RÉPUBLIQUE ALGÉRIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Abderrahmane Mira de Béjaïa
Faculté des Sciences Exactes
Département d'Informatique
Centre de recherche sur l'information scientifique et technique

Mémoire de Fin de Cycle

En vue de l'obtention du diplôme de Master recherche en Informatique

Spécialité : Réseaux et Systèmes Distribués

Thème

Algorithmes évolutionnaires parallèles sur plateforme de calcul volontaire, application aux problèmes de permutations

Réalisé par :

M^r DAOUDI Nassim & M^r MEHDI Juba

Devant le jury composé de :

Présidente :	M^{me}	YAICI	Malika
Examinatrice :	M^{lle}	DJERROUD	Souhila
Examinatrice :	M^{me}	GHIDOUCHE	Kahina
Encadreur :	M^r	SIDER	Abderrahmane (Université de béjaia)
Co-encadreur :	M^r	BENDJOUDI	Ahcène (CERIST)
Co-encadreur :	M^{me}	SILHADI	Malika née MEHDI (CERIST)

Promotion 2013

⁂ ⁂ ⁂ *Remerciements* ⁂ ⁂ ⁂

Nous rendons grâce à notre Dieu, le tout puissant et miséricordieux, pour nous avoir donné le courage et la patience pour mener à bout ce modeste travail et qui nous a procuré ce succès.

Nos plus sincères remerciements s'adressent à notre encadreur M^{r} SIDER Abdrahmen et nos deux co-encadreurs M^{r} BENDJOUDI Ahcène et M^{me} SILHADI Malika née MEHDI , pour leurs confiance, disponibilité et leurs encouragements.

Nous remercions tous particulièrement les membres de jury, en l'occurrence M^{me} YAICI Malika, M^{lle} DJERROUD Souhila et M^{me} GHIDOUCHE Kahina d'avoir accepter d'évaluer notre travail et pour l'intérêt qu'ils y portent. Ainsi que tous les enseignants qui ont contribué à notre formation.

Un grand merci à toute l'équipe du CERIST, qui nous ont bien accueilli et nous ont offert une meilleure atmosphère pour débuter notre travail.

Un grand merci à toute l'équipe du Centre de calcul de l'université de béjaia, de nous avoir permis l'accés aux matrériels nécessaires à la réussite de notre travail.

Un grand merci à nos familles, pour leur soutien permanent et indéfectible qui nous ont permis de chercher au plus profond fond de nous même la force, la volonté et la persévérance à même d'arriver à cet instant des plus important de notre vie.

Un merci pudique à nos amis, nos collègues en Master 2 et tous ceux qui ont contribué de près ou de loin à la concrétisation de cette œuvre.

⁂⁂⁂⁂ *Dédicaces* ⁂⁂⁂⁂

Je dédie ce modeste travail :

A mes chèrs parents pour leur sacrifices depuis qu'ils m'ont mis au monde.

À mes frères et sœurs Malika, Nacer, Kamel et Moumen.

À mes amis(es) ceux avec j'ai parcouru le long chemain avec tant de peine et de joie, Yasmine, Kinza, Zahia, Youba, Walid mamou, Yazid, Fares, Lila, souhila, Nawel, ainsi que tous ceux qui m'ont aider de prés ou de loin.

À mon binôme Daoudi Nassim et sa famille.

Mr MEHDI Juba

Je dédie ce travail.

À mes parents

À mes frères et soeurs .

Vous vous êtes dépensés pour moi sans compter.

En reconnaissance de tous les sacrifices consentis par tous et chacun pour me permettre d'atteindre cette étape de ma vie.

À tous mes amis et amies et plus particulièrement mon binôme MEHDI Juba.

À tous ceux qui m'ont aidé de près ou de loin dans la réalisation de ce travail.

Et à tous les étudiants de la promotion Informatique.

Mr DAOUDI Nassim

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Le calcul scientifique devient de jours en jours plus complexe suivant l'évolution de la science. Utiliser des machines afin de résoudre des équations complexes devient un must dont on peut plus se défaire, en effet ces dernières arrive à faire des calculs gigantesques à une vitesse qu'on spécifiera de miraculeuse.

Cependant on s'est vite heurté à des problèmes dont la complexité est telle que même nos formidables machines calculatrices ont échoué à résoudre. Faute de temps de résolution, ses problème dis NP-Hard sont longtemps resté classé comme insolubles en matière d'optimalité.

L'apparition des nouvelles techniques de calcul rapproché nommées méta-heuristiques nous ont permis de nous rapprocher de la solution optimale, parmi ces méthodes on retrouve les fameux algorithmes génétiques auquel nous nous intéresserons au cours de notre travail et dont les études n'arrêtent pas de se succéder, pour cause d'avoir fait leur preuve en matière d'optimalité des résultats et du temps de calcul obtenu. Cependant même avec de pareilles techniques les machines utilisées lors des calculs se sont toujours montrées aussi inefficaces en matière de temps de calcul.

La solution qui a été proposé et qui s'est d'ailleurs montré très efficace est de partager le calcul entre plusieurs machines pour obtenir à la fin les même résultats

en un temps beaucoup moins important. A partir de la, des plateformes de parallélisation et distribution de tâches de calcul entre différente machine ont pu voir le jour.

BOINC, SiCortex, Beowulf ou autres sont toute des plateformes de calcul distribué, permettant de relier un grand nombre de machines pour participer à un seul et même calcul sous un concept appelé le calcul volontaire.

Avoir un large éventail de machine effectuant des tâches d'un même calcul est certes un grand privilège mais possède aussi ses contraintes, en effet on ne peut se permettre une exécution anarchique des tâches entres les différente machine, c'est ici que nous venons à nous demander : De quel manière devrait-on distribuer les tâches sur les différentes machines participantes au calcul ? Comment récupérer les résultats et surtout comment savoir que ses derniers sont valide ou erroné ?

Ce document présentera le plus gros de notre travail pour essayer de répondre à ces questions, le premiers chapitre résumera les éléments clé nécessaire pour la bonne compréhension de notre travail, le deuxième est une études des travaux existant dans ce domaine, suivie d'une présentation des Algorithme génétiques, le quatrième est une étude approfondie de la plateforme BOINC et de ces composants, et pour finir le cinquième et dernier chapitre constituera la présentation et l'étude de notre proposition.

1

Généralités sur les architectures parallèles et le calcul distribué

Introduction

Avant de nous plonger dans ce qui est le cœur de notre travail d'études, nous devons d'abord faire un tour d'horizon sur ce que sont les architectures parallèles et distribués les plus importantes dans le cadre de notre travail, vu l'influence de ces dernières sur les résultats que nous sommes supposés atteindre. Ce chapitre aura donc pour mission de survoler ce concept sans trop s'y approfondir.

1.1 Les architectures parallèles

Les architectures parallèles sont nombreuses et leur nombre ne cesse d'augmenter grâce aux prouesses dont font preuve les constructeurs dans des buts d'améliorations. Il a fallut donc mettre au point une classification afin de les organiser et de les distinguer selon la forme de leurs parallélismes. Les classifications les plus importantes sont listées comme suit [8] :

- Classification de Flynn.
- Classification basée sur l'arrangement de la mémoire et la communication entre les processeurs.
- Classification basée sur l'interconnexion entre les processeurs les modules mémoire.
- Classification basée sur la nature caractéristique des processeurs.

Notre travail d'étude s'intéressera à la classification basée sur l'arrangement de la mémoire et la communication entre les processeurs.

1.1.1 Classification basée sur l'arrangement de la mémoire et la communication entre les processeurs

Les modèles les plus importants pris en compte par cette classification sont :

- Modèle à mémoire partagée.
- Modèle à mémoire distribuée.
- Modèle hybride.

1.1.1.1 Modèle à mémoire partagée

Ce modèle partage une mémoire globale constituée d'un ou plusieurs modules mémoire entre les différents processeurs. L'accès à ces modules mémoire est équitable entre les processeurs et chaque module mémoire est vu comme étant un seul espace d'adressage par tous les processeurs. Les modules mémoires peuvent être

centralisés ou distribués sur plusieurs processeurs. La figure ci-dessous schématise ce que nous venons d'expliquer [8] [9] :

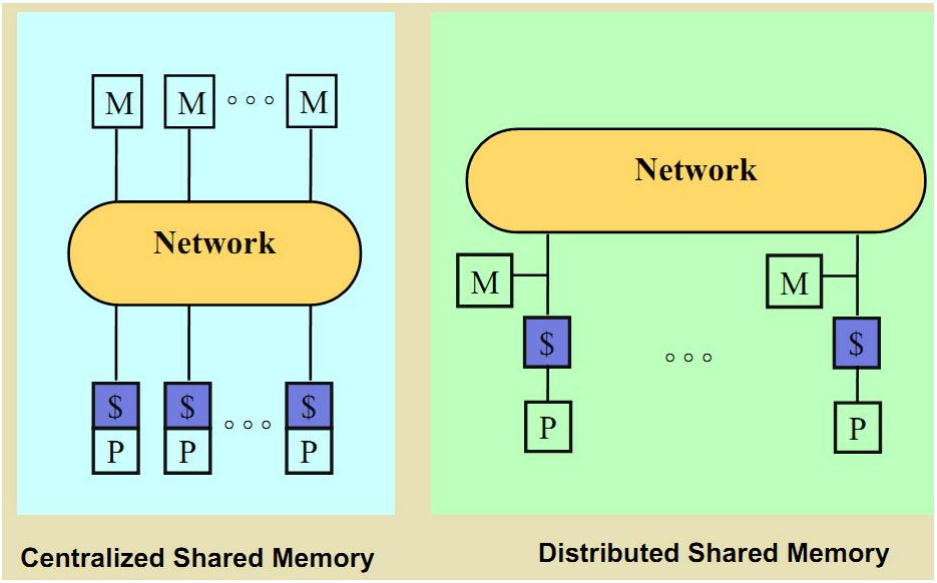


FIGURE 1.1 – Modèle d'architecture parallèle à mémoire partagée [9]

NB : $ est le coût d'accès du processeur à la mémoire.

1.1.1.2 Modèle à mémoire distribuée

Ce modèle est différent du précédent dans le fait que chaque unité de cette architecture est un calculateur complet intégrant à sa composition un processeur, une mémoire et un système d'entrée/sortie [8]. Un processeur ne peut avoir accès qu'à sa propre mémoire privée. Les communications entre les processeurs se font à partir d'opérations d'entrée/sortie dont résulte des messages circulant dans des bus à travers le réseau [8] [9]. Le schéma suivant schématise les précédentes explications :

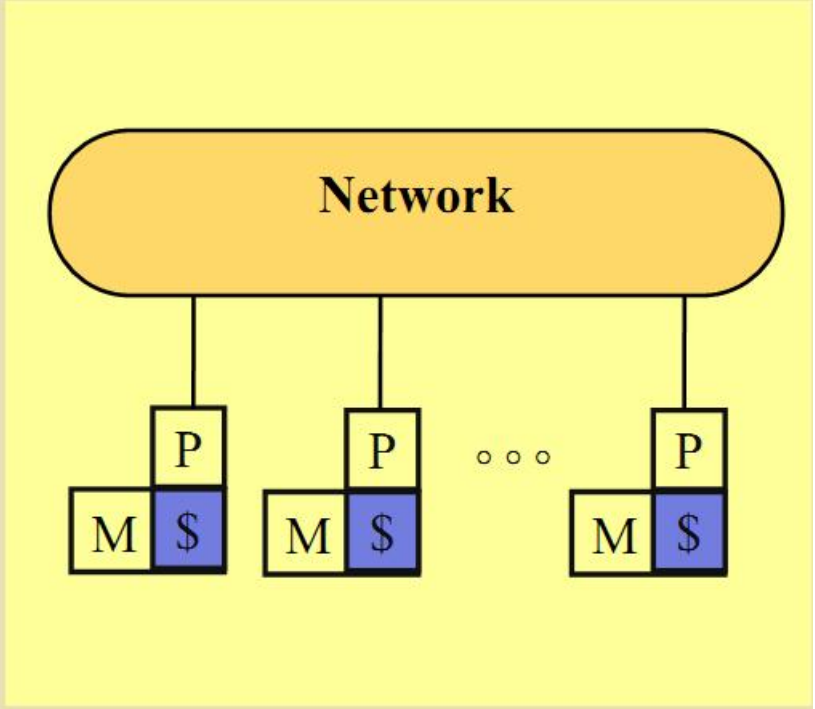


FIGURE 1.2 – Modèle d'architecture parallèle à mémoire distribuée [9]

NB : $ est le coût d'accès du processeur à la mémoire.

1.1.1.3 Modèle hybride

Ce modèle regroupe les caractéristiques des deux modèles précédents. Dans un modèle hybride le système est un ensemble de nœuds se constituant chacun d'une mémoire privée, d'un ou plusieurs processeurs(Clusters) et d'un système d'entrée/-sortie.Les nœuds se partagent une mémoire globale et peuvent communiquer entre eux par échange de messages à travers le réseau [8] [10]. Cette hybridation donne comme résultat une facilité de modification et une amélioration du temps de réponse d'une application en mémoire partagée et de cacher les mécanismes de communication complexe au programmeur [11]. Le schéma suivant donne une illustration de ce modèle :

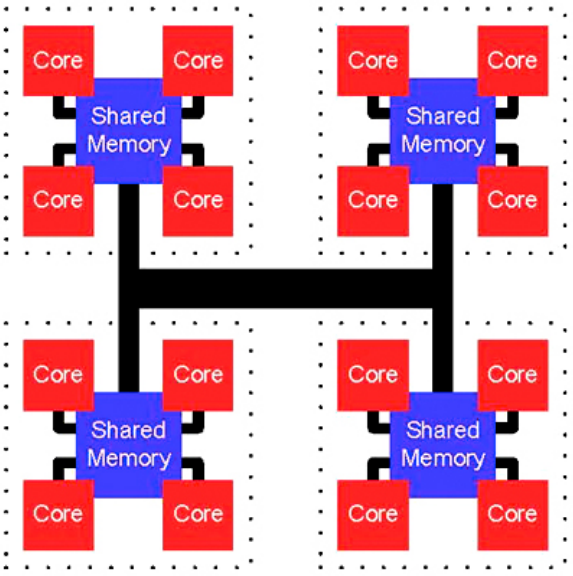


FIGURE 1.3 – modèle hybride d'une architecture parallèle [12]

1.2 Le calcul distribué

L'informatique distribué ou encore le calcul distribué, relie un nombre important d'ordinateurs, créant ainsi un système doté d'une puissance de calcul qui peut dépasser celle des supercalculateurs, généralement un calcul distribué est réalisé sur des clusters de calcul. Le calcul distribué est un domaine de recherche des sciences mathématique et informatique, qui fait face à des problèmes algorithmiques (distribution des donnée, ordonnancement, équilibrage de charge . . . etc.), système, logiciel . . . etc.

1.2.1 Modèle de déploiement

Plusieurs modèles ont été apparues, suivant ces derniers un calcul distribué peut être réalisé sur des grappes de calcul (clusters), les réseaux de stations de travail, les grilles de calcul et les grilles de PC (Desktop Grid).

1.2.1.1 Les grappes de calcul

Une grappe de calcul (cluster) est un ensemble d'ordinateurs reliés entre eux par un réseau généralement fiable, dont le but est d'effectuer des calculs complexes. Des dispositifs de sécurité sont met en place pour que La disponibilité de la grappe soit permanente (protéger la grappe contre les coupures d'électricité, utilisation d'un réseau fiable et rapide ...etc.). La grappe est généralement gérée d'une manière centralisé pour limité la communication avec le monde extérieur.

1.2.1.2 Les grilles de calcul

Une grille informatique est une composition de plusieurs ordinateurs sous réseau. Chaque utilisateur offre sa puissance de calcul et de stockage de données. Plusieurs domaines (biologie, astronomie, médecine ...etc) ont recoure aux grilles informatique afin de résoudre certains problèmes qui demande une puissance de calcul important. Plusieurs projets de grille sont met en place, comme exemple [26] :

- le projet EGEE (Enabling Grids for E-sciencE), visant à mettre en place l'infrastructure d'une grille pluridisciplinaire, disponible partout en Europe, 24 heures sur 24.
- le projet LCG (LHC Computing Grid), l'infrastructure de grille dédiée aux expériences de l'accélérateur de particules LHC (ou Large Hadron Collider) du CERN.

1.2.1.3 Le calcul par volontariat (les grilles de PC)

Le calcul par volontariat utilise la puissance d'un grand nombre d'ordinateur mis à disposition par des volontaire dans l'objectif est de participer à des calculs intensifs.

Les grilles de PC se distinguent des autres types de grilles par la capacité de calcul dont elles disposent. En général les grilles de calcul sont composées d'un ensemble de cluster ou de supercalculateur (de 10 à 100) et qui sont connectés par un réseau

fiable et rapide, or les grilles de pc utilisent les ordinateurs personnels des volontaires allant de 10000 à 1000000 ordinateurs. De plus, contrairement aux grilles de PC, les autres types de grille nécessite des systèmes de gestion et de réservation de ressource, parmi les intergiciels les plus utilisés nous citons, Globus, Logion, Unicore ... etc.

Les ordinateurs des volontaires sont reliés par le réseau internet, ce qui permet de former une grille en associant un très grand nombre d'ordinateur hétérogènes. Depuis leurs apparitions (les grilles de PC) à ce jour plusieurs projets scientifiques les utilisés, nous citons comme exemple climateprediction@home, seti@home ... etc.

1. **L'approche centralisée :** Le modèle centralisé est basé sur l'architecture client/serveur. Dans ce modèle les clients peuvent participer à plusieurs projets en même temps, ces derniers sont formé de plusieurs applications à exécuter, ils sont souvent hébergés sur un même serveur ou sur des serveurs distincts.

   Les clients qui participent à un projet donné, récupèrent des tâches depuis le serveur, et ils lui renvoient les résultats à la fin de l'exécution, ce dernier (le serveur) valide les résultats reçus suivant une stratégie de validation et les sauvegardes dans une base de données.

   Plusieurs intergiciels sont basés sur ce paradigme tel que BOINC (Berkeley Open Infrastructure of Network Computing), qui est l'un des systèmes de calcul volontaire à grand échelle les plus utilisé et sophistiqué au monde. L'inconvénient majeur d'une telle approche est que si un serveur tombe en panne c'est tous les projets qu'il héberge qui sont paralysé, et leurs calculs ne progresses plus [17].

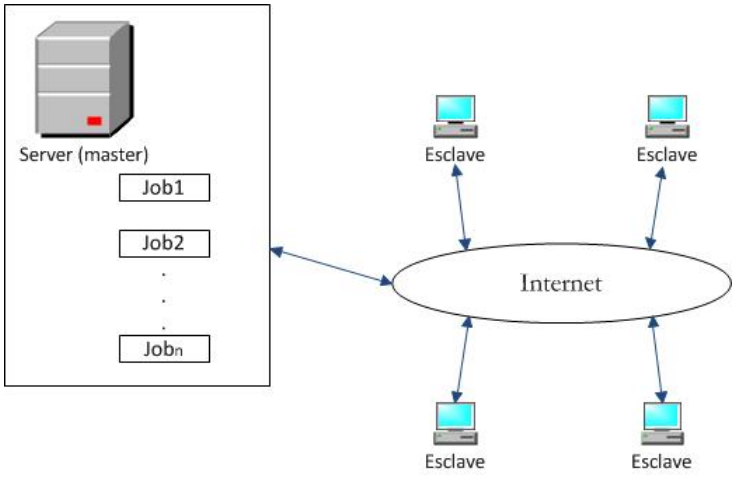


FIGURE 1.4 – approche centralisée

2. **L'approche semi-centralisée :** se distingue de l'approche centralisée dans la soumission des tâches. Le principe de cette approche est simple, en effet, un client voulant exécuter son application, l'envoi au serveur, ce dernier envoi les tâches au volontaire par l'intermédiaire d'un coordonnateur centralisé et toutes les machines disponible participent à cette exécution [17] [27].

