École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
5e année
2013 - 2014

Rapport PFE

Le problème de Consolidation de Serveurs (hors-ligne)

Encadrants
Vincent T'Kindt
vincent.tkindt@univ-tours.fr

Étudiants
Lei SHANG
lei.shang@etu.univ-tours.fr

Université François-Rabelais, Tours

DI5 2013 - 2014

Version du 24 février 2014

Table des matières

Table des figures

Liste des tableaux

Liste des algorithmes

Introduction

Ce projet est développé dans le cadre du projet de fin d'études (noté PFE) au département informatique, école d'ingénieurs Polytech'Tours. Le sujet, qui est de nature de recherche scientifique, consiste à résoudre un problème d'ordonnancement NP-complet rencontré par les fournisseurs d'infrastructure comme Amazon Cloud. On cherche à ordonnancer des machines virtuelles (tâches des clients) sur des machines physiques de façon optimale pour minimiser le coût d'utilisation.

Basé sur le travail existant, les travaux à réaliser sont :

- Corriger et améliorer la méthode heuristique existante.
- Implémenter une autre méthode heuristique basée sur le solveur CPLEX.
- Effectuer des recherches sur le Preprocessing du modèle pour améliorer le fonctionnement de la méthode exacte.
- Effectuer des campagnes de tests.
- Analyser et comparer les résultats obtenus.

La première partie de ce rapport portera sur la présentation et la modélisation du problème, les travaux qui ont déjà été effectués seront aussi présentés dans cette partie.

La deuxième partie présentera les travaux réalisés, y compris la réalisation des méthodes de résolution, le travail sur le technique de Preprocessing ainsi que les analyses et les comparaison des résultats de tests.

La troisième partie s'agit de la gestion du projet, y compris la planification et la gestion de versions.

On parlera à la fin sur les difficultés rencontrées et la conclusion.

Contexte du projet

Cette partie présente le contexte du projet, le problème à traiter et les travaux existants au début de ce projet.

2.1 Contexte du problème

Depuis la démocratisation du concept de Cloud, plus en plus d'entreprises et organisations tendent à confier leurs applications aux fournisseurs de plateforme Cloud. En faisant ça, les entreprises n'ont pas besoin d'acheter et de maintenir leurs propres matériels, l'environnement fourni par les fournisseurs est prêt à l'emploi et largement personnalisable.

Sur le côté des fournisseurs (Amazon et Google par exemple), ils ont construit leurs super data centres pour avoir une puissance de calcul considérable et suffisante pour de nombreux clients. Par exemple pour Amazon Cloud Computing, selon une recherche effectuée en 2013 par l'entreprise NetCraft [1], le nombre des serveurs Amazon destiné aux applications web a atteint 158 mille.

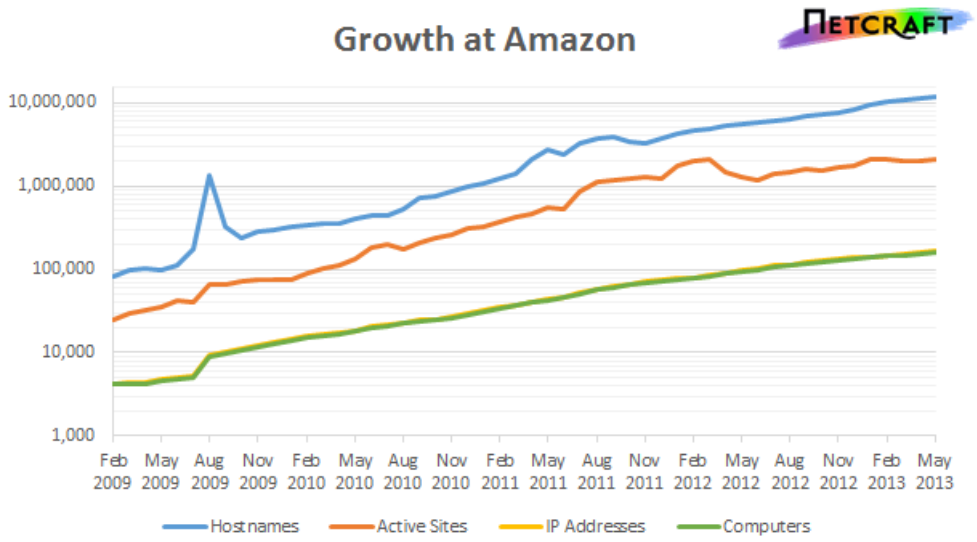

FIGURE 2.1 – Croissance chez Amazon1

Ayant tellement nombreux de serveurs en utilisations, le coût généré devient énorme, seulement le coût de consommation d'électricité est déjà considérable. Les fournisseurs sont donc obligés d'optimiser chaque phase de production pour minimiser le coût.

Avec la technologie de virtualisation, chaque tâche de client est considérée comme une machine virtuelle, plusieurs tâches peuvent s'exécutent en même temps sur une seule machine physique. Alors on a maintenant une question d'ordonnancement : à l'instant donné, quelle tâche doit être exécutée sur quelle machine ? Cette question n'est pas facile à répondre à cause de nombreuses contraintes à respecter, qui seront expliquées dans la section suivante.

1. http://news.netcraft.com/archives/2013/05/20/amazon-web-services-growth-unrelenting.html

2.2 Description du problème

Ayant présenté le contexte du problème à traiter, dans cette partie, on va expliquer un peu en détail le problème en introduisant les contraintes à respecter et les objectifs à atteindre. Ils seront décrits d'une façon moins rigoureuse pour être plus compréhensible. La modélisation mathématique du problème peut être trouvée dans le chapitre suivant.

Les tâches seront placées sur des machines connectées en respectant les contraintes suivantes :

2.2.1 Contraintes de préaffectation

Il existe des contraintes qui influence directement sur le résultat de l'ordonnancement :

1. Une tâche est planifiée d'être exécutée sur certains instants de temps.
2. Une tâche ne peut être exécutée que sur certaines machines physiques.

2.2.2 Contraintes sur l'utilisation de machines

Pour héberger une tâche, la machine doit avoir assez de ressources (CPU, GPU, RAM, HDD).

Une tâche peut être migrée d'une machine à l'autre, pendant la migration cette tâche utilise les ressources de toutes ces 2 machines. Puisque la migration est effectuée en parallèle de l'exécution de la tâche sur la machine de source, alors seulement les tâches qui ont été exécutée pendant assez de temps peuvent être migrées.

Une tâche qui est planifiée d'être exécutée peut éventuellement être suspendue si elle est du type préemptable. Pour réveiller cette tâche, il faut d'abord la recharger, ceci va prend du temps.

2.2.3 Contraintes sur l'utilisation de réseaux

Il peut y avoir des affinités entre les tâches. En ce cas, les tâches concernées ont besoin de communiquer entre elles pendant exécution, alors la bande passante du réseau doit être suffisante pour cette communication.

2.2.4 Objectifs

Il y a 2 objectifs à atteindre : minimisation du coût total et minimisation de la durée de reconfiguration.

Pour le premier, voici les coûts qu'on doit considérer :

- Le coût d'utilisation des ressources (CPU, GPU, RAM, HDD) de machines.
- Le coût de suspension de tâches.
- Le coût de rester allumé pour les machines. On peut le considérer comme le coût de consommation.

2.3 Eléments existants

Dans cette partie on va présenter le travail qui a déjà été réalisé avant ce projet.

2.3.1 Modélisation mathématique

Le modèle mathématique du problème a déjà été créé par Vincent T'Kindt et ses collègues. Toutes les contraintes ainsi que les fonctions objectifs sont déjà représentées sous forme de formulaire. Cette modélisation peut être transmise directement en programme CPLEX, qui peut ensuit trouver la solution optimale.

Le modèle complet peut être trouvé dans le chapitre suivant.

2.3.2 Programme de la méthode exacte

Le programme qui exprime le problème selon le modèle et qui utilise la librairie CPLEX pour trouver la solution optimale a déjà été réalisé par Vincent T'Kindt. Néanmoins, le problème étant prouvé NP-complet, quand on a une grande instance du problème, le programme CPLEX a besoin d'énormément de temps pour trouver une solution. Pour cette raison, on a besoin de chercher une approche heuristique, qui nous permet de trouver très vite une solution assez bonne mais pas forcément optimale.

Même si ce programme CPLEX qui est une approche exacte, n'est peut-être pas très adapté pour les gros instances du problème, il est indispensable dans le test, pour donner la solution optimale qui nous sert ensuite d'évaluer les appoaches heuristiques.

2.3.3 Programme de la méthode heuristique

Une première approche heuristique de liste a été aussi réalisée dans le cadre de PFE (2012 - 2013) de Cyrille PICARD. L'idée de l'approche et les algorithmes concernés ont été decrits dans son rapport de PFE. Par contre, à cause de la complexité du problème et la limite du temps, son implémentation contient des bugs qui rendent le programme incorrect. Alors sur la base de ce programme, j'ai essayé dans un premier temps de corriger les erreurs trouvées et d'améliorer cette implémentation.

2.3.4 Programme testeur

Un testeur a été aussi réalisé par Vincent T'Kindt. Ce testeur contient une partie de génération des instances de problème qui couvre 6 scénarios. Pour chaque instance de problème généré, le testeur appelle le programme de résolution pour trouver la solution et faire des statistiques sur les résultats obtenus.

Modèle mathématique

La modélisation de ce problème a déjà été faite avant ce projet, mais je la remets ici pour assurer la complétude du rapport.

3.1 Notions

Le tableau3.1 est un récapitulatif des notions du problème.

