Optimisation parallèle hybride sur système pair à pair[1]

M. Mezmaz, N. Melab and E-G. Talbi

Laboratoire d'Informatique Fondamentale de Lille
LIFL - UMR CNRS 8022- Bâtiment M3
INRIA Futurs-DOLPHIN
59655 Villeneuve d'Ascq Cédex - FRANCE
{mezmaz,melab,talbi}@lifl.fr

Résumé Les problèmes d'optimisation combinatoire, du domaine académique ou du monde réel, sont majoritairement NP-difficiles. Pour des instances de dimensions importantes et/ou avec un nombre de contraintes élevé, la résolution sera donc toujours limitée par les ressources utilisées. L'utilisation du déploiement à large-échelle de type pair à pair (P2P) présente alors un intérêt double. Il permet d'aboutir efficacement à des solutions satisfaisantes en un temps raisonnable par une distribution des calculs, mais aussi le déploiement de modèles coopératifs hybrides naturellement parallèles afin d'améliorer la qualité des solutions produites et la robustesse des algorithmes mis en oeuvre. Les systèmes de calcul P2P tels que XtremWeb [8] [7] , JNGI/JXTA [10] ou Condor, ne diffèrent pas fondamentalement dans leur paradigme "coordinator/worker" pour lequel aucune forme de communication n'est permise entre les workers. Ceci nous a conduit à proposer un nouveau modèle de coordination : P2P-Linda, appelé ainsi parce qu'il est inspiré de Linda. Avec ce modèle, nous avons pu transformer un des middlewares P2P (XtremWeb) pour l'adapter aux besoins intrinsèques des méta-heuristiques coopératives. En terme de validation, nous avons expérimenté nos travaux sur un problème de flowshop. Les résultats obtenus sur un réseau d'une centaine de machines montrent l'efficacité et la robustesse de l'approche proposée.

Mots-Clefs. Calcul P2P ; Modèle de coordination ; Optimisation combinatoire multiobjectif ; méta-heuristiques parallèles hybrides ; Problème du flowshop ;

1 Introduction

Il existe en pratique une large variété de problèmes d'optimisation combinatoire. Ces problèmes sont souvent de grande taille car ils sont caractérisés par un nombre important de variables de décision et de contraintes. Ils sont également NP-difficiles pour la plupart. Même si la complexité des problèmes est réduite par l'utilisation de méta-heuristiques et l'hybridation de plusieurs algorithmes, il n'en demeure pas moins que le parallélisme s'avère un moyen efficace pour améliorer la performance de l'exécution, notamment l'utilisation du parallélisme à grande échelle basé sur les grilles de calcul ou les systèmes P2P. Dans cet article, nous nous intéressons aux méta-heuristiques parallèles hybrides sur des systèmes P2P. A notre connaissance, à ce jour, aucun travail de recherche n'a été publié sur cette thématique. Les systèmes P2P sont composés de centaines voire de milliers de PC volontaires. Il existe aujourd'hui des intergiciels (middlewares) tels que XtremWeb [8] [7], JNGI/JXTA [10], etc. pour l'exploitation de ces systèmes. Ces intergiciels sont très adaptés aux applications de type multiparamètres comme les simulations avec différents jeux de paramètres. Cependant, ils n'offrent pas de mécanismes de communication entre pairs (tâches). En effet, les seules communications autorisées sont de type père-fils (déploiement de tâches) et vice-versa (retour de résultats). L'objectif de notre travail s'articule autour de l'étude des schémas d'hybridation/coopération entre méthodes d'optimisation et leur adaptation en vue de leur passage à l'échelle. Ceci nous conduit à développer un nouveau langage de coordination : P2P-Linda, avec lequel XtremWeb a été étendu. L'expérimentation et la validation du modèle proposé ont été menées sur un algorithme qui appellé AGMA [3], après qu'une version P2P tolérante aux pannes de cet algorithme ait été développée.
La suite de l'article est organisée de la manière suivante : La **section 2** introduit la problématique de l'optimisation combinatoire multiobjectif. La **section 3** porte sur le calcul P2P. Dans la **section 4**, nous décrivons les expériences réalisées et les résultats obtenus. L'article sera clôturé par la **section 5** où nous présentons les conclusions et les perspectives de ce travail.

[1] Ce travail a été réalisé dans le cadre du projet GGM de l'ACI "Masse de données".

2 Optimisation combinatoire

Dans cette section, nous présontons quelques notions liées aux problèmes d'optimisation combinatoire multiobjectif, ainsi que les différentes méthodes de leur résolution, notamment l'hybridation de méta-heuristiques. Nous y décrivons également deux modèles parallèles connus des méta-heuristiques.

2.1 Problèmes d'optimisation combinatoire multiobjectifs

L'*optimisation combinatoire* est une discipline qui s'intéresse aux problèmes requérant de trouver la meilleur configuration parmi un ensemble fini mais très grand de configurations possibles. Chacune des configurations constitue *une solution* au problème et appartient a un espace appelé *espace de recherche*. Le choix de la solution se fait en fonction d'un ou de plusieurs *critères*. Dans l'optimisation combinatoire, les termes *critère*, *attribut*, et *objectif* sont utilisés indifféremment. Un problème peut être donc *uni-objectif* ou *multiobjectif*, selon qu'on s'intéresse respectivement à un ou plusieurs objectifs. Un*coût* est associé à chaque solution qui sera , selon le nombre d'objectifs du problème, une simple valeur ou un vecteur de valeurs. Résoudre un *problème d'optimisation combinatoire* (POC) revient donc à trouver la ou les solution(s) de coût *optimal*. Si la notion d'optimalité est simple a définir dans le cas d'un problème uni-objectif, elle n'est pas aussi évidente pour un problème multiobjectif. Généralement, l'optimalité pour les problèmes multiobjectifs est définie a l'aide de la relation de *dominance* entre vecteurs.

