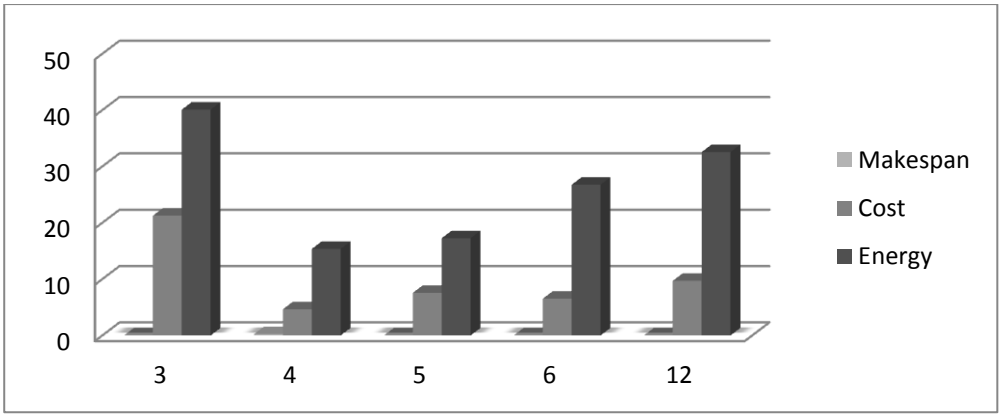


Figure 4.8. Gain sur HEFT (%) en fonction du nombre de processeurs.

Le tableau 4.4 et la figure 4.8 montrent que DVFS-MODPSO améliore les résultats obtenus par HEFT, pour l'instance du *workflow* synthétique. Le temps d'exécution total (*makespan*) est réduit de 0,05%, le coût est réduit de 9,93% et la consommation d'énergie est réduite de 26,40%. La figure 4.8 montre un histogramme des gains moyens obtenus par notre approche, en fonction du nombre de processeurs utilisés.

- **Améliorations en fonction du type de *workflow* réel**

Le tableau 4.5 et la figure 4.9 illustrent les gains obtenus par notre approche selon le type d'applications du monde réel. Le tableau 4.5 montre également les détails des gains obtenus lors du redimensionnement du nombre de processeurs.

	Nbr.	**Gain sur HEFT (%)**		
Workflow	Proc.	*Makespan*	Coût	Énergie
hybride	3	4,05	11,55	04,02
	4	0,28	05,66	05,07
	5	0,00	13,47	11,24
	6	0,00	01,17	12,65
	12	0,43	22,14	07,61
	moyenne	**0,95**	**10,80**	**08,12**
parallèle	3	2,35	56,40	25,52
	4	11,47	20,94	32,71
	5	0,00	21,42	11,66
	6	0,93	11,70	31,54
	12	0,00	0,31	3,06
	moyenne	**2,95**	**22,15**	**20,9**

Tableau 4.5. Améliorations en fonction des applications du monde réel

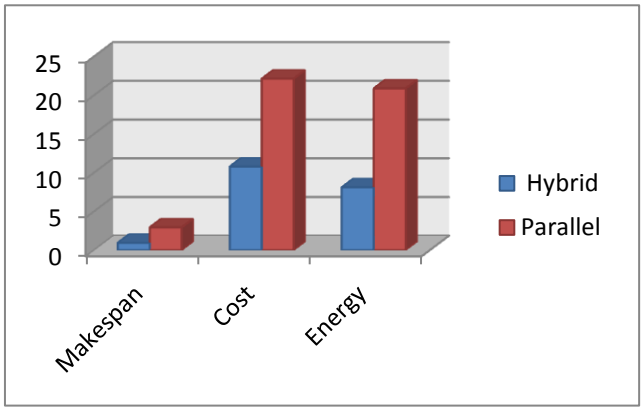


Figure 4.9. Amélioration en fonction des applications du monde réel.

Comme on peut le voir dans le tableau 4.5, les métriques de qualité de service sont améliorées par rapport à HEFT en moyenne de 0,95% pour le *makespan*, 10,8% pour le coût et 8,12% pour la consommation d'énergie, lors de l'utilisation du *workflow* hybride.

Les métriques sont améliorées en moyenne de 2,95% pour le *makespan*, 22,15% pour le coût et 20,9% pour la consommation d'énergie, lors de l'utilisation du *workflow* parallèle. Ces évaluations confirment les résultats obtenus, lors de l'utilisation du *workflow* synthétique.

On remarque qu'en utilisant notre algorithme DVFS-MODPSO, nous sommes en mesure d'améliorer non seulement le coût et l'énergie, mais aussi le *makespan* pour lequel l'algorithme HEFT est censé donner de bons résultats. Ces résultats restent vrais pour l'ordonnancement de tous types de *workflow*, et sur n'importe quel nombre de ressources.

Dans nos expériences, nous avons limité le nombre de ressources utilisées à 12, comme nous avons constaté que, pour ces types de *workflow*, certaines ressources ne sont pas du tout utilisées. La figure 4.10 montre les charges de ressources (pourcentage de tâches affectées à chaque ressource) lors de l'ordonnancement du *workflow* synthétique sur 12 ressources.

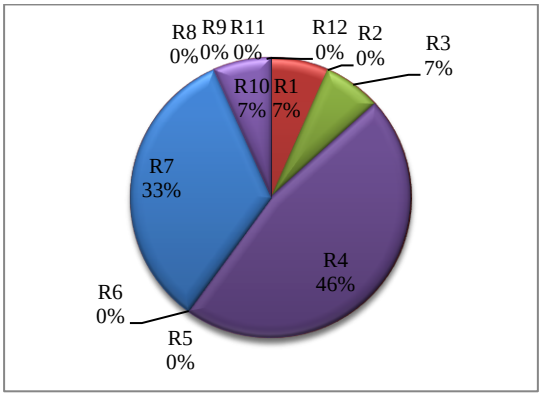


Figure 4.10. Charges des ressources pour le *workflow* synthétique.

En outre, même si le nombre de ressources augmente, le coût total et la consommation totale de l'énergie ne peuvent pas toujours êtres diminués, comme illustré dans la figure 4.11.