Données générales du problème :	
T,	horizon de planification,
N,	le nombre de tâches,
M,	le nombre de machines physiques,
Pour les tâches :	
n_i^c	la charge requise par la tâche i en termes de CPU,
n_i^g	la charge requise par la tâche i in termes de GPU,
n_i^r,	la quantité de RAM requise par la tâche i,
n_i^h,	la capacité de disque dur requise par la tâche i,
$u_{i,t}$,	un booléen indiquant si la tâche i est susceptible d'être traitée au temps t,
$A = (a_{i,k})_{i,k}$,	matrice des affinités,
$Q = (q_{i,j})_{i,j}$,	matrice des pré-affectation,
$b_{i,j}$,	la bande passante requise par les tâches i et j pour communiquer,
R_i,	un booléen indiquant si la tâche i est préemptable ou non,
ρ_i,	a pénalité unitaire par tâche suspendu i, si préemptable ($R_i = 1$), 0 sinon,
mt_i	le temps de migration de la tâche i, avec $mt_i = \lceil \frac{n_i^h + n_i^r}{b_{i,i}} \rceil$,
$rt_{i,j}$	le temps de reprise i sur la machine j, avec $rt_i = \lceil \frac{n_i^r}{v_j} \rceil$,
Pour les machines physiques :	
m_j^c	la charge maximale acceptée par CPUs de la machine j,
m_j^g	la charge maximale acceptée par GPUs de la machine j,
m_j^r,	la quantité de RAM de la machine j,
m_j^h,	la capacité du disque dur de la machine j,
m_j^b,	le maximum de bande passante qui peut être utilisé par n'importe quelle tâche sur cette machine,
α_j^c,	le coût d'utilisation du CPU de la machine j,
α_j^g,	le coût d'utilisation du GPU de la machine j,
α_j^r,	le coût d'utilisation d'un Mb de RAM de la machine j,
α_j^h,	le coût d'utilisation d'un Mb de capacité disque de la machine j,
β_t,	le coût d'une machine en marche à l'instant t,
v_j,	la rapidité de la machine j pour charger le contexte d'exploitation,
Pour le réseau :	
$G = (V, E)$,	le graphe modélisant le réseau,
b,	la bande passante maximale associée pour toutes les arrêtes $e_{\ell,\ell'} \in E$,
$\mathcal{P}$,	un ensemble de chemins entre deux machines, un chemin est un ensemble d'arrêtes,
$P_{\ell,\ell'}$,	l'ensemble des couples de machine (j, j') qui utilise l'arrête $e_{\ell,\ell'}$,

FIGURE 3.1 – Tableau des variables

3.2 Objectifs

La variable de décision du problème est la suivante qui représente l'ordonnancement en sortie.

$$x_{i,t}^{j} = \begin{cases} 1 & \text{si la tâche } i \text{ est traitée sur la machine } j \text{ dans l'intervalle } [t, t+1] \\ 0 & \text{sinon} \end{cases}$$

On définit aussi un ensemble de variables supplémentaires suivantes :

- $y_{i,i\prime,t}^{j,j\prime} = 1$ si les deux tâches i et $i\prime$ sont exécutées respectivement sur les machines j et $j\prime$ à l'instant t et ont besoin de communiquer sur le réseau. Au cas de la migration de tâche i de la machine j vers la machine $j\prime$, $y_{i,i,t}^{j,j\prime} = 1$. Dans les autres cas, on a $y_{i,i\prime,t}^{j,j\prime} = 0$.
- $z_{t,j} = 1$ si à l'instant t la machine est en état allumé, c'est-à-dire elle exécute au moins une tâche en ce moment.
- z_t est le nombre de machines en marche à l'instant t.
- $d_{i,t}$ est la durée des opérations de reconfiguration de la tâche i à l'instant t. Ca peut être la durée d'une opération de $resume$ commançant à t ou bien la durée d'une opération de migration terminant à t.

Maitenant avec les variables et les notions déclarées avant, voici les fonctions objectifs à minimiser.

$$\begin{aligned} TC &= \sum_{t=1}^{T}\sum_{i=1}^{N}\sum_{j=1}^{M} x_{i,t}^{j}(\alpha_j^c n_i^c + \alpha_j^g n_i^g + \alpha_j^h n_i^h + \alpha_j^r n_i^r) \\ &+ \sum_{t=1}^{T}\sum_{i=1}^{N}\sum_{j,k=1;k\neq j}^{M} y_{i,i,t}^{j,k}(\alpha_k^h n_i^h + \alpha_k^r n_i^r) + \sum_{t=1}^{T}\sum_{i=1}^{N}(1 - \sum_{j=1}^{M} x_{i,t}^{j})\rho_i u_{i,t} + \sum_{t=1}^{T}\beta_t z_t \\ RE &= \sum_{i=1}^{N}\sum_{t=1}^{T} d_{i,t} \end{aligned}$$

3.3 Contraintes

$$\sum_{i=1}^{N} n_i^c x_{i,t}^j \leq m_j^c \qquad \forall t = 1, \ldots, T; \quad \forall j = 1, \ldots, M \qquad \text{(A)}$$

$$\sum_{i=1}^{N} n_i^g x_{i,t}^j \leq m_j^g \qquad \forall t = 1, \ldots, T; \quad \forall j = 1, \ldots, M \qquad \text{(B)}$$

$$\sum_{i=1}^{N} n_i^h x_{i,t}^j + \sum_{j,k=1;k\neq j}^{M} n_i^h y_{i,i,t}^{j,k} \leq m_j^h \qquad \forall t = 1, \ldots, T; \quad \forall j = 1, \ldots, M \qquad \text{(C)}$$

$$\sum_{i=1}^{N} n_i^r x_{i,t}^j + \sum_{j,k=1;k\neq j}^{M} n_i^r y_{i,i,t}^{j,k} \leq m_j^r \qquad \forall t = 1, \ldots, T; \quad \forall j = 1, \ldots, M \qquad \text{(D)}$$

$$y_{i,i\prime,t}^{j,j\prime} \geq x_{i,t}^j + x_{i\prime,t}^{j\prime} - 1 \qquad \forall t = 1, \ldots, T; \forall i = 1, \ldots, N; \quad \forall i\prime = i+1, \ldots, N donta_{i\prime} = 1; \quad \forall j = 1, \ldots, M; \forall j\prime = j+1, \ldots, M \qquad \text{(E)}$$

$$y_{i,i,t}^{j,j\prime} \geq (x_{i,t_1}^j - \sum_{k=1}^{M} \sum_{t\prime=t_1+1}^{t_2-1} x_{i,t\prime}^k + x_{i,t2}^{j\prime} - 1) \qquad \forall t = 1, \ldots, T; \forall i = 1, \ldots, N; \quad \forall j, j\prime = 1, \ldots, M etj \neq j\prime; \quad \forall t_1 = t, \ldots, Min(t + mt_i, T) \quad \forall t_2 = t_1 + 1, \ldots, T \qquad \text{(F)}$$

$$mt_i(x_{i,t}^j - \sum_{k=1}^{M} \sum_{t\prime=t+1}^{t_1-1} x_{i,t\prime}^k + x_{i,t_1}^{j\prime} - 1) \leq \sum_{t\prime=max(t+1-mt_i,0)}^{t} x_{i,t}^j \qquad \forall t = 1, \ldots, T; \forall i = 1, \ldots, N; \quad \forall j, j\prime = 1, \ldots, M etj \neq j\prime; \quad \forall t_1 = t+1, \ldots, T \qquad \text{(G)}$$

$$\sum_{j=1}^{M} x_{i,t}^j = u_{i,t} \qquad \forall i = 1, \ldots, N, dontR_i = 0; \quad \forall t = 1, \ldots, T \qquad \text{(H)}$$

$$\sum_{j=1}^{M} x_{i,t}^j \leq u_{i,t} \qquad \forall i = 1, \ldots, N, dontR_i = 1; \quad \forall t = 1, \ldots, T \qquad \text{(I)}$$

$$x_{i,t}^j \leq u_{i,t} q_{i,j} \qquad \forall t = 1, \ldots, T; \forall i = 1, \ldots, N; \quad \forall j = 1, \ldots, M \qquad \text{(J)}$$

$$\sum_{(j,j\prime)\in P_{l,l\prime}, j<j\prime} \; (\sum_{i,i\prime=1, i<i\prime, a_{i,i\prime}=1}^{N} b_{i,i\prime} y_{i,i\prime,t}^{j,j\prime} + \sum_{i}^{N} b_{i,i} (y_{i,i,t}^{j,j\prime} + y_{i,i,t}^{j\prime,j})) \leq b \qquad \forall t = 1, \ldots, T; \forall e_{l,l\prime} \in E \qquad \text{(K)}$$

$$z_{t,j} \geq x_{i,t}^{j} \qquad \forall t = 1, \ldots, T; \forall i = 1, \ldots, N; \forall j = 1, \ldots, M \quad \text{(L)}$$

$$z_t = \sum_{j=1}^{M} z_{t,j} \qquad \forall t = 1, \ldots, T \quad \text{(M)}$$

$$x_{i,t_2}^{j} \leq 1 - x_{i,t_1}^{j} + x_{i,t_1+1}^{j} \qquad \forall i = 1, \ldots, N, dontR_i = 1; \forall j = 1, \ldots, M; \forall t_1 = 1, \ldots, T - rt_{i,j}; \forall t_2 = t_1 + 1, \ldots, t_1 + rt_{i,j} \quad \text{(N)}$$

$$d_{i,t_1} \geq (t_2 - t_1)(x_{i,t_1-1}^{j} - \sum_{k=1}^{M} \sum_{t=t_1}^{t_2-1} x_{i,t}^{k} - 1 + x_{i,t_2}^{j}) \qquad \forall i = 1, \ldots, N, dontR_i = 1; \forall j = 1, \ldots, M; \forall t_1 = 1, \ldots, (T - rt_{i,j} - 1); \forall t_2 = t_1 + rt_{i,j} + 1, \ldots, T \quad \text{(O)}$$

$$d_{i,t_1} \geq mt_i(x_{i,t_1}^{j} + x_{i,t_1+1}^{j'} - 1) \qquad \forall i = 1, \ldots, N; \forall j, k = 1, \ldots, M, j \neq k; \forall t_1 = 1, \ldots, (T - mt_i - 1); \forall t_2 = t_1 + mt_i + 1, \ldots, T \quad \text{(P)}$$

Travaux réalisés

Cette partie présente les travaux réalisé dans ce projet. Puisqu'il y a une grosse partie de travail qui concerne les nombreux tests et les analyses donc un autre chapitre indépendent est créé pour la présenter.

4.1 Finition de la méthode heuristique de liste (H1)

Les algorithmes et le programme de H1 ont été principalement réalisés par Cyrille PICARD avant ce projet. Cependant, ce travail n'a pas été vraiment fini et il existe des problèmes dans les algorithmes ainsi que dans l'implémentation.