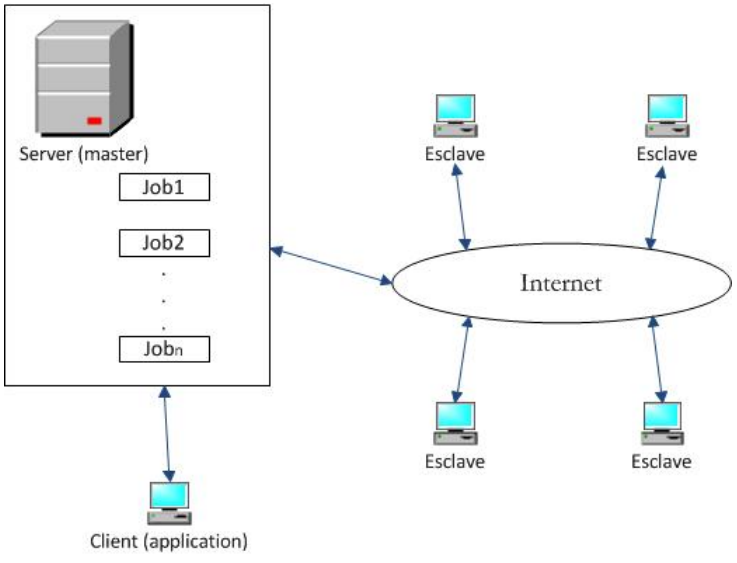


FIGURE 1.5 – approche semi-centralisée

3. **L'approche distribuée :**Les systèmes pair-à-pair ou P2P considèrent que toutes les machines sont égales. Généralement ce type d'architecture est utilisée pour l'exécution des applications parallèle ou échange de fichier.

   Nous distinguons trois types des systèmes P2P, P2P pur, P2P hybride et le P2P à base de DHT (distributed hash table).

   Zorilla est l'un des intergeciels qui utilise cette approche, il permet d'utiliser plusieurs clusters comme s'il s'agissait d'un ensemble uniforme de calculateurs. L'ordonnancement des tâches, est complètement différent des deux systèmes introduits en haut. C'est le nœud émetteur qui se charge de trouver d'autres nœuds qui vont participer à l'exécution de son application. Le nœud demandeur transmit un message à tous ses voisins du réseau qui eux même le propagent à leur voisins. Un nœud recevant un message et s'il a suffisamment de ressources pour participer à ce calcul, informe le nœud d'origine. Si le nombre de nœuds disponible est suffisant, la distribution des tâches sera faite et l'exécution commence. [17] [27]

Conclusion

Dans ce chapitre nous avons présenté briévement les différente architecture parallèle et distribué, dans le prochain chapitre nous allons présenter un état de l'art exposant ainsi notre problématique et les différants travaux qui ont été fait dans ce domaine.

2

Etat de l'art sur l'implémentation des AG sous des plateformes de calcul volontaire

Introduction

La puissance de calcul est une contrainte que les scientifiques et ingénieurs doivent gérer quotidiennement pour assurer le bon déroulement de leurs travaux et cela en plusieurs domaines. L'apparition des ordinateurs a su faire évoluer les sciences d'un grand pas et cela en offrant plus de puissance de calcul mais la limite de ces derniers a fait que le besoin en puissance de calcul soit toujours présent. En effet ce besoin n'a cessé et ne cesse de s'accroitre au fil du temps.

Parmi les domaines nécessitant le plus de puissance de calcul, l'optimisation combinatoire. En effet trouver la meilleure solution réalisable pour un problème donné n'est pas toujours évident pour ne pas dire impossible dans des délais raisonnables quand l'ensemble des solutions de bases réalisables dépasse un certain nombre. On parle alors de problèmes dis NP-Difficile, pour ce genre de problème est nécessaire une très grande quantité en matière de puissance de calcul afin de ne pas dépasser un temps de résolution dit acceptable.

L'exactitude des résultats est l'un des critères fondamentaux des mathématiques, cependant en optimisation combinatoire, résoudre un problème NP-Difficile avec exactitude est quasi impossible surtout lorsque le problème en lui-même est d'un degré élevé. On doit alors faire appel à des méthodes approchées afin d'optimiser au mieux les résultats obtenus.

Beaucoup de scientifiques se sont alors penchés vers les algorithmes évolutionnaire. S'inspirant de la théorie de l'évolution, ces algorithmes ont la propriété d'utiliser itérativement des processus aléatoires afin de faire évoluer un ensemble de solutions à un problème donné et ainsi trouver les meilleurs résultats.

Cependant, on a eu beau essayer, le besoin de puissance de calcul est toujours resté présent, plus encore il s'est accru. C'est alors que les scientifiques de ce domaine ont fait preuve d'ingéniosité en mettant en place des machines dotées de plusieurs processeurs et capable d'exécuter un large nombre de calculs en parallèle.

Mais ne nous bernant pas d'illusion, tout a un prix et il se trouve que le prix de ces infrastructures soit très élevé pour des machines dont les limites sont très vite atteintes.

A cet effet, nous pouvons nous pencher vers l'une des autres techniques développés par nos chercheurs qui est le calcul distribué. Ainsi notre système de résolution sera doté d'un nombre indéterminé de machines fonctionnant en parallèle et offrant leurs ressources pour la résolution du problème. Nous anéantissons ainsi la limite en ressources et les coûts élevés auxquels nous faisions face en utilisant une machine

multi-coeurs.

De cet aspect ont été développé des plateformes de calcul volontaire tel que BOINC, XtremeWeb et Condor qui utilisent les ressources de clients volontaires via le réseau internet pour résoudre des problèmes de calcul dans différent domaine : Biologie, Physique, Chimie, etc.

Pour avoir les meilleurs résultats, des infrastructures matérielle et logicielle ont été mise en oeuvre, nous nous intéresserons le plus au grilles de calcul qui sont une infrastructure virtuelle qui exploite la puissance de calcul de milliers d'ordinateurs afin de donner l'illusion d'un ordinateur virtuel très puissant.

Des méthodes algorithmiques plus complexes ont été mises en oeuvre afin de subvenir aux différents besoins du calcul distribués. Dérivant de la famille des algorithmes évolutionnaires on présente les algorithmes génétiques(AGs). Sans détailler leur fonctionnement, ils sont utilisés afin d'obtenir une solution approchée à un problème d'optimisation.

2.1 Problématique

Les problèmes d'optimisation résident dans plusieurs domaines industriels et scientifique tel que la planification des processus industriels dans les chaines de fabrications, assemblage de fragments ADN, problème d'affectation quadratique ... etc. un grand nombre de ces problèmes sont classés comme NP-difficile, ce qui rend leurs résolution un vrai défi pour l'optimisation combinatoire.

Deux grandes méthodes de résolution sont apparues, les méthodes exactes et les méthodes approchées. Les méthodes exactes visent à trouver une solution exacte au problème traité ce qui est impossible dans le cas des problèmes complexes. Et les méthodes approchées qui tente de se rapprocher du mieux de l'optimum, les métaeuristiques est l'une des variétés des méthodes approchées qui englobe la famille des algorithmes évolutionnaires plus exactement dans notre travail nous utiliserons les

algorithmes génétiques pour la résolution d'un problème de permutation.

L'utilisation des algorithmes génétiques sur un problème de permutation à large échelle, nécessite une puissance de calcul importante pour aboutir à des résultats s'approchant de l'optimum. Afin d'obtenir la puissance nécessaire plusieurs modèle de parallélisation d'AGs ont été apparus (master/slave, cellulaire, insulaire et hybride), pour pouvoir parlléliser l'execution de l'AG sur plusieurs machine.

Des infrastructures parallèles et distribués ont été mises en place pour permettre l'exécution parallèle des applications demandant de grande capacité de calcul. Le calcul volontaire est l'une des infrastructures distribués qui a démontré ses épreuves vue le nombre important de projets scientifique et industriels qui l'utilise, nous traitons ici l'un des problèmes de permutations (flowshop) sur une plateforme de calcul volontaire en parallèlisant l'exécution de l'AG sur les volontaire.

Notre objectif est d'implémenter un projet BOINC pour la parallélisation d'un algorithme génétique, afin d'aboutir à cet objectif nous devons faire :

- Une étude profonde des systèmes de calcul volontaire plus particulièrement BOINC.
- Une implémentation d'un algorithme génétique avec integration d'une méthode de la recherche locale Hill Climbing, pour le problème du flow hop, basé sur le Framework open source paradisEO.
- Une proposition d'un modèle de parallèlisation de l'AG qui convient avec l'architecture de BOINC.
- Une Implémentation d'une application BOINC afin de distribuer le calcul réalisé par l'AG
- Réalisation d'expériences sur le système BOINC installé.

2.2 Les méthodes de résolution des problèmes d'optimisation combinatoire

Résoudre un problème dans un ensemble discret et fini semble facile en théorie, en effet il suffit d'essayer toutes les combinaisons possible jusqu'à trouver une solution satisfaisant les contraintes du problème. Mais en pratique l'énumération de toutes les solutions peut prendre trop de temps avant de trouver la solution voulu. De ce fait deux grandes familles des méthodes d'optimisation sont apparues, les méthodes exactes et les méthodes approchées dans ces dernières nous allons présenter l'une des variantes des méta-heuristiques nommée les Algorithmes génétiques souvent noté AG.

2.3 Les Algorithmes génétiques

Les algorithmes génétiques est l'une des méta-heuristiques qui ont prouvé leur efficacité. Les Algorithmes génétiques sont utilisés pour résoudre certains problèmes d'optimisation combinatoire dits de class NP-difficile, leurs aspect aléatoire de résoudre ces problèmes a eu souvent plus de chance de se rapproché de la solution optimale. Les AGs sont apparus au début des années 70 (1975), Holland a introduit cette théorie basée essentiellement sur l'évolution naturelle des êtres vivant (exposée par Charles Darwin en 1960), en 1989, Goldberg ajouta à la théorie de Holland deux principes qui ont rendu l'utilisation des algorithmes génétiques encore plus efficace. Les algorithmes génétiques explorent un espace de recherche fini et discret afin de tomber sur la solution qui se rapproche mieux de la solution optimale. Leurs principe est de faire évoluer une population d'individus en leurs subissant des reproductions et des mutations afin d'avoir une nouvelle population (les fils), on change alors les parents les moins efficace par leur fils. L'exploration de l'espace de recherche dans certains problèmes prend un temps important, une parallélisation de l'exécution des AGs est alors inévitable. [2]

2.4 Le calcul volontaire

de multiples architectures parallèles (supercalculatteur) et distribués(les grille de calcul, le calcul volontaire ...etc) existent, mais leur objectif est toujours de réduire de temps d'exécution. le principe de calcul par volontariat est d'utilisé les ressources non exploités des ordinateurs se trouvant sur le net, cette idée va permettre d'avoir une puissance de calcul important vu le nombre d'ordinateurs connecté à internet de nos jours.[17]

2.5 *Berkeley Open Infrastructure of Network Computing*(BOINC)

BOINC est l'une des plateforme basées sur le calcul volontaire (*Grid PC*), developpée par des chercheurs de l'université de Berkeley. BOINC vous permet de manipuler la puissance de votre PC utilisé par les projets aux quels vous avez attaché le manager. Son architecture suit le même principe que celle du client/serveur, plusieurs serveurs de projets peuvent exister sur le net, ainsi, vous pouvez participer à plusieurs projets en même temps (pour plus de détail voir chapitre 4). [17]

2.6 Étude de l'existant sur l'implémentation des AG sous une plateforme de calcul volontaire

Les algorithmes génétiques ont longtemps été un sujet de recherche afin de résoudre les problèmes d'optimisation les plus ardus. Des chercheurs de l'université d'Extremadura, Espagne ont proposé en 2011 une caractérisation des tolérances aux pannes des algorithmes génétique dans un Desktop Grid system [28]. Selon ces travaux on peut réduire l'impact d'une panne en :

– Augmentant la taille de la population de la première génération afin d'indemniser les clients qui vont être perdu au cours des générations.

– Utilisant le phénomène de désabonnement d'un hôte : une ressource peut être disponible après avoir été indisponible.

Ces solutions sont certes d'un grand apport, mais aucun modèle de parallélisassions n'a été proposé pour les mettre en pratique, et le problème de distributions des tâches entre les différents clients subsiste toujours.

Par ailleurs les résultats des recherches que l'ont peut retrouver en [25] démontre que nous pouvons rester dans une logique séquentielle tout en exécutant un algorithme génétique de manière parallèle. Cette approche est mise en pratique grâce au modèle insulaire qui est décris en détail en [10].

Cependant une question perturbante que nous somme amené à nous poser et que nous formulons de la manière suivante : Pouvons nous implémenter de telles méthodes, qui ont été destiné au départ à une architecture en Grid, sous une plateforme telle que BOINC qui est à la base une architecture client/serveur.

Nous retrouvons une application antérieure des plus intéressantes en [25], En effet les auteurs de ce dernier article ont pu démontrer prodigieusement que l'on pouvait facilement implémenter un modèle de distribution, destiné à la base à être implémenté sous une architecture autre que client/serveur, sous une plateforme basé sur un tel type d'architecture comme BOINC.

Conclusion

Dans ce chapitre nous avons vu les travaux les plus importants auxquels notre travail se réfère, nous avons aussi introduit rapidement les algorithmes génétiques, le calcul volontaire et une présentation de la plateforme BOINC, que nous détaillerons dans les futures chapitres.

3

Les algorithmes génétiques

Introduction

Les méthodes d'optimisations combinatoires se composent de deux grandes catégories : exacte et approchée.

Les méthodes de la première catégorie sont classées comme des méthodes itératives qui convergent vers un minimum local. Les méthodes exactes ont prouvé leurs efficacités quand l'évaluation de la fonction objective est rapide. Dans la plus part du temps les problèmes d'optimisations nécessitent un temps de calcul important, les méthodes approchées semblent les mieux adaptées dans ce genre de problèmes.

Les méthodes approchées (méta-heuristique) se basent généralement sur l'aléatoire pour mieux explorer l'espace de recherche, parmi ces méthodes figure la famille des algorithmes évolutionnaires qui s'inspirent de la théorie de l'évolution pour ré-

soudre divers problèmes, généralement des problèmes d'optimisation combinatoire. Les Algorithmes génétiques souvent noté AG font partie des algorithmes évolutionnaires, les AGs simulent le mécanisme de reproduction des êtres vivant dans la nature, leur robustesse et leur capacité d'offrir plusieurs solutions considérablement bonnes d'un problèmes font d'eux une des approches les plus utilisées pour la résolution des problèmes dits NP-Hard.Le schéma suivant décrit les déférentes approches de résolution des problèmes d'optimisations [1] :

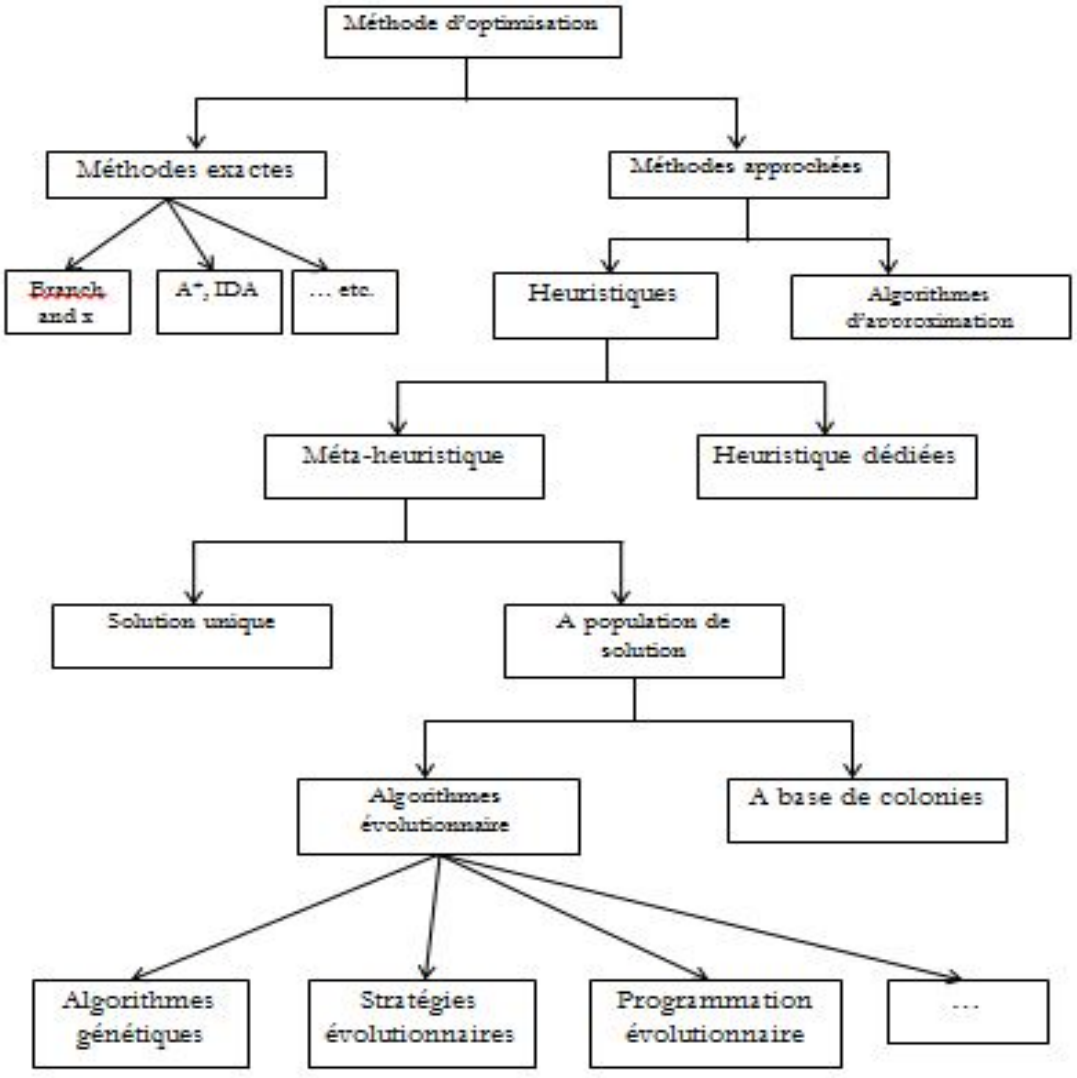


FIGURE 3.1 – Shémas illustrant les différentes méthodes de résolution

3.1 Historique

Les algorithmes génétiques tirent leurs nom de la théorie de l'évolution des espèces introduite par Charles Darwin en 1960 dans son livre intitulé *l'origine des espèces au moyen de la sélection naturelle ou la lutte pour l'existence dans la nature*[2], il expliqua alors que l'évolution des espèces suit quatre lois à savoir :

- La loi de croissement et de reproduction.
- La loi d'hérédité.
- La loi de variabilité.
- Loi de la multiplication des espèces qui a pour conséquence la sélection naturelle.

En 1975, Holland introduisit la théorie des algorithmes génétiques basés sur la théorie de l'évolution naturelle des espèces. En 1989 Goldberg publia un ouvrage dans lequel il ajouta à la théorie de Holland les deux principes suivant :

- Un individu est lié à un environnement par son code d'ADN.
- Une solution est liée à un problème par son indice de qualité.[2]

3.2 Vocabulaire

Le vocabulaire utilisé par cette méta-heuristique est tiré directement de la génétique, chaque population (génération) est constituée d'un ensemble fini d'individus, ou chaque individu est un chromosome. Un chromosome est considéré comme un ensemble de gènes, ces derniers représentent les caractéristiques d'un individu. Un allèle est la valeur prise par chaque gène (caractéristique).

La Recombinaison ou la reproduction est l'opérateur naturel de l'évolution, techniquement deux individus se reproduisent en mixant une partie de leurs gènes (cross-over).

La mutation est une autre technique qui permet d'obtenir de nouveaux individus. Elle consiste en un changement de la valeur d'un gène d'une manière aléatoire pour

un chromosome donné. [3]

3.3 Principe de fonctionnement

Un ensemble de solutions appelé population d'individus est évalué par les AGs, un codage est choisi pour décrire chaque individu, le choix d'un bon codage réduit la complexité final de l'algorithme implémenté. Un individu (chromosome) est une solution possible pour le problème à résoudre. À chaque individu est affectée une fonction de fitness qui représente la fonction objective à optimiser. Les meilleurs individus sont ensuite sélectionnés pour se reproduire en leurs appliquant des croisements (cross-over) et des mutations avec deux probabilités Pc et Pm, fixées à l' avance, de ce fait une nouvelle population de solutions apparaît. Ce processus se poursuit jusqu'à atteindre un certain critère d'arrêt, on parle alors de la convergence de l'algorithme.