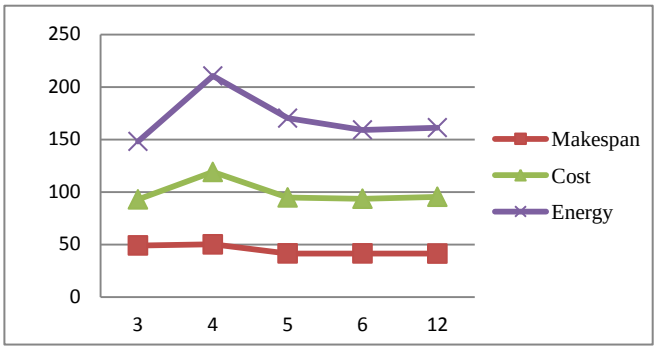


Figure 4.11. Évolutions des QoS en fonction du nombre de ressources pour le *workflow* synthétique.

De cette figure, nous pouvons voir que les paramètres de qualités de service:

(1) diminuent tout d'abord quand le nombre de ressources augmente jusqu'à atteindre le nombre 6. Ceci peut être expliqué par le fait que lorsque le nombre de ressources est augmenté, il y'aura moins de tâches à être exécutées sur une ressource. Cela permettra aux tâches de prolonger leur temps d'exécution, ainsi, les ressources auront plus de chances de réduire leurs tensions et fréquences, ce qui peut être très efficace pour réduire la consommation de l'énergie totale.

(2) Après avoir atteint 6 ressources, les métriques de qualité de service commencent à augmenter. La raison est que les temps de traitements sont dominés par les communications inter-processeurs, diminuant ainsi les possibilités de baisser les tensions et les fréquences de ressources. Par conséquent, un seuil sur le nombre de ressources qui optimise des métriques de QoS peut être obtenu.

4.5. Conclusion

Dans ce chapitre, nous avons présenté un nouvel algorithme, nommé DVFS-MODPSO (*DVFS- Multi-Objectif Discrete Particle Optimization*), pour l'ordonnancement de *workflows* dans des environnements distribués, tels que le HaaS dans un *cloud computing.*

DVFS-MODPSO optimise simultanément plusieurs objectifs contradictoires à savoir, le *makespan*, le coût et l'énergie, dans un espace discret. L'algorithme propose un ensemble de solutions non-dominées, offrant ainsi plus de flexibilité aux utilisateurs pour évaluer leurs préférences et choisir un ordonnancement qui répond au mieux à leurs exigences, en terme de qualité de service.

Notre approche exploite l'hétérogénéité et l'aspect orienté marché des ressources du *cloud*, afin de trouver des solutions, qui optimisent le *makespan* et le coût. En outre, elle utilise la technique de la variation dynamique de la tension et de la fréquence (DVFS) pour réduire la consommation énergétique.

Nous avons évalué notre algorithme, DVFS-MODPSO par simulation, en utilisant à la fois des *workflows* synthétiques et des *workflows* scientifiques issus du monde réel, et ayant des structures différentes. Les résultats obtenus montrent que DVFS-MODPSO est capable de produire un ensemble de bonnes solutions Pareto. Les résultats montrent également que notre approche améliore de façon significative les résultats de HEFT. Elle l'améliore, non seulement en termes de coût et de la consommation d'énergie mais aussi en termes de *makespan* pour lequel il est censé donner de bons résultats.

Conclusion générale et perspectives

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 dans un environnement élastique sous contraintes souvent contradictoires pour les exploitants et les utilisateurs soulève plusieurs défis scientifiques.

Cette thèse s'intéresse particulièrement à la problématique d'ordonnancement des *workflows* d'exécution des services du *cloud* respectant, d'une manière efficace, multiple contraintes. Ce problème est considéré comme étant un problème d'optimisation multiobjectif, en raison de la nature conflictuelle des métriques de QoS à prendre en compte. Nous avons développé des méthodes à base de métaheuristiques pour résoudre cette problématique, en nous focalisant essentiellement sur deux modèles de services de *cloud computing* , notamment l'*Infrastructure as a Service*, IaaS et le *Hardware as a Service*, HaaS.

Au niveau de l'infrastructure *IaaS,* nous avons dans un premier temps résolu le problème d'ordonnancement de *workflows,* en utilisant une approche par agrégation, puis en utilisant une approche Pareto.

Concernant l'approche par agrégation, nous avons adapté l'algorithme génétique (GA) pour le problème d'ordonnancement afin d'optimiser le *makespan*, le coût, la fiabilité et la disponibilité, en mettant en œuvre une nouvelle fonction de *fitness* générique et plusieurs opérateurs de croisements. Les résultats expérimentaux obtenus montrent la supériorité du croisement élitiste. Nous constatons que l'algorithme GA donne de bons résultats d'ensemble pour toutes les métriques de QoS. Ces métriques atteignent plus de 70% de leur meilleure valeur lorsque celles-ci sont considérées d'une façon équitable.

Notre deuxième contribution dans cette approche consiste à gérer les contraintes de QoS et les contraintes fortes. Nous avons développé un algorithme génétique amélioré GA* qui optimise le *makespan*, le coût, la fiabilité et la disponibilité, sous plusieurs contraintes QoS

(délai, budget,…) et plusieurs contraintes fortes. Dans cet algorithme, nous avons mis en œuvre l'operateur *d'Infection Virale* pour traiter les contraintes fortes. De plus, nous avons intégrer une nouvelle fonction de pénalité à la fonction de *fitness* afin de respecter les contraintes de QoS. Les résultats des expériences effectuées montrent l'efficacité de GA* dans le traitement des différentes contraintes considérées. En outre les résultats prouvent que GA* est plus rapide que GA, à trouver une solution au problème d'ordonnancement de *workflows* lorsque plusieurs contraintes fortes sont exigées.

Quant à ce qui concerne l'approche Pareto pour l'ordonnancement de *workflows* au niveau IaaS; l'objectif était d'optimiser le *makespan*, le coût, la fiabilité et la disponibilité simultanément. Nous avons développé l'algorithme NSGA-II qui est basé population et l'algorithme PAES basé sur la recherche locale, afin de fournir un ensemble de solutions Pareto optimal. Les résultats montrent que les deux algorithmes génèrent chacun un bon ensemble de solutions non dominées. L'analyse statistique des résultats ont montré que NSGA-II obtient de meilleurs résultats que PAES pour les deux types de *workflows* étudiés.