Les travaux réalisés concernent donc la correction des algorithmes ainsi que ses implémentations. Cette partie a duré plus longtemps que ce que nous avions prévu. Finalement la partie principale du programme est quasiment réécrit. La plus part de travail se porte sur la programmation C++ qui n'est pas très pertinent d'être présentée dans ce rapport. Par contre, nous listons ici les algorithmes (pseudo-code) principals que nous avons créés ou modifiés.

4.1.1 Algorithmes

Nous listons dans cette section les algorithmes critiques de la méthode H1. Nous avons fait des efforts pour rendre la présentation assez synthétique et compréhensible.

Avant d'entrer dans les algorithmes, nous déclarons ici des structrues de données globales qui peuvent être utilisées sans déclaration dans n'importe quel algorithme.

1. ListeGPU1 : la liste des tâches non-préemptable ayant des besoins en GPU et CPU et tel que besoins HDD $>$ besoins RAM.
2. ListeGPU2 : la liste des tâches non-préemptable ayant des besoins en GPU et CPU et tel que besoins HDD $<$ besoins RAM.
3. ListeCPU1 : la liste des tâches non-préemptable ayant des besoins uniquement en CPU et tel que besoins HDD $>$ besoins RAM.
4. ListeCPU2 : la liste des tâches non-préemptable ayant des besoins uniquement en CPU et tel que besoins HDD $<$ besoins RAM.
5. ListeTachesPr : la liste de toutes les tâches préemptable.
6. ListePrGPU1 : la liste des tâches préemptable ayant des besoins en GPU et CPU et tel que besoins HDD $>$ besoins RAM.
7. ListePrGPU2 : la liste des tâches préemptable ayant des besoins en GPU et CPU et tel que besoins HDD $<$ besoins RAM.
8. ListePrCPU1 : la liste des tâches préemptable ayant des besoins uniquement en CPU et tel que besoins HDD $>$ besoins RAM.
9. ListePrCPU2 : la liste des tâches préemptable ayant des besoins uniquement en CPU et tel que besoins HDD $<$ besoins RAM.
10. ListeOrdo : la liste de l'ordonnancement en sortie.

Le premier algorithme1 est l'algorithme général qui est le plus synthétique. L'idée de l'algorithme2 est de déterminer les intervalles *stable*, c'est-à-dire toutes les tâches du même intervalle ontle même Ui (prévue à être exécutée). L'algorithme CréerListeServeurTriée3 trie la liste des serveurs selon le coût normalisé croissant. L'algorithme TrierTâches5 trie la liste de tâches selon leur priorités pour un intervalle et une machines donnée. L'algorithme Ordonnancer6 effectue l'ordonnancement. L'algorithme AllumageMachine8 détermine est-ce qu'on peut rallumer une machine pour mettre la tâche, si oui c'est la quelle.

Algorithme 1 Algorithme Général

Entrée

ListOrdo : le tableau qui contient la matrice d'affectation des VMs sur les machines phisiques.
IsFeasible : booléan qui indique si on a bien ordonnancé toutes les tâches non-préemptables.
coûtTotal : le coûtTotal selon l'ordonnancement effectué.

Sortie

ListOrdo, IsFeasible, coûtTotal

```
Début
    IsFeasible = true ;
    CalculerIntervalle() ;
    CréerListeServeurTriée() ;
    pour chaque intervalle i faire
        CréerListesTâche(i) ;
        Ordonnancer(ListeOrdo, i) ;
    fin pour
    /* Maintenant on a fini l'ordonnancement des tâches et on peut déjà calculer le coût total. */

    coûtTotal = CalculerCoûtTotal() ;
    /* Voir si la solution est feasable */
    pour chaque intervalle i faire
        pour chaque tâche t faire
            si (t n'est pas préemptable ET ListeOrdo[i][t] == -1) alors
                IsFeasible = false ;
                retourner ;
            fin si
        fin pour
    fin pour
Fin
```

Algorithme 2 CaculerIntervalle

Sortie

```
1:  ListeIntervalle : le tableau des intervalles trouvés
2:  Début
3:      Entier nbIntervalle=0 ;
4:      ListeIntervalle[0].BorneInf = 0 ;
5:      ListeIntervalle[0].BorneSup = 0 ;
6:      pour t allant de 0 à T-2 faire
7:          pour iTache allant de 0 à N-1 faire
8:              si u(iTache,t) !=u(iTache,t+1) alors
9:                  nbIntervalle = nbIntervalle+1 ;
10:                 ListeIntervalle[nbIntervalle-1].BorneSup = t ;
11:                 ListeIntervalle[nbIntervalle].BorneInf = t+1 ;
12:                 break ;
13:             fin si
14:         fin pour
15:     fin pour
16:     nbIntervalle=nbIntervalle+1 ;
17:     ListeIntervalle[nbIntervalle-1].BorneSup = T-1 ;
18: Fin
```

Algorithme 3 CréerListeServeurTriée

Sortie

```
1:  ListeServeur : le tableau des serveurs triés par ordre croissant en fonction du coût normalisé
2:  Début
3:      Réel CoutTotal, SommeCaract ;
4:      pour i allant de 0 à M-1 faire
5:          ListeServeur[i].IndiceServeur=i ;
6:          CoutTotal=mc(i)*alphac(i) +mg(i)*alphag(i) + mr(i)*alphar(i)+ mh(i)*alphah(i) ;
7:          SommeCaract = mc(i)+mg(i)+mr(i)+mh(i) ;
8:          ListeServeur[i].CoutNorm = CoutTotal / SommeCaract ;
9:      fin pour
10:     TrierParCoutNorm(ListeServeur) ;
11: Fin
```

Algorithme 4 CréerListesTâches

Entrée

indice : l'indice de l'intervalle sur lequel on travaille

Sortie

ListeTachesPr : la liste des tâches préemptable.

/∗ Les tâches sont réparties dans ces 8 listes selon leurs besoins en GPU, CPU, HDD, RAM et si la tâche est préemptable. ∗/

ListeGPU1, ListeCPU1, ListePrGPU1, ListePrCPU1 //Tâches non-préenptables

ListeGPU2, ListeCPU2, ListePrGPU2, ListePrCPU2 //Tâches préenptables

Début

pour tâche de 0 à N()-1 **faire**
si (u(tâche, temps)=1) **alors**
/∗ Cette tâche peut être exécutée en ce moment ∗/
si (R(tâche==0)) **alors**
si (ng(tâche)>0) **alors**
si (nh(tâche) > nr(tâche)) **alors**
Répartir cette tâche à ListeGPU1 ;
sinon
Répartir cette tâche à ListeGPU2 ;
fin si
sinon
/∗ ng(tâche)=0 ∗/
si (nh(tâche) > nr(tâche)) **alors**
Répartir cette tâche à ListeCPU1 ;
sinon
Répartir cette tâche à ListeCPU2 ;
fin si
fin si
sinon
/∗ Cette tâche est préemptable ∗/
Répartir cette tâche à ListeTâchesPr ;
si (ng(tâche)>0) **alors**
si (nh(tâche) > nr(tâche)) **alors**
Répartir cette tâche à ListePrGPU1 ;
sinon
Répartir cette tâche à ListePrGPU2 ;
fin si
sinon
/∗ ng(tâche)=0 ∗/
si (nh(tâche) > nr(tâche)) **alors**
Répartir cette tâche à ListePrCPU1 ;
sinon
Répartir cette tâche à ListePrCPU2 ;
fin si
fin si
fin si
fin si
fin pour

Fin

Algorithme 5 TrierTâches

```
Entrée
indice : indice de l'intervalle
indiceServeur : indice de la machine
list : une liste de tâches à trier
taille : la taille de la liste
Sortie
list : liste de tâches triée par prio décroissant
Début
    Entier IB ;
    Entier WG ;
    Entier MachineRecevoir ; /* La durée d'exécution actuelle de la tâche */
    Entier duree ;
    pour iboucle de 0 à taille faire
        list[iboucle].prio = 0 ;
        IB = 0 ;
        WG = 0 ;
        MachineRecevoir = 0 ;
        duree = GetDureeExeActuelle(indice, list[iboucle].IndiceVM) ;
        si (la tâche n'est pas encore affectée) alors
            si (cette tâche était affectée sur cette machine à l'intervalle précédent) alors
                /* La tâche a plus de priorité */
                IB = M() ;
            fin si
            /* Si migration pas possible */
            si (duree!=0 et duree<mt(list[iboucle].IndiceVM)) alors
                Chercher indiceInterval qui est le dernier intervalle où la tâche était
                exécutée.
                si (indiceInterval est trouvé et la machine correspondante est la même)
                alors
                    WG = indiceServeur ;
                sinon
                    /* ça veut dire pas possible */
                    WG = -M() ;
                fin si
            sinon
                pour iboucle2 de 0 à <M() faire
                    si (ListeMachine[iboucle2] peut recevoir la tâche) alors
                        MachineRecevoir++ ;
                    fin si
                fin pour
                WG = M() - MachineRecevoir ;
            fin si
            list[iboucle].prio = IB + WG ;
        fin si
        /* Trier les tâches en priorité décroissante */
        SortListByPrio(list) ;
    fin pour
Fin
```

Algorithme 6 Ordonnancer

Entrée

indiceInter : L'intervalle à traiter.

Sortie

ListeOrdo : Une variable globale qui représente le résultat de l'ordonnancement.