Définition.1 :
Soient $X = (X_1, .., X_N)$ *et* $Y = (Y_1, .., Y_N)$ *deux vecteurs et i,j* $\in$ *[1,N]*
X *domine* $Y \Longleftrightarrow (\forall i, X_i \leq Y_i) et (\exists j, X_j < Y_j)$

La relation de dominance constitue un *ordre partiel*, ce qui implique que plusieurs solutions optimales peuvent exister comme l'illustre la figure 1. Cet ensemble est appelé *front Pareto*.

Définition.2 :
Soit E un ensemble de vecteurs, et F le front Pareto de E,
$X \in F \Longleftrightarrow \nexists\ Y \in E\ /\ Y$ *domine* X

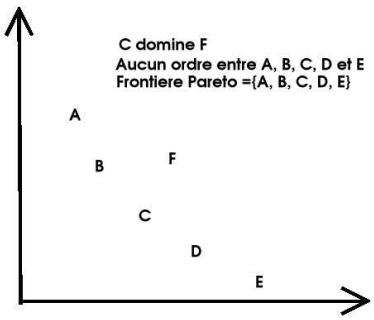

Fig.1. L'optimalité au sens Pareto

De nombreuses méthodes de résolution ont été développées. Elles sont classées généralement en deux grandes catégories : *les méthodes exactes* et *les méthodes approchées*. Le principe d'une méthode exacte consiste à énumérer, souvent de manière implicite, l'ensemble des solutions de l'espace de recherche. Cependant, la plupart des POCs sont *NP-complets*. Cela signifie qu'il n'existe vraisemblablement aucun algorithme capable de les résoudre en un nombre d'opérations polynomial par rapport à la taille des instances. On doit donc souvent renoncer à rechercher la ou les solution(s) optimale(s). Souvent, on est contraint de se contenter d'algorithmes approchés. Les méthodes approchées constituent une alternative très intéressante pour traiter les problèmes de grande taille si l'optimisation n'est pas primordiale. Une méthode approchée peut être soit une *heuristique* ou une *méta-heuristique*. A l'inverse d'une heuristique, conçue pour solutionner un POC précis, une métaheuristique s'adapte et s'applique à un grand nombre de POCs.

2.2 Hybridation et parallélisme

L'analyse des différentes méthodes de résolution, notamment les metaheuristiques, a révélé certains aspects de leur nature. En effet, certaines méthodes ont un comportement *explorateur*, tandis que d'autres ont un comportement plutôt *exploiteur*. La notion d'exploitation insiste sur la capacité d'une méthode à examiner en profondeur une zone de l'espace de recherche, alors que a la notion d'exploration met en avant

sa capacité de découvrir de nouvelles zones de recherche prometteuses. Pour avoir des méthodes ayant un comportement à la fois explorateur et exploiteur, des *hybridations* de métaheuriqtiques ont été proposées. Les meilleurs résultats sur de nombreux POC, académiques ou pratiques, sont obtenus grâce aux *méthode hybrides* [13]. Cependant, le temps de calcul nécessaire pour les méthodes hybrides peut devenir prohibitif. Une voie pour résoudre ce problème est leur parallélisation. L'exploitation d'environnements d'exécution parallèles à grande échelle permet, outre la distribution des calculs, le déploiement de modèles de résolution robustes basés sur l'hybridation d'algorithmes. Parmi les schémas de parallélisation de métatataheuristiques possibles, deux vont particulièrement nous intéresser :
Le modèle multidépart : consiste à lancer plusieurs métaheuristiques en parallèle. Les processus parallèles peuvent être indépendants ou coopératifs
Le modèle insulaire : est inspiré de certains comportements observés dans les niches écologiques. Dans ce modèle [5], plusieurs populations de solutions semi-indépendantes et faiblement couplées évoluent simultanément. La coopération prend la forme de flux migratoire de solutions. Diverses stratégies sont envisageables quant au choix des solutions à émettre vers les populations voisines, de la manière de les assimiler, de la topologie du modèle coopératif et enfin du critère de décision d'immigration.

3 Calcul P2P

C'est quoi un *calcul P2P* ? Les plateformes de *calcul P2P* existant, sont-elles exploitables pour le déploiement des méthodes hybrides multiobjectives ? Si la réponse est non, et nous allons voir par la suite que c'est le cas, quelles sont leurs limites et les solutions pour y remédier ? Autant de questions pour lesquelles, nous essayerons de répondre dans cette section. Nous allons voir que toutes ces interrogations nous ont conduit à proposer un modèle de coordination P2P : *P2P-Linda*, ainsi qu'une extension de XtremWeb par ce modèle.