Au niveau de l'infrastructure HaaS, notre objectif était d'ordonnancer le *workflow* tout en minimisant le coût, le *makespan* et l'énergie consommée simultanément. Nous avons développé une méthode basée sur *l'optimisation multiobjectif par essaim particulaire* combinée avec la technique DVFS, DVFS-MODPSO. Nous avons adapté le PSO pour le problème d'ordonnancement de *workflows* considéré. DVFS-MODPSO exploite l'hétérogénéité et l'aspect orienté marché des ressources du *cloud* pour optimiser le *makespan* et le coût. En outre, il utilise la technique DVFS pour réduire l'énergie consommée. Les résultats des expériences montrent que cet algorithme est capable de produire un ensemble de bonnes solutions non dominées. Les résultats montrent également que cette approche améliore de façon significative les résultats de HEFT, sur toutes les métriques de QoS, y compris le *Makespan*.

Actuellement, dans la plupart des offres commerciales des services fournis par les fournisseurs *cloud*, l'expression des exigences utilisateurs se limite à la définition de seuils de déclenchement rattachés à des métriques techniques de bas niveau (e.g. consommation mémoire, utilisation CPU) sans lien évident direct avec des SLA (*Service Level Agreement*) de niveau applicatif. De nombreux travaux académiques tentent d'adresser ces limitations; nous y avons contribué, à notre manière dans cette thèse, en apportant des solutions globales

d’ordonnancement des *workflows* dans le *cloud* basées sur des multiples exigences utilisateurs définies dans le SLA.

Les perspectives d’évolution de ce travail concernent différents points :

Dans la partie concernant l’ordonnancement de *workflows* au niveau HaaS, nous n'avons considéré que l'énergie consommée par un processeur pour calculer l’énergie totale. L’idée est de généraliser le modèle en prenant en compte d'autres sources de consommation d'énergie, telles que les disques durs par exemple.

Les algorithmes d’ordonnancement basés sur l’approche Pareto, que nous avons proposés génèrent des ensembles de solutions non dominées de grande taille. La prochaine étape consiste à réduire cet ensemble, afin de permettre au décideur de mettre en place des méthodes interactives pour l’aide à la décision multicritère.

L’hybridation de nos algorithmes avec d’autres métaheuristiques ou des recherches locales, afin d’accélérer les convergences des algorithmes et d’améliorer la qualité des solutions obtenues est en cours de développement.

Dans cette thèse, nous avons évalué les algorithmes proposés via la formalisation des modèles de configuration et de comportement des infrastructures *cloud computing* existantes. Les résultats de simulation sont prometteurs. La prochaine étape serait d’intégrer et d’exécuter ces algorithmes dans des systèmes de gestion de *workflows* existants tels que le Toolkit *cloud*bus par exemple (Buyya, 2009) avant de les appliquer dans un vrai *cloud computing* .
L’exploitation des puissances de calcul fournies et le passage réelle à l’échelle dans un vrai *cloud* permet d’envisager une parallélisation de nos algorithmes, afin d’améliorer la robustesse et de résoudre le problème d’ordonnancement multiobjectif de *workflows* à très grande échelle.

Le problème d’ordonnancement de multiples instances de *workflows* demeure un problème de recherche important. Ce problème se pose lorsque l’ordonnanceur reçoit plusieurs *workflows* d'un seul ou plusieurs utilisateurs, à exécuter sur le même pool de ressources disponibles.

La *scalabilité* est considérée comme un aspect important du *cloud computing.* Les ressources ne peuvent pas évoluer indéfiniment, et quand une limitation des ressources est rencontrée, un fournisseur de services peut décider de déléguer des tâches à d'autres fournisseurs d'une façon transparente aux utilisateurs, afin d'éviter les pénalités de violation du SLA. Un tel scénario ouvre des possibilités de recherche dans la gestion des SLA. L'aspect « *Intercloud* » apportera son lot de problèmes pour la mise en œuvre de nos algorithmes dans un environnement hétérogène distribué.

Dans cette thèse, nous nous sommes concentrés sur l'ordonnancement statique de *workflows*. Il serait intéressant de prendre aussi en compte l'aspect dynamique de l'environnement du *cloud,* afin d'établir le plan d'ordonnancement, à l'aide des informations du *cloud* mises à jour en « *temps réel* ». Le problème du déploiement fiable et de la reconfiguration dynamique d'applications arbitraires réparties (représentées sous forme des *workflows*) impliquant éventuellement la migration des données et des machines virtuelles dans le *cloud* devraient nécessiter des aménagements à nos algorithmes. La prise en compte de la dynamicité dans les modèles nécessite d'utiliser, en complément, les modèles probabilistes, par exemple.

La gestion automatique/autonomique de l'élasticité aura un impact majeur qu'il faudra à l'occasion étudier par rapport à nos algorithmes. L'élasticité dans le domaine du *cloud* computing désigne le (re)dimensionnement et placement dynamique des ressources d'une application (*workflow*), en fonction de la charge à laquelle elle est soumise par ajout / suppression / migration / modification de la configuration de ces ressources. Ce problème représente l'une des principales promesses et un enjeu majeur du *cloud computing.* Les solutions actuelles le traitent insuffisamment et d'une manière naïve. A priori, la solution de ce problème est fondée sur l'hypothèse qu'un module d'analyse et de décision existe dans l'infrastructure *cloud,* pour donner les ordres à effectuer lors de la reconfiguration dynamique (i.e. combien ajouter/retirer d'instances de tel type de composant/ressource) - module qui, à notre connaissance, n'existe pas à ce jour. Il faudra alors étendre nos recherches pour prendre en compte ces aspects.

Références bibliographiques

(Abrishami, 2013) Abrishami, S., Naghibzadeh, M. and Dick H. J. E. Deadline-Constrained Workflow Scheduling Algorithm for Infrastructure as a Service. Future Generation Computer Systems, 29(2013)158-169, 2013.

(Adam, 2008) Adam, A. et Hemert, J. V. Scientific workflow: a survey and research directions. In Proceedings of the 7th Int. Conf. on Parallel processing and applied mathematics, PPAM'07, pages 746- 753, Berlin, Heidelberg, 2008.

(Agrafiotis, 2001) Agrafiotis, D.K. Multiobjective optimization of combinatorial libraries. IBM Journal of Research and Development, 45(2001) 545–566.

(Altintas, 2004) Altintas, I., Berkley, C. Jaeger, E., Jones, M., Ludäscher, B. and Mock, S. Kepler: An Extensible System for Design and Execution of Scientific Workflows, Proc. of 16th Int. Conf. on Scientific and Statistical Database Management (SSDBM 2004), pages 423-424, Santorini Island, Greece, June 2004.

(Amaz, 2012) Amazon Web Services LLC. Amazon elastic compute cloud (amazon ec2) http://aws.amazon.com/ec2/, 2012.