Début

```
    indiceAllume : L'indice de la machine qui sera allumée
    /* Affecter d'abord les tâches non préemptables */
    pour indiceServeur de 0 à NbServeur faire
        si (toute les tâches non-préemptables sont affectées) alors
            break ;
        fin si
        /* Ordonnancer les tâches non-préemptables sur le serveur indiceServeur */
        TrierTâches(ListeGPU1, indiceServeur) ;
        TrierTâches(ListeGPU2, indiceServeur) ;
        TrierTâches(ListeCPU1, indiceServeur) ;
        TrierTâches(ListeCPU2, indiceServeur) ;
        OrdoListeTache(ListeGPU1, indiceInter, indiceServeur, true) ;
        OrdoListeTache(ListeGPU2, indiceInter, indiceServeur, true) ;
        OrdoListeTache(ListeCPU1, indiceInter, indiceServeur, true) ;
        OrdoListeTache(ListeCPU2, indiceInter, indiceServeur, true) ;

        /* Ordonnancer les tâches préemptables sur le serveur indiceServeur */
        TrierTâches(ListePrGPU1) ;
        TrierTâches(ListePrGPU2) ;
        TrierTâches(ListePrCPU1) ;
        TrierTâches(ListePrCPU2) ;
        OrdoListeTache(ListePrGPU1, indiceInter, indiceServeur, false) ;
        OrdoListeTache(ListePrGPU2, indiceInter, indiceServeur, false) ;
        OrdoListeTache(ListePrCPU1, indiceInter, indiceServeur, false) ;
        OrdoListeTache(ListePrCPU2, indiceInter, indiceServeur, false) ;
    fin pour

    /* Pour le reste des tâches prémptable, on regarde si on a besoin de rallumer des machines */

    tant-que (Il y a encore des tâches préemptables non affectées) faire
        indiceAllume = AllumageMachine(indiceInter) ;
        si (indiceAllume==-1) alors
            /* Aucune machine ne peut être allumée, arrêter. */
            retourner ;
        sinon
            TrierTâches(ListePrGPU1) ;
            TrierTâches(ListePrGPU2) ;
            TrierTâches(ListePrCPU1) ;
            TrierTâches(ListePrCPU2) ;
            OrdoListeTache(ListePrGPU1, indiceInter, indiceServeur, true) ;
            OrdoListeTache(ListePrGPU2, indiceInter, indiceServeur, true) ;
            OrdoListeTache(ListePrCPU1, indiceInter, indiceServeur, true) ;
            OrdoListeTache(ListePrCPU2, indiceInter, indiceServeur, true) ;
        fin si
    fin tant-que
Fin
```

Algorithme 7 OrdoListeTache

Entrée

ListeTâche : La liste des tâches.
indiceI : L'indice d'intervalle.
indiceS : L'indice du serveur.
canTurnOn : un booléan pour indique quand le serveur n'est pas allumé est-ce qu'on a le droit de l'allumer et affecter au-dessus.

Sortie

ListeOrdo : L'affectation des tâches est enregistrée dans cette liste qui est aussi la sortie du programme.

```
Début
    Entier interInf, interSup, indiceVM, indiceVM2 ;
    Entier dureeExeActuelle ;
    interInf = ListeIntervalle[indiceI].BorneInf ;
    interSup = ListeIntervalle[indiceI].BorneSup ;

    pour chaque tâche t dans ListTâche faire
        si (t n'est pas encore affectée sur cet intervalle) alors
            si (Le serveur indiceS possède assez de ressource pour t) alors
                pour chaque tâche t2 qui a une affinité avec t et qui a été affectée sur un autre serveur indiceS2 faire
                    si ( false == CalculFesabiliteReseau(t1.indiceVM, t2.indiceVM, indiceS, indiceS2, indiceI) ) alors
                        Cette tâche t ne peut pas affectée sur ce serveur, car le réseau ne le permet pas.
                        Continuer voir la tâche suivante dans ListTâche.
                    fin si
                fin pour

                /* Maintenant on est sûr que la tâche peut être affectée, alors on l'affecte... */
                MaJReseau(t1.indiceVM, t2.indiceVM, indiceS, indiceS2, indiceI) ;
                pour chaque instant de temps i faire
                    Mettre à jour ListeOrdo[i][t.indiceVM] ;
                    Mettre à jour les caractéristiques de ce serveur(indiceS) ;
                fin pour
            fin si
        fin si
    fin pour
Fin
```

Algorithme 8 AllumageMachine

```
Entrée
indiceInter : L'intervalle à traiter.
Sortie
indiceAllume : indice de la machine à allumer. -1 s'il y en a pas.
Début
        inf : instant inférieur de l'intervalle
        sup : instant supérieur de l'intervalle
        critère : la différence entre le coût de rallumer la machine et le coût de ne pas la rallumer.
        Plus c'est moins, mieux c'est.
        critèreMin : le minimum du critère.

        inf = ListeIntervalles[indiceInter].BorneInf ;
        sup = ListeIntervalles[indiceInter].BorneSup ;
        critèreMin = 0 ;
        pour chaque machine m faire
                si (m n'est pas allumée) alors
                        critère = (sup - inf + 1) * beta(m) ; //Le coût d'être ON
                        pour (chaque tâche t préemptable qui n'est pas encore affectée) faire
                                si (m peut recevoir t) alors
                                        critère = critère + (sup - inf + 1 ) * CalculCoutAffectation(t,m)
                                        - rho(t) ;
                                fin si
                        fin pour
                        si (critère < critèreMin) alors
                                critèreMin = critère ;
                                indiceAllume = m.indiceServeur ;
                        fin si
                fin si
        fin pour
        si (critèreMin < 0) alors
                retourner indiceAllume ;
        sinon
                retourner -1 ;
        fin si
Fin
```

4.2 Réalisation de la méthode heuristique basée sur Cplex (H2)

La deuxième méthode heuristique est basée sur la méthode exacte du solveur Cplex. Il s'agit de chercher de bonnes valeurs de certains paramètres pour rendre la méthode en heuristique, qui peut nous donner une solution faisable (donc pas forcément optimale) dans un temps limité.

Les deux paramètres Cplex qui nous intéressent sont :

- TimeLimit : durée maximum de résolution. Cplex s'arrête au bout de ce temps même s'il n'a pas encore trouver la solution optimale.
- EpGap : tolérance d'écart entre la solution trouvée et la borne inférieure. Si Cplex trouve une solution faisable qui n'est pas optimal mais qui est tolérée par cet écart alors il va aussi s'arrêter en retourner cette solution convenable.

4.2.1 Détermination des paramètres

Pour trouver de bonnes valeurs pour ces deux paramètres, nous avons réalisé une analyse sur les fichiers log Cplex engendrés pendant la résolution de certaines instances. En fait quand Cplex résoud un problème, il enregistre fréquemment les informations sur la déviation entre la solution actuelle et la LB, avec aussi le temps écoulé. A partir de ces informations, nous pouvons chercher la corrélation entre ces deux facteurs. La figure **??** a été ensuite réalisée pour illustrer cette relation.

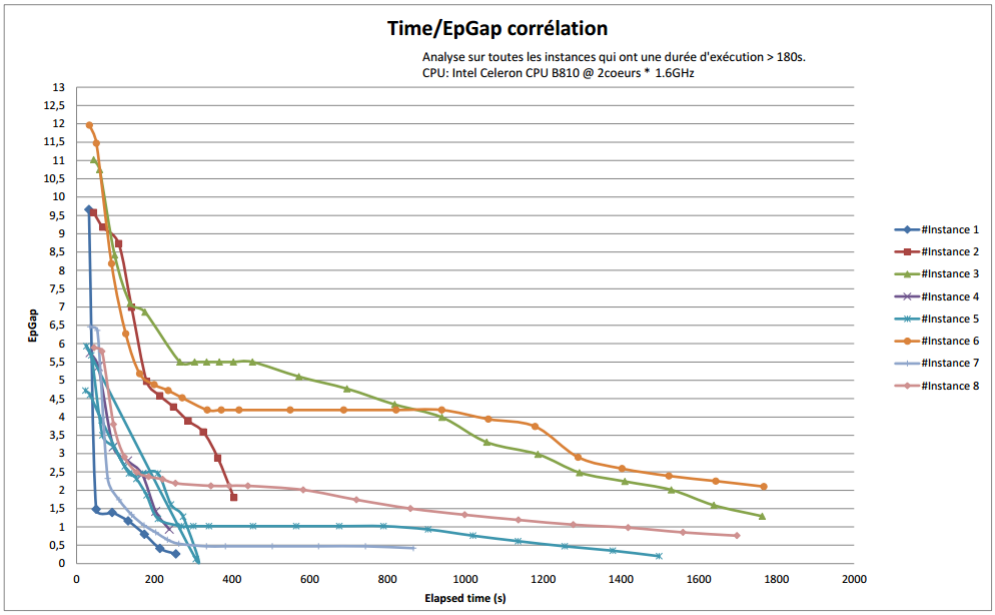

FIGURE 4.1 – L'évolution de la déviation selon le temps écoulé

En analysant cette figure, nous avons choisi heuristiquement 2% comme la valeur pour le paramètre *EpGap* et 400s pour *TimeLimit*. D'abord une déviation de 2% reste très faible donc admissible, ensuite nous pouvons constater aussi après 400s la solution ne peut plus être améliorée facilement.

4.3 Mise en oeuvre du Preprocessing

4.3.1 Principe

Le *Preprocessing* est un prétraitement réalisé sur le modèle mathématique du problème avant de passer ce modèle au solveur. L'idée du Preprocessing est de fixer autant que possible de variables dans le modèle pour réduire la taille du problème et donc accélérer la résolution qui suit.

Pour le faire, il faut avoir une borne supérieure (UB) en entrée. Ça peut être fournit par les deux méthodes heuristiques que nous avons parlées. Ensuite il faut faire une relaxation continue sur le modèle pour obtenir un modèle LP dont les variables sont de type réel entre 0 et 1.

Nous pouvons alors lancer la résolution LP sur ce modèle. La valeur objective obtenue est donc une borne inférieure (LB) du problème initial. Si toutes les variables de la solution valent 0 ou 1, alors le travail est fini ; sinon nous commenceons à fixer les variables qui ne sont pas entier de façons suivante :

1. Si le fait de fixer une variable à 0 va nous amener une solution qui viole la UB, alors il faut fixer cette variable à 1.
2. Équalement, si le fait de fixer une variable à 1 va nous amener une solution qui viole la UB, alors il faut fixer cette variable à 0.

3. S'il n'y a aucune variable qui peut être fixée, alors le Preprocessing est fini. Sinon il faut relancer la résolution LP et répéter le Preprocessing.

Pour expliquer comment peut-on savoir un fixage à 0 ou à 1 peut violer la UB, il faut d'abord introduire les notions *Reduced-cost* et *Pseudo-cost*.

En effet après la résolution LP d'un problème, Cplex va attacher un *Reduced-cost* à chaque variable qui est hors la *Base*. Le *Reduced-cost* signifie tout simplement l'influence d'un ajustement de la variable sur la valeur de la fonction objectif. Par exemple si nous voulons fixer à 1 une variable qui vaut 0.8, alors nous savons que la nouvelle valeur objective sera $sol + (1 - 0.8) * \textit{Reduced-cost}$ dont sol est la valeur objective avant le fixage.