L'algorithme génétique de base est définit par les paramètres suivants : T initialisé à zéro définit le numéro de la génération à laquelle appartient une population P ; une fonction *évaluer(P(T))* qui permet de calculer la valeur de la fonction de fitness de chaque individu ; une fonction *sélectionner(P(T))* pour sélectionner les meilleurs individus de la population P(T) qui participeront à la génération de la population suivante ; *croisement(P(T))*, *Mutation(P(T))* qui permettent d'effectuer respectivement un croisement ou une mutation sur les individus déjà sélectionnés et *remplacer(P(T))* pour remplacer les anciens de P(T) par leurs descendants, enfin une nouvelle génération est créé. L'algorithme de base est comme suit [4] :

```
Début
    Entrée : instance du problème.
    Sortie : une solution.
    T=0 ; initialisation de la population (T)
    Evaluer (p(T)) ;
    Tant que (critère d'arrêt non satisfait) Faire
        T=T+1 ;
        Sélectionner (p(T)) ;
        Croisement (p(T)) ;
        Mutation (p(T)) ;
        Remplacer (P(T))
        Evaluer (p(T)) ;
    Fin ;
Fin.
```

3.3.1 Le codage

Pour appliquer les AGs à un problème spécifique, nous devons d'abord définir une représentation génétique appropriée pour la solution. Nous devons trouver un moyen d'encoder n'importe quelle solution du problème dans un chromosome. Cette structure partagée par tous les chromosomes est appelé la représentation génétique ou codage, deux type de codages s'offrent à nous [2] [3] :

– **codage binaire :** dans ce premier chaque individu (chromosome) est représenté par un tableau binaire ou chaque case (gène) peut prendre l'une des valeurs (allèle) suivante 0,1. Un individu est représenté comme suit :

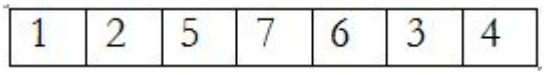
Dans ce type de codage on utilise généralement la distance de Hamming entre

1	0	0	1	1	1	0

deux solutions, ceci compte le nombre des bits différents entre les deux. Un inconvénient de cette méthode est le fait d'avoir deux éléments voisins en terme de distance de Hamming ne code pas nécessairement deux éléments proches

dans l'espace de recherche. Ce dernier peut être évité en utilisant le codage de Gray qui se définit comme suit : entre deux éléments n et n+1, qui sont voisins dans l'espace de recherche un seul bit diffère.

– **codage réel :** : un individu est représenté par un tableau d'entiers, l'ensemble de définition est l'ensemble des entiers naturel 1..n tel que n est la taille du problème à résoudre :



3.3.2 Fonction de fitness

Après avoir choisi un codage des chromosomes une phase d'évaluation des individus (chromosomes) se présente, afin de donner une valeur à leurs fonctions de fitness (fonction objectif à optimiser). Cette fonction exprime sous forme mathématique les objectifs à atteindre, sa valeur reflète l'efficacité de la solution, la persistance de cette dernière dépend de la formulation de cette fonction [3].

3.3.3 Population

La population est l'ensemble des solutions (individus) du problème. Au départ une population initiale est choisi, soit d'une manière aléatoire soit en suivant une méta-heuristique pour avoir une population initiale diversifié ce qui peut aider l'algorithme à atteindre l'optimum plus rapidement [3].

3.3.4 La sélection

L'opérateur de sélection est chargé de sélectionner les individus qui vont servir de parents pour créer la nouvelle génération. La probabilité qu'un individu soit sélectionné est généralement liée à son efficacité (la valeur de sa fonction de fitness). Il existe quatre méthodes de sélection différentes [2].

3.3.4.1 La sélection par loterie ou roulette

La probabilité qu'un individu soit sélectionné est relativement liée à la valeur de sa fonction de fitness, plus un individu est adapté au problème plus il a de la chance d'être sélectionné, en commence par tourner la roue, a son arrêt, l'individu sur lequel la balle s'est arrêté sera sélectionné.

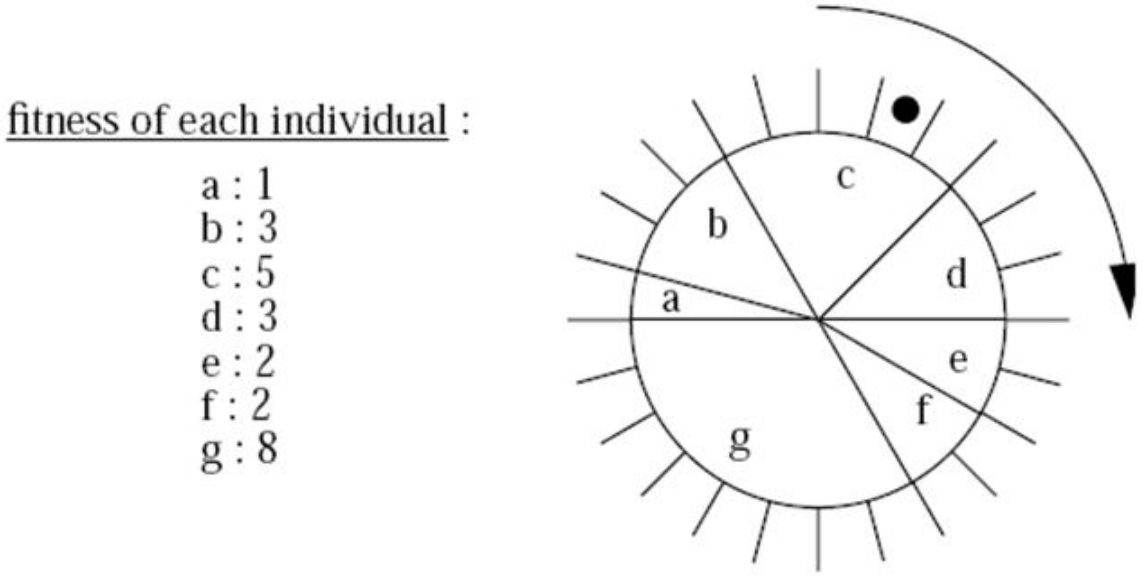


FIGURE 3.2 – La sélection par roulette [3]

Cette méthode présente plusieurs inconvénients, en effet lors de la sélection, les parents qui seront sélectionnés peuvent être des individus avec une mauvaise valeur de leurs fonctions de fitness, ce qui donne une nouvelle population avec des parents pratiquement tous mauvais, alors que le principe des algorithmes génétiques veut que les meilleurs individus soient sélectionnés pour créer la nouvelle génération. Dans d'autres circonstances, un individu peut être dominant en étant localement supérieur ce qui entraine une perte de diversité, on se retrouve face à un problème connu sous le nom de *la convergence prématurée*, l'évolution se met à stagner et on restera sur un optimum local, sans jamais atteindre la solution optimale.

3.3.4.2 La sélection élitiste

Le principe de cette méthode est comme suit, un trié décroissant selon la fitness des individus sera fait, par la suite on sélectionne les n individus qui vont nous servir de parents pour générer la prochaine population. Cette méthode nous amène généralement à une convergence prématurée, plus rapidement que la première. Ceci est dû à la pression de la sélection, où il n'y aura presque ni de diversité ni de variance des individus du point de vu de la sélection.

3.3.4.3 La sélection par tournoie

Un tirage avec remise de deux individus se fait, celui ayant une fitness plus élevé sera retenu avec une probabilité p (0.5<p<1). Ce processus se poursuit jusqu'à atteindre les n individus qui nous serviront de parents pour la prochaine génération. La pression de la sélection est relative selon la valeur de p. généralement cette méthode nous permet d'avoir des résultats plus satisfaisant que les autres.

3.3.4.4 La sélection universelle stochastique

Cette méthode consiste à sélectionner les individus en se basant sur la valeur attendue e_n d'un individu x_n tel que : e_n=[f_n/($\sum_{j=1}^{N} f_j$)]*N.
Où N est le nombre d'individus et f_n est la valeur de la fonction de fitness de l'individu x_n. Sans entrer dans les détails cette méthode est approuvée peu efficace vu qu'elle introduit peu de diversité, et donc elle conduit l'algorithme a une convergence prématurée plus rapidement que toutes les autres méthodes.

3.3.5 Le cross-over (croisement)

Après avoir sélectionné les meilleurs individus de la population comme parents, la phase de la reproduction (croisement) s'enchaine, en commençant par Sélectionner deux solutions P1 et P2 de l'ensemble des parents, ensuite un tirage aléatoire d'une (ou plusieurs) position (s) inter-gène (point de croisement) sera fait dans chacun

des parents. Pour finir on échange les sous-chaines produites de chacun des parents avec une probabilité Pc, ce qui produit deux nouvelles solutions C1 et C2 appelées enfants. Trois mécanismes de cross-over existent vis-à-vis les points de croisement [2] :

3.3.5.1 Un seul point de croisement

Un seul point de croisement sera désigné aléatoirement, on échange en suite les deux sous-chaines pour avoir les fils comme le montre la figure suivante :

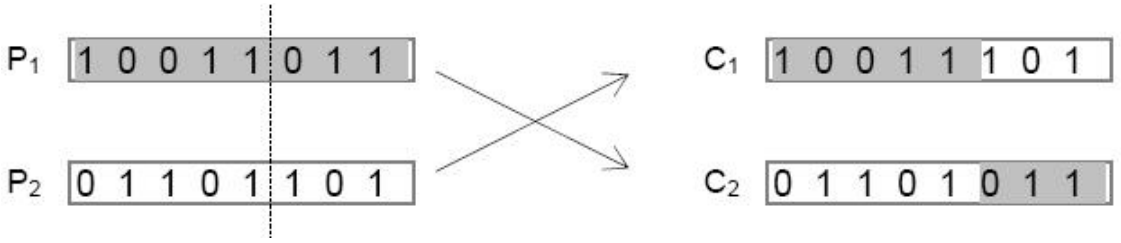


FIGURE 3.3 – le croisement en un point [5]

3.3.5.2 Deux points de croisement

Deux points de croisement seront fixés sur chacun des parents, ce qui va produire plus de sous-chaines qui seront échangées pour produire les deux fils comme l'illustre la figure qui suit :

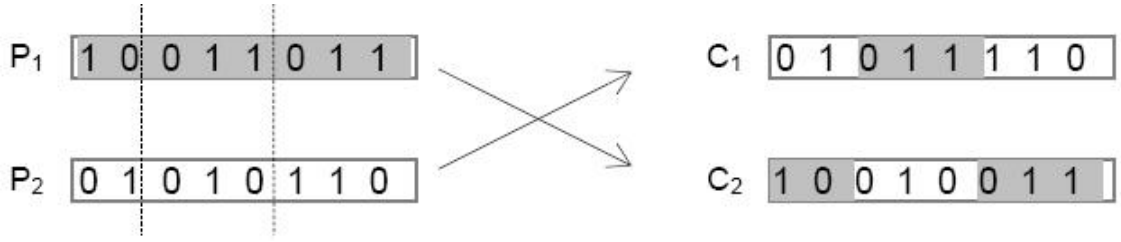


FIGURE 3.4 – croisement en deux points [5]

3.3.5.3 Croisement uniforme

Dans le cas d'un problème modélisé en codage binaire, le croisement uniforme semble très utile. Chaque gène d'un enfant est créé en copiant le gène correspondant de l'un des parents, en suivant un masque ayant la même langueur que les parents, généré aléatoirement. Pour chaque couple de solutions (parents) un masque binaire est généré, lorsque le masque à pour valeur 1, le gène du premier enfant est copié du premier parent et le gène du deuxième enfant du deuxième parent, et vis vers sa, avec une probabilité de croisement Pc. Comme l'illustre la figure suivante :

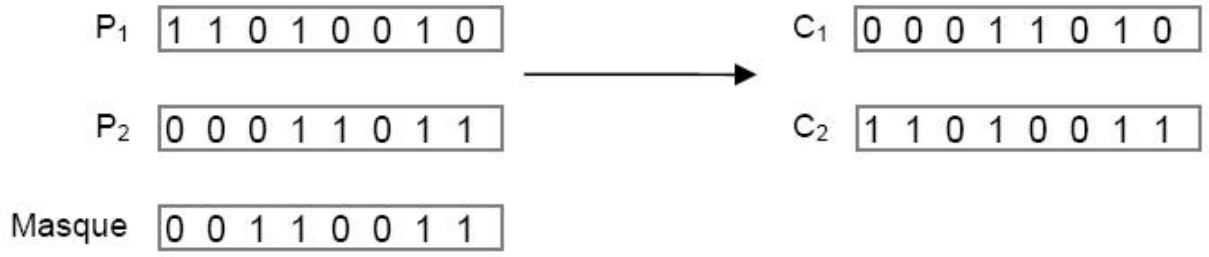


FIGURE 3.5 – croisement uniforme [5]

3.3.6 La mutation

La mutation est définie comme étant le changement de la valeur d'un gène avec une probabilité Pm. Dans le cas d'un codage binaire cela consiste simplement en l'inversion de la valeur du bit en question.

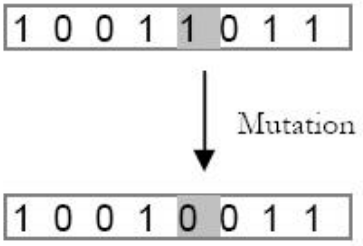


FIGURE 3.6 – Mutation [5]

Selon la probabilité Pm, la mutation modifie aléatoirement les caractéristiques d'un individu, ce qui permet d'introduire plus de diversité au sein de la population courante. L'opérateur de mutation nous offre plusieurs avantages entre autres on peut citer :

- Il garantit la diversité de la population.
- Il permet de limiter les risques de la convergence prématurée. [2]

3.3.7 L'opérateur de remplacement

Le remplacement est l'opérateur qui consiste à réintroduire les descendants obtenus, dans la nouvelle population. Cette opération peut être faite d'une manière totale ou partielle. Généralement les meilleures solutions remplacent les plus mauvaises, on obtient alors, une amélioration de la génération précédente. Lorsque la nouvelle génération n'est constituée que des enfants on parle de remplacement générationnel. Un autre mécanisme de remplacement connu sous le nom de remplacement élitiste, le principe de ce dernier, garde l'individu le plus performant d'une génération à la suivante [3].

3.3.8 La convergence d'un AG (critère d'arrêt)

L'évolution d'un AG est un processus qui ne s'arrête jamais. Lors de son évolution, on ne peut pas s'assurer qu'un AG s'arrête sur la solution optimale. Un AG est une méta-heuristique donc dans certain cas on n'arrive même pas à trouver l'optimum. Ainsi l'évolution de l'AG doit converger (s'arrêter) à un moment donné. Trois critères d'arrêt qu'on peut utiliser s'offre à nous :

- L'AG s'arrête après un certain nombre de génération, mais ceci peut être couteux en terme de temps d'exécution si le nombre d'individus a évalué dans chaque population est important.
- L'AG s'arrête lorsque la population n'évolue plus.
- L'AG s'arrête si la valeur de la fonction de fitness de l'un des individus dépasse

un seuil fixé au départ.

Comme les AGs sont basé sur l'aléatoire, l'AG aura un comportement différent et des résultats différents, pour des paramètres et populations identiques. De ce fait pour évaluer correctement un AG, il faut l'exécuter plusieurs fois et analysé statistiquement les résultats obtenus [2] [3].

3.3.9 Les paramètres d'un AG

3.3.9.1 La taille de la population

La convergence d'un AG est liée à la taille de la population à évoluer. Lorsque celle-ci est très grande, la diversité augmente ce qui diminue la convergence de l'AG et le temps d'exécution augmente, dans ce cas la recherche de l'optimum risque de tourner au rond et affecté l'efficacité de l'AG. Par contre si la taille de la population est très petite, l'AG risque de converger rapidement vers un optimum local. La taille de la population doit être moyenne (entre 25 et 100) [6].

3.3.9.2 Le taux de croisement

Il détermine le nombre d'individus qui seront croisés parmi ceux qui remplaceront l'ancienne population. Le croisement s'applique avec une probabilité Pc, si cette valeur est très élevée, de nouvelles structures (individus) sont introduit dans la prochaine génération. Dans cas contraire (la valeur de Pc est très basse), l'évolution de la population risque de stagner. En général, Pc varie entre 0.25 et 0.7 [6].

3.3.9.3 Le taux de mutation

La mutation s'applique avec une probabilité Pm. Si cette valeur est élevée la recherche devient purement aléatoire, la population est diversifiée et l'AG perd de son efficacité. Dans le cas où la valeur de Pm est très petite, il y aura un manque de diversification et l'évolution risque de stagner. La valeur de Pm généralement comprise entre 0.001 et 0.0001 [6].

3.4 Les algorithmes génétiques parallèles

Il a été prouvé que l'efficacité d'un AG pour trouver une solution optimale est déterminée par la taille de la population évaluée. Avec une population importante et plus de diversité l'AG peut explorer un espace de recherche important et avoir un optimum global du problème traité. Mais évaluer une population importante nécessite plus de puissance de calcul ainsi que de mémoire de stockage.

L'un des avantages majeurs des AGs est leurs structure de base, qui leurs permet d'être parallélisés plus facilement. En effet toutes les opérations génétiques en particulier l'évaluation des individus (calcul de la valeur de la fonction de fitness), sont hautement parallélisable, sauf les phases de sélection et de remplacement qui dépendent de l'architecture parallèle suivit.

Plusieurs modèles de parallélisation sont apparu offrant ainsi plus d'avantages et de facilité de trouver l'optimum global pour un AG. Les quatre modèles les plus connus sont : le modèle maitre/esclave, modèle avec migration qui se compose de : gros grain (îlot/insulaire) et grain fin (cellulaire), le modèle hybride [3] [7].

3.4.1 Le modèle maître/esclave

Ce modèle est constitué de deux entités, le maitre et les esclaves, comme le décrit la figure 7. Dans ce modèle une seule population est évoluée, cette dernière réside dans le programme master (maître).

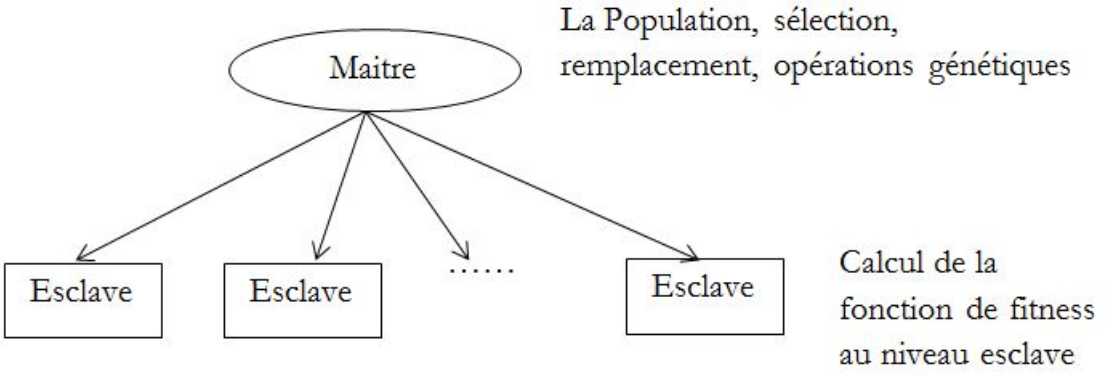


FIGURE 3.7 – Le modèle maître esclave

Le maître qui dispose de la population à faire évoluer, gère l'AG (sélection, remplacement et opérateurs génétiques) et on fait parallélisé la phase d'évaluation des individus qui est la plus couteuse dans un AG.

À premier temps, La station maître envoi le calcul des fonctions de fitness aux stations esclaves, une fois fait les esclaves renvoient les résultats au maître qui continu l'exécution du reste des phases de l'AG.

De cette manière le comportement de l'AG parallèle est exactement comme celui en séquentiel, sauf que le temps d'exécution de l'AG sera réduit grâce a la parallélisation de la phase d'évaluation. Ce modèle nous permet de faire évoluer une population importante et donc d'avoir un AG plus efficace et avoir un optimum global du problème traité.

Deux variante de ce modèle se présente : le synchrone où le maître attend les valeurs des fonctions de fitness de tous les individus, avant de passer à la prochaine génération. L'asynchrone où le maître n'attend pas que toutes les valeurs soient arrivés [7].

3.4.2 Modèle avec migration

Dans ce modèle nous avons des sous-populations différentes, et un nouvel opérateur appelé *migration* est introduit. La population principale est divisée en plusieurs

sous-populations et les individus d'une même population peuvent se reproduire.

La migration est l'opérateur par lequel les meilleurs individus sont échangés entre les populations, ce qui va apporter un nouveau matériel génétiques ainsi on aura plus de diversité. La migration peut se faire d'une population vers n'importe quelle autre, dans ce cas en parle de modèle *insulaire* ou *îlot* (gros grains). Dans le cas où la migration se fait uniquement entre les populations voisines alors on parle de modèle *cellulaire* (grains fins) [7].