(Armbrust, 2009) Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R. Konwinski, A., Lee, G. Patterson, D., Rabkin, A. and Stoi, I. Above the clouds: A Berkeley View of cloud computing, Technical Report, UCB/EECS-2009-28, UC Berkeley Reliable Adaptive Distributed Systems Laboratory, EECS Department, University of California, Berkeley, February, 2009.

(Armbrust, 2010) Armbrust, M. Fox, A., Griffith, R. Joseph, A. D., Randy, H., Katz, R., Konwinski, A., Lee, G., David, A., Patterson, D., Rabkin, A., Stoica, I. and Zaharia, M. A view of cloud computing. Commun. ACM, 53(2010)50-58.

(Aydin, 2004) Aydin, H., Melhem, R., Moss, D., Meja-Alvarez, P. Power-aware scheduling for periodic real-time tasks, IEEE Trans. Comput. 53 (2004) 584-600.

(Baremetal, 2014) Baremetalcloud, http://baremetalcloud.com/index.php/en/, 2014.

(Baumgartner, 2004) Baumgartner, U., Magele, C., Renhart, W. Pareto optimality and particle swarm optimization, IEEE Transactions on Magnetics, 40(2004)1172-1175.

(Berman, 2001) Berman, F., Chien, A., Cooper, K., Dongarra, J., Foster, I.D., Gannon, D., Johnsson, L., Kennedy, K., Kesselman, C., Mellor-Crummey, J., Reed, D., Torczon, L. and Wolski, R. The GrADS Project: Software Support for High-Level Grid Application Development, International Journal of High Performance Computing Applications (JHPCA), 15(2001)327-344.

(Berriman, 2004) Berriman, G. B., Deelman, E., Good, J. C. J., Jacob, J. C., Katz, D. S., Kesselman, C., Laity, A. C. T., Prince, A., Singh, G. and Su, M. Montage: a Grid-Enabled Engine for Delivering Custom Science-Grade Mosaics on Demand, Proc. of the International Society for Optical Engineering, 5493(1), pages 221-232, Kiev, Ukraine, May 2004.

(Berro, 2001) Berro, A. Optimisation multiobjectifs et stratégies d'évolution en environnement dynamique, Thèse de doctorat, Université des Sciences Sociales Toulouse 1, 2001.

(Best, 2007) Best, B. D., Halpin, P. N., Fujioka, E., Read, A. J., Qian, S. S., Hazen, L. J. and Schick, R. S. Geospatial web services within a scientific workflow: Predicting marine mammal habitats in a dynamic environment. Ecological Informatics, vol 2, no. 3, pages 210-223, 2007.

(Bhanu, 1994) Bhanu, B. and Lee, S. Genetic Learning for Adaptive Image Segmentation. Kluwer Academic Publishers, Boston, 1994.

(Brooks, 2000) Brooks, D. Bose, P., Schuster, S.E., Jacobson, H., Kudva, P.N., Buyuktosunoglu, A., Wellman, J.-D., Zyuban, V., Gupta, M., Cook, P.W. Power-aware

microarchitecture: design and modeling challenges for the next-generation microprocessors, IEEE Micro, 20 (2000)26-44.

(Buyya, 2004) Buyya, R. and Venugopal, S. The Gridbus Toolkit for Service Oriented Grid and Utility computing : An Overview and Status Report, Proc. of 1st IEEE International Workshop on Grid Economics and Business Models (GECON 2004), 19-36, Seoul, Korea, April 2004.

(Buyya, 2008) Buyya, R. Yeo, C. S. and Venugopal, S. Market-Oriented cloud and Atmospheric computing : Hype, Reality, and Vision, Proc. of 10th IEEE Int. Conf. on High Performance computing and Communications (HPCC-08), 5-13, Dalian, China, September, 2008.

(Buyya, 2009) Buyya, R. Yeo, C. S. Venugopal, S. Broberg, J. and Brandic, I. cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Comp. Syst., 25(2009)599-616.

(Buyya, 2009b) Buyya, R., Pandey, S. and Vecchiola, C. Cloudbus toolkit for market-oriented cloud computing. In Proceedings of the 1st Int. Conf. on cloud computing, vol. 5931 of LNCS, pages 24-44. Springer, Germany, December 2009.

(Brucker, 2004) Brucker, P. Scheduling Algortihms, Springer-Verlag, Berlin Heidelberg, Hollinsworth, 2004.

(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.

(Cerny,1985) Cerny, V.A. Thermodynamical approch to the traveling salesman problem: An efficient simulation algorithm. Journal of Optimization Theory and Applications, 45(1985)41-51.

(Chang-Tian , 2012) 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.

(Chow, 2004) Chow, C., Tsui, H. Autonomous agents response learning by multi-species particle swarm optimization, Proceedings of the IEEE Congress on Evolutionary Computation, Vol. 1, pages 778-785, IEEE Press, Portland, Oregon, USA, June 2004.

(Churches, 2005) Churches, D., Gombas, G., Harrison, A., Maassen, J., Robinson, C., Shields, M., Taylor, I. and Wang, I. Programming Scientific and Distributed Workflow with Triana Services, Concurrency and Computation: Practice and Experience, 18(2005)1021-1037.

(Clerc, 2005) Clerc, M. L'optimisation par Essaims Particulaires. Hermes Science Publications, 2005.

(Coa, 2003) Coa, J., Jarvis, S., Saini, S. and Nudd, G. GridFlow: Workflow Managament for Grod computing, Proc. of 3rd International Symposium on Cluster Computing and the Grid, pages 198-205, Tokyo, Japan, May 2003.

(Coello, 2002) Coello, C., Veldhuizen, D., Lamont, G. Evolutionary Algorithms for Solving Multi-Objective Problems, Kluwer Academic Publishers. 2002.

(Coello, 2002b) Coello, C.A., Lechuga, M. S. MOPSO: a proposal for multiple objective particle swarm optimization, Proceedings of the 2002 IEEE Congress on Evolutionary Computation, Vol. 2, pages 1051-1056, IEEE Press, 2002.

(Coello, 2004) Coello Coello, C.A., Toscano Pulido, G., Salazar Lechuga, M. Handling multiple objectives with particle swarm optimization, IEEE Transactions on Evolutionary Computation, Vol. 8, N° 3, pages 256-279, 2004.

(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.

(Collette, 2002b) Collette, Y. et Siarry, P. Optimisation multiobjectif, Eyrolles, 2002.