Le *Pseudo-cost* a un sens similaire, mais c'est pour les variables qui sont dans la *Base*.

Nous avons dejà une librairie C++ *PreLib* qui est une implémentation de la technique du Preprocessing issue d'un autre projet. Pour l'employer dans ce projet, nous avons besoin de faire des adaptations, sinon la parite essentielle du Preprocessing est déjà prête à utiliser.

4.3.2 Valide inequalities

Selon l'analyse du premier test 5.4 du Preprocessing, nous avons constaté que les LBs pendant le Preprocessing ne sont pas assez proches de la solution optimale, ce qui gêne la performence du Preprocessing. Pour améliorer cette situation, nous décitons d'ajouter des *Valide Inequalities* dans le modèle LP du problème.

Les *Valide Inequalities* (ou Cut, Coupe) sont des contraintes supplémentaires qui sont redondantes, mais qui peuvent peut-être accélérer la résolution. Si ces coupes sont bien conçues, le solveur peut alors en profiter dans la procédure *Branch & Cut* pour éliminer très vite les branches inintéressantes ; mais l'ajout de ces coupes peut aussi amener un modèle grossi, ce qui peut au contraire ralentir la résolution. Donc le choix sur les coupes doit se faire avec modération. La conception, l'implémentation et le choix des coupes font une grande partie dans le travail de ce projet.

Nous distinguons deux types de coupe : la coupe dépendante du problème et la coupe classique qui ne dépend pas le problème. La premère coupe dépend la particularité du problème, elle est souvent sur l'utilisation des ressources. La coupe classique est inventée à partir des contraites existantes d'une façons mathématique. Dans les deux sections qui suivent, nous présentons une coupe dépendante du problème et une coupe classique que nous avons créé.

Cuts sur les contraintes des ressources

Nous avons créé une coupe pour chaque utilisation des ressources CPU/GPU/HDD/RAM. L'idée est de dire : si la tâche i ne peut pas être affectée au serveur j à cause de la capacité résiduelle de CPU/GPU du serveur, alors pour toute les tâches qui demandent plus de CPU/GPU que la tâche i, cette affectation ne peut pas être effectuée non plus.

Cette contrainte peut être exprimée de façon suivante, nous respectons la même règle de notations que dans le modèle mathématique :

$Si\ n^c_{i\prime} \geq n^c_i\ alors$

$$x_{i,j,t} + x_{i\prime,j,t} \leq (m^c_j - \sum_{k=1;k\neq i,i\prime;u_{k,t}=q_{k,j}=1}^{N} n^c_k x_{k,j,t})/n^c_i \qquad \forall t = 1,\ldots,T, tq\ u_{i,t} = u_{i\prime,t} = 1;$$

$$\forall j = 1,\ldots,M, tq\ q_{i,j} = q_{i\prime,j} = 1 \qquad \text{(A)}$$

Si $n^g_{i\prime} \geq n^g_i$ *alors*

$$x_{i,j,t} + x_{i\prime,j,t} \leq (m^g_j - \sum_{k=1;k\neq i,i\prime;u_{k,t}=q_{k,j}=1}^{N} n^g_k x_{k,j,t})/n^g_i \qquad \forall t = 1,\ldots,T, tq\ u_{i,t} = u_{i\prime,t} = 1;$$

$$\forall j = 1,\ldots,M, tq\ q_{i,j} = q_{i\prime,j} = 1 \qquad \text{(B)}$$

À noter que nous n'avons pas besoin de considérer ici la contraite de préaffectation.

Les cuts sur les ressources HDD/RAM ont le même principe sauf que ces ressources puissent aussi être occupées par l'opération de la migration. Nous pouvons appliquer les mêmes cuts comme pour CPU/GPU mais la prise en compte de la migration peut rendre le cut plus strict.

Si $n^h_{i\prime} \geq n^h_i$ *alors*

$$x_{i,j,t} + x_{i\prime,j,t} \leq (m^h_j - \sum_{k=1;k\neq i,i\prime;u_{k,t}=q_{k,j}=1}^{N} n^h_k x_{k,j,t} - \sum_{k=1;k\neq i,i\prime}^{N}\sum_{l=1;l\neq j}^{M} n^h_k y^{l,j}_{k,k,t})/n^h_i \qquad \forall t = 1,\ldots,T, tq\ u_{i,t} = u_{i\prime,t} = 1;$$

$$\forall j = 1,\ldots,M, tq\ q_{i,j} = q_{i\prime,j} = 1 \qquad \text{(C)}$$

Si $n^r_{i\prime} \geq n^r_i$ *alors*

$$x_{i,j,t} + x_{i\prime,j,t} \leq (m^r_j - \sum_{k=1;k\neq i,i\prime;u_{k,t}=q_{k,j}=1}^{N} n^r_k x_{k,j,t} - \sum_{k=1;k\neq i,i\prime}^{N}\sum_{l=1;l\neq j}^{M} n^r_k y^{l,j}_{k,k,t})/n^r_i \qquad \forall t = 1,\ldots,T, tq\ u_{i,t} = u_{i\prime,t} = 1;$$

$$\forall j = 1,\ldots,M, tq\ q_{i,j} = q_{i\prime,j} = 1 \qquad \text{(D)}$$

Le résultat du test correspondant peut être trouvé dans le chapitre des tests 5.5.

1-Cuts

1-cuts par Osorio et al.(2002) sont des coupes générées à partir des contraintes de types $d^T x \leq b$ avec $d_1 \geq d_2 \geq \cdots \geq d_n > 0$. Ce sont donc des contraintes redondantes qui peut pourtant plus efficaces. Par exemple pour la contrainte $x_1 + 2x_2 + 2x_3 \leq 3$ dont les variables sont binaires, on peut en déduire un 1-cuts $x_2 + x_3 < 1$, car si $x_2 = x_3 = 1$ la contrainte originale sera violée.

Il existe déjà l'algorithme[?] pour générer automatiquement les 1-cuts, alors nous l'avons réalisé et ensuite appliqué sur les contraintes de ressources dans le Preprocessing. Le résultat du test montre que ces coupes ont bien un effet possitif pour fixer plus de variables surtout pour les permiers 3 scénarios. Cependant, nous avons aussi aperçu que le nombre de coupes générées est considérable pendant cette démarche, ce qui peut potentiellement avoir un effet négatif sur le temps de résolution, parce que quand nous avons de nombreux contraintes ajoutés, Cplex va alors mettre plus de temps pour traiter ces contraintes.

Pour résoudre ce problème, nous nous posons la question : combien de 1-cuts devons-nous générer et quelles sont les contraintes prioritaires. Empiriquement, nous décitons de trier les contraintes par ordre croissante de la partie droite de l'équation (noté RHS) car quand le RHS est plus petit, ça génère des coupes plus contraignantes. Après, en considérant la partie gauche de l'équation, nous avons aussi essayé une deuxième approche, c'est de trier les contraintes par ordre décroissante de LRS/RHS, dont LRS est la somme des coefficients à gauche de l'équation.

Pour la question sur le nombre de coupes à générer, nous avons fait un test sur des instances choisies avec un nombre différent des coupes pour voir c'est combien le seuil pour chaque scénario. Ensuite avec le résultat des scénarios 4, 5 et 6 comme échantillons, nous avons trouvé une fonction qui peut donner un seuil selon le nombre de tâche et le nombre de machine du problème. La recherche de cette fonction est basé sur la procédure de "Multiple Linear Regression". Un outil en ligne [1] a été utilisé pour cette recherche.

Les tests et analyses effectués sont décrits dans la section 5.6.

1. http://www.xuru.org/rt/MLR.asp

Tests

Pour tester le fonctionnement des différentes méthodes de résolution dans ce projet, plusieurs tests ont été effectués sur un ensemble d'instances du problème. Cet ensemble contient 6 scénarios dont chacun est composé par 20 instances du problème. Toutes ces données de test sont générées par le programme Testeur.

Dans la partie suivante, les trois premiers tests sont pour comparer les trois méthodes de résolutions : méthode exacte, H1 et H2. Les sections après sont les tests sur le Preprocessing.

5.1 Méthode exacte - solveur Cplex

Dans un premier temps, le test du solveur Cplex est effectué car les solutions trouvées par le solveur peuvent nous servir à évaluer la performance des autres méthodes de résolution.

Le tableau 5.1 représente les statistiques effectuées sur le résultat de test du solveur Cplex. Les significations des colonnes sont :

1. Sc(N/M) : numéro de scénario et le nombre de VM et de serveur physique.
2. #Infeas : le nombre des instances qui sont prouvées comme "Infaisable" par le solveur.
3. #Opt : le nombre des instances qui sont résolues avec la solution optimale trouvée.
4. #Mem : le nombre des instances pour lesquelles le solveur n'a pas pu trouver la solution optimale à cause de la limite de l'espace mémoire.
5. #Tim : le nombre des instances pour lesquelles le solveur n'a pas pu trouver la solution optimale à cause de la limite du temps.
6. T_{min}, T_{avg}, T_{max} : le temps (minimum, moyenne et maximum) de résolution en seconde. Nous ne considérons pas les instances infaisable.

Sc(N/M)	#Infeas	#Opt	#Tim	#Mem	T_{min}	T_{avg}	T_{max}
Sc1(8/2)	2	18	0	0	0,02	0,03	0,04
Sc2(11/3)	9	11	0	0	0,10	0,28	0,59
Sc3(15/4)	1	19	0	0	0,31	2,55	30,09
Sc4(18/5)	0	20	0	0	1,12	99,69	783,99
Sc5(21/5)	3	10	6	1	23,10	1048,50	1800,31
Sc6(24/6)	2	6	8	4	124,91	1188,18	1800,49

TABLE 5.1 – Résultat de test du solveur Cplex

A partir du scénario 4, on commence à avoir des grosses instances pour lesquelles le solveur n'a pas pu trouver la solution optimale ($T_{max} = 1800$) à cause de la limite de temps.