3.1 Calcul distribué et systèmes P2P

Deux définitions du *calcul P2P* semblent faire l'unanimité, ce sont celles cités dans [11] et [1]. Dans [11],P2P est défini comme *une classe d'applications qui exploite les ressources -stockage, cycles de calcul, contenu et présence humaine- disponibles aux bords d'Internet.* Dans [1], *P2P a été défini comme un système distribué décentralisé, qui s'auto-organise et dans lequel toutes ou la plupart des communications sont symétriques.* Même si la deuxième définition est plus 'jolie', de nombreux scientifiques lui préfèrent la première définition, puisque celle-ci s'applique à des systèmes déjà admis comme P2P à savoir Seti@Home [2], Napster, JXTA/JNGI[10] et XtremWeb [8] [7]. Les dernier middleware cité est celui sur lequel nous avons travaillé. XtremWeb est un projet académique dont le but est la conception et le développement d'un environnement de calcul P2P pour des applications sécurisées et généralistes. Elles sont généralistes dans le sens qu'à la différence des projets comme SETI@Home, XtremWeb n'est pas dédié à une application particulière, mais configurable pour une classe d'applications. A l'instar de nombreux systèmes de calcul P2P, XtremWeb distingue trois rôles essentiels pour un pair : client, serveur et worker. Le rôle principal du **serveur**, appelé également coordinateur, est d'assurer la mise en relation entre les demandes de ressources de calcul émanant des clients et les ressources mises à disposition par les Workers. Le rôle du **worker** est de fournir les ressources de la machine d'un utilisateur pour l'exécution d'un calcul XtremWeb. Un **client** soumet une tâche en fournissant la référence de l'application et l'ensemble des paramètres qui la définissent. L'une des limites des plateformes de *calcul P2P* que nous avons testé est l'absence d'un mécanisme permettant aux workers d'échanger directement ou indirectement des données. Nous entendons par indirectement un échange par l'intermédiaire d'un troisième pair, le dispatcher par exemple. En effet, ces intergiciels sont construits pour supporter des applications de type *Single Program Multiple Data (SPMD).* Les applications *SPMD* ont démontré, dès le départ, qu'elles sont de bonnes candidates pour être distribuées. Les intergiciels de *calcul P2P* supposent que l'application à traiter est divisible en plusieurs morceaux de code dont l'exécution de chacun est indépendante des autres. Cependant, dans la plupart des applications sur lesquelles travaille la communauté de l'optimisation combinatoire, un mécanisme de collaboration entre workers est indispensable. D'où la nécessité d'adapter les intergiciels de calcul P2P par un modèle de coordination comme *Linda.*

3.2 Un nouveau modèle de coordination : P2P-Linda

Historiquement *Linda* [6] est l'un des premiers langages de coordination à être mis au point. Il se base sur le paradigme de la communication générative. Dans ce paradigme, l'échange de messages entre les processus se fait via une mémoire partagée dans laquelle un processus expéditeur dépose son message et de laquelle un processus récepteur le récupère. L'intérêt de ce paradigme est d'assurer une indépendance spatiale et temporelle entre les processus : un processus ignore l'endroit, i.e. position dans le réseau, et le moment auquel s'exécutent les autres processus.

Dans *Linda*, la mémoire partagée et un message sont appelés respectivement *espace de tuples* (ET) et *tuple*. ET est une collection de tuples, non pas un ensemble, puisque plusieurs exemplaires d'un même tuple peuvent s'y trouver simultanément. Un tuple est une suite finie et ordonnées de champs typés dont le premier est généralement une étiquette. Chaque champ contient soit une valeur typée, soit un appel de processus appelé *champ processus*. Un tuple qui ne contient que des valeurs typées est appelé *tuple de données*, ou *tuple passif*. Un tuple qui contient au moins un champ processus est appelle *tuple de processus*, ou *tuple actif*. Si un tuple contient plusieurs champs processus, la version originale de *Linda* ne précise pas si un ou autant de processus seront générés (Dans *C-Linda* plusieurs processus seront créés). Le résultat de l'exécution d'un processus sera affecté au champ processus correspondant. Dès qu'un tuple de processus termine son exécution, il se transforme en un tuple de données en remplaçant les champs processus par les valeurs de retour de leur processus respectifs. Contrairement à un tuple de données, qui est une entité passive, un tuple de processus est une entité active qui échange des tuples en générant, lisant et consommant d'autres tuples. L'action des processus sur ET se fait par l'intermédiaire de quatre *primitives*, ou *opérations*, définies dans le modèle :

- *out(T)* et *eval(F)* : selon que le tuple est passif(T) ou actif(F), une des deux primitives est utilisée pour le déposer dans ET.
- *in(P)* et *rd(P)* : sont respectivement employées pour prendre et lire, de ET, un tuple quelconque correspondant au *pattern* P. Contrairement à un tuple lu, un tuple pris est supprimé de ET.

Un pattern est lui-même une sorte de tuple passif contenant au moins un champ vide qu'on appelle *champ formel*. Par convention le nom d'un champ formel commence par un point d'interrogation (?). Un pattern P correspond à tuple T signifie que si un champ non formel, respectivement formel, de P et un champ de T ont le même ordre, alors ils ont la même valeur, respectivement sont du même type. Le pseudo-code, ci dessous, illustre le fonctionnement de *Linda*. Il s'agit de trouver tous les nombres premiers inférieurs à N.