(Cybok, 2008) Cybok, D. Workflow Management for Grid computing - A Grid Workflow Infrastructure, msg systems ag, Munich, Germany, 2008.

(Darwin, 1859) Darwin, C. On the origin of the species by means of natural selection: Or, the

preservation of favored races in the struggle for life. John Murray, 1859.

(Davenport, 1993) Davenport, T.H. Process innovation: reengineering work through information technology. Harvard Business School Press, Boston, MA, USA, 1993.

(Deb, 2001) Deb, K., Multi-Objective Optimisation using Evolutionary Algorithms, Edited by John Wiley & Sons, Chichester, 2001.

(Deb, 2002) Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. A fast and elitist multi-objective genetic algorithm: NSGA-II. IEEE Transaction on Evolutionary Computation, 6(2002)181-197.

(Deelman, 2003) Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G. and Vahi, K. Mapping Abstract Complex Workflows onto Grid Environments, Journal of Grid computing, 1(2003) 25-39.

(Deelman, 2005) Deelman, E., Singh, G., Su, M., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G. B., Good, J., Laity, A., Jacob, J. C. and Katz, D. S. Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems. Scientific Programming, 13(2005)219-237.

(DeJong, 1975) De Jong, K. The analysis of the behaviour of class of genetic adaptative systems. Thèse de doctorat, Department of computer Science, University of Michigan, Ann Arbor, Michigan, 1975.

(Edgeworth, 1881) Edgeworth. F.Y. Mathematical Physics. P. Keagan, London, 1881.

(Egwutuoha, 2012) Egwutuoha, I.P., Chen, S., Levy, D., Selic, B. and Calvo, R. A Proactive Fault Tolerance Approach to High Performance Computing (HPC) in the cloud. In The 2nd Int. Conf. on Cloud and Green Computing, pages 268-273, Xiangtan, Hunan, China, 2012.

(El-kenawy, 2012) 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.

(Eucal, 2014) Inc. Eucalyptus Systems. Eucalyptus cloud http://www.eucalyptus.com/eucalyptus-cloud, 2014.

(Eugen, 2011) Eugen, F., Rilling, L. and Morin, C. Energy-Aware Ant Colony Based Workload Placement in clouds. Proceedings of the 2011 IEEE/ACM 12th Int Conf. on Grid Computing pages 26-33, 2011.

(Fahringer, 2005) Fahringer, T., Jugravu, A., Pllana, S., Prodan, R., Seragiotto J., C. and Truong, H.L. ASKALON: A Tool Set for Cluster and Grid Computing, Concurrency and Computation: Practice and Experience, 17(2005)143-169.

(Fang, 2011) Fang, L., Jin, T., Jian, M., Robert, B., John Messina, L.B. and Leaf, D. NIST Cloud Computing Reference Architecture. 2011.

(Fatemipour, 2012) Fatemipour, F. and Fatemipour, F. Scheduling Scientific Workflows using Imperialist Competitive Algorithm. International Conference on Industrial and Intelligent Information (ICIII 2012) IPCSIT, Vol. 31, p218-225, Singapore 2012.

(Fonseca, 1993) Fonseca, C. M. and Fleming, P. J. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In S. Forrest, editor, Proceedings of the 5th Int. Conf. on Genetic Algorithms, pages 416-423, Morgan Kaufmann publishers, 1993.

(Fonseca, 1995) Fonseca, C. M. and Fleming, P. J. An overview of evolutionary algorithms in multiobjective optimization. Evolutionary Computation, 3(1995)1-16.

(Foster, 1999) Foster, I. and Kesselman, C. The grid: blueprint for a new computing infrastructure. Morgan Kaufmann Publishers Inc., 1999.

(Fourman, 1985) 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.

(Garfinkel, 1999) Garfinkel, S. and Abelson, H. Architects of the Information Society: 35 Years of the Laboratory for Computer Science at Mit. MIT Press, Cambridge, MA, USA, 1999.

(Ge, 2005) Ge, R., Feng, X. and Cameron, K. W. Performance-constrained Distributed DVS

Scheduling for Scientific Applications on Power-aware Clusters. In Proc. the ACM/IEEE Conference on Supercomputing, pages 34-44, November 2005.

(Ge, 2010) 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.

(Geelan, 2008) Geelan, J. Twenty One Experts Define Cloud Computing. Electronic Magazine, available at http://virtualization.sys-con.com/node/612375.

(Gil, 2007a) Gil, Y., Deelman, E., Ellisman, M. H., Fahringer, T., Fox, G. Gannon, D., Goble, C. A., Livny, M., Moreau, L. and Myers, J. Examining the Challenges of Scientific Workflows. IEEE Computer, 40(207)24-32.

(Glover, 1986) Glover, F. Tabu Search - part i. ORSA Journal on Computing, 1(1986)190-206.

(GoG, 2014) Gogrid cloud hosting http://www.gogrid.com/products/cloud-hosting, 2014.

(Goldberg, 1989) Goldberg, D E. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, 1989.

(Grid, 2014) Grid5000. http://www.grid5000.fr, 2014 http://www.openstack.org/, 2014.

(Gruman,2008) Gruman, G. and Knorr, E. What cloud computing really mean. InfoWorld, Electronic Magazine available at http://www.infoworld.com/d/cloud-computing/what-cloud-computing -really-means-031.

(Guo, 2012) Guo, L., Shao, G. and Zhao, S. Multi-objective Task Assignment in cloud computing by Particle Swarm Optimization. In Proceedings of 8th Int. Conf. on Wireless Communications, Networking and Mobile computing , pages 1-4, 2012.

(Guzek, 2012) 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.

(GW, 1998) Groupware & workflow. 16 février 1998.

(Hakem, 2007) Hakem, M. and Butelle, F. Reliability and Scheduling on Systems Subject to Failures. Int. Conf. on Parallel Processing (ICPP), Sept. 2007.

(Hamilton, 2009) Hamilton, J. Cooperative expendable micro-slice servers (CEMS): low cost, low power servers for Internet-scale services, In: Proc of CIDR, 2009.

(Hoffa, 2008) Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K., Berriman, B., and Good, J. On the Use of cloud computing for Scientific Workflows, Proc. of Fourth IEEE Int. Conf. on e-Science, pages 640-645, Indianapolis, Indiana, USA, December 2008.

(Holland, 1962) Holland, J. H. Outline for a Logical Theory of Adaptive Systems, Journal of the association of computing machinery 3(1962)297-314.