3.4.2.1 Modèle îlot ou insulaire (gros grains)

Dans un tel modèle l'algorithme génétique sera distribué, ce modèle est utilisé généralement dans les réseaux hétérogènes. La figure suivante décrit le fonctionnement de ce modèle :

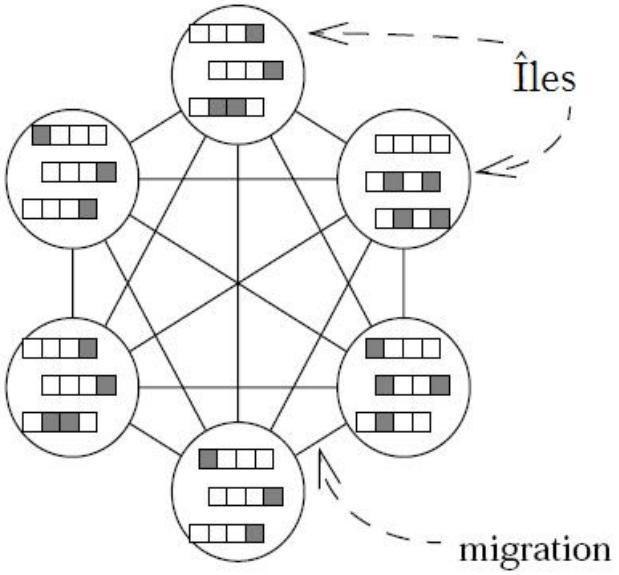


FIGURE 3.8 – Le modèle îlot [3]

Sur chaque île réside une population où toutes les phases de l'AG seront exécutées d'une manière séquentielle (évaluation, sélection, croisement, mutation et remplacement), la communication entre les îles se fait rarement. Dans le cas où le nombre de processeurs n'est pas important, le modèle îlot est le mieux adapté pour avoir de

meilleurs résultats [7].

3.4.2.2 Le modèle cellulaire (grains fins)

Contrairement au modèle insulaire, la topologie du modèle cellulaire est sous forme de grille, où chaque point d'intersection est un processeur. Comme le montre la figure suivante :

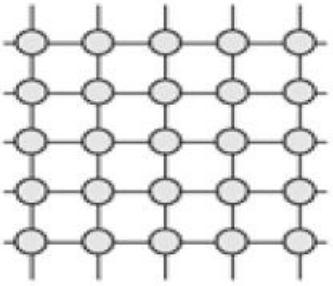


FIGURE 3.9 – Le modèle cellulaire [7]

Dans ce modèle la population principale est décomposée en sous-populations très petites, généralement, elles se composent d'un seul individu. Chaque processeur exécute l'AG sur une sous-population constitué de l'individu dont il dispose et les individus de ses voisins à un saut.

En générale un processeur joue le rôle du maître (root), ce processeur décide de l'arrêt de l'algorithme, suivant les informations qu'il reçoit des autres processeurs sur l'évolution de la population [7].

3.4.3 Le modèle hybride

Pour mieux utiliser les caractéristiques des trois modèles vus en haut, atteindre de bonnes performances et réduire le coût de calcul, nous pouvons utiliser une combinaison des trois modèles pour obtenir un nouveau, dit modèle hybride.La figure suivante représente un modèle hybride qui mélange entre les avantages du modèle maître/esclave et le modèle îlot [7] :

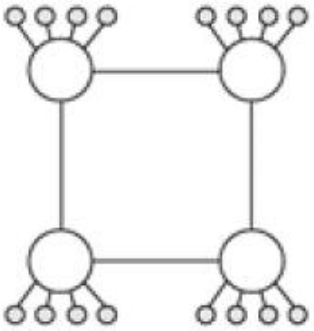


FIGURE 3.10 – Le modèle hybride

Conclusion

Plusieurs méta-heuristiques sont apparues pour avoir de meilleurs résultats compte à la résolution des problèmes d'optimisation combinatoire. Les AGs ont prouvé leurs efficacité de résoudre ce genre de problèmes, et ceci grâce à leurs aspect naturel et aléatoire qui leurs permet d'explorer au mieux l'espace de recherche, et avoir l'optimum global.

Dans ce chapitre nous avons présenté le fonctionnement des algorithmes génétiques séquentiels et parallèles, nous avons fait le point sur les différents modèles parallèles et les différentes fonctions a définir pour assurer le bon fonctionnement de l'AG et de trouver ainsi une solution s'approchant le mieux de l'optimum en un temps qui reste résonable.

4

Berkeley Open Infrastructure of Network Computing (BOINC)

Introduction

L'une des solution pour le problème de la puissnace de calcul dans plusieurs problème est la parallèlisation ou le partage du travaille entre plusieurs nœuds d'un réseau. Dans ce chapitre nous allons vous présenter l'une des plateforme de parallèlisation ou de distribution de travail sur un réseau à large échelle telque internet, BOINC (*Berkeley Open Infrastructure of Network Computing*) à permit la naissance de plusieurs projets scientifiques, économiques, industriels ....

4.1 *Berkeley Open Infrastructure of Network Computing*

BOINC (Berkeley Open Infrastructure of Network Computing), est une plateforme de calcul distribué basée sur le modèle de calcul volontaire, crée par des chercheurs de l'université de Berkeley (Calefornie,USA), elle-même créatrice du projet SETI@home en 1992, SETI@home est l'un des projets de calcul distribué, qui a pour mission de chercher l'existence de civilisations extraterrestres.

BOINC a été créée afin de résoudre des problèmes (projets) dits complexes qu'un seul ordinateur ne peut faire, à ce stade, BOINC utilise les ressources des ordinateurs personnels (volontaires) se trouvant sur internet. Les volontaires peuvent ainsi choisir un ou plusieurs projets auxquels ils participeront, en exécutant un logiciel client (boinc manager) qui va recevoir les portions de calcul à faire pour le(s) projet(s) choisi(s).

4.2 Terminologie de BOINC

Avant d'aller plus loin dans ce chapitre certains termes qu'utilise BOINC doivent être définit :

- *Application* : un programme ayant pour but de résoudre un problème écrit sous C, C++ ou Fortran avec l'utilisation de l'API BOINC pour la distribué.
- *BOINC manager* : interface graphique de BOINC avec laquelle un participant peut gérer les calculs sur les différents projets.
- *Crédit* : points obtenu par les participants lorsqu'ils retournent un résultat valide.
- *FLOPS* : nombre d'opération par seconde faite par un processeur.
- *Equipe* : une équipe est un ensemble de participant qui travaille sur les mêmes projets.
- *Master URL* : le master url est utilisé pour accéder au site web du projet.

- *Participant (volontaire)* : un participant est un internaute qui offre la puissance de son PC pour un ou plusieurs projets BOINC.
- *Projet BOINC* : c'est un ensemble d'applications distribué définit par un master URL.
- *Quorum* : nombre d'utilisateur qui doivent valider un même résultat pour qu'il soit considéré comme fiable.
- *Work Unit (unité de travail)* : c'est un calcul qui doit être exécuté, et est associé à une application et à un fichier d'entrée.

4.3 Principe de fonctionnement

BOINC vous permettra d'utiliser uniquement la puissance non exploitée de votre processeur à des fin scientifique (calcul distribué), dans un temps normal, vous utilisez 1% de votre CPU, BOINC dans ce cas utilisera les 99% restant, mais si vous avez besoin de plus de puissance BOINC s'adaptera automatiquement et réduit son utilisation de lui-même.

BOINC utilise votre connexion internet pour communiquer avec le serveur et récupérer les tâches à vous faire exécuter, après les avoir téléchargées il utilisera un espace de votre disque dure pour les stockées et effectuer le calcul sans être connecté au serveur, ainsi qu'une petite partie de la mémoire vive pour pouvoir exploiter les données dont l'application a besoin pour s'exécuter.

4.4 Volatilité et disponibilité des ressource

Les ressources qu'utilise un projet BOINC, sont réparties entre l'ensemble des volontaires qui sont connectés à ce projet. En effet à tout moment un client peut déconnecter ou reprendre le contrôle de son ordinateurs ce qui rend les ressources volatile, c'est-à-dire, à n'importe quel moment une ressource peut ne plus être disponible, ce qui va conduire une tâche à ne pas se terminé correctement : exécution

d'une autre application, panne, redémarrage de la machine, activité clavier/souris dans le cas d'un économiseur d'écran. Plusieurs niveaux de disponibilité existent [14] :

- La disponibilité de la machine : la machine est allumée et le client BOINC fonctionne.
- La disponibilité du processeur : les ressources de calcul sont attribuées à une tâche donnée.
- La disponibilité de l'application : une tâche peut être interrompue par le client BOINC et attribuer la ressource à un autre projet.

4.5 Architecture de BOINC

L'architecture de BOINC est basée sur le modèle client/serveur. Les projets qui sont identifié par une URL, résident dans le serveur. Lorsqu'un client a besoin d'une nouvelle tâche, il le demande au niveau du serveur qui va lui allouer une.

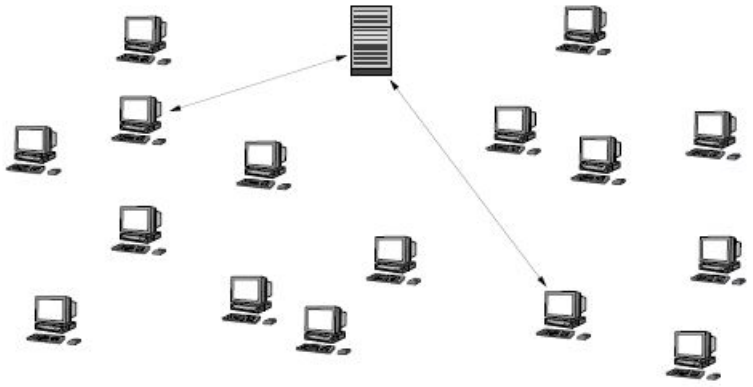


FIGURE 4.1 – Client/Sevrveur

Afin de remédier au problème de sabotage des résultats, chaque tâche est désignée à N clients, si un certain nombre de résultats diffèrent, le serveur redistribue la tâche à N autres clients.[15]

4.5.1 Le client BOINC

Un utilisateur voulant participer à un projet avec la puissance de son ordinateur, commence par le téléchargement et l'installation du client BOINC connu aussi sous le nom *core* BOINC (BOINC manager), ce dernier se trouve généralement sur le site web de chaque projet ou sur le site officiel de BOINC. L'utilisateur s'enregistre sur le site web du projet (interface web). Après être enregistré, Il reçoit un e-mail contenant un identificateur unique (User ID), pour rejoindre un projet l'utilisateur doit insérer son identifiant ainsi que le Master URL du projet sur une interface graphique qu'offre le client BOINC, ce processus est connu sous le nom de l'attachement du client à un projet. L'utilisateur peut s'attacher à plusieurs projets, et spécifier le taux de puissance consacré à chaque projet.

L'utilisateur a le contrôle total sur le client BOINC, il peut spécifier le temps durant lequel le client peut exécuter les tâches, ou l'exécution se fait uniquement durant l'inactivité de l'utilisateur. Il peut aussi spécifier la quantité maximale et minimale de travail qui sera offerte au client, durant l'exécution, si cette quantité dépasse son minimum, le client contacte automatiquement le serveur pour lui envoyer plus de travail. Le client peut exécuter les tâches sans être connecté, et par la suite, dès qu'une connexion avec le serveur est disponible les résultats sont envoyés automatiquement. La file d'attente locale de travail est géré par défaut selon la politique FIFO, mais comme un client peut s'attacher à plusieurs projets alors la nécessité de pouvoir attribuer les priorités de calcul au projet est devenu indispensable. Le client envoi aussi le temps CPU consacré à chaque résultat obtenu, afin de calculer le montant crédit (système de crédit est conçu dans le but de mettre les volontaires en concurrence.) qui lui est attribué.

Notons que le client BOINC est disponible sous Windows, Unix et Mac OS, mais comme il est open source donc il peut être adapté à toutes autres plateformes.[17]

4.5.1.1 La sécurité du client

La sécurité du client est assurée par le hachage des données et la signature du haché par une clé privé. Pour protégé le client contre les exécutables modifiés, les exécutables sont signés par une clé privé RSA et les clients BOINC vérifie la signature avec la clé public du projet. Dans le cas où la clé privé n'est pas utilisée elle sera conservée dans une machine qui n'est pas connectée à internet et ceci pour minimiser la probabilité que la clé soit piratée.

Au cours de cette procédure le client récupère les adresses des serveurs d'ordonnancement à partir de la page web du projet et se connecte à l'un d'eux. BOINC ne protège pas les machines contre les erreurs de programmation et les applications malveillantes, i-e un client peut être victime d'une application a des fins non scientifique, et ne protège non plus pas les communications entre les clients et les serveurs (les attaques de type sniffing [2]).

BOINC sécurise les clients en empêchant la surcharge du CPU, la mémoire et l'espace réservé au calcul, ceci en bloquant les applications qui excèdent les limites d'usage prédéfini par le projet.[17] [18]

4.5.2 Le serveur BOINC

Trois composantes essentielles forme le serveur BOINC : Web interface à travers laquelle les volontaires peuvent consulter leurs crédits, leurs préférence et consulter des news sur le projet. Le Task Server ou le serveur de tâche qui a pour rôle de créer des tâches, les distribué aux clients et récupérer les résultats à la fin de l'exécution. Le Data Server ou le serveurs de données qui permet le téléchargement des fichiers d'entrés et la distribution des fichiers de sorties, ces derniers sont relative à un projet. BOINC se compose essentiellement de deux entités (le client BOINC et le serveur

2. écouter une ligne par laquelle transitent des paquets de données pour récupérer à la volée (et illégalement) les paquets qui peuvent être intéressants (qui contiennent par exemple le mot « password »...)

BOINC) comme l'illustre la figure suivante : [16]

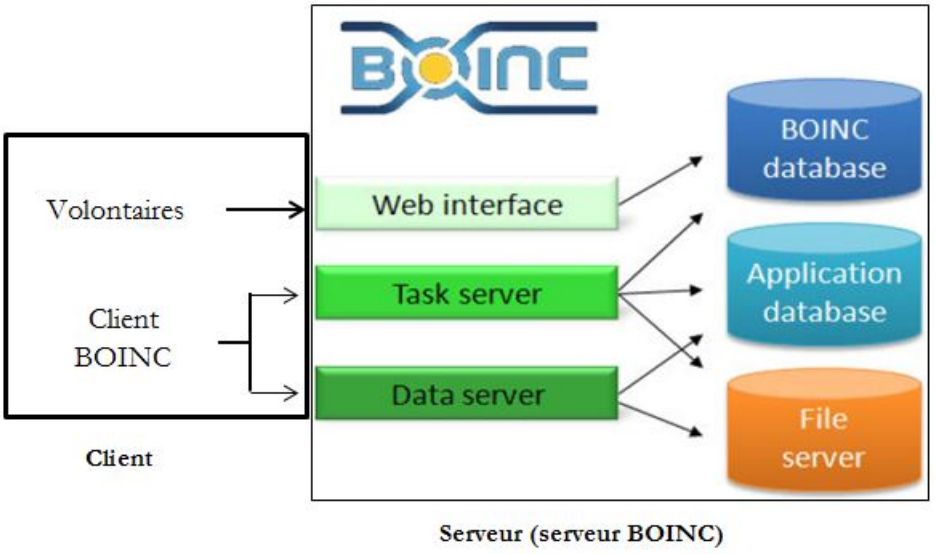


FIGURE 4.2 – Le serveur BOINC

4.5.2.1 Interface web

Lorsque vous créez un nouveau projet en exécutant la commande make_project , un site web est créé pour ce projet. Si votre projet s'appelle teste alors l'accès a la page principale du site se fait via l'URL http ://a.b.c.d/teste, tel que a.b.c.d est l'adresse IP de la machine où le site est hébergé.

Cette interface permet aux utilisateurs (volontaires) de se connecter à leurs espaces personnels, modifier leurs préférences, modifier le profil, voir les nouvelles versions de l'application et consulter les statistiques liées à la participation à ce projet.

Ce site web peut être modifié par l'administrateur en redéfinissant l'ensemble des pages d'une manière est ce qu'il présente au mieux son projet pour avoir plus de participants figure 4.3.

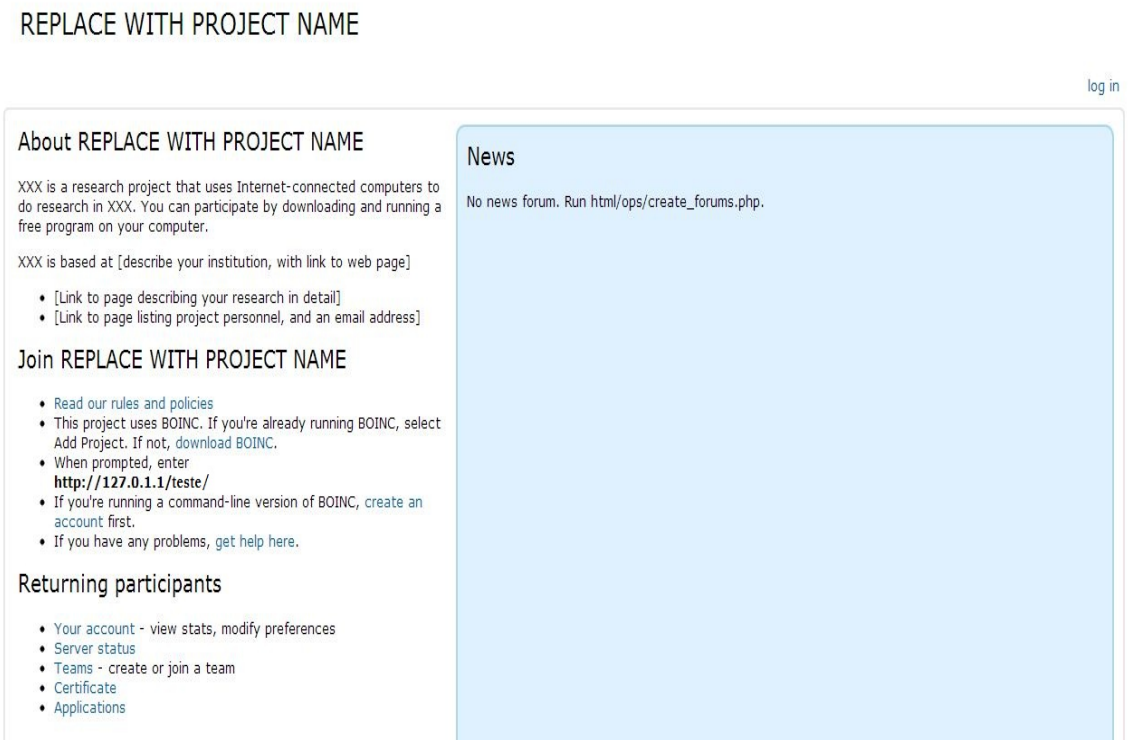


FIGURE 4.3 – Web interface

Une partie administrative de ce site est mise en place pour bien géré le site web du projet, l'administrateur accéde à cette partie via l'URL http ://a.b.c.d/teste_ops. Pour des raisons de sécurité la partie admin est protégée par un fichier .htacces et un fichier de configuration qui permet à l'administrateur de spécifier la politique de protection de cette interface.

Cette interface permet de multiples manipulations entre autres : le parcours de la base de données, les profils des utilisateurs, créer et modifier les applications, annuler des unités de travail.