Algorithm 1 Exemple Linda : Calcul de tous les nombres premiers inférieurs à N

```
// procedure pour verifier si un nombre est premier
procedure est_premier(i : entier) :booléen
for j :=2 to sqrt(i) do
  rd("premier",i, ?ok)
  if ok et (i mod j=0) then return false
end for
return true;
end procedure
// procedure principale
procedure main()
for i :=2 to N do
  eval("premier",i,est_premier(i)) // Lancement de N-1 tuples de processus
end for
for i :=2 to N do
  rd("premier",i, ?ok) // Lecture des résultats
end for
end procedure
```

Les N-1 processus lancés dans l'exemple ci-dessous, sont synchronisés : si l'entier I est inférieur à J alors le processus *est_premier(I)* va finir avant *est_premier(J)*. Une implémentation simple du problème des philosophes, juste avec *Linda* et sans sémaphores, est tout à fait possible. L'exemple précédent donne une idée de la capacité expressive du modèle, relativement aux peu de concepts qui le définissent, ainsi que de son pouvoir de synchronisation. Cependant, la nature des applications P2P, notamment l'importance du temps de communications et la volatilité des pairs, nous a conduit à étendre le modèle par trois types de primitives qui sont des version modifiées des primitives de base de *Linda* :

Primitives non bloquantes : *rd* et *in* sont des primitives bloquantes, à l'inverse de *eval* et *out*. Autrement dit, après chaque appel, le processus appelant se bloque tant qu'aucun tuple correspondant n'a pas été trouvé. Ceci peut s'avérer pénalisant en terme de ressources de calculs. Pour cette raison, nous avons repris les versions non bloquantes, pour *rd* et *in*, proposées dans les extensions :

- *inp(P)* et *rdp(P)*, idem que *in* et *rd*, sauf que si aucun tuple ne correspond au pattern P, *inp* et *rdp* renvoient *false* au lieu d'un tuple.

Primitives de groupe : Dans les applications parallèles, souvent la même primitive est évoquée plusieurs fois de suite avec des arguments différents (voir l'exemple précédent). Étant donnée, la durée des communications, relativement très longue dans un système a large échelle, une extension de *Linda* par des versions manipulant un groupe de tuples rendrait les applications exploitant le modèle plus performantes. Ceci nous a conduit à définir pour chacune des primitives précédentes une primitive de groupe équivalente.

- *ing(P)* et *rdg(P)*, idem que *in* et *rd*, sauf que *ing*, resp. *rdg* prend, resp. lit, tous les tuples dont le pattern correspond à P.
- *rdpg(P)* et *outpg(P)* sont les versions non bloquantes pour lire ou prendre un groupe de tuples ayant le même pattern.
- *evalg(F1,...)*, resp. *outtg(T1,...)*, est la version groupe de *eval*, resp. *out*. Elle ajoute à ET, en un seul appel, plusieurs tuples actifs, resp passifs.

Primitives de mise à jour : Les primitives de base de *Linda* sont soit dédiées à consulter, i.e.prendre ou lire, ou à ajouter des tuples. Une modification d'un tuple se traduit donc par l'exécution de deux primitives successives : un *in* suivi d'un *out*. Ce qui est, dans une application P2P, très coûteux en temps de communication, sans oublier tous les problèmes de synchronisation qu'engendrerait une machine qui tomberait en panne après l'exécution du *in*. D'où la nécessite d'enrichir le modèle par une primitive atomique dédiée à la mise à jour d'un tuple.

- *updateg(P, T)*, resp. *update(P, T)*, met à jour tous les tuples, resp. un tuple quelconque, dont le pattern correspond a P en leur, resp lui, affectant la valeur T.

4 Implémentation et expérimentation

Dans [3], une méthode hybride appelée *Adaptative Genetic Memitic Algorithm* (AGMA) a été présentée pour résoudre un problème du flowshop de permutation bi-objectif(FPB). Dans cette section, après avoir expliqué comment *P2P-Linda* a été impémenté sur *XtremWeb*, nous allons présenter FPB. Nous enchaînerons par las suite par une description de AGMA, de sa version P2P, et de la façon avec laquelle la tolérance aux pannes a été gérée. La section sera clôturée en présentant les résultats des expériences menées.

4.1 Implémentation de P2P-Linda sur XtremWeb

L'implémentation de ce modèle sur XtremWeb a été faite concrètement par l'ajout d'un *gestionnaire d'ET*, greffé sur le dispatcher, et d'une *API P2P-Linda/C*, rattachée au worker. Les workers ont été étendu par l'*API P2P-Linda/C* afin que l'API soit accessible aux processus qui s'y exécutent. Le *gestionnaire d'ET* a été placé au niveau du dispatcher à cause de l'architecture de XtremWeb : c'est le seul pair visible de tous les workers. L'*API P2P-Linda/C* est un ensemble de fonctions dont chacune est une implémentation d'une primitive de P2P-Linda. Deux fichiers *P2PLinda.o* et *P2PLinda.h* sont nécessaires à un programmeur pour exploiter l'API. Pour comprendre comment l'*API P2P-Linda/C* communique avec le *gestionnaire d'ET*, Il faut savoir que XtremWeb offrait déjà une classe java appelée *Client.java*. Cette classe sert d'interface avec le dispatcher, en utilisant un shell. Les fonctions de l'*API P2P-Linda/C* communiquent avec le *gestionnaire*

d'ET par l'intermédiaire de *Client.java*. Quand un programme invoque l'une des fonctions de l'API, cette fonction encapsule dans un message les arguments de l'appel, i.e. tuple(s) et/ou pattern. Un mécanisme combinant *tube* et *fork* a été utilisée pour que la fonction en question envoie ce message à *Client.java*. *Client.java* prend le relai par la suite en le faisant transmettre,par un appel RMI, pour le dispatcher. Le dispatcher se charge de son côté de l'aiguiller vers le *gestionnaire d'ET*. Le *gestionnaire d'ET* gère les tuples en les considérant comme des enregistrements d'une base de données. Il traduit les messages, provenant de l'*API P2P-Linda/C*, en requêtes SQL. Dans XtremWeb, MySQL doit être installé pour que le dispatcher fonctionne. A l'installation, une base de données (BD) XtremWeb est donc créée avec un certain nombre de tables. Nous avions eu à modifier les scripts d'installation pour qu'une autre table soit rajoutée a la BD. Cette table sera utilisée par le *gestionnaire d'ET* comme un ET pour les tuples passifs. Les tuples actifs, quant à eux, sont stockées sur une autre table déclarée par XtremWeb pour gérer les processus. Les deux tables constituent donc ET.