(Hu, 2003) Hu, X., Eberhart, R. S. Y. Particle Swarm with Extended Memory for Multiobjective Optimization. Proceedings of the 2003 IEEE Swarm Intelligence Symposium, pages 193-197, IEEE Press., 2003

(Jang, 2012) 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.

(Jin, 2001) Jin, Y., Okabe, T., Sendho, B. Dynamic weighted aggregation for evolutionary multiobjective optimization: why does it work and how?. Proceedings of the Genetic and Evolutionary Conference, pages 1042-1049, Morgan Kaufman Publishers, 2001.

(Justesen, 2009) Justesen, P D. Multi-objective Optimization using Evolutionary Algorithms, University of Aarhus, Department of Computer Science, Denmark, 2009.

(Juve, 2008) Juve, G. and Deelman, E. Resource provisioning options for large-scale scientific workflows. In: Proceedings of the 3rd International Workshop on Scientific Workflows and Business Workflow Standards in e-Science (SWBES '08), 2008.

(Kanoh, 1996) Kanoh, H., Hasegawa, K., Matsumoto, M., Kato, N and Nishihara, S. Solving Constraint Satisfaction Problems by a Genetic Algorithm Adopting Viral Infection, Institute of Information Sciences and Electronics, University of Tsukuba, Japan, 1996.

(Kennedy,1985) Kennedy, J. and Eberhart, R. C. Particle Swarm Optimization. In : Proceedings of the IEEE Int. Conf. on Neural Networks IV, pages 1942-1948, Perth, Australia, 1995.

(Kennedy, 2001) Kennedy, J., Eberhart, R.C., Shi, Y. Swarm Intelligence, Morgan Kaufmann Academic Press. 2001

(Khan, 2009) Khan, S. U., Ahmad, I. A Cooperative Game Theoretical Technique for Joint Optimization of Energy Consumption and Response Time in Computational Grids, IEEE Trans on Parallel and Distributed Systems, 21(2009)346 - 360.

(Knowles, 1999) Knowles, J.D. and Corne, D.W. The Pareto archived evolution strategy: A new baseline algorithm for multiobjective optimisation. 1999 congress on evolutionary computation, pages 98-105, Piscataway, NJ. Washington, DC, USA: IEEE Service Center, July 1999.

(Laszewski, 2004) Laszewski, G. V., Amin, K., Hategan, M., Zaluzec, N. J., Hampton, S. and Rossi, GridAnt, A. A Client-Controllable Grid Workflow System, Proc. of 37th Hawaii Int. Conf. on System Sciences (HICSS-37), pages 210-219, Hawaii, USA, January 2004.

(Lee, 2009) Lee, Y. C. and Zomaya, A. Y. Minimizing Energy Consumption for Precedence-Constrained Applications Using Dynamic Voltage Scaling. In Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID), pages 92–99, IEEE Computer Society, Washington, DC, USA, 2009.

(Lee, 2011) Lee, Y. C. Zomaya, A.Y. Energy Conscious Scheduling for Distributed Computing Systems under Different Operating Conditions, IEEE Trans. Parallel Distrib. Syst., vol. 22, no. 8, 1374–1381, Aug. 2011.

(Lee, 2012) Lee, Y. C. and Zomaya, A.Y. Energy efficient utilization of resources in cloud computing systems. The Journal of Supercomputing, 60(2012)268-280

(Lenstra, 1978) Lenstra, J.K. and. Rinnooy Kan, A.H.G. Complexity of scheduling under precedence constraints, Operations Research, 26(1978)22–35.

(Li, 2003) Li, X. A non-dominated sorting particle swarm optimizer for multiobjective optimization. Proceedings of the 2003 Genetic and Evolutionary Conference, pages 37-48, LNCS 2723, Springer.

(Li, 2009) Li, Y. Liu, Y. Qian, D. A heuristic energy-aware scheduling algorithm for heterogeneous clusters, In ICPADS. IEEE, pages 407-413, 2009.

(Li, 2011) 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.

(Lin, 2009) Lin, C., Shiyong Lu, S., Fei, X., Chebotko, A., Pai, D., Lai, Z., Fotouhi, F. et Hua, J. A Reference Architecture for Scientific Workflow Management Systems and the VIEW SOA Solution. IEEE Transactions on Services computing, 2(2009)79-92.

(Lin, 2011) Lin, C., Lu, S. Scheduling Scientific Workflows Elastically for Cloud Computing. In IEEE 4th Int. Conf. on Cloud Computing, pages 746-747, 2011

(Liu, 2010) Liu, K., Yang, Y., Chen, J., Liu, X., Yuan, D. and Jin, H. A Compromised-Time- Cost Scheduling Algorithm in SwinDeW-C for Instance-intensive Cost-Constrained Workflows on cloud computing Platform, International Journal of High Performance computing Applications, 24(2010)445-456.

(Liu, 2013) 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.

(Ludäscher, 2009) Ludäscher, B., Weske, M., Mcphillips, T. et Bowers, S. Scientific Workflows: Business as Usual? In Proceedings of the 7th Int. Conf. on Business Process Management, BPM '09, pages 31–47, Berlin, Heidelberg, 2009. Springer-Verlag.

(McFedries, 2008) McFedries, P. The cloud is the computer. IEEE Spectrum Online, Electronic Magazine available at http://spectrum.ieee.org/computing /hardware/the-cloud-is-the-computer.

(Mell, 2011) Mell, P. and Grance, T. The NIST Definition of Cloud Computing , September 2011.

(Mezmaz , 2010) Mezmaz, M., Lee, Y.C., Melab, N., Talbi, E.G., Zomaya, A. Y. A bi-objective hybrid

genetic algorithm to minimize energy consumption and makespan for precedence-constrained applications using dynamic voltage scaling, Evolutionary Computation (CEC), 2010 IEEE Congress on, pages 1–8, July 2010.

(Micros, 2014) Microsoft Windows azure http://www.windowsazure.com/en-us/, 2014.

(Miettinen, 1999). Miettinen, K. Nonlinear Multiobjective Optimization. Kluwer Academic Publishers, 1999.

(Moore, 1999) Moore, J. and Chapman, R. Application of Particle Swarm to Multiobjective Optimization. Department of Computer Science and Software Engineering, Auburn University. (Unpublished manuscript), 1999.