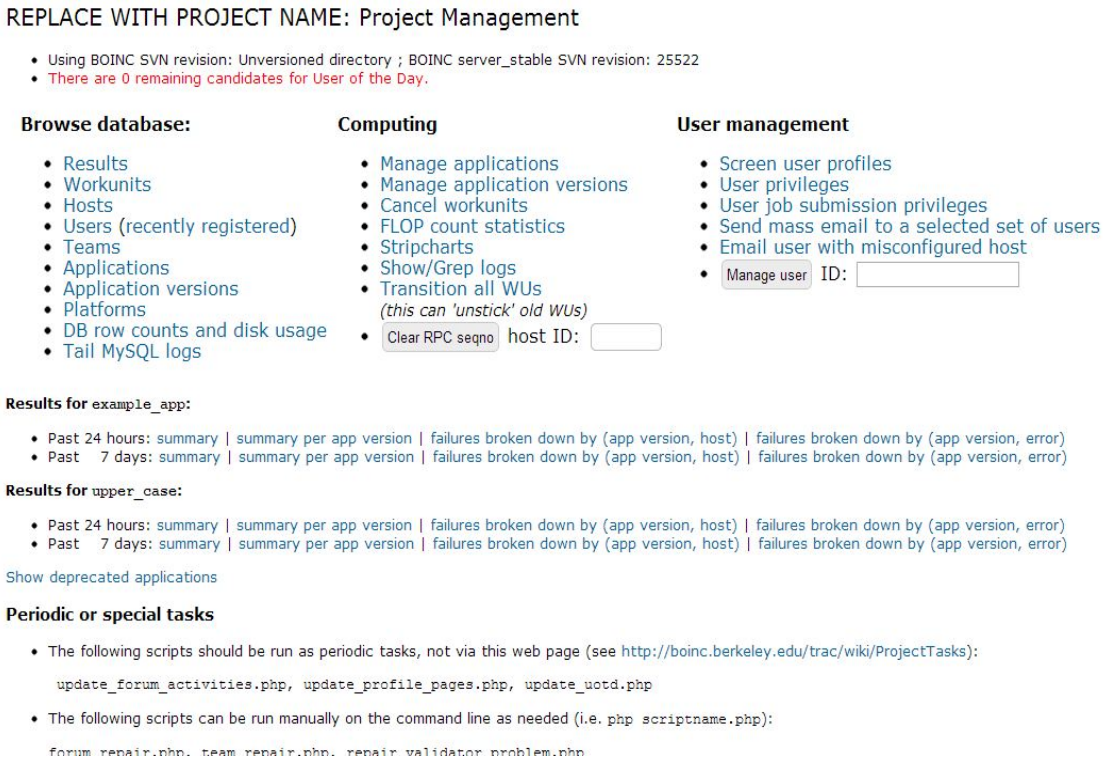


FIGURE 4.4 – Espace administrateur

4.5.2.2 Task server (le serveur de tâches)

Les projets de calcul volontaire sous BOINC utilisent un serveur de tâche pour retourner les résultats des tâches terminées et demander de nouvelles tâches. la figure suivante décrit la composition du serveur de tâche :

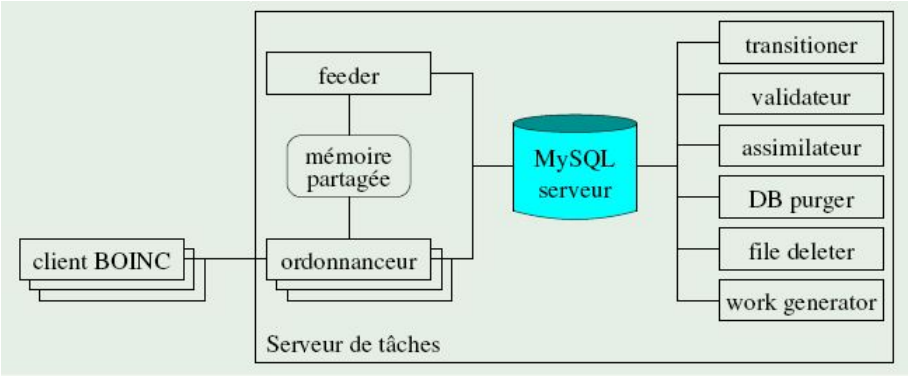


FIGURE 4.5 – Task server

1. ***L'ordonnanceur*** : les clients BOINC communiquent directement avec l'ordonnanceur, en lui envoyant des requêtes pour recevoir de nouvelles tâches ou indiquer la terminaison de certaines. Son rôle principal est de déterminer quelles sous-tâche sont envoyées aux clients, cette décision tient compte des paramètres de chaque sous-tâche et de chaque projet. Chaque demande comporte une description du client émetteur. Suite à la réception d'une requête le Scheduler déclenche un certain nombre d'opération sur la base de donnée : la mise à jour du compte utilisateur, des instances de travaille ... etc. Les opérations de lecture et écriture de la base de données sont exécuté par l'intermédiaire du *feeder* en utilisant une mémoire partagée chargée par ce dernier. Le *Scheduler* est considéré comme un programme CGI[3], exécuté sur un serveur apache à chaque fois qu'un client se connecte au projet et demande du travail. Le client communique avec le serveur d'ordonnancement via des requêtes http sous cette forme :

```
<scheduler_request>
    <platform_name> i686-pc-linux-gnu </ PLATFORM_NAME>
    <core_client_major_version> 1 </ core_client_major_version>
    <core_client_minor_version> 1 </ core_client_minor_version>
    ...  autres éléments
</ Scheduler_request>
```

Où on trouve des informations relatives au client tel que sa plateforme, la version du client et d'autres éléments. Une reponse est envoyé du serveur au client :

```
<scheduler_reply>
    [<message Priority='low'> texte arbitraire </ message> ...  ]
    [<request_delay> 3600 </ request_delay>]
    [<redirect> URL </ redirect>]
    ...  autres éléments
</ Scheduler_reply>
```

Plusieurs éléments sont indiquer dans la réponse : *message priority* qui indique si le message doit être visible par le participant (priorité élevé) ou il doit juste être inscrit dans un fichier .log (priorité faible), *request_delay* qui indique au

3. (common geteway interface)

client émettre à la fin du délai d'attente, et *redirect* pour rediriger le client vers d'autres serveurs d'ordonnancement. [18] [19] [20]

2. ***Le feeder*** : le feeder rend l'accès du scheduler à la base de données plus rapide, en faisant un pré-chargement dans la mémoire partagée, pour améliorer la performance de BOINC en limitant le nombre de requête envoyé au serveur de base de données par le client BOINC. La mémoire partagée contient en général l'ensemble des applications disponibles pour un projet, et les différentes versions de ces applications. ainsi qu'une mémoire cache d'une taille fixe contenant les instances/job non envoyées. Un sémaphore est utilisé par BOINC pour synchroniser l'accès à la mémoire partagée et minimiser les conflits. Dans les premières versions de BOINC les choix des sous-tâche qui sont misent dans la mémoire partagée ont été choisies aléatoirement, en raison des problèmes de performance, les nouvelles versions de BOINC charge les sous-tâches en suivant la politique FIFO.

3. ***Le transitioner*** : le *transitioner* est un programme offert par BOINC, son rôle est d'examiner les sous-tâche ayant subi un changement d'état, les WU ne possèdent pas un état général mais un ensemble de sous-état, par exemple si une tâche est prête pour la validation. Le transitioner est l'un des programmes qui consomme le plus de puissance CPU, il est donc généralement divisé en de nombreux processus où chacun est responsable d'un ensemble de WU (workUnit).

4. ***DB purger*** : le *DB purger* ou database purger, est un utilitaire qui fait partie de la plateforme BOINC, permetant d'écrire les résultats des *WorkUnit* sous format XML, puis les supprimées de la base de données, afin d'eviter la surcharge de la base de données. Le DB purger possède plusieurs options en ligne de commande : *-Min_age_day-N* (purgé uniquement les WU avec un mode_time qui égale au moins a N), *-Daily_dir* (écrire l'archive dans un nouveau répertoire chaque jour). les archives ont des noms de la forme suivante

wu_archive_time où "time" est la date de création de l'archive.

5. ***File deleter*** : le file deleter, qui a comme rôle la suppression des fichiers d'entrée et de sortie des WU qui ont été exécuté. La politique de suppression par défaut est la suivante : les fichiers d'entrée sont supprimés si tous les résultats de la WU en question sont rapportés ou le délai de livraison est expiré, et les fichiers de sortie sont supprimés après que le résultat canonique est choisi et est validé, car ces fichiers peuvent être nécessaires pour la validation. Avant de supprimer ces fichiers du serveur web, ils sont copié dans la base de données, il est donc possible de récupérer des informations sur les WU et les résultats, même après la suppression des fichiers et l'achevage de ces WU.

6. ***Validator*** : le validateur est une composante logicielle décrite par le développeur du projet (propre à un projet). Son rôle principal est le choix d'un résultat canonique pour une workUnit (unité de travaille), en effet lors de la création d'une unité de travail un Quorum lui est fixé, ce dernier représente le nombre de calcul qui sont validé pour considérer un résultat comme canonique. A son lancement le validateur vérifié s'il y a de nouveaux résultats non valider dans la base de données, dans ce cas il exécute une fonction spécifique à l'application qui lui permettra de comparer les résultats et définir un comme canonique. Une fois le résultat canonique est défini, le validateur mis à jour les crédits des utilisateurs ayant renvoyé un résultat correct. Deux validateurs standards sont fourni par BOINC : *sample_bitwise_validator* requiert une majorité strict, deux résultats sont équivalant s'il y a une correspondance octet par octet entre les deux, ce type de validateur est utilisé généralement si le projet utilise une redondance homogène, qui considère tous les résultats comme valide.

7. ***L'assimilator*** : tout comme le validateur le fonctionnement de l'assimilateur est définit par le développeur, sa fonction principale consiste en l'insertion du résultat canonique dans la base de données scientifique, ensuite les résul-

tats peuvent être retiré de la base de données de BOINC. dans le cas où le traitement d'une sous-tâche est interrompu d'une manière non récupérable, le démon (service qui s'exécute en arrière plan) enregistre l'échec dans une base de données ou alerte l'administrateur du projet.

8. ***Le Work generator (le générateur de travail)*** : le générateur de travail ou des WorkUnit, est spécifique à une application, créé par le développeur en spécifiant : le quorum maximal pour chaque unité de travail, le nombre maximal d'erreur acceptable par le validateur, le nombre total de résultat, sont des paramètres fixés lors de la création d'une tâche par le générateur.
9. ***La base de données MySQL*** : une base de données MySql est disponible pour stocker toute information en relation avec le serveur BOINC, les projets, les clients BOINC, les WU et leurs résultats, les applications et leurs versions. en général tout état du serveur, est enregistré dans la BDD, qui est interrogé par les différents démons introduit en haut.

4.5.2.3 Data Server

Un client BOINC est relié directement au data server qui lui permet de télécharger des fichiers de données pour exécuter une application et sauvegarder les résultats après l'exécution.

4.5.2.4 La sécurité du serveur BOINC

Les projets BOINC sont une cible de trois attaques essentielles. D'abord, un projet BOINC peut subir un DoS[4] en retournant un grand nombre de résultat. Pour le résoudre une taille maximale et une signature sont fixés pour les fichiers output lors de la création de l'unité de travail.

Aussi, un projet peut être victime d'un sabotage des résultats, i-e de faux résultats seront retournés à chaque fois. Pour remédier à ce problème, BOINC procède

4. déni de service

par une duplication des instances d'une tâche, envoyer chaque instance à plusieurs clients pour effectuer le calcul et fixer un nombre de résultats erronés qui seront accepté. Au retour des résultats si le nombre d'erreur dépasse celui déjà fixé, BOINC redistribue les instances sur un nouveau groupe de client, sinon un résultat canonique sera fixé et validé. Dans la plus part du temps il y a toujours des erreurs dans certains résultats, par conséquent, les projets qui exigent que tous les résultats sront correcte ne devrait pas utiliser BOINC.

Certain participants renvoient de faux résultats d'une manière redondante et ceci juste pour faire augmenter leur crédit, BOINC résous se problème par un principe simple qui est : un participant augmente son crédit si et seulement s'il retourne des résultats correcte.

Pour mieux sécurisé les accès aux serveurs il est préférable d'utiliser le protocole SSH ou un autre protocole crypté, et de faire tourner le serveur derrière un pare-feu pour minimiser le taux d'intrusion.[18] [17]

4.5.2.5 Communication client BOINC/serveur BOINC

Un client BOINC est relié directement au data server qui lui permet de télécharger des fichiers de données pour exécuter une application et sauvegarder les résultats après l'exécution.

La communication entre le client et le data server se fait via des requêtes http, généralement un serveur de données est mis en place en utilisant un serveur web avec un programme CGI fourni par BOINC. Le protocole de communication prend en charge les transferts partiels de fichiers, si le transfert d'un fichier est interrompu à l'octet N, à la prochaine communication le transfert débutera de l'octet N et il n'a pas besoin de tout refaire.

Lors d'un téléchargement d'un fichier de données une requête GET sur l'URL est envoyée, et pour sauvegarder les fichiers de résultat sur le serveur une requête POST est transmise au démon. Un mécanisme de sécurité est met en place pour

empêcher la sauvegarde non autorisé d'un grand nombre de quantité de donnée.[20]

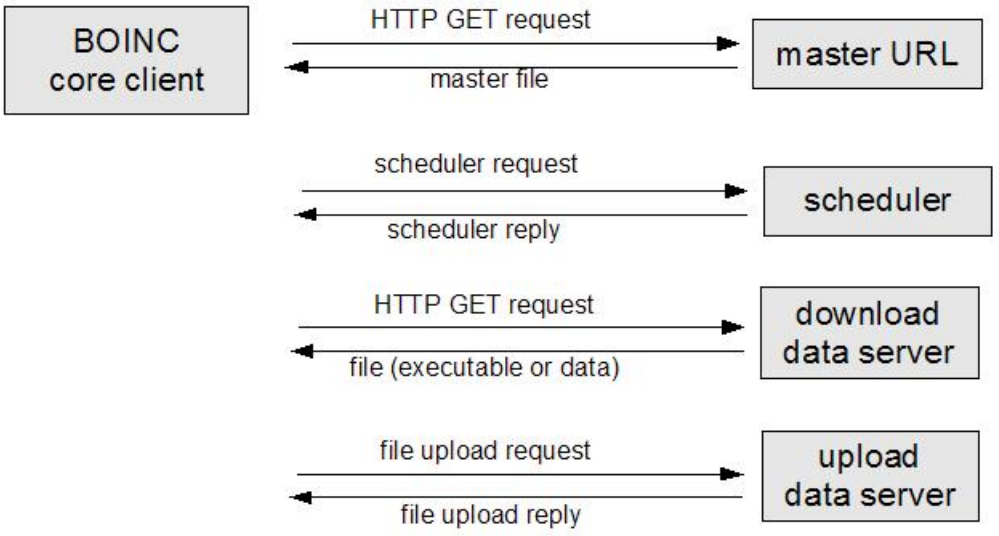


FIGURE 4.6 – Communication Client BOINC/Serveur d'ordonnancement

- Le client BOINC accède à la page web du projet (master URL), où il obtient une liste des serveurs d'ordonnancement.
- Un échange de requête/réponse entre le client et le serveur d'ordonnancement se déclenche, les réponses contiennent généralement une description des WU et une liste d'URL de leurs fichiers d'entrée et de sortie.
- Le client télécharge des fichiers de données et applications d'un ou plusieurs serveurs de données, et commence l'exécution des applications.
- A la fin de l'exécution, le client sauvegarde les résultats sur le data server. Un protocole est implémenté sur ce dernier pour empêcher les attaques de type DoS.

4.6 Un projet BOINC

Un projet BOINC est définit par un ensemble d'application, identifié par un Master URL, les projets sont indépendant entre eux, donc un serveur peut contenir plusieurs projets différents. Chaque projet possède, une base de données MySQL créer automatiquement par BOINC lors de la création de celui-ci, un validateur, un assimilateur et un générateur de WU, qui sont décrit par le développeur ainsi qu'une partie du site web du projet. (Les étapes à suivre pour la création d'un projet BOINC sont définit dans l'Annexe). Compte au reste des composant dont un projet aura besoin pour fonctionner correctement, sont offert directement par BOINC.

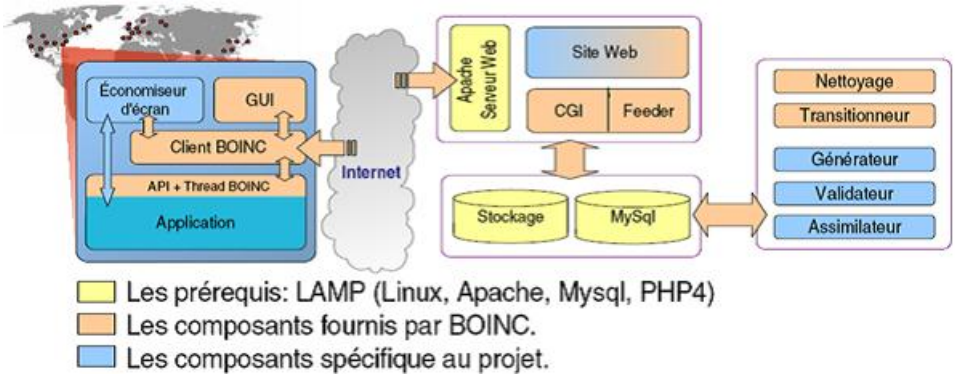


FIGURE 4.7 – Les composents d'un projet BOINC

cettefigure décrit les différents éléments que BOINC nous offre, et les composants qu'un développeur doit créer lui-même pour le bon fonctionnement de son projet.[22]

Conclusion

Le nombre de participants (volontaires) ne cesse d'augmenter depuis sa création (BOINC) à ce jour, d'après des statistique faites par les membres de http ://boinc-af.org [24], il serait qu'en novembre 2012, BOINC a environ 341 000 ordinateurs participant au calcul volontaire, ce qui donne une puissance soit de 42% du plus puissant ordinateur du monde le TITAN-Cray XK7 hébergé aux USA, avec une puissance de calcul qui atteint les 17.590 petaFlops.

Dans ce chapitre nous avons pu exposer l'architecture de BOINC en détail ce qui va nous permettre dans le chapitre suivant d'implémenter un modèle de parallèlisation des AGs sous BOINC. Cette implémentation va nous résoudre l'un des problèmes d'optimisation combinatoire (FlowShop), en un temps beaucoup plus résonable que celui capturé en séquentiel.

5

Conception et implémentation

Introduction

Après la présentation des chapitres précédents qui regroupent la majorité des notions théorique nécessaire à la résolution de notre problématique, nous en venons à ce dernier chapitre qui représente la phase applicative qui déterminera les résultats auxquels notre dur labeur de recherche aura aboutit.

Dans ce présent chapitre nous allons présenter notre application implémenté sous paradisEO et BOINC, notre solution à la problématique ainsi que des testes que nous avons pu faire sur plusieurs machines en utilisant des machines virtuelles.

5.1 Présentation des outils de développement

Nous allons vous présenter brièvement les outils que nous allons utiliser, BOINC est déjà présenté en détail dans le chapitre 4, nous allons donc nous focalisé sur ParadisEO, le langage R et VirtualBox

5.1.1 ParadisEO

ParadisEO est une plate-forme logicielle facilitant l'utilisation, le développement et la comparaison de métaheuristiques (algorithmes d'optimisation visant à résoudre des problèmes d'optimisation difficiles) classiques, multiobjectifs, parallèles et hybrides. ParadisEO est développé par l'équipe DOLPHIN du laboratoire IRINA, Lille, France. Paradiseo se compose de 4 modules qui sont comme l'illustre la figure 5.1 :

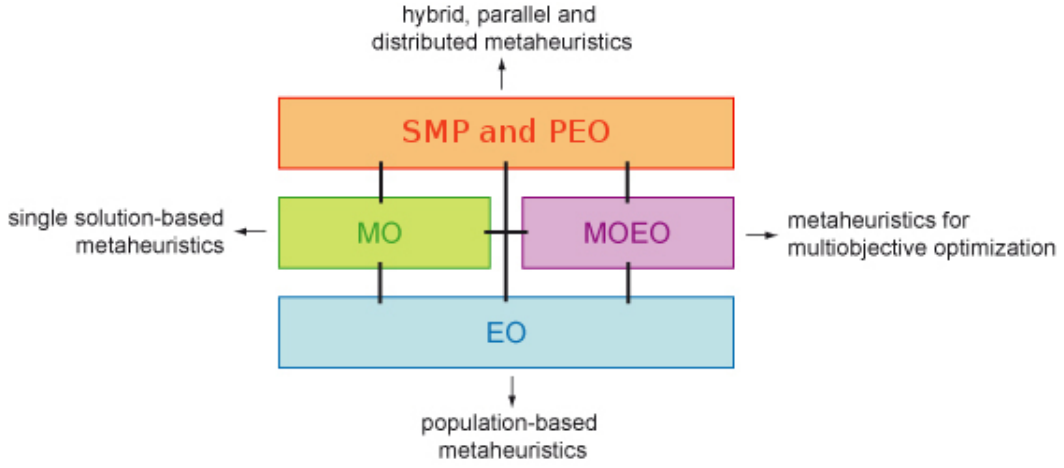


FIGURE 5.1 – Les composants de paradisEO

- **EO** : pour les Métaheuristiques à population de solutions (les AGs).
- **MO** : pour les Métaheuristiques à solution unique.
- **MOEO** : Métaheuristiques pour l'optimisation multiobjectif.
- **PEO** : Métaheuristiques hybrides, parallèles et distribuées.

ParadisEO offre des implémentations de nombreuses métaheuristiques classiques de la littérature ainsi que des métaheuristiques novatrices. ParadisEO permet également la parallélisation des métaheuristiques.

ParadisEO est utilisé pour la résolution de problèmes d'optimisation de grande taille issus de nombreux domaines, tels que : le transport, la logistique, les systèmes de télécommunication ou la bioinformatique. Paradiseo est un framework C++ open-source, il est portable sous windows, unix, MacOs.

Dans notre travail nous avons utiliser ParadisEO, pour développer une application qui peut résoudre le problème du flow-shop en utilisant les algorithmes génétiques.

5.1.2 Le langage R

R est un logiciel de calcul scientifique interactif libre qui possède une large collection d'outils statistiques et graphiques, Des versions compilées de R sont disponibles pour Linux, Windows et Mac OS X.