4.2 Problème du flowshop de permutation bi-objectif

Le problème de flowshop fait partie de la famille des problèmes d'ordonnancement. La variante du flowshop de permutation étudiée ici est une version simple parmi les nombreuses extensions de ce problème. Dans le flowshop, on dispose de plusieurs machines dans un atelier. La particularité de cet atelier est que les travaux, qui y sont réalisés, utilisent les machines dans le même ordre, suivant le principe de la chaîne de production. Dans un flowshop de permutation, on suppose, en plus, qu'on trouve sur chaque machine la même séquence de travaux, ceux-ci ne peuvent pas se doubler. Une solution sera donc une permutation de travaux et le but est de trouver la ou les permutations optimisant un certain nombre de critères. Ici, deux critères seront pris en compte : Date de fin des travaux, appelé aussi *makespan*, et somme des retards, appelé aussi *tardiness*. la valeur de *makspan* correspond a la date de fin du dernier travail sur la dernière machine. Dans le *tardiness*, pour chaque travail un temps, auquel celui-ci devrait finir, est donné. Au-delà de ce temps, le travail est dit en retard. Le *makespan* a été prouvé NP-difficile à partir de trois machines [9]. Le *tardiness* est, quant a lui, NP-difficile même sur une seule machine, sauf sur une machine dans le cas où toutes les dates de fin sont dépassées [4].

4.3 AGMA

L'algorithme AGMA, présenté ci-dessous, combine un algorithme génétique (AG) et un algorithme mimétique (MA). Nous ne donnerons pas ici les détails et paramètres de l'AG. Au besoin, le lecteur pourra les trouver dans [3]. L'AG utilise notamment deux paramètres : une archive Front Pareto $PO*$ et un ratio P_{PO*} de progression du front $PO*$. A chaque génération, ces deux paramètres sont mis à jour. Si une progression non significative est constatée (inférieure à un seuil fixé α), on procède à une intensification de la recherche se traduisant par l'application de MA à la population courante pendant une seule génération. L'application de MA retourne un front $PO*'$ avec lequel l'archive Pareto $PO*$ de l'AG sera mise à jour.

Algorithm 2 L'algorithme génétique hybride AGMA

```
Générer une population initiale
while Critère d'arrêt non vérifié do
  Appliquer l'AG sur une génération
  Mettre à jour PO* et P_PO*
  if P < α then
    Appliquer une génération de MA sur la population courante (Algorithme 3) et récupérer PO*'
    Mettre à jour avec PO*' le front PO* et P_PO*
  end if
end while
```

L'algorithme MA, présenté ci-dessous, consiste à sélectionner des solutions aléatoirement à partir de la population courante de l'AG. Le crossover est ensuite appliqué à ces solutions pour produire de nouvelles

solutions. Parmi ces nouvelles solutions, on retient celles non dominées pour constituer un nouveau front PO'. Ensuite, pour chaque solution de PO' on calcule le voisinage par une méthode de recherche locale (RL). Parmi les solutions constituant ce voisinage, les solutions non dominées sont retenues et insérées dans le front PO'.

Algorithm 3 L'algorithme mimétique MA

```
while Critère d'arrêt non vérifié do
  Appliquer le crossover aux solutions choisies aléatoirement dans la population courante N pour produire un
  ensemble N' de nouvelles solutions.
  Déterminer dans N' l'ensemble PO' de solutions non dominées.
  for chaque solution de PO' do
    Calculer le voisinage V
    PO' = V ⋃ PO'
  end for
end while
```

4.4 AGMA P2P tolérant aux pannes

Dans AGMA P2P, deux types de tuples passifs et un seul type de tuples actifs vont se retouver dans ET. Les tuples passifs sont soit des individus migrants ou des individus Pareto retrouvés par une RL. Les tuples actifs sont ceux qui sont en attente d'un worker oisif, ils correspondent a des RLs. Les tuples ont donc trois formes d'étiquettes possibles "mX/Y", "pX/Y", et "rlX/Y", avec X le numéro d'un AGMA et Y le numéro de la génération actuelle de l'AGMA X. La parallélisation de AGMA a été faite, comme l'illustre la figure 1, en exploitant les deux modèles insulaire et multidépart décrits dans la section 2.2.

Le modèle insulaire est déployé sur des clients XtremWeb où chaque AGMA s'exécute sur un client distinct. La migration des individus d'un AGMA à un autre se fait à l'aide des primitives *outg* et *ing* du modèle P2P-Linda. La figure 2 est un diagramme de séquence montrant un scénario de migration de deux solutions de AGMA 3 vers AGMA 5. Au temps T1, AGMA 3 dépose deux solutions dans ET qui seront récupérées par AGMA 5 au temps T2. Les deux solutions de l'exemple sont étiquetées par "m3/9".