(Netjinda, 2012) Netjinda, N., Sirinaovakul, B. and Achalakul, T. Cost Optimization in cloud Provisioning using Particle Swarm Optimization. In Proceedings of 9th IEEE Int. Conf. on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, pages 1-4, 2012.

(Nimb, 2014) Nimbus cloud http://www.nimbusproject.org/, 2014.

(O'Brien, 2004) A. O'Brien, S. Newhouse, and J. Darlington. Mapping of scientific workflow within the e-protein project to distributed resources. In Proceedings of the UK e-Science AllHandsMeeting, Nottingham, UK, September 2004.

(Oinn, 2004) Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M. Greenwood, M. Carver and, T. Glover, K. Pocock, M. R. Wipat, A. and Li, P. Taverna: A Tool for the Composition and Enactment of Bioinformatics Workflows, Bioinformatics, 20(17), 3045-3054, 2004.

(OpenN, 2014) OpenNebula Project (OpenNebula.org). Opennebula open-source cloud http://www.opennebula.org/ , September 2012.

(OpenS, 2014) OpenStack.org. Openstack: the open source cloud operating system, 2014.

(Pandey, 2008) Pandey, S., Chao, J., Voorsluys, W., Rahman, M., Buyya, R. Gridbus Workflow Management System on Clouds and Global Grids, Proc. of IEEE 4th Int. Conf. on e-Science, 1(1), pages 323-324, December 2008.

(Pandey, 2010) Pandey, S., Wu, L., Guru, S. and Buyya, R. A particle swarm optimization based heuristic for scheduling workflow applications in cloud computing environments. 24th IEEE Int'l Conference on Advanced Information Networking and Applications (AINA), pages 400-407, Perth, Australia, 2010.

(Papadimitriou, 1982) Papadimitriou, C. and Steiglitz, K. Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall, 1982.

(Pareto, 1996) Pareto, V. Cours d'économie politique. Rouge, Lausanne, 1996.

(Parsopoulos, 2002) Parsopoulos, K.E., Vrahatis, M.N. Particle Swarm Optimization in Multiobjective Problems, Proceedings of the 2002 ACM Symposium on Applied Computing , pages 603-607, ACM Press, 2002.

(Parsopoulos, 2004) Parsopoulos, K.E., Tasoulis, D., Vrahatis, M.N. Multiobjective optimization using parallel vector evaluated particle swarm optimization, Proceedings of the IASTED Int. Conf. on Artificial Intelligence and Applications, Vol. 2, pages 823- 828, Acta Press. 2004.

(Racks, 2014). Rackspace open cloud http://www.rackspace.com/cloud/, 2014.

(Raque, 2005) Raquel, C.R., Naval, J.P.C. An effective use of crowding distance in multiobjective particle swarm optimization, Proceedings of the 2005 Genetic and Evolutionary Computation Conference, pages 257-264, ACM Press, 2005.

(Ray, 2002) Ray, T. Liew, K.M. A swarm metaphor for multiobjective design optimization, Engineering Optimization, 34(2002)141-153.

(Reyes-Sierra, 2006) Reyes-Sierra, M. Coello C.A. Multi-Objective Particle Swarm Optimizers: A survey of the state-of-the-art, International Journal of Computational Intelligence Research, 2(2006)287-308.

(Ried, 2011) Ried, S., Kisker, H., Matzke, P., Bartels, A. and Lisserman, M. Understanding and quantifying the future of cloud computing. Technical report, 2011.

http://www.forrester.com/rb/Research/sizing_cloud/q/id/58161/t/2.

(Rimal, 2009) Rimal, B. P., Choi, E. and Lumb, I. A taxonomy and survey of cloud computing systems. In INC, IMS and IDC, 2009, NCM'09, Fifth Int. Joint Conf. on INC, IMS and IDC, pages 44-51, 2009.

(Salehi, 2010) Salehi, M.A. and Buyya, R. Adapting market-oriented scheduling policies for cloud computing , Proceedings of the 10th Int. Conf. on Algorithms and Architectures for Parallel Processing (ICA3PP 2010), pages 351-362, Busan, Korea, 2010.

(Sarda, 2011) Sarda, K., Sanghrajka, S. and Sion, R. Cloud Performance Benchmark Series : Amazon EC2 CPU Speed Benchmarks, Technical Report, cloud computing Center & Network Security and Applied Cryptography Lab, Stony Brook University, New York, USA. 2011.

(Schaffer, 1985) 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.

(Siarry, 2003) Siarry, P. Dréo, J. Pétrowski, A. Taillard, E. Métaheuristiques pour l'optimisation difficile, Eyrolles, 2003.

(Singh, 1996) Singh, M.P. and Vouk, M.A. Scientific workflows: scientific computing meets transactional workflows. In Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the- Art and Future Directions,, pages 28-34, Univ. Georgia, Athens, GA, USA, 1996.

(Singh, 2005) Singh, G., Kesselman, C., Deelman, E.: Performance impact of resource provisioning on workflows. Tech. rep., University of Southern California, Information Sciences Institute, 2005.

(SoftLayer, 2014) SoftLayer, http://www.softlayer.com/, 2014.

(Srikantaiah, 2009) Srikantaiah, S., Kansal, A., Zhao, F. Energy aware consolidation for cloud computing. Cluster Computing, 12(2009)1-15.

(Sriniva, 1994) Sriniva, N. and Kalyanmoy, D. Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. Evolutionary Computation, 2(1994)221–248.

(Stewart, 1991) Stewart B.S. and White, C.C. Multiobjective A*. Journal of the ACM, 38(1991)775-814.

(Sublime, 2012) Sublime, J, Yassa, S., Jo, G-S. A genetic algorithm with the concept of viral infections to solve hard constraints in workflow scheduling, In proceeding of: Korea Intelligent Information System Society, At Seoul National University, Seoul, Volume: 한국지능정보시스템학회 학술대회논문집, pages 95-100, 2012.

(Talbi, 2009) Talbi, E. G. Metaheuristics: from design to implementation. Wiley, 2009.

(Topcuoglu, 2002) 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.

(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.

(Vaquero, 2009) Vaquero Gonzalez, L.M., Rodero Merino, L., Caceres, J. and Lindner, M. A break in the clouds: towards a cloud definition. Computer Communication Review, 39(2009)50–55.

(Vasic, 2009) Vasic, N., Barisits, M., Salzgeber, V., Kostic, D. Making cluster applications energy-aware. In: ACDC. Proc. of the 1st Workshop on Automated Control for Datacenters and Clouds, pages 37-42, Barcelona, Spain, 2009.