5.2 Méthode heuristique de liste

Le tableau 5.2 montre le résultat de test de la méthode heuristique de liste (on l'appelle H1 pour raison de simplicité). Dans ce tableau, les colonnes D_{min}, D_{avg} et D_{max} signifient la déviation entre la solution

trouvée par H1 et la solution trouvée par le solveur Cplex. A noter que pour les 3 premiers scénarios le solveur a trouvé la solution optimale pour toutes les instances donc cette déviation peut montrer la qualité de notre méthode H1 par rapport à la solution optimale. A partir du scénario 4 on commence à avoir des instances pour lesquelles le solveur a trouvé une solution faisable mais pas optimale à cause de la limite du temps ou de l'espace mémoire, alors dans ce cas la déviation est aussi affectée.

Sc(N/M)	#Infeas	#Solved	T_{min}	T_{avg}	T_{max}	D_{min}	D_{avg}	D_{max}
Sc1(8/2)	2	18	0.00	0.00	0.00	0%	19%	67%
Sc2(11/3)	9	11	0.00	0.00	0.00	0%	17%	46%
Sc3(15/4)	7	13	0.00	0.00	0.00	5%	24%	58%
Sc4(18/5)	11	9	0.00	0.00	0.00	10%	27%	37%
Sc5(21/5)	16	4	0.00	0.00	0.01	30%	36%	43%
Sc6(24/6)	20	0	0.00	0.01	0.02	*	*	*

TABLE 5.2 – Résultat de test de la méthode heuristique de liste (H1)

On trouve souvent 0.00 seconde dans les colonnes de temps, ce qui signifie que le temps de résolution de H1 est très court.

Par rapport à la déviation, la performance de H1 n'est pas très stable : pour certaines instances du problème, H1 a trouvé la solution optimale ($D_{min} = 0\%$) mais on a aussi dans le Sc1 $D_{max} = 67\%$ qui n'est pas très optimiste. Pour le scénario 6, H1 n'a résolu aucune instance donc la déviation n'est pas calculée.

5.3 Méthode heuristique de Cplex

Le tableau 5.3 montre le résulat de test sur la méthode heuristique basée sur le solveur Cplex (H2).

Sc(N/M)	#Infeas	#Solved	T_{min}	T_{avg}	T_{max}	D_{min}	D_{avg}	D_{max}
Sc1(8/2)	2	18	0.02	0.07	0.15	0%	0%	1%
Sc2(11/3)	9	11	0.08	0.29	0.97	0%	0%	1%
Sc3(15/4)	1	19	0.64	1.37	5.21	0%	0%	2%
Sc4(18/5)	0	20	1.53	80.33	400.32	0%	0%	2%
Sc5(21/5)	3	17	1.68	240.31	400.41	0%	2%	4%
Sc6(24/6)	2	18	2.08	286.79	410.43	-1%	1%	7%

TABLE 5.3 – Résultat de test de la méthode heuristique de Cplex (H2)

Puisque la méthode H2 est basée sur le solveur Cplex, pour les petites instances de problème elle a trouvé des solutions qui sont très proches des solutions optimales. Pour les grandes instances (à partir du scénario 4), les solutions qu'elle a trouvées sont aussi très intéressantes avec la déviation maximale égale à 7% qui reste acceptable. Le temps maximum de résolution est vers 200 secondes qui provient de l'implémentation de H2.

5.4 Preprocessing sans coupes supplémentaires

A partir de cette section, nous présentons les tests réalisés sur le technique de Preprocessing. On rappelle ici l'idée du Preprocessing est de fixer autant que possible de variables dans le modèle pour réduire la taille du modèle. Pour le faire, il faut avoir une borne supérieure (UB) et une borne inférieure (LB) qui sont

assez proches de la solution optimale. Nous avons lancé d'abord le Preprocessing sur toutes les variables booléennes sans ajoutant des coupes supplémentaires.

Dans le tableau 5.4 on peut trouver des statistiques sur la qualité des LB et des UB ainsi que la proportion de variables fixées pendant le Preprocessing. Les colonnes s'interprêtent comme le suivant :

1. DevLB : la déviation (minimum, moyenne et maximum) entre la solution optimale et LB.
2. DevUB : la déviation (minimum, moyenne et maximum) entre UB et la solution optimale.
3. Fixed : la proportion du nombre de variables fixées par rapport au nombre de toutes les variables ayant passé le preprocessing.

	DevLB			DevUB			Fixed		
Sc(N/M)	min	avg	max	min	avg	max	min	avg	max
Sc1(8/2)	0.00%	4.83%	17.07%	0.00%	0.04%	0.65%	0.00%	9.25%	51.83%
Sc2(11/3)	0.04%	7.21%	17.13%	0.00%	0.01%	0.15%	0.00%	13.90%	66.67%
Sc3(15/4)	1.10%	4.01%	10.12%	0.00%	0.36%	1.80%	0.00%	0.88%	10.96%
Sc4(18/5)	0.05%	5.28%	12.56%	0.00%	0.53%	1.81%	0.00%	2.86%	55.64%
Sc5(21/5)	1.82%	5.03%	8.94%	0.56%	1.94%	7.62%	0.00%	0.00%	0.00%
Sc6(24/6)	3.94%	5.91%	8.72%	0.30%	1.75%	5.40%	0.00%	0.09%	0.56%

TABLE 5.4 – Résultat de test du Preprocessing sur toute les variables booléannes

Nous constatons que le Preprocessing naïf fonctionne mais pas suffisamment bien. Il ne fixe pas beaucoup de variables donc il n'aide pas la résolution de Cplex. Souvent, ce problème survient quand l'UB ou la LB n'est pas assez bonne. En comparant la qualité de LB et d'UB, on peut trouver que relativement les UB sont assez proches que la solution optimale, en revanche les LB ne sont pas très bonnes.

Pour améliorer les LB, nous avons cherché d'ajouter des contraintes supplémentaires (Cuts) au modèle LP du problème, pour enfin améliorer le fonctionnement du Preprocessing.

5.5 Preprocessing avec coupe 1

Nous avons d'abord relancé le test de Preprocessing avec la coupe 1 (cf 4.3.2) qui est problème-dépendante concernant les contraintes de ressources. Malheureusement, l'ajout de cette coupe n'a apporté aucun changement au résulat de test (identique que 5.4). Nous pouvons donc dire que la coupe 1 n'est pas utile pour le fixage des variables pendant le Preprocessing, mais c'est très probable qu'elle est efficace dans le MIP. Néanmoins, à cause de la limite de temps du projet, nous n'avons pas pu tester la performance de cette coupe dans le MIP.

5.6 Preprocessing avec coupe 2

Cette partie contient le résultat (tableau5.5) de test de Preprocessing avec la coupe 2 (aussi notée 1-Cuts, cf 4.3.2).

Dans le résultat, les trois colonnes d'UB restent les mêmes car le Preprocessing n'affecte que sur les LB. Nous pouvons trouver qu'avec la coupe 2, le Preprocessing a pu fixé plus de variables qu'avant, surtout pour les premiers 3 scénarios. En conséquence, les LB ont été aussi améliorées (la déviation entre la solution optimale et la LB devient plus petite).

En résumé, la coupe 2 est utile pour renforcer le Preprocessing mais le résultat n'est pas encore assez bon pour bien accélérer la résolution après. A partir de scénario 4, le Preprocessing n'aide quasiment plus.

	DevLB			DevUB			Fixed		
Sc(N/M)	min	avg	max	min	avg	max	min	avg	max
Sc1(8/2)	0.00 %	2.87%	17.07 %	0.00%	0.04%	0.65%	0.00%	43.91 %	100.00%
Sc2(11/3)	0.04 %	4.54%	11.97 %	0.00%	0.01%	0.15%	0.00%	15.80 %	66.67%
Sc3(15/4)	0.44 %	3.47%	9.40 %	0.00%	0.36%	1.80%	0.00%	2.22 %	10.96%
Sc4(18/5)	0.05 %	4.94%	11.60 %	0.00%	0.53%	1.81%	0.00%	2.91 %	55.64%
Sc5(21/5)	1.82 %	4.72%	8.82 %	0.56%	1.94%	7.62%	0.00%	0.00 %	0.00%
Sc6(24/6)	3.64 %	5.65%	8.45 %	0.30%	1.75%	5.40%	0.00%	0.09 %	0.56%

TABLE 5.5 – Résultat de test du Preprocessing avec la coupe 2

5.7 Preprocessing avec la coupe 2 renforcée

Selon les tests que nous avons fait, la coupe 2 a des effects positifs pour le Preprocessing. Nous avons donc effectué des démarches pour encore renforcer la coupe 2.

Dans la phase de fixage, le Preprocessing peut fixer des variables en générant des contraintes, mais quand le nombre de contraintes devient plus en plus grand, ça peut au contraire ralentir la résolution de Cplex dans la phase suivante. Dans notre cas, puisque la coupe 2 va générer beaucoup de contraintes, nous essayons donc de trouver un seuil supérieur pour le nombre des contraintes générés. Mise en place de ce paramètre doit diminuler autant que possible le temps utilisé pour la résolution MIP, tout en assurant que le nombre de variables fixées ne soit pas affecté.

Pour trouver la relation entre le nombre de contraites générées à partir de la coupe 2 et le temps dépensé pendant la résolution Cplex, nous avons conçu deux tests d'analyse.

Premier test de l'analyse du nombre de coupes à générer

Les principles du premier test sont :

1. Pour chacun des scénarios 4, 5 et 6, nous choisissons 5 instances de problème qui peuvent être résolues à l'optimalité dans un temps modéré (pas trop court pour être observable ni trop long pour ne pas dépasser la limite de temps).
2. Pour chaque instance, nous faisons un série de tests avec un seuil de nombre de coupes différents. Ce seuil commence à 200 puis s'incrémente d'un pas de 200 jusqu'à le nombre total des coupes qu'on peut générer.
3. Après le test, pour chaque scénario, nous cherchons le seuil qui donne un temps moyen d'exécution minimum pour les 5 instances choisies.
4. Pour que les coupes générées soient les plus contraignantes, nous avons trié avant la génération de coupe 2, les contraintes en ordre croissante de RHS (cf 4.3.2).

Le résultat du test (tableau5.6) nous a donné des échantillons pour étudier la corrélation entre le meilleur seuil et les caractéristiques de scénarios.