Lors de la phase d'hybridation, i.e. exécution de MA, les recherches locales (RLs), initiées sur une partie des solutions, sont soumises au dispatcher. Ce dernier aura donc la charge de les déployer, selon le modèle multidépart, sur les Workers. La figure 3 montre un exemple de l'exécution de ce modèle. A un temps T1, l'AGMA 3 initie, avec *eval*, deux RLs sur solution02 et solution08. L'exécution de *out*, au temps T2, bloque AGMA 3 tant que les deux RLs ne sont pas terminées. Avant qu'une RL se termine, elle dépose d'abord, dans ET, les solutions trouvées en les étiquetant par "p3/9". La fin de l'exécution des deux RLs transforme le tuple actif, étiqueté "rl3/9", en tuple passif. Ce qui permet à AGMA 3, au temps T3, de le prendre et ainsi d'être débloquée. AGMA 3 récupère, avec outg, les nouvelles solutions trouvées par les deux RLs.

Les deux figures 2 et 3 montrent donc des exemples de variation des solutions selon l'axe des abscisses pour les processus, l'axe des coordonnées pour le temps, et les primitives exécutées.

Dans une application P2P, il est indispensable d'avoir un mécanisme de tolérance aux pannes. Les applications P2Ps, auxquelles on s'intéresse dans l'optimisation combinatoire, nécessitent un calcul intensif de plusieurs jours, voire plusieurs semaines. Le réseau et les machines n'étant pas fiables, l'exécution de ces applications n'est pas à l'abri des pannes matérielles et/ou logicielles. Dans AGMA P2P, chaque client aura la charge de gérer le problème de pannes en utilisant le mécanisme de *checkpointing*, sauvegarde/restauration des données. Un checkpoint peut se faire de deux façons : par le middleware ou par l'application. Le principe du checkpoint fait par le middleware est de sauvegarder, selon une certaine stratégie, tous les segments de données d'un processus ainsi que les valeurs actuelles des registres du processeur. L'inconvénient de cette technique est l'impossibilité de relancer un processus sur un autre OS que celui auquel il a commencé son exécution. Par contre, un checkpoint fait par l'application est indépendant de l'OS, et les informations sauvegardées sont moins importantes, puisque quelques variables seulement seront sauvegardées. Étant donnée l'hétérogénéité des OSs dans une application P2P, nous avons opté pour un checkpoint applicatif en sauvegardant les variables suivantes : l'archive du front Pareto $PO*$, le ratio P_{PO*} de progression du front