5.1.3 VirtualBox

VirtualBox est un logiciel de virtualisation de systèmes d'exploitation. En utilisant les ressources matérielles de l'ordinateur (système hôte), VirtualBox permet la création d'un ou de plusieurs ordinateurs virtuels dans lesquels s'installent d'autres systèmes d'exploitation (systèmes invités).

5.2 Conception

5.2.1 Présentation du problème du FlowShop

Le problème du Flow-shop est un problème d'ordonnancement de jobs. Disposant de n jobs J_1, J_2,..., J_n et de m machines ce problème nous demande d'ordonnancer ces n jobs sur les m machines sachant que chaque job J_i est composé de m tâches consécutives notées O_{ij} avec un temps d'exécution P_{ij}. Ainsi O_{ij} se lit de la manière suivante : la $j^{iéme}$ opération du job J_i doit être exécutée sur la machine m_i.

Entre les opérations successives d'un même job (O_{ij} et O_{ij+1}) existe une contrainte de précédence et nous ne pouvons allouer qu'un seul job à la fois à chaque machine. Nous sommes amenés à trouver une séquence de jobs sur chaque machine de façon à minimiser le temps total d'exécution de tous les jobs.

5.2.2 Résolution par permutation

Nous pouvons résoudre le problème précédent en utilisant la technique des permutations, ce qui consiste à avoir une même séquence de jobs sur toutes les machines. A chaque tâche O_{ij} est associé un temps de départ P_{ij} ainsi qu'un temps d'exécution S_{ij} sur une machine M_j. La fonction à minimiser (le makespan) se calcule alors comme suit :

$C_{max} = \max \{P_{iM} + S_{iM} \mid \text{i [1,n]}\}$

S_{iM} et P_{iM} représenterons ainsi la date de début de la tâche O_{iM} et sa durée d'exécution sur la machine.

L'exploration de l'ensemble des ordonnancements pour le problème peut ainsi se faire en travaillant les permutations.

5.2.3 Objectifs de notre travail

L'objectif principal de ce travail est d'implémenter une métaheuristique (algorithmes génétique) pour résoudre le problème du FlowShop en utilisant la plateforme ParadisEO, tout en proposant un modèle de parallèlisation de l'AG sous BOINC, qui va nous permettre de résoudre le problème en moins de temps.

Pour aboutir aux résultats attendus nous commençons par une exécution séquentielle de l'AG sur une seule machine tout en observant l'évolution du temps d'exécution en variant la taille de la population et l'instance du problème. Ensuite nous ferons un deuxième teste sur 10 ordinateurs dotés d'une machine virtuelle (ubuntu 12.04 32bits avec un seul cœur) avec implémentation du modèle parallèle de l'AG en utilisant BOINC. Enfin nous allons exécuter l'AG sur 10 machines dotés

d'une machine virtuelle, en offrant à la machine virtuel 8 cœurs.

5.2.4 Présentation de l'application (séquentielle)

Le diagramme suivant représente l'ensemble des classes dont notre application est constituée, toutes les classes héritent de la classe EO quielle même hérite de la calsse eoObject, comme l'illustre la figure 5.2 :

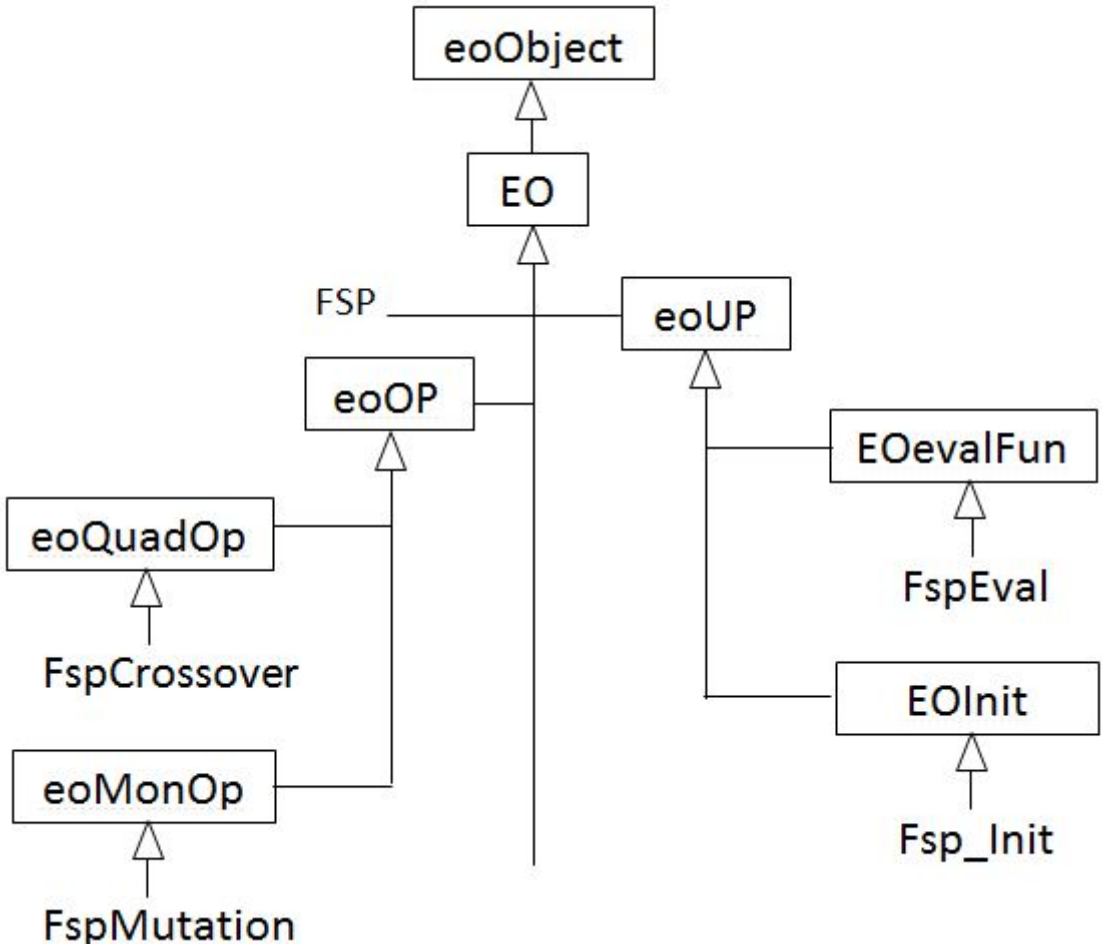


FIGURE 5.2 – diagramme de classe explicatif

Notre application consiste en une implémentation d'un algorithme génétique pour la résolution du problème du flow-shop sous la plateforme paradisEO. Dans cette dernière le sequelette de l'AG est déja disponible, en effet il suffit juste de redéfinir les composants qui dépendent du problème a traité, un ensemble de classe nous a

été offert :

- le namespace ins (fichier instance_fsp.cpp et instance_fsp.h) sert à générer une instance flowshop et contient la matrice des données qui sert pour le calcul du makespan.
- la classe FSP (fichier fsp.cpp et fsp.h) représente une solution au problème du flowshop.
- la classe FSPInit (fichier fsp_init.cpp et fsp_init.h) implémente la méthode d'initialisation d'une solution d'une façon aléatoire.
- la classe FSPFitness (fichier FSPFitness.h) représente le type de donnée utilisé pour la valeur de la fonction objectif (la fitness).
- La classe FSPEval (fsp_eval.cpp, fsp_eval.h).

Et nous avons développer la classe FspCrossover ainsi que la classe FspMutation :

- La classe FspCrossover : hérite de la classe eoQuadOp qui utilise un opérateur avec deux arguments d'entrée (les deux parents) pour produire deux fils en sortie.
- La classe FspMutation : qui hérite de la classe eoMonOp, cette dernière n'a besoin que d'un seul argument (le fils sur lequel la mutation sera appliquée) pour s'exécuter.

5.2.5 Choix du modèle de distribution

Nous retrouvons en [5] une étude très détaillée de ce qui est des modèle de distribution les plus important bien que majoritairement ils ont été réfléchit pour une architecture distribuée en grilles. Chaque modèle est plus au moins intéressant et contraignant pour notre cas, mais notre attention à était attiré en particulier l'un d'eux : Le modèle insulaire, et cela pour des raisons bien précises :

- Il est simple à mettre en place.
- Sont implémentation sur BOINC API ne nécessite aucune modification des composantes de cette dernière (Ceci sera expliqué dans une section ultérieur.)

– Il est très flexible et dans notre cas nous pourrions avoir la possibilité de modifier son fonctionnement même en cours d'exécution sans pour autant altérer les résultats obtenus.

5.2.6 Fonctionnement du modèle insulaire

Le principe de ce modèle comme illustré dans le chapitre 3, est de diviser la population en sous populations beaucoup plus petites mais avec un large nombre d'individus. Chaque sous population est affectée à une machine (un client) qui se chargera de calculer son meilleur individu. Durant l'exécution chaque machine communique son résultat à une autre (phénomène de migration). A la fin de l'exécution on obtient ainsi le meilleur résultat parmi tous les autres meilleurs résultats envoyés.

5.3 Implémentation

Notre application se compose de deux parties, une partie séquentielle sous paradisEO et la parallélisation de cette dernière sous BOINC. Dans ce qui suit nous allons présenter les deux cas de figure.

5.3.1 Implémentation séquentielle sous ParadisEO

5.3.1.1 Codage de l'AG

Dans le cas d'un problème de permutation le choix du codage à utiliser se focalise le plus souvent sur le codage réel. En effet, dans le problème du flow-shop nous disposons d'un ensemble de jobs à exécuter, le séquencement de ces jobs est sans répétition et est représenté un individu dans l'AG, qui est (individu) une permutation d'entiers de 0 à n-1, avec n le nombre de jobs.
Ex : nous supposons que nous disposons de 10 jobs à exécuter, le séquencement de l'exécution des jobs peut être représenté comme suit : 0 5 2 3 9 1 8 7 4 6.

5.3.1.2 La taille de la population

Comme nous l'avons déjà défini dans le chapitre consacré aux algorithmes génétiques, une population est l'ensemble des individus que nous voulons faire évoluer, dans le cas du problème du flow-shop une population est l'ensemble des jobs que nous ordonnons à chaque fois d'une manière différente.

Ex : nous donnons l'exemple d'une population de 5 individus :

2 5 9 8 7 6 3 0 4 1

0 5 7 1 3 6 9 4 8 2

9 6 3 2 5 8 7 4 1 0

0 1 2 3 4 5 6 7 8 9

9 8 7 4 5 6 1 2 3 0

5.3.1.3 La méthode d'initialisation

La méthode d'initialisation est la méthode avec lequelle nous créons la population de départ. dans notre cas nous avons opté pour une méthode d'initialisation aléatoire. L'initialisation est réalisée par l'exécution du code suivant dans la fonction Main du projet :

```
FSPInit init_sol;
eoPop <FSP> pop (pop_size, init_sol);
```

Où pop_size représente la taille de la population et init_sol l'objet d'initialisation.

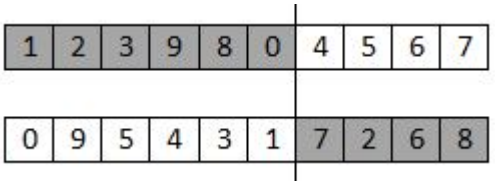
5.3.1.4 Opérateur de sélection

Nous avons choisi la sélection stochastique, qui se base essentiellement sur la valeur attendu d'un individu, Pour plus de détail sur le fonctionnement de la méthode stochastique voir chapitre 3.

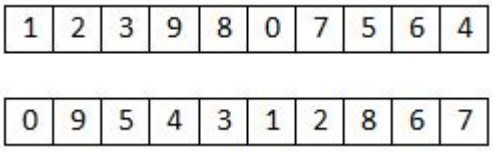
5.3.1.5 Le croisement

Nous avons utilisé un croisement en un point, le point de croisement sera choisi aléatoirement, dans le travail que nous avons réalisé nous avons fixé le taux de croisement à 1.

Ex : nous supposons les deux parents suivant avec le point de croisement à la position 6 :



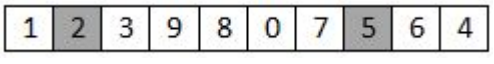
Après l'exécution du croisement nous obtenons les deux fils suivant :



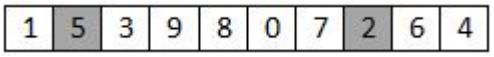
5.3.1.6 La mutation

Dans le cas des problèmes codé en réel, la mutation consiste à échanger la position de deux jobs (dans notre cas) dans l'individu en question, les deux positions à faire permuter sont choisies aléatoirement.Dans notre travail nous avons fixé le taux de mutation à 0.4.

Ex : nous supposons l'individu suivant : Après l'exécution le choix des deux positions



à faire permuter et l'exécution nous pouvons obtenir l'individu suivant :



5.3.1.7 L'opérateur de remplacement

Dans la phase de remplacement chaque parent à la probabilité de 0.7 de rester et de faire partie de la prochaine population, cette probabilité va nous permettre de sauvegarder le patrimoine génétique de l'ancienne génération.

5.3.1.8 Critère d'arrêt

Le critère d'arrêt, représente une condition sur laquelle l'AG arrête son exécution, plusieurs critères existent : critère de temps, critère se basant sur le nombre de génération, c'est ce dernier que nous avant utilisé dans notre application. L'AG

```
eoGenContinue <FSP> cont (1000) ;
```

s'arrête lorsque l'on atteint la 1000^{ieme} génération.

Après avoir déclarer tous les paramètre dont l'AG à besoin pour s'exécuter correctement, nous déclarons un objet de type *eoEasyEA* et nous appelons la fonction *ea(pop)* pour l'exécution de l'AG, comme suit :

```
eoEasyEA <FSP> ea (cont, eval, select, transform, merge, reduce);

ea (pop) ;
```

5.3.2 Implémentation prallèle de l'AG sous BOINC

Après avoir choisi le modèle insulaire pour parallèliser l'exécution de l'AG, nous allons dans ce qui suit décrire les modification que nous avons pu apporté à ce modèle afin qu'il soit possible de l'integré et de l'implémenter sous BOINC.

Bien que le fondement de ce modèle (modèle insulaire) soit principalement basé sur la communication inter-clients son objectif est de retourné le meilleur individu des meilleurs individus obtenu comme résultat par chaque client. De ce fait nous pouvons considérer la modification suivante :

5.3.2.1 Première phase

Dans cette promière phase le serveur envoi à chaque client un AG pour l'exécuter, comme l'illustre la figure 5.3 :

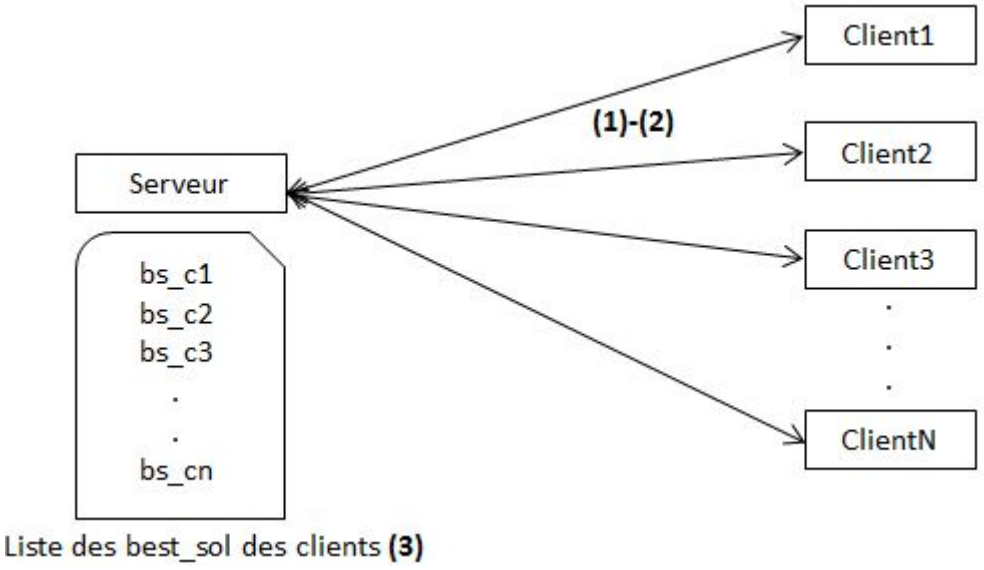


FIGURE 5.3 – première phase du modèle de distribution proposé

- **(1)** Le serveur envoi l'ensemble des tâches aux clients disponibles.
- **(2)** Les clients ayant terminé l'exécution renvoient le résultat au serveur.
- **(3)** A la fin de l'étape 2 le serveur dispose de l'ensemble des meilleures solutions calculées par les différents clients.

5.3.2.2 Deuxième phase (phase de validation)

Une fois la première phase est achevé, le serveur dispose d'une liste de résultats, cette dernière n'est pas encore validé. Dans cette phase le serveur réenvoi la liste des

résultats aux différents clients disponible pour validé les résultats obtenus. la figure 5.4 décrit ce processus :

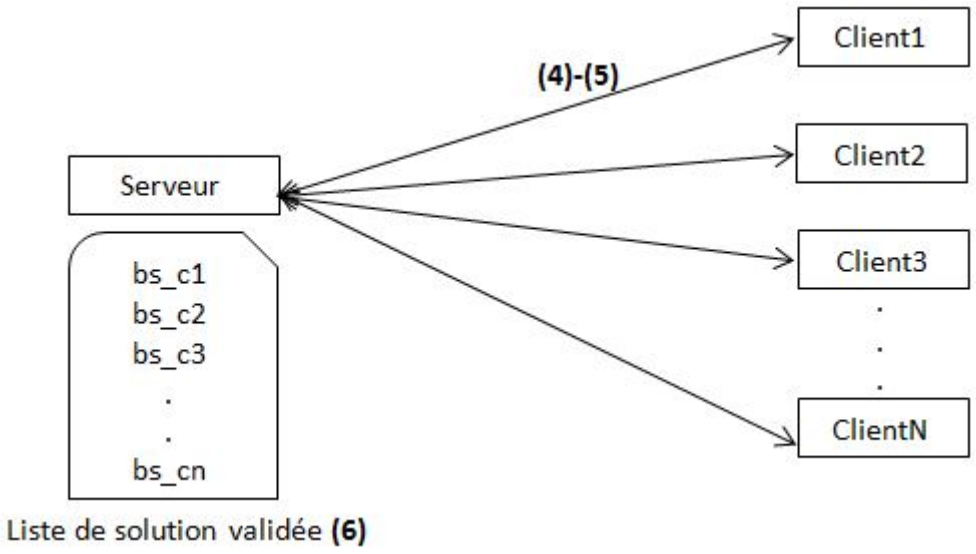


FIGURE 5.4 – Deuxième phase du modèle (la validation)

- **(4)** Le serveur envoi des sous ensemble de solutions aux clients afin de recalculer leur valeur de fitness et de s'assurer qu'il s'agit bien d'une permutation.
- **(5)** Les clients renvoient les valeurs obtenus au serveur.
- **(6)** Le serveur et selon les résultats reçus valide la liste des solutions dont il dispose.

5.3.2.3 Troisième phase (Le choix de la meilleure solution)

À la fin de la phase de validation, le serveur envoi la liste des résultats validé aux clients afin de faire un tri et de trouver la meilleure solution parmis les autres :

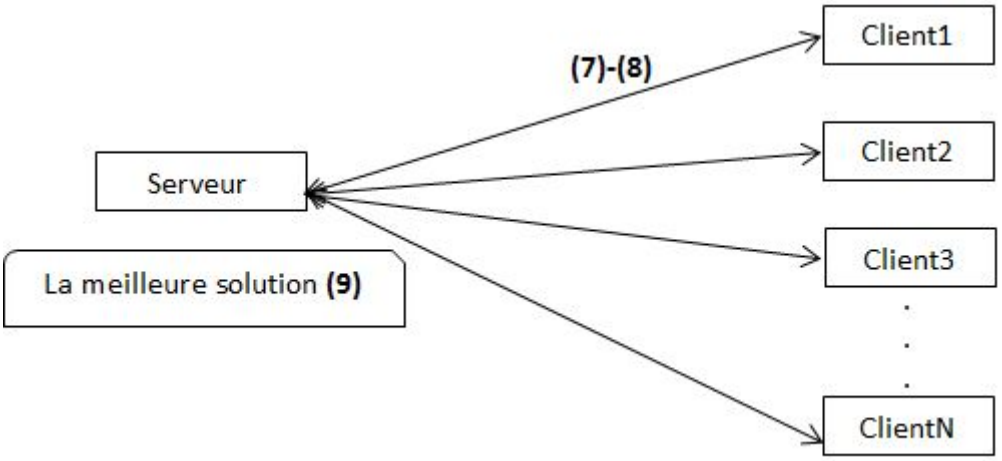


FIGURE 5.5 – Quelque instances du problème du flowshop (Eric Taillard)

- **(7)** le serveur choisi un client parmi les clients disponible et ayant de meilleure performance et lui envoi la liste des solutions pour pouvoir trouver la meilleure parmi autres.
- **(8)** Le client en question renvoi la solution obtenu au serveur.
- **(9)** A la fin de l'étape 8 le serveur dispose de la meilleure solution trouvée par l'exécution de l'application.