Sc	N	M	BestSeuil
4	18	5	600
5	21	5	400
6	24	6	1600

TABLE 5.6 – Résultat du premier test d'analyse du seuil de la coupe 2

Ce résultat montre que quand N (le nombre de tâches) augmente, BestSeuil diminue et quand M (le nombre de machines) augmente, BestSeuil augmente. Empiriquement, nous avons fait une "Multiple Linear

Regression" [1] sur ces données de test pour trouver une fonction qui peut donner le meilleur seuil pour un scénario. La fonction trouvée (notée Seuil1) est :

$$BestSeuil = -66.67N + 1400M - 5200 \quad \text{(A)}$$

Elle s'applique pour les scénarios après le 4, car les 3 premiers sont assez facile.

Néanmoins, la corrélation entre le temps écoulé et le nombre de contraintes n'est pas très évidente dans le cas de Cplex. Ça concerne le mécanisme au sein de Cplex que nous ne pouvons pas savoir donc cet approche est avant tout empirique.

Deuxième test de l'analyse du nombre de coupes à générer

Une deuxième analyse (tableau5.7) a été faite pour chercher le meilleur seuil de la coupe 2. Cet analyse est presque pareil que la première sauf que cette fois au lieu de trier les contraintes en ordre croissante de RHS, nous les trions en ordre décroissante de LHS/RHS, dont LHS signifie le somme de tous les coefficients à gauche de l'équation.

Sc	N	M	BestSeuil
4	18	5	1800
5	21	5	1200
6	24	6	2600

TABLE 5.7 – Résultat du deuxième test d'analyse du seuil de la coupe 2

La fonction trouvée (notée Seuil2) cette fois est :

$$BestSeuil = -200N + 2000M - 4600 \quad \text{(B)}$$

Test de la coupe 2 avec seuil

Après mettre en place la fonction de seuil, nous avons relancé encore 2 fois le test de la coupe 2 avec la fonction de seuil différente.

D'abord sur le fixage des variables, avec la mise en oeuvre du seuil, le nombre de variables fixées reste le même pour tous les 6 scénarios, sauf 3 instances (tableau5.8) qui sont affectées par la première fonction de seuil (Seuil1) :

Sc-id	NoSeuil	Seuil1	Seuil2
3-11	485	484	485
3-19	677	666	677
4-8	101	0	101

TABLE 5.8 – Nombre des variables fixées des 3 instances particulières

Ensuite sur le temps total de la résolution (Preprocessing + MIP), voici (tableau5.9) la déviation des 2 cas avec seuil par rapport au cas sans seuil. Seulement les instances dont les solutions optimales sont trouvées par toutes les trois méthodes (NoSeuil, Seuil1, Seuil2) sont considérées.

Nous pouvons trouver qu'avec l'ajout du seuil, par rapport au cas sans seuil, nous pouvons gagner beaucoup de temps en moyenne. Le Seuil1 est meilleur que le seuil2 pour les scenarios 4 et 5 mais pas 6. Donc aucun seuil peut dominer l'autre.

De plus, nous voulons maintenant faire un autre tableau5.10 de déviation mais cette fois par rapport au MIP naïf (cf 5.1) sans coups 2 ajouté.

1. http://www.xuru.org/mlr

	Seuil1			Seuil2		
Sc	DevMin	DevAvg	DevMax	DevMin	DevAvg	DevMax
1	-40,00%	-6,93%	12,50%	-40,00%	-6,09%	0,00%
2	-8,64%	2,41%	15,52%	-4,69%	3,12%	19,75%
3	-14,03%	-2,16%	12,84%	-13,59%	1,46%	36,63%
4	-42,53%	-6,54%	39,40%	-42,75%	-4,16%	23,02%
5	-56,47%	-13,39%	22,21%	-34,76%	-12,49%	23,78%
6	-50,45%	-22,79%	51,32%	-66,12%	-23,16%	13,95%

TABLE 5.9 – Déviation de temps avec les 2 fonctions de seuil par rapport au cas sans seuil

	Seuil1			Seuil2		
Sc	DevMin	DevAvg	DevMax	DevMin	DevAvg	DevMax
1	-50,00%	110,65%	350,00%	-50,00%	108,80%	300,00%
2	-9,09 %	68,03 %	160,00%	-9,09 %	69,02 %	150,00%
3	-13,43%	64,99 %	156,41%	-12,40%	69,33 %	143,59%
4	-59,14%	14,56 %	75,40 %	-71,09%	20,22 %	91,65 %
5	-43,79%	5,06 %	74,50 %	-33,12%	9,00 %	88,87 %
6	-62,88%	-5,71 %	79,70 %	-57,16%	-8,30 %	35,32 %

TABLE 5.10 – Déviation de temps avec les 2 fonctions de seuil par rapport au MIP sans Preprocessing

A partir du tableau5.10 nous pouvons observer l'amélioration que nous avons apportée jusqu'à présent, c'est-à-dire le Preprocessing plus la coupe 2 avec seuil. Le temps que nous avons gagné est considérable.

Les trois tableaux ci-après5.11 sont toujours pour comparer les trois tests (NoSeuil, Seuil1 et Seuil2), mais d'une autre façon. Dans ces tableau la déviation est calculée par

$$Dev = (Sol - Min(Sol_{NoSeuil}, Sol_{Seuil1}, Sol_{Seuil2}))/Min(Sol_{NoSeuil}, Sol_{Seuil1}, Sol_{Seuil2})$$

C'est-à-dire la déviation entre la solution donnée et la meilleure solution des trois cas. Cette fois toutes les instances faisables sont considérées sauf les instances qui sont invalides à cause de la limite de mémoire.

A partir de la colonne D_{avg} nous pouvons constater que si nous prenons compte de toutes les instances alors le seuil 1 peut donner en général une solution qui est la meilleure entre les trois cas.

NoSeuil												
Sc	n	m	#InFea	#Opt	#Tim	#Mem	T_{min}	T_{avg}	T_{max}	D_{min}	D_{avg}	D_{max}
1	8	2	2	18	0	0	0,01	0,07	0,11	0,00%	0,00%	0,00%
2	11	3	9	11	0	0	0,20	0,40	0,81	0,00%	0,00%	0,00%
3	15	4	1	19	0	0	0,75	2,97	27,94	0,00%	0,00%	0,00%
4	18	5	0	20	0	0	1,40	112,68	1100,71	0,00%	0,00%	0,00%
5	21	5	3	7	8	2	60,48	996,29	1809,03	0,00%	0,34%	4,02%
6	24	6	2	6	6	6	162,41	986,31	1809,76	0,00%	0,06%	0,62%

Seuil1												
Sc	n	m	#InFea	#Opt	#Tim	#Mem	T_{min}	T_{avg}	T_{max}	D_{min}	D_{avg}	D_{max}
1	8	2	2	18	0	0	0,01	0,06	0,09	0,00%	0,00%	0,00%
2	11	3	9	11	0	0	0,20	0,40	0,74	0,00%	0,00%	0,00%
3	15	4	1	19	0	0	0,73	2,83	26,05	0,00%	0,00%	0,00%
4	18	5	0	20	0	0	1,42	106,37	1102,95	0,00%	0,00%	0,00%
5	21	5	3	10	6	1	40,31	963,81	1803,47	0,00%	0,34%	4,02%
6	24	6	2	6	6	6	155,72	884,55	1809,45	0,00%	0,06%	0,62%

Seuil2												
Sc	n	m	#InFea	#Opt	#Tim	#Mem	T_{min}	T_{avg}	T_{max}	D_{min}	D_{avg}	D_{max}
1	8	2	2	18	0	0	0,01	0,06	0,09	0,00%	0,00%	0,00%
2	11	3	9	11	0	0	0,20	0,41	0,97	0,00%	0,00%	0,00%
3	15	4	1	19	0	0	0,75	2,92	26,36	0,00%	0,00%	0,00%
4	18	5	0	20	0	0	1,39	106,86	1066,17	0,00%	0,00%	0,00%
5	21	5	3	9	6	2	43,63	892,31	1808,87	0,00%	0,37%	3,98%
6	24	6	2	6	8	4	92,53	917,72	1811,88	0,00%	0,42%	2,06%

TABLE 5.11 – NoSeuil vs Seuil1 vs Seuil2

Test avec AddMIPStart

En plus de la mise en place des seuils, nous avons aussi testé le fonctionnement de AddMIPStart [2]. Avec AddMIPStart, nous pouvons fournir une solution comme le point de départ de la résolution MIP. Ça peut donc peut-être gagner du temps pour Cplex. Mais le souci est que l'application de cette fonction va peut-être changer la startégie de branchement du Cplex donc nous ne sommes pas sûr que ça marche toujours.

Le tableau5.12 montre la déviation du temps quand on fait AddMIPStart sur une fonction de seuil. $Deviation = (T_{Seuil+AddMIPStart} - T_{Seuil})/T_{Seuil}$. Comme l'habitude, on prend en compte seulement pour les instances qui sont résolues à l'optimalité ou sont prouvées infaisables. On peut trouver que généralement plus le scénario est dur plus l'ajout d'AddMIPStart peut accélérer la résolution.

Déviation du temps						
	Seuil1 vs Seuil1+AddMIPStart			Seuil2 vs Seuil2+AddMIPStart		
sc	min	avg	max	min	avg	max
1	-66,67%	12,35 %	52,63 %	-66,67 %	13,61 %	91,80%
2	-15,52%	4,20 %	13,04 %	-21,57 %	1,00 %	13,04 %
3	-30,94%	0,54 %	20,50 %	-43,10 %	-2,69 %	22,06%
4	-84,47%	-3,57 %	31,80 %	-112,46 %	1,78 %	48,69%
5	-59,01%	-6,76 %	55,32 %	-68,91 %	-0,19 %	32,25%
6	-68,94%	-8,28 %	21,71 %	-51,68 %	-10,87%	6,14%

TABLE 5.12 – Déviation suivant l'ajout de AddMIPStart sur les fonctions de Seuil

Dans les tableaux5.13 et 5.14, $Deviation = (Sol - V)/V$ dont $V = Min(Sol_{Seuil+AddMIPStart}, Sol_{Seuil})$.