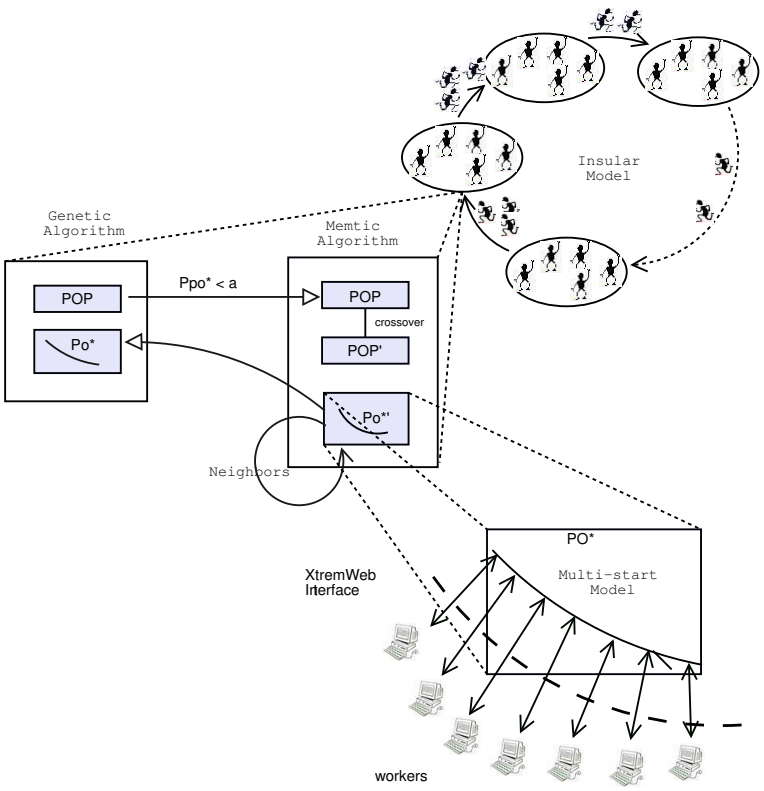

Fig. 1. Les modèles insulaire et multidépart dans AGMA

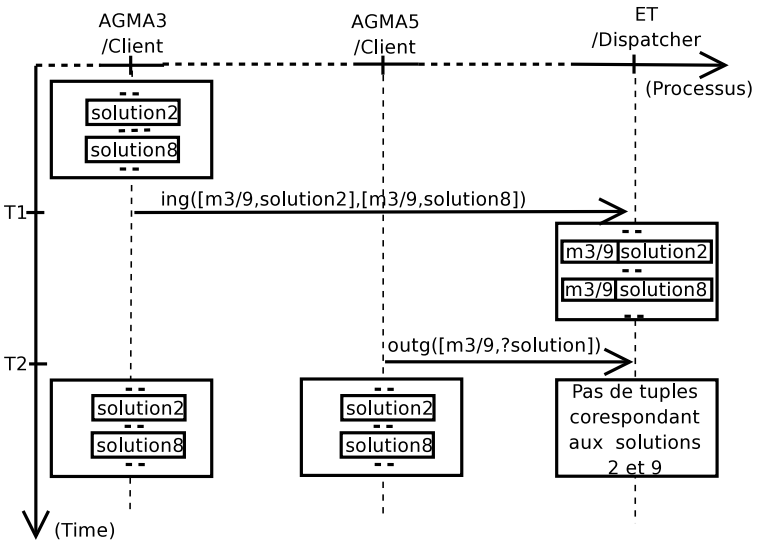

Fig. 2. Modèle insulaire

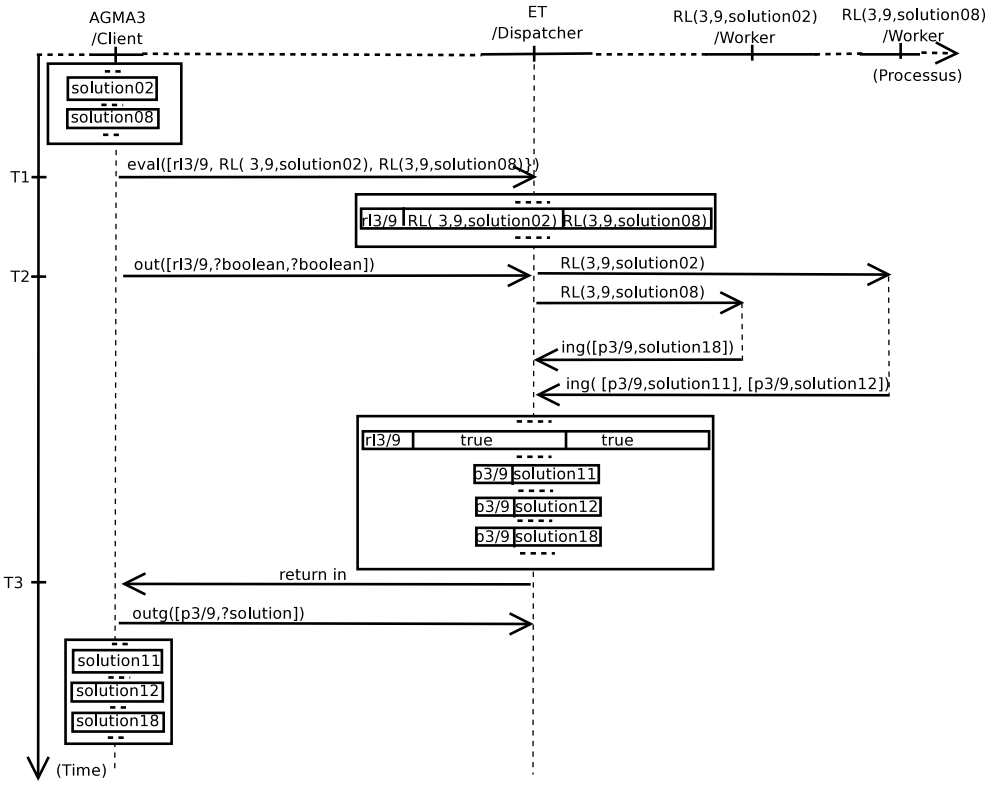

Fig. 3. Modèle multidépart

$PO*$, la population actuelle, et le numéro de la génération courante. Dans notre stratégie de checkpoint, la sauvegarde intervient après chaque exécution de l'algorithme mimétique.

4.5 Expérimentations

Dans nos expérimentations, nous avons considéré une instance de *200 jobs sur 10 machines*. Les paramètres du modèle insulaire sont fixés de la manière suivante : le processus de migration intervient toutes les 10 générations, le nombre de migrants à chaque opération de migration est fixé à 20 si la taille de l'archive Pareto de l'AGMA est supérieure à 20 et la totalité dans le cas contraire. La taille de la population dans chaque AGMA est fixée à 100. La plate-forme matérielle d'expérimentation est le réseau d'enseignement de Polytech'Lille composé de 120 PC sous Linux Debian. Les caractéristiques de ces PC sont données dans le tableau ci-dessous.

Processeur	**Nombre**
AMD Duron(tm) Processor	14
Celero(Coppermine)	14
Intel(R) Celeron(R)CPU 2.00 GHz	8
Intel(R) Celeron(R)CPU 2.20 GHz	28
Intel(R) Celeron(R)CPU 2.40 GHz	21
Intel(R) Celeron(R)CPU 1400 MHz	7
Pentium III(Katmai))	28
Total	120

Dans nos expérimentations, nous avons considéré trois versions de déploiement de AGMA.

1. *Version1* : c'est celle présentée dans [3], elle exploite seulement le modèle multidépart. L'AG est exécuté sur une seule machine et la phase d'hybridation est déployée sur une machine parallèle (IBM-SP2 du CRI, Lille1). Suivant le modèle maître-esclave, les déploiements des RLs sont à l'initiative du maître.
2. *Version2* : idem que version1, sauf que la phase d'hybridation est déployée de manière distribuée sur un ensemble de workers XtremWeb suivant un modèle de vol de cycles et le déploiement est à l'initiative des workers.
3. *Version3* : cette version exploite les deux modèles, i.e insulaire et multidépart, trois AGMA déployés sur des machines clientes coopèrent suivant le modèle insulaire. Chaque AGMA est une implémentation de la version2.