5.4 Présentation des résultats

Dans le site suivant (http ://mistic.heig-vd.ch/taillard/problemes.dir/ ordonnancement.dir/flowshop.dir/best_lb_up.txt) qui un certains nombre de résultats, fait par le professeur Eric Taillard (*university of Applied Sciences of Western Switzerland*). Eric Taillard, à proposer plusieurs instances du problème en leurs spécifiant un ID . Dans ce qui suit nous allons reprendre quelque ID (figure 5.6) des instances afin de pouvoir éffectuer des testes.

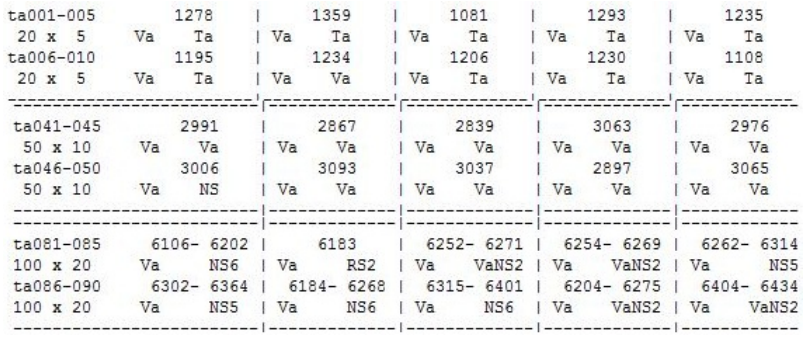


FIGURE 5.6 – Troisième phase du modèle (Le choix de la meilleure solution)

5.4.1 Résultats obtenus en séquentiel (sur une seule machine)

Dans cette partie du travail nous avons fait plusieurs séries d'exécutions répétées sur des populations différentes en changeant d'ID (instance du problème selon le site), afin d'avoir des contraintes de nombre de machines et de taille d'individu différentes sur une machine à processeur core 2 Duo Intel 2.00 Ghz, 2 Go de ram et un système 32 bits. Nous avons répéter l'exécution de chaque ID (10 fois) car l'AG est basé sur l'aléatoire. Les résultats de ces tests sont montrés dans les tableaux suivants :

1. **Première série : id = 10, taille de l'individu (nombre de jobs) = 20, nombre de machines = 10, taille de la population = 100**

Exécutions/ID	1	2	3	4	5	6	7	8	9	10
Temps d'exécution en seconde	11.678	11.685	11.779	11.713	11.762	11.671	11.751	11.744	11.744	12.522
Fitness	1136	1151	1188	1185	1161	1140	1149	1155	1174	1174

TABLE 5.1 – Réultats du premier teste ID=10, taille de la population=100

Le tableau suivant résume les résultats obtenus en faisant point sur le max, le min et la moyenne du temps d'execution aissi que la valeur de la fonction de

fitness :

	Max	Min	Moyenne
Temps d'exécution	12.522	11.671	11.804
Fitness	1188	1136	1161

TABLE 5.2 – Analyse des valeurs obtenus ID=10, taille de la population=100

2. **Deuxième série : id = 10, taille de l'individu = 20, nombre de machines = 10, taille de la population = 200**

Exécutions/ID	1	2	3	4	5	6	7	8	9	10
Temps d'exécution en seconde	18.700	18.638	18.721	18.849	18.679	11.671	18.548	18.771	18.979	18.702
Fitness	1152	1133	1179	1136	1127	1174	1136	1152	1137	1152

TABLE 5.3 – Réultats du premier teste ID=10, taille de la population=200

le tableau suivant résume les résultats obtenus en faisant point sur le max, le min et la moyenne du temps d'execution aissi que la valeur de la fonction de fitness :

	Max	Min	Moyenne
Temps d'exécution	18.979	18.548	18.756
Fitness	1179	1127	1147

TABLE 5.4 – Analyse des valeurs obtenus ID=10, taille de la population=200

3. **Troisième série : id = 10, taille de l'individu = 20, nombre de machines = 10, taille de la population = 1000**

le tableau suivant résume les résultats obtenus en faisant point sur le max, le min et la moyenne du temps d'execution aissi que la valeur de la fonction de fitness :

Exécutions/ID	1	2	3	4	5	6	7	8	9	10
Temps d'exécution en seconde	74.280	74.227	74.159	74.400	74.179	74.247	74.484	74.188	74.479	74.549
Fitness	1127	1123	1142	1147	1156	1127	1128	1108	1127	1134

TABLE 5.5 – Réultats du premier teste ID=10, taille de la population=1000

	Max	Min	Moyenne
Temps d'exécution	74.549	74.159	74.3192
Fitness	1156	1108	1132

TABLE 5.6 – Analyse des valeurs obtenus ID=10, taille de la population=1000

4. **Première série : id = 41, taille de l'individu = 50, nombre de machines = 10, taille de la population = 100**

Exécutions/ID	1	2	3	4	5	6	7	8	9	10
Temps d'exécution en seconde	40.114	39.425	39.348	39.402	39.361	39.413	40.382	39.419	39.702	39.977
Fitness	3308	3226	3276	3262	3285	3262	3283	3269	3247	3262

TABLE 5.7 – Réultats du premier teste ID=41, taille de la population=100

le tableau suivant résume les résultats obtenus en faisant point sur le max, le min et la moyenne du temps d'execution aissi que la valeur de la fonction de fitness :

	Max	Min	Moyenne
Temps d'exécution	40.382	39.348	39.654
Fitness	3308	3226	3268

TABLE 5.8 – Analyse des valeurs obtenus ID=41, taille de la population=100

5. **Deuxième série : id = 41, taille de l'individu = 50, nombre de machines = 10, taille de la population = 200**

Exécutions/ID	1	2	3	4	5	6	7	8	9	10
Temps d'exécution en seconde	63.109	63.287	63.393	64.100	63.715	63.590	63.747	63.620	63.505	63.659
Fitness	3292	3292	3202	3255	3238	3288	3239	3235	3304	3246

TABLE 5.9 – Réultats du premier teste ID=41, taille de la population=200

le tableau suivant résume les résultats obtenus en faisant point sur le max, le min et la moyenne du temps d'execution aissi que la valeur de la fonction de fitness :

	Max	Min	Moyenne
Temps d'exécution	64.109	63.287	63.572
Fitness	3304	3202	3259

TABLE 5.10 – Analyse des valeurs obtenus ID=41, taille de la population=200

6. **Troisième série : id = 41, taille de l'individu = 50, nombre de machines = 10, taille de la population = 1000**

Exécu/ID	1	2	3	4	5	6	7	8	9	10
Temps d'exécution en seconde	255.708	256.056	256.083	255.533	256.808	255.922	256.294	256.167	256.807	256.538
Fitness	3212	3230	3238	3209	3190	3282	3219	3236	3248	3282

TABLE 5.11 – Réultats du premier teste ID=41, taille de la population=1000

le tableau suivant résume les résultats obtenus en faisant point sur le max, le min et la moyenne du temps d'execution aissi que la valeur de la fonction de fitness :

	Max	Min	Moyenne
Temps d'exécution	256.808	255.533	255.191
Fitness	3282	3190	3234

TABLE 5.12 – Analyse des valeurs obtenus ID=41, taille de la population=1000

7. **Première série : id = 81, taille de l'individu = 100, nombre de machines = 20, taille de la population = 100**

Exécu/ID	1	2	3	4	5	6	7	8	9	10
Temps d'exécution en seconde	144.025	144.000	143.821	143.845	143.925	143.463	143.849	143.651	144.197	143.953
Fitness	6945	6949	6857	6913	6998	6993	6912	6960	6951	6936

TABLE 5.13 – Réultats du premier teste ID=81, taille de la population=100

le tableau suivant résume les résultats obtenus en faisant point sur le max, le min et la moyenne du temps d'execution aissi que la valeur de la fonction de fitness :

	Max	Min	Moyenne
Temps d'exécution	144.197	143.463	143.872
Fitness	6998	6857	6941

TABLE 5.14 – Analyse des valeurs obtenus ID=81, taille de la population=100

8. **Deuxième série : id = 81, taille de l'individu = 100, nombre de machines = 20, taille de la population = 200**

Exécu/ID	1	2	3	4	5	6	7	8	9	10
Temps d'exécution en seconde	232.023	231.758	232.374	232.052	231.121	233.177	231.783	232.405	232.085	231.669
Fitness	6979	7114	6952	6861	6868	6929	6921	6934	6900	6938

TABLE 5.15 – Réultats du premier teste ID=81, taille de la population=200

le tableau suivant résume les résultats obtenus en faisant point sur le max, le min et la moyenne du temps d'execution aissi que la valeur de la fonction de fitness :

	Max	Min	Moyenne
Temps d'exécution	233.177	231.121	231.844
Fitness	7114	6861	6939

TABLE 5.16 – Analyse des valeurs obtenus ID=81, taille de la population=200

9. **Troisième série : id = 81, taille de l'individu = 200, nombre de machines = 20, taille de la population = 1000**

Exécu/ID	1	2	3	4	5	6	7	8	9	10
Temps d'exécution en seconde	936.964	1219.947	932.916	937.885	936.016	931.767	934.848	940.145	1208.100	930.709
Fitness	6887	6887	6873	6997	6956	6888	6927	6851	6930	6881

TABLE 5.17 – Réultats du premier teste ID=81, taille de la population=1000

le tableau suivant résume les résultats obtenus en faisant point sur le max, le min et la moyenne du temps d'execution aissi que la valeur de la fonction de fitness :

	Max	Min	Moyenne
Temps d'exécution	1219.947	931.767	990.929
Fitness	6997	6851	6908

TABLE 5.18 – Analyse des valeurs obtenus ID=81, taille de la population=1000

5.4.1.1 Analyse des résultats obtenus en séquentiel

Après avoir récupéré et traité les résultats de nos séries d'exécutions nous avons pu mettre en place les tableaux 5.19 et 5.20 qui représentent respectivement le temps d'exécution et la valeur de la fitness en fonction de la taille de la population et de l'ID de l'instance :

Taille pop/id	10	41	81
100	11.8049	39.6543	143.8728
200	18.7564	63.5725	231.8445
1000	74.3192	255.1916	990.9297

TABLE 5.19 – résumé des temps d'exécution

Taille pop/id	10	41	81
100	1161	3268	6941
200	1147	3259	6939
1000	1132	3234	6908

TABLE 5.20 – résumé valeurs de la fonction de fitness

Les résultats des deux tableaux précédents nous ont ensuite permis de donner lieu aux deux diagramme (5.7 et 5.8) suivants en utilisant le langage R, qui les illustrent avec une bonne précisions nous permettant ainsi de mieux les analyser et de mieux les interpréter :

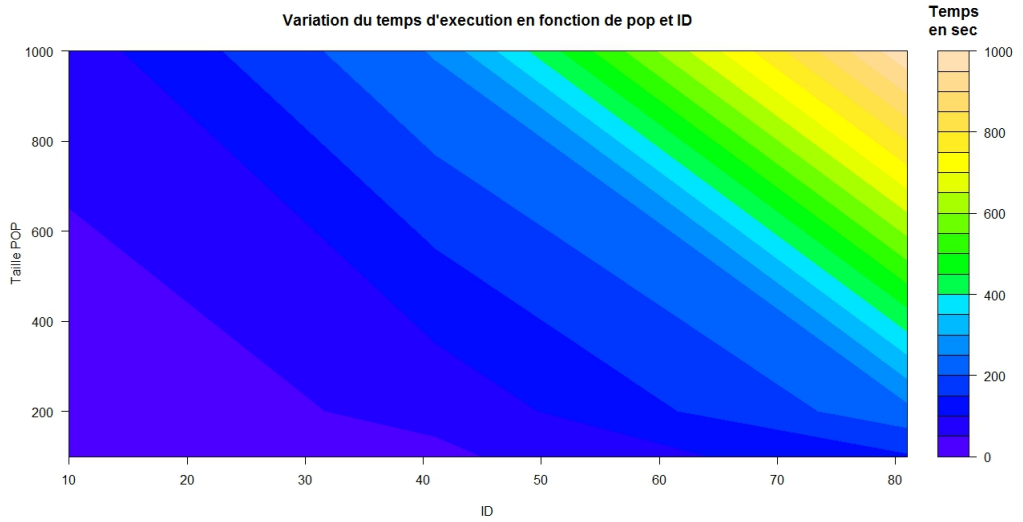


FIGURE 5.7 – variation du temps en fonction de la taille de la population et l'instance du problème

Nous voyons clairement dans ce graphe la relation qui existe entre la taille de la population l'ID(ou plus précisément la taille de l'individu) et le temps d'exécution. En effet plus la population est de taille importante et plus le temps d'exécution est grand et il en va de même pour la taille de l'individu. Le temps d'exécution arrive à des valeurs extrêmes lorsque la taille de la population et la taille de l'individu sont tout deux de tailles importantes.

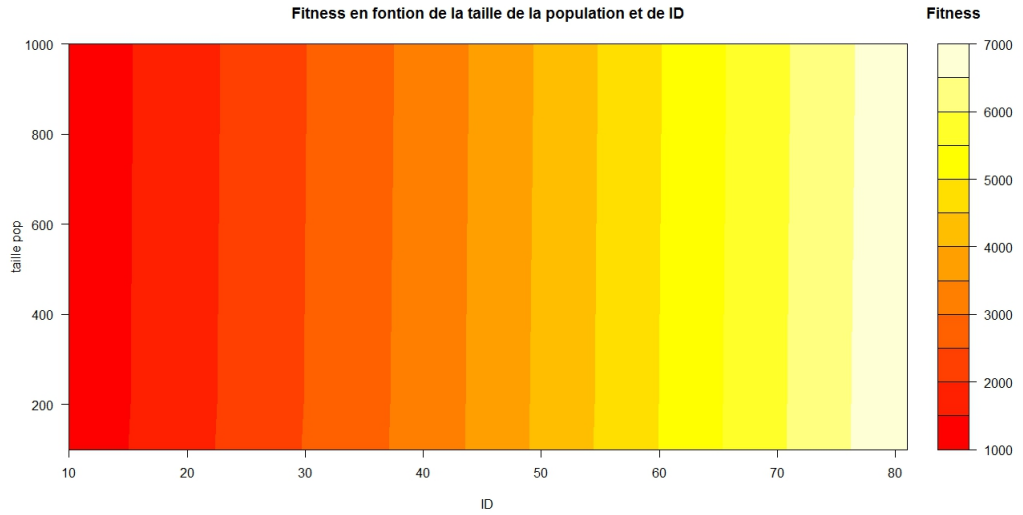


FIGURE 5.8 – variation de la fitness en fonction de la taille de la population et l'instance du problème

Il est clair d'après ces résultats que ce qui est le plus contraignant reste le temps d'exécution qui varie selon deux facteurs essentiels : la taille de la population et la taille de l'individu. Dans les cas pratiques réels ces deux facteurs sont souvent de très grandes tailles alors qu'on arrive déjà à un temps d'exécution qui se mesure en milliers de secondes avec une taille de population égalant à 1000 et celle de l'individu égalant à 200. Mais alors comment faire pour diminuer ce temps d'exécution qui semble n'avoir aucune borne ? La distribution des tâches du calcul entre plusieurs machine peut être une bonne initiative ayant déjà fait ses preuves antérieurement comme expliqué en [1], [2] et [3].

5.4.2 Résultats obtenus avec une exécution sous BOINC

5.4.2.1 Teste sur 10 machine avec un seul cœur

Maintenant que notre modèle est en fin prêt à être implémenter, après la création de notre projet en suivant les étapes décrites à l'Annexe A et l'ajout de notre application comme décris à L'Annexe B, nous avons fait une nouvelle série de tests en distribué avec 10 clients connectés au projet et possédant les caractéristiques

suivantes :

- RAM : 2Go
- Processeur : GenuineIntel Intel(R) Core(TM) i3 CPU 540 @ 3.07GHz

Nous avons utilisé des machines virtuelles sur chaque ordinateur (vitualBox), ayant les caractéristiques suivantes :

- Système : ubuntu 12.04 32bits.
- Processeur : nous avons utilisé un seul cœur de 3.07GHz.
- RAM : 912 Mo.

Les résultats obtenus sont décrit dans les tableaux suivants :

Clients	temps d'exécution en sec
1	101.25
2	100.00
3	100.702
4	101.289
5	100.358
6	100.658
7	102.387
8	100.725
9	99.991
10	100.280

TABLE 5.21 – Les temps d'exécution obtenus sur les 10 machines

Une possible représentation graphique de ces résultats est donnée comme suit :

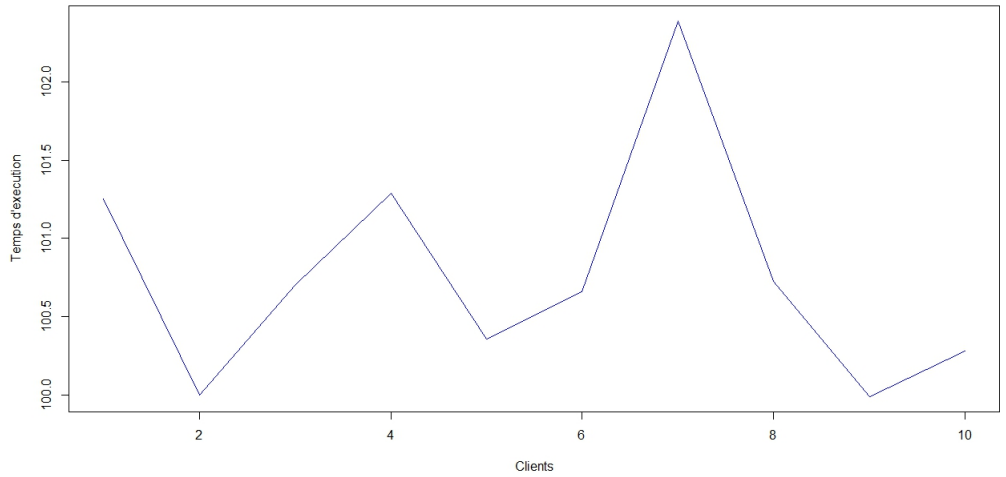


FIGURE 5.9 – variation des temps d'exécution sur les différents clients

Nous obtenons ainsi :

Temps d'exécution de tâche max	Temps d'exécution de tâche min	Temps d'exécution de tâche moyen	Temps d'exécution de l'AG
102.387	99.991	100.764	6941

TABLE 5.22 – résumé des temps d'exécution

En comparaison avec les résultats des premiers tests effectués en séquentiel et dont le temps d'exécution moyens était de 990.9297 secondes, nous avons pu obtenir un gain en temps d'exécution estimé à 96.7%, soit un temps d'exécution obtenu de 9.67 fois plus petit que le temps séquentiel.

5.4.2.2 Teste sur 10 machines avec 8 cœurs

Même si les machines que nous avons utilisées durant les tests antérieurs sont dotées de processeurs à 8 cœurs, nous n'en avons exploité qu'un seul. Durant la présente étape nous allons refaire les tests effectués en distribués en utilisant 10 ordinateurs doté d'une machine virtuelle avec 8 cœurs. Nos résultats sont ainsi présentés dans le tableau suivant :

Clients	Nombre de tâches exécutées	temps nécessaire
1	3	100.039
2	0	NAN
3	0	NAN
4	2	101.259
5	0	NAN
6	1	102.606
7	4	103.017
8	0	NAN
9	0	NAN
10	0	NAN

TABLE 5.23 – Les temps d'exécution obtenus sur les 10 machines

Ces résultats ne montrent aucune amélioration dans le temps d'exécution de l'application, cependant nous constaterons que le nombre de machines participantes dans le calcul a nettement diminué, le fait est que chaque machine peut exécuter jusqu'à 8 tâches en parallèle.

Conclusion

Nous avons ainsi conclus notre travail par ce chapitre qui était une suite de tests dont certains était avec une proposition mise en place après une étude détaillé. La comparaison des résultats de ces tests nous ont permis de démontré l'efficacité de cette dénière en matière d'amélioration du temps de calcul.