(Verma, 2012) Verma, A., Kaushal, S. Deadline and Budget Distribution based Cost- Time Optimization Workflow Scheduling Algorithm for cloud. In Proceedings of the IJCA on International Conference on Recent Advances and Future Trends in Information Technology (iRAFIT '12), pages 1-4, 2012.

(Vouk, 2008) Vouk, M. A. cloud computing_issues, research and implementations. Journal of

Computing and Information Technology, 16(2008)235-246.

(Wang, 2008) Wang, L., Tao, J., Kunze, M., Castellanos, A., C. Kramer, D. and Karl, W. Scientific cloud computing : Early Definition and Experience, Proc. of the 10th IEEE Int. Conf. on High Performance computing and Communications, pages 825-830, Washington, DC, USA, September, 2008.

(Wang, 2010) Wang, L., Von Laszewski, G. and Dayal, J. Towards Energy Aware Scheduling for Precedence Constrained Parallel Tasks in a Cluster with DVFS. IEEE/ACM CCGRID'10, 368–377, Australia, 2010.

(Wang, 2011) 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.

(Weske, 2007) Weske, M. Business process management: Concepts, languages, architectures. Springer Verlag, first édition, Novembre 2007.

(WfMC, 1999) Workflow Management Coalition. Terminology and Glossary (WFMC-TC-1011). 1999.

(WfMC, 1995) WfMC. The Workflow Reference Model. Document Number TC00- 1003. Document Status - Issue 1.1. 19 Jan 1995.

(Wood, 2007) Wood, T., Shenoy, P., Venkataramani, A., Yousif, M. Black-box and gray-box strategies for virtual machine migration, Proceedings of the 4th USENIX conference on Networked systems design & implementation, pages 17-17, Cambridge, MA, April 11-13, 2007.

(Yang, 2008) Yang, Y., Liu, K., Chen, J., Liu, X., Yuan, D. and Jin, H. An Algorithm in SwinDeW-C for Scheduling Transaction-Intensive Cost-Constrained cloud Workflows, Proc. of 4th IEEE Int. Conf. on e-Science, pages 374-375, Indianapolis, USA, December 2008.

(Yao, 1995) Yao, R., Demers, A., Shenker, S. A scheduling model for reduced cpu energy. In: IEEE Sym. on Foundations of Computer Science, FOCS.95, pages 374-382, Oct. 1995.

(Yassa, 2011) Yassa, S., Kadima, H. Négociation et composition dynamique des services basée sur le SLA dans un cloud. Workshop sur l'Evolution du principe de Réutilisation entre Composants, Services et cloud Services(RCS2), Lille, juin 2011.

(Yassa, 2012) Yassa, S., Chelouah, R., Kadima, H., Granado, B. A Genetic Algorithm Approach to QoS based Workflow Scheduling in cloud computing Environment. Proceeding of ICDSD'12 International Conference en Distributed Systems and Decisions, ISSN 2335-1012, Oran-Algeria, November 21-22, 2012.

(Yassa, 2012b) Yassa, S., Chelouah, R., Kadima, H., Granado, B. A PSO-based Heuristic for energy-aware scheduling of Workflow applications on cloud computing, 25th European Conference on Operational Research, Lithuania 2012.

(Yassa, 2013) Yassa, S., Chelouah, R., Kadima, H., Granado, B. A Genetic Algorithm Approach to a Cloud Workflow Scheduling Problem with Multi-QoS Requirements, 26th European Conference on Operational Research, Rome 1-4 July, 2013.

(Yassa, 2013a) Yassa, S, Sublime, J., Chelouah, R., Kadima, H, Jo, G-S., Granado, B. A Genetic Algorithm for Multi-Objective Optimization in Workflow Scheduling with Hard Constraints, International Journal of Metaheuristics, 2(2013)415 – 433.

(Yassa, 2013b) Yassa, S., Chelouah, R., Kadima, H., Granado, B. Multi-Objective Approach for Energy-Aware Workflow Scheduling in cloud computing, The Scientific World Journal, vol. 2013, Article ID 350934, 13 pages, 2013.

(You, 2008) Yu, J., Buyya, R. and Ramamohanarao, K. Workflow Scheduling Algorithms for Grid Computing. 146(2008)173-214, Springer Heidelberg.

(Yu, 2004) Yu, J. and Buyya, R. A Novel Architecture for Realizing Grid Workflow Using Tuple Spaces, Proc. of the 5th IEEE/ACM International Workshop on Grid computing , pages 119-128, Pittsburgh, USA, November, 2004.

(Yu, 2005) Yu, J. and Buyya, R. A Taxonomy of Workflow Management Systems for Grid

Computing, Journal of Grid Computing, 3(2005)171-200.

(Zhang, 2002) Zhang, Y., Sharon X., Hu, D., Chen, Z. Task scheduling and voltage selection for energy minimization. In: Design Automation Conf., pages 183-188, New Orleans, Louisiana, USA, June 10-14, 2002,

(Zhao, 2004) Y. Zhao, M. Wilde, I. Foster et al., Grid middleware servicesfor virtual data discovery, composition, and integration, in Proceedings of the 2nd Workshop on Middleware for Grid computing (MGC '04), pp. 57–62, Ontario, Canada, October 2004.

(Zhu, 2003) Zhu, D., Melhem, R., Childers, B.R. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems, IEEE Transactions on Parallel and Distributed Systems, 14(2003)686–700.

(Zhu, 2004) Zhu, D., Mosse, D., Melhem, R. Power-aware scheduling for and/or graphs in real-time systems, IEEE Transactions on Parallel and Distributed Systems, 15(2004)849–864.

(Zhuo, 2005) Zhuo, J. and Chakrabarti, C. An efficient dynamic task scheduling algorithm for battery powered DVS systems, In: Asian South Pacific Design Automation Conf., pages 846-849, Shanghai, China Jan. 2005.

(Zitzler, 1999) Zitzler, E. Thiele, L. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approaches. IEEE Transactions on Evolutionary Computation, 3(1999)257-271.

(Zitzler, 2002) Zitzler, E., Laumanns, M. and Thiele, L. SPEA2: Improving the strength pareto evolutionary algorithm. In Proc. of Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN 2001), pages 95-100, 2002.

(Zitzler, 2004) 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.

(Zitzler, 2004b) Zitzler, E. and Kunzli. S. Indicator-based selection in multiobjective search. Parallel Problem Solving from Nature, PPSN VIII, 3242(2004)832-842.