La figure 4 illustre les fronts Pareto obtenus avec les version 1 et 2 après 80 RLs. Les deux fronts sont sensiblement les mêmes. Toutefois, la version 2 présente l'avantage d'être tolérante aux pannes. La version 1, n'étant pas tolérante aux pannes et ne supportant pas le passage à l'échelle, elle s'est limitée à 80 RLs. A l'inverse, avec la version 2 une exécution de plus longue durée est possible. La figure 5 illustre le résultat obtenu après 350 RLs, contrairement à la version1, l'execution s'est arrêtée une fois que la convergence est atteinte. L'exécution a duré environ une semaine et 10 pannes ont été constatées et donc autant d'opérations de restauration ont été effectuées. Le front Pareto obtenu avec la version 2 effectuant 350 RLs est nettement meilleur que celui de 80 RL de la version 1. Notons que ceci n'a été possible que par l'utilisation d'une version passant à l'échelle et tolérante aux pannes. La figure 6 permet de comparer les fronts Pareto obtenus avec les versions 2 et 3 et montre l'apport du modèle insulaire. Sur 80 RLs, on constate déjà que le front obtenu avec la version 3 et meilleur que celui obtenu avec la version 2.

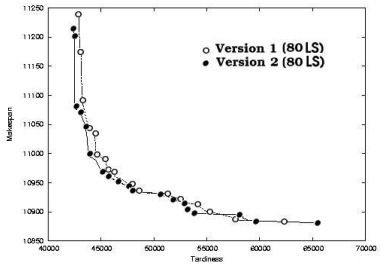

Fig.4. Versions1/2,80 RLs

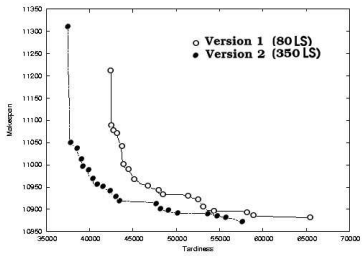

Fig.5. Version1/2,80/350 RLs

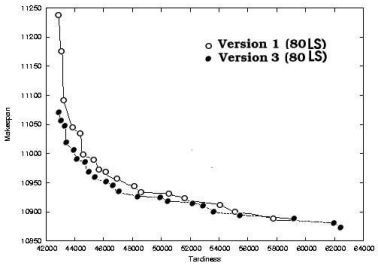

Fig.6. Versions1/3,80 RLs

5 Conclusions et perspectives

L'hybridation de métaheuristiques aux comportements complémentaires permet d'améliorer la qualité et la robustesse des solutions en optimisation combinatoire [12]. Cependant, son exploitation sur des applications de taille industrielle n'est possible que par l'utilisation d'une puissance de calcul importante. Le parallélisme à grande échelle basé sur l'utilisation des systèmes P2P s'avère aujourd'hui un moyen potentiel pour disposer d'une telle puissance. A notre connaissance, aucun travail de recherche n'a été publié sur les métaheursitiques parallèles hybrides à grande échelle. Les intergiciels de calcul P2P sont à ce jour insuffisants pour le déploiement de telles applications. En effet, ils nécessitent une extension logicielle pour la mise en oeuvre de la coopération. Pour répondre à ce besoin, nous avons étendu XtremWeb. L'extension consiste en une implémentation d'un nouveau modèle de coordination :*P2P-Linda*. Celle-ci constitue la première contribution de notre travail. La deuxième contribution de notre travail a consisté à repenser les modèle insulaire et multidépart tout en gérant leur déploiement et leur tolérance aux pannes. Les résultats obtenus après plusieurs jours, sur 120 machines, d'exécution montrent que le passage à l'échelle permet de mieux exploiter l'hybridation, et d'améliorer la qualité des solutions. En perspective, nous envisageons de mener d'autres expérimentations sur un réseau plus large comme la grille expérimentale nationale GRID'5000 (5000 PC). Cela permettra d'avoir une meilleure connaissance sur les limites de l'hybridation en termes de convergence des solutions.

Références

1. http ://www.cs.rice.edu/Conferences/IPTPS02/, 2002.

2. D.P. Anderson, J. Cobb, E. Korpela, , M. Lepofsky, and D. Werthimer. SETI@home : An Experiment in Public-Resource Computing. *Communications of the ACM*, Vol. 45(No. 11) :56–61, Nov. 2002.
3. M. Basseur, F. Seynhaeve, and E-G. Talbi. Adaptive mechanisms for multi-objective evolutionary algorithms. In *Congress on Engineering in System Application CESA'03*, pages 72–86, Lille, France, 2003.
4. T. Belding. The distributed genetic algorithm revisited. In D. Eshelmann editor, editor, *Sixth Int. Conf. on Genetic Algorithms*, San Mateo, CA, 1995. Morgan Kaufmann.
5. J.P. Cohoon, S.U. Hedge, W.N. Martin, and D. Richards. Punctuated equilibria : A parallel genetic algorithm. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, page 148. Lawrence Erlbaum Associates, 1987.
6. D.Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems*, 7 :80–112, 1985.
7. G. Fedak, C. Germain, V. Neri, and F. Cappello. XtremWeb : building an experimental platform for Global Computing. *Workshop on Global Computing on Personal Devices (CCGRID2001), IEEE Press*, May 2001.
8. C.Germain V.Neri G.Fedak and F.Cappello. Xtremweb : a generic global computing platform. *CCGRID'2001 Special Session Global Computing on Personal Devices*, 2000.
9. Du J. and Leung J. Y.-T. Minimizing total tardiness on one machine is np-hard. *Mathematics of operations research*, 15 :483–495, 1990.
10. J.Verbeke and N.Nadgir. Jngi : P2p distributed computing,. *Sun Microsystems, Inc., Palo Alto, California.*, Sep. 2003.
11. Shirky.C. What is p2p... and what isn't ? www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html, 2000.
12. E-G. Talbi. A Taxonomy of Hybrid Metaheuristics. *Journal of Heuristics, Kluwer Academic Publishers*, Vol.8 :541–564, 2002.
13. D.H. Wolpert and W.G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.