Conclusion générale

Dans ce travail, embelli par sa modestie, nous avons acheminé les étapes avec égards pour chaque information récolté même la plus moindre afin d'arrivé à proposé une solution à une problématique qui n'est pas sans importance dans le milieu des systèmes distribués.

Nous avons ainsi fait le tour des termes techniques les plus utilisés dans la terminologie de notre travail et qui est aussi le jargon du domaine du calcul parallèle et distribué, afin que le lecteur ne se perde pas dans les termes riches en complexités de ce dernier.

Le deuxième chapitre quand à lui, nous a servis de support pour poser notre problématique, à la quelle nous avons fait suivre un résumé d'une étude détaillé de sujets publiés ayant pu abordés similairement des traits de nos questionnements.

Les Algorithmes génétique étant notre cas d'étude pratique principale, nous avons trouvé nécessaire de détailler ce qu'ils sont, leur fonctionnement, comment les modélisé et comment les implémenté dans le chapitre trois.

Comment ne pas parler de BOINC ? L'outil star de notre travail, nous avons consacré le chapitre quatre entièrement à l'étude de cette plateforme en mettant en évidence son architecture, les outils nécessaire à son fonctionnement et son fonction-

nement lui-même.

Le chapitre cinq représente quand à lui le cœur de notre travail nous y avons détaillé, les outils que nous avons utilisé en jumelage avec BOINC, une modélisation de notre solution avec une méthode d'implémentation, une technique de validation de résultats et enfin une série de tests qui ont prouvé l'efficacité de notre modèle.

Bien que ce travail ait constitué un long parcours pour arriver a une sortie possible, la solution obtenu étant fonctionnel, facile à mettre en œuvre et efficace, elle peut encore être amélioré, nous avons des idées pouvons être posés comme perspectives d'amélioration, nous les citons comme suit :

- étudier le problème du Scheduling, en effet cela serait intéressant de pouvoir classé les machines participantes afin de mieux gérer leurs puissances
- améliorer le système de validation dans le cas ou le nombre de résultats serait trop important, étudier une manière de paralléliser cette dernière serait d'un grand apport.

A

Installation et configuration d'un projet et d'un serveur BOINC

Introduction

L'installation et la configuration de BOINC Server peut être déroutante et parfois même mener à la confusion surtout pour les novices qui doivent appréhender une multitude d'outils et parvenir à une configuration homogène permettant l'harmonie de fonctionnement entre ces derniers. Cela est dû d'une part au manque de détails dans la documentation officielle de la plateforme et d'autre part à l'omission de mises à jours aux documentations annexes. Nous avons donc trouver intéressant pour ne pas dire primordial de montrer les différentes étapes que nous avons suivis durant notre travail afin de mener à bien ces deux processus qui pourraient sembler pour

certains ardus.

A.1 Installation des outils nécessaires au fonctionnement de BOINC Server

Ce que l'ont peux constater aux premiers abords, c'est que les outils nécessaires au bon fonctionnement de la plateforme sont très nombreux et doivent être compatibles entre eux dans leurs versions respectives. Nous avons donc essayé de trouver une solution faisant abstraction de ces outils ainsi que de leur nombre tout en s'assurant de leur installation et de leur bon fonctionnement, après tout faire abstraction des opérations compliquées à l'utilisateur tout en s'assurant de la fiabilité du service offert n'est il pas la devise d'un bon développeur ? Nous somme ainsi tomber sur un package très utile développé par la communauté Debian, il se nomme boinc-server-maker. L'installation de ce package permet l'installation de tout les outils nécessaires au fonctionnement de notre plateforme et cela sans se soucier du nombre ou des noms de ceux-ci. Son installation est assez simple sous Linux il suffit d'exécuter la commande : **sudo apt-get install boinc-server-maker** A noter que lors de l'installation un mot de passe sera demander, pour l'outil MySQL, ce mot de passe sera ensuite utilisé par chaque client BOINC afin d'accéder à la base de données du projet, il est donc tres important de s'en rappeler.

A.2 Création d'un utilisateur et d'un groupe pour BOINC

Le client BOINC lors de son exécution fait en permanence plusieurs appels système. Pour des raisons de sécurité, le mieux serait de lui créer un utilisateur et un groupe. Ceci ce fait grâce aux deux instructions :

```
sudo adduser boincadm
sudo usermod -aG boincadm www-data
```

A.3 Téléchargement et compilation du code source BOINC

Le code source contient les scripts de plusieurs plusieurs programmes faisant fonctionner le serveur et permettant des actions tel la création de projet, la distribution des tâches aux clients, la gestion des erreurs, ... etc.

```
git clone git://boinc.barkeley.edu/boinc-v2.git boinc
cd boinc
./_autosetup
./configure --disable-client --disable-manager
make
```

A.4 Création et configuration d'un projet BOINC

A.4.1 Création et configuration d'une base de données MySQL pour le projet

Avant de créer le projet en lui-même la base de donnée liée à ce projet doit exister et les privilèges de l'utilisateur BOINC être fixés comme listés dans les instructions qui suivent :

```
mysql -u root -p;

DROP DATABASE IF EXISTS $dbname;

CREATE USER '$dbuser'@'localhost' IDENTIFIED BY '$dbpasswd';

GRANT ALL PRIVILEGES ON $dbname.* TO '$dbuser'@'localhost';

GRANT CREATE,DROP ON *.* TO 'db_user'@'localhost'

GRANT
SELECT,INSERT,UPDATE,DELETE,CREATE,REFERENCES,INDEX,ALTER,CRE
ATE TEMPORARY TABLES, LOCK TABLES ON  `db_name`.* TO
'db_user'@'localhost';
```

A.4.2 Création d'un projet BOINC

Ceci se fait grâce au programme make_project se trouvant dans le dossier tools du code source. Ainsi pour créer un projet BOINC nous devons exécuter l'instruction qui suit :

A.4.2.1 Explication des paramètres

- **- -url_base** : URL par défaut du projet.
- **- -db_name** : Nom de la base de données liée au projet.
- **- -db_user** : Nom de l'utilisateur de la base de données.
- **- -db_passwd** : Mot de passe pour la base de données.
- **- -drop_db_first** : Supprime la base de données d'une précédente installation.
- **- -delete_prev_inst** : remplace la racine du projet si elle existe.
- **- -project_root** : Racine du projet.
- **$projectname** : Nom court du projet.
- **$projectnicename** : Nom long du projet.

```
sudo ./tools/make_project \
 --url_base "$hosturl" \
 --db_name "$dbname" \
 --db_user "$dbuser" \
 --db_passwd "$dbpasswd" \
 --drop_db_first  \
 --delete_prev_inst \
 --project_root /var/www/boinc/$projectname
 --delete_prev_inst \
 --project_root /var/www/boinc/$projectname \
  "$projectname" "$projectnicename"
```

A.4.3 Modifier les permissions des dossiers du projet

Afin de permettre l'accès de différents dossiers du projet à partir de n'importe quelle machine volontaire liée au projet, il est nécessaire de modifier les permissions de ces derniers et cela en exécutant les instructions suivantes à partir du dossier du projet :

```
sudo chown boincadm:boincadm  -R.
sudo chmod g+w -R.
sudo chmod 02770 -R upload html/cache html/inc html/languages
      html/languages/compiled html/user_profile
```

A.4.4 Lancement du projet

A partir de cet instant précis le projet est totalement opérationnel et nous pouvons désormais lancer ses programmes d'arrière plan (Aussi appelés daemons). Pour ce faire il suffit de lancer à partir du dossier du projet le script :

```
sudo ./bin/start
```

A.4.5 Configuration de l'interface web

Chaque projet BOINC dispose d'une interface web accessible sous réseau et permettant à partir de n'importe quelle machine connectée de consulter la page web dédiée au projet ou d'admistrer la base de donnée du projet si permission accordée. De ce fait la configuration d'une telle interface repose d'une part sur la protection de la page d'administration par un mot de passe et d'autre part de configurer le serveur apache afin de rendre les différentes pages web du site fonctionnelles.

A.4.5.1 Protection de l'interface d'administration par un mot de passe

Ceci se fait tout simplement par une protection htaccess, les pages web de cette interface se trouvant dans le dossier html/ops du projet il suffit d'exécuter l'instruction :

```
sudo htpasswd -c html/ops/.htpasswd USERNAME
```

A.4.5.2 Configuration du serveur apache

Au moment de la création du projet, un fichier dont l'extention est.httpd.conf est généré. Ce fichier contient les configurations nécessaires au bon fonctionnement de l'interface web via apache. Donc afin de configurer ce dernier il suffit de faire en sorte qu'il prenne en considération ce fichier. Ceci se fait comme suit :

```
sudo cp ${projectname}.httpd.conf  /etc/apache2/sites-available/ \
sudo a2ensite  ${projectname}.httpd.conf \
 sudo /etc/init.d/apache2 reload
```

Conclusion

Dans cette partie nous avons essayé au mieux de détailler les différentes étapes qui nous ont permis l'installation et la configuration de la plateforme BOINC ainsi que la création et la configuration d'un Projet BOINC. Ceci dit ces étapes sont fonctionnelles au moment de la rédaction de ce travail et peuvent ne pas fonctionner ultérieurement à cause d'éventuelles mises à jour.

B

Ajout et déploiement d'applications

Introduction

Après la création d'un projet BOINC, la première étape que l'utilisateur voudra surement effectuer est celle de l'ajout d'applications, de la création des tâches et de leurs distributions sur les différents clients participant au projet. Ce chapitre a pour but de montrer les différentes étapes par lesquelles nous sommes passés pour accomplir ce fait.

B.1 Ajout d'applications

Les applications du projet BOINC créé sont stockées au niveau du dossier app/ se trouvant dans l'arborescence du projet. Afin d'ajouter une application au projet, nous devons procéder par étapes comme suit :

- Créer les dossiers de l'application dont la racine primaire est app/ et suivant une architecture bien spécifique qui a été définit par les concepteurs de la plateforme.
- Mettre en place l'exécutable de l'application.
- Ajouter l'entrée de l'application à la base de données du projet.

B.1.1 Création des dossiers de l'application

Ces dossiers sont le nom de l'application, les différentes versions de l'application (un dossier par version), les différentes plateformes sur lesquelles l'application est supposé tourné (un dossier par plateforme, et doivent être créées selon une certaine architecture à partir du dossier " app/ " du projet et dans ce cas un schéma serait beaucoup plus explicatif qu'un texte :

B.1.2 Mettre en place l'exécutable de l'application

L'exécutable de l'application probablement créé doit être présent dans le dernier dossier de l'arborescence précédemment créée (cela va de soit que pour plusieurs versions ou plusieurs plateformes, plusieurs exécutables doivent être ajoutés). Le nom de cet exécutable doit avoir une forme bien spécifique selon les règles établies par les concepteurs dela plateforme comme suit : *NomApp_VersionApp_NomPlatforme.*

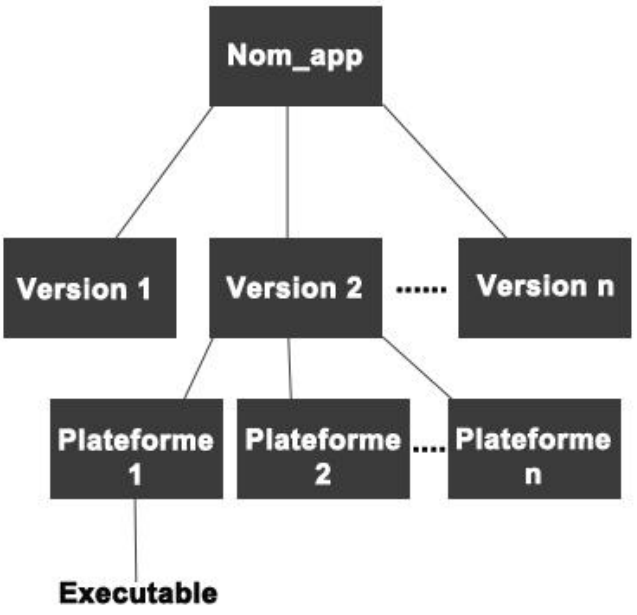


FIGURE B.1 – Arborescence des dossiers de l'application

B.1.3 Ajouter l'entrée de l'application à la base de données de projet

Pour ce faire nous devons d'abord éditer le fichier " project.xml " afin d'y ajouter les entrées nécessaire à notre application. Ce fichier se présente comme suit dans sa forme par défaut : Nous nous contenterons d'ajouter les entrées de l'application et des plateformes nécessaires suivant le schéma du fichier. Ceci fait nous pouvons alors lancer le script suivant à partir de la racine du projet : *bin/xadd* Ce script ajoutera les entrées nécessaires à la base de données du projet. Et en fin :*sudo ./bin/update_versions* Ce qui rendra les versions opérationnelles.

B.2 création de tâches

Les applications et les données du projet peuvent être partagées entre plusieurs tâches. Pour créer une tâche certains aspects doivent être connus

– Les fichiers d'entrées (Input files) : Ce sont les fichiers à partit des quels une

```
<boinc>
 <app>
  <name>example_app</name>
  <user_friendly_name>exampleAPP</user_friendly_name>
 </app>
 <platform>
  <name>i686-pc-linux-gnu</name>
  <user_friendly_name>Linux/x86 32bit</user_friendly_name>
 </platform>
 <platform>
  <name>x86_64-pc-linux-gnu</name>
  <user_friendly_name>Linux/amd64 64bit</user_friendly_name>
 </platform>
</boinc>
```

application travail, ils contiennent généralement des données que l'on envoi à une application afin d'y effectuer des opérations et des transformations.

- Le fichier work unit : Ce fichier est spécifique à chaque tâche, son emplacement est impérativement dans le dossier "templates/" du projet et contient une description de comment la tâche doit fonctionner.
- Le fichier result : Ce fichier est aussi spécifique à chaque tâche et doit se trouver comme son prédécesseur dans le dossier "templates/" du projet. Ce fichier contient une description de la forme du résultat attendu à l'exécution de la tâche.

Malheureusement il nous ais impossible ici de détailler le fonctionnement et l'architecture des fichiers " templates ". il faudrait surement tout un chapitre pour chacun des deux fichiers. Cependant nous invitons le lecteur intéréssé par l'obtention de plus d'information à consulter [1], il pourra y trouver une description détaillé sur ce

sujet.

Une fois les fichiers templates créés et mis en place, nous pouvons créer des taches à l'application en exécutant le script :

```
sudo ./bin/create_work -appname nom_app -wu_name test -wu_template
templates/nom_wu -result_template templates/nom_result in
```

Explication des paramètres

- **-appname** : nom de l'application.
- **-wu_name** : nom de la tâche.
- **-wu_template** : nom et emplacement du fichier work unit.
- **-result_template** : nom et emplacement du fichier result.
- **in** : nom du fichier d'entrée de l'application.

Conclusion

Nous avons montré dans ce chapitre et dans le cadre de notre travail, les étapes que nous avons suivis pour répondre aux questions de l'ajout d'application à un projet BOINC et de la création des tâches pour une application et ce pour enlevé tout ambigüité dans l'esprit du lecteur disvirant utilisé la plateforme ou seulement avoir connaissance du travail nécessaire.

Bibliographie

[1] Jean-Charles Boisson, "Algorithmes génétiques et calcul haute performance", Université de Reims. 2011.

[2] Radet Francois-Gérard, Soquet Amédée, "Algorithmes génétiques", 2004.

[3] Xavier Hüe, "Genetic Algorithms for Optimisation Background and Applications", Université de Edinburgh, 1997.

[4] MOHAMED ANOUAR TALEB, "Parallèlisation d'un algorithme génétique pour le problème d'ordonnancement sur machine unique avec temps de réglages dépendants de la séquence", Université à Chicoutimi, Avril 2008.

[5] El Maskaoui, Zakaria, "Application de la programmation orientée objets à l'optimisation discrète sous contraintes des structures métalliques formées de poutres via les algorithmes génétiques", 2007.

[6] LASSOUAOUI Nadia, HAMAMI Latifa, NOUALI Nadia, "Les algorithmes génétiques application à la segmentation des images", 2004.

[7] Antony Sam James, "Architectural Overview and Evaluation of Implementing a Parallel Genetic Algorithm in a distributed volunteering network", The University of YORK Department of computer Science, 2011

[8] Parallel computing for real time signal processing and control, TOKHI.M 2003

[9] Introduction to prallel architecture and the world of multicores, R. Govindarajan, 2011

[10] Distributed Shared Memory,Humayun Arafat 2012 OHIO state university

[11] Distributed shared memory : concept and systems, Jelica protic, Milio Tomaservic, University of Belgrade 1996.

[12] http ://www.embedded.com/design/mcus-processors-and-socs/4007623/Getting-started-with-multicore-programming-Part-1.

[13] Jimmy Broché, Aureliano Naviliat, Marie Rodriguez, Loïc Vaes ULB, "SETI@Home et le calcul distribué volontaire au service de la science".

[14] Paul Malécot, Derrick Kondo, Gilles Fedak, "XtremLab : une plateforme pour l'observation et la caractérisation des grilles de PC sur Internet", "Laboratoire de Recherche en Informatique Université Paris XI", octobre 2006.

[15] Sylvain DAHAN, "Mécanismes de recherche de services extensibles pour les environnements de grilles de calcul", thèse de doctorat, université de Franche-Comte, décembre 2005.

[16] Raphaël BOLZE, "Analyse et déploiement de solutions algorithmiques et logicielles pour des applications bioinformatiques à grande échelle sur la grille", thèse de doctorat, université de Lyon, Octobre 2008.

[17] Walid Saad. "Intégration des intergiciels Boinc et Condor et tolérance aux pannes dans BonjourGrid", mémoire de Master, Université de Tunis, école supérieure des sciences et techniques de Tunis, séptembre 2009.

[18] Laurent Birtz, "Le logiciel de calcul distribué Snowflakes", mémoire, Sherbrooke, Québec, Canada, février 2009.

[19] David P. Anderson, Eric Korpela, Rom Walton, "High-Performance Task Distribution for Volunteer Computing", article, Space Sciences Laboratory University of California, Berkeley.

[20] http ://boinc.berkeley.edu.

[21] Christian Ulrik Søttrup, Jakob Gregor Pedersen, "Developing Distributed Computing Solutions Combining Grid Computing and Public Computing", article, Department of Computer Science University of Copenhagen, 1st March 2005.

[22] Paul Malécot, "Les grilles de PC XtremWeb et BOINC", Laboratoire de Recherche en Informatique Paris XI, Orsay, Octobre 2006.

[23] http ://wiki.backtrack-fr.net/index.php/Introduction_au_spoofing, _sniffing,_et_aux_brutes_forceurs.

[24] http ://boinc-af.org. le 14/01/2013 à 19h00.

[25] Malek SMAOUI FEKI Viet Huy NGUYEN and Marc GARBEY, "Parallel Genetic Algorithm Implementation for BOINC", Department of Computer Science, University of Houston, Texas.

[26] http ://cc.in2p3.fr/Grille-informatique.

[27] Vincent Leroy,"Les réseaux pair-à-pair au service du calcul distribué", 1 février 2007.

[28] Nate Cole, Travis Desell, Daniel Lombrana Gonzalez, Francisco Fernandez de Vega, Malik Magdon-Ismail, Heidi Newberg, Boleslaw Szymanski, and Carlos Varela. "Evolutionary Algorithms on Volunteer Computing Platforms : The MilkyWayHome Project", article,Université d'Extremadura, 2011.

Résumé

La résolution des problèmes NP-Hard a longtemps été un sujet de recherche aux buts variant du rapprochement à la solution optimale à l'optimisation du temps de recherche dans l'espace de solutions. L'apparition des systèmes parallèles et distribués a pu guider les recherches vers des techniques de résolutions aux résultats de plus en plus optimales en divisant en ensemble de tâches ,destinées a être distribué sur différents calculateur, un plus grand problème de calcul. Comment diviser un problème en tâches ? Comment distribuer les tâches ? Sur quelle plateforme devrait-on effectuer le calcul ? C'est a ses questions que nous avons essayer de répondre dans ce travail en proposant un modèle de distribution que nous avons implémenté sous la plateforme BOINC appliqué à l'exemple du problème du Flow-Shop.

Mots clés **:** calcul volontaire, grilles de PC , les algorithmes génétiques.

Abstract

Resolving NP-Harp problems have been for a long time a research subject involving goals varying between getting close of the optimal solution and optimizing the researching time in the solution space. The apearance of parallel and distributed systems leaded the reseachs to technics of resolution giving results increasingly optimal by dividing in a set of tasks intended to be distributed on different calculators, a bigger problem. How to divide a problem in tasks ? How to distribute tasks ? What plateform should we use for computing ? We tried to answer these questions in this work by proposing a destribution model that we implement on the BOINC platform applied to the FlowShop problem.

Keywords **:** volonteer computing, desktop grid, genetic algorithm.