Seuil1												
Sc	n	m	#InFea	#Opt	#Tim	#Mem	T_{min}	T_{avg}	T_{max}	D_{min}	D_{avg}	D_{max}
1	8	2	2	18	0	0	0,01	0,06	0,09	0,00 %	0,00 %	0,00%
2	11	3	9	11	0	0	0,20	0,40	0,74	0,00 %	0,00 %	0,00%
3	15	4	1	19	0	0	0,73	2,83	26,05	0,00 %	0,00 %	0,00%
4	18	5	0	20	0	0	1,42	106,37	1102,95	0,00%	0,00%	0,00%
5	21	5	3	10	6	1	40,31	924,69	1803,47	0,00%	0,31%	3,15 %
6	24	6	2	6	8	4	155,72	968,17	1809,45	0,00%	0,20%	2,02%

Seuil1+AddMIPStart												
Sc	n	m	#InFea	#Opt	#Tim	#Mem	T_{min}	T_{avg}	T_{max}	D_{min}	D_{avg}	D_{max}
1	8	2	2	18	0	0	0,01	0,08	0,19	0,00 %	0,00 %	0,00%
2	11	3	9	11	0	0	0,22	0,41	0,83	0,00 %	0,00 %	0,00%
3	15	4	1	19	0	0	0,83	2,87	26,74	0,00 %	0,00 %	0,00%
4	18	5	0	20	0	0	1,59	104,40	852,77	0,00 %	0,00 %	0,00%
5	21	5	3	9	5	3	25,57	931,82	1803,49	0,00 %	0,00 %	0,00 %
6	24	6	2	6	6	6	101,76	978,29	1809,59	0,00%	0,22%	1,31%

TABLE 5.13 – Seuil1 vs Seuil1+AddMIPStart

Comme ce que dont nous nous sommes inquiétés, AddMIPStart a bien changer la stratégie de branchement du Cplex. Le nombre des instances qui ont atteint la limite du temps et de la mémoire devient

2. http://pic.dhe.ibm.com?topic=%2Filog.odms.cplex.help%2FContent%2FOptimization%2FDocumentation%2FOptimization_Studio%2F_pubskel%2Fps_usrmancplex1850.html

différent que sans AddMIPStart. Selon D_{avg}, nous pouvons quand même dire que l'AddMIPStart a des effects positifs sur la qualité de solutions.

Seuil2												
Sc	n	m	#InFea	#Opt	#Tim	#Mem	T_{min}	T_{avg}	T_{max}	D_{min}	D_{avg}	D_{max}
1	8	2	2	18	0	0	0,01	0,06	0,09	0,00%	0,00%	0,00%
2	11	3	9	11	0	0	0,20	0,41	0,97	0,00%	0,00%	0,00%
3	15	4	1	19	0	0	0,75	2,92	26,36	0,00%	0,00%	0,00%
4	18	5	0	20	0	0	1,39	106,86	1066,17	0,00%	0,00%	0,00%
5	21	5	3	9	6	2	43,63	892,07	1808,87	0,00%	0,66%	8,62%
6	24	6	2	6	8	4	92,53	998,04	1811,88	0,00%	0,18%	1,76%

Seuil2+AddMIPStart												
Sc	n	m	#InFea	#Opt	#Tim	#Mem	T_{min}	T_{avg}	T_{max}	D_{min}	D_{avg}	D_{max}
1	8	2	2	18	0	0	0,01	0,15	0,66	0,00%	0,00%	0,00%
2	11	3	9	11	0	0	0,22	0,40	0,81	0,00%	0,00%	0,00%
3	15	4	1	19	0	0	0,81	2,80	24,85	0,00%	0,00%	0,00%
4	18	5	0	20	0	0	1,59	105,66	996,10	0,00 %	0,00%	0,00%
5	21	5	3	8	6	3	25,83	918,98	1809,21	0,00%	0,38%	2,92%
6	24	6	2	6	6	6	89,18	965,00	1811,99	0,00%	0,12%	0,79%

TABLE 5.14 – Seuil2 vs Seuil2+AddMIPStart

Cette fois avec AddMIPStart sur Seuil2, nous avons une meilleure déviation moyenne des solutions avec AddMIPStart. Comme le cas du Seuil1, le *#Opt*, *#Tim*, *#Mem* sont tous affectés.

En conclusion, pour bien optimiser la résolution exacte de ce problème, il vaut mieux de faire d'abord le Preprocessing avec la coupe 2 et le seuil 2, puis donner une solution de départ avec le lancement de MIP. La solution de départ corresponds à la UB du Preprocessing.

Tableaux supplémentaires

Les tableaux suivants concernent des comparaisons entre MIP seul, Seuil1 et Seuil2+AddMIPStart.

Tableau 5.15 contient des statistiques sur les résultats de résolution pour les 3 méthodes. Il s'agit des nombres d'instances dans chaque état de résolution : optimalité, limite de temps, limite de mémoire.

				MIP			Seuil1			Seuil2+AddMIPStart		
Sc	n	m	Infea	Opt	Tim	Mem	Opt	Tim	Mem	Opt	Tim	Mem
1	8	2	2	18	0	0	18	0	0	18	0	0
2	11	3	9	11	0	0	11	0	0	11	0	0
3	15	4	1	19	0	0	19	0	0	19	0	0
4	18	5	0	19	1	0	20	0	0	20	0	0
5	21	5	3	6	10	1	10	7	0	8	6	3
6	24	6	2	5	13	0	6	8	4	6	6	6

TABLE 5.15 – Statsistiques sur les résultats de résolution

Tableau 5.16 compare les temps d'exécution pour les 3 méthodes. L'analyse est faite sur les instances pour lesquelles toutes les 3 méthodes ont trvoué la solution optimale ou ont prouvé qu'elles sont infaisables.

	MIP			Seuil1			Seuil2+AddMIPStart		
Sc	T_{min}	T_{avg}	T_{max}	T_{min}	T_{avg}	T_{max}	T_{min}	T_{avg}	T_{max}
1	0,01	0,03	0,04	0,01	0,07	0,20	0,01	0,15	0,66
2	0,04	0,17	0,59	0,03	0,24	0,74	0,03	0,24	0,81
3	0,31	2,45	30,09	0,22	2,70	26,05	0,22	2,68	24,85
4	1,12	99,69	783,99	1,42	106,37	1102,95	1,59	105,66	996,10
5	0,95	248,50	769,13	0,67	222,59	919,01	0,66	265,22	1022,32
6	1,19	365,32	1788,43	2,26	205,93	663,83	2,26	200,79	658,59

TABLE 5.16 – Temps de résolution pour les instances résolues

Tableau 5.17 peut montrer la qualité des solutions trouvées par les 3 méthodes. Dans le tableau, $Dev = (Sol - V)/V$ dont $V = Min(Sol_{MIP}, Sol_{Seuil1}, Sol_{Seuil2+AddMIPStart})$. Cet analyse est faite sur les instances pour lesquelles :

- Aucunne méthode a atteint la limite de mémoire
- Toutes les méthodes ont trouvé une solution faisable

	MIP			Seuil1			Seuil2+AddMIPStart		
Sc	Dev_{min}	Dev_{avg}	Dev_{max}	Dev_{min}	Dev_{avg}	Dev_{max}	Dev_{min}	Dev_{avg}	Dev_{max}
1	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
2	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
3	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
4	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
5	0,00%	0,49%	5,38%	0,00%	0,42%	5,44%	0,00%	0,26%	1,68%
6	0,00%	0,07%	0,36%	0,00%	0,21%	1,87%	0,00%	0,43%	3,91%

TABLE 5.17 – Déviation de solutions

Tableau 5.18 compare le pourcentage des variables fixées pour Seuil1 et Seuil2+AddMIPStart. Il n'y a pas beaucoup de différence puisque comme nous avons expliqué, il n'y a que 3 instances pour lesquelles le nombre des variables fixées est différent pour ces 2 méthodes.

	Seuil1			Seuil2+AddMIPStart		
Sc	Fix_{min}	Fix_{avg}	Fix_{max}	Fix_{min}	Fix_{avg}	Fix_{max}
1	0,00%	43,92%	100,00%	0,00%	43,92%	100,00%
2	0,00%	15,80%	66,67%	0,00%	15,80%	66,67%
3	0,00%	**2,21%**	**15,77%**	0,00%	**2,22%**	**16,03%**
4	0,00%	**2,86%**	55,64%	0,00%	**2,91%**	55,64%
5	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
6	0,00%	0,09%	0,56%	0,00%	0,09%	0,56%

TABLE 5.18 – Pourcentage des variables fixées

Gestion du projet

La gestion du projet est mise en oeuvre depuis le tout début du projet. La planification que nous allons présenter a été bien respectée. La gestion de versions nous a aussi beaucoup servi.

6.1 Planification du projet

6.1.1 Découpage des tâches

Ce projet consiste globalement à 5 tâches à réaliser, qui sont décrites au-dessous.

- Corriger et améliorer la méthode heuristique.
- Réaliser une méthode heuristique basée sur le solveur CPLEX.
- Effectuer des recherches sur le Preprocessing du modèle de problème pour améliorer le fonctionnement de la méthode exacte.
- Effectuer des campagnes de tests.
- Analyser et comparer les résultats obtenus.
- Développer de nouvelles méthodes de résolution.

Reprise de l'existant

Cette première tâche consiste à étudier le problème et reprendre les travaux qui ont déjà été effectués. C'est la tâche de base pour la suite du projet.

Amélioration et complétion du programme heuristique

Parmi les travaux déjà réalisés, l'implémentation de l'algorithme heuristique reste à être corrigé et amélioré. La deuxième tâche du projet est alors de modifier le code pour faire fonctionner correctement ce programme heuristique.

Lancement de tests et de comparaisons

Après avoir fini la modification du programme heuristique, il faut faire le test à l'aide du programme Testeur. Ensuite il faut comparer les résultats donnés par l'heuristique et les résultats donnés par la méthode exacte. Les aspects à comparer comprennent le nombre d'instances résolu, le temps d'exécution pour trouver la solution et la déviation entre les résultats, etc.

Réalisation de la seconde méthode heuristique

Cette tâche est ajoutée au mi-cours du projet. L'idée est d'ajouter des caractéristiques heuristiques dans la méthode exacte pour avoir un meilleur compromis entre la qualité du résultat et le temps dépensé pour chercher la solution.

Pour le faire, on va mettre en place des paramètres CPLEX pour bien contrôler le temps d'exécution. Par exemple on peut arrêter le programme au bout de 3 minutes d'exécution, ou bien on peut l'arrêter quand on est sûr que la solution actuelle a au moins 95% de qualité par rapport à la solution optimale, etc.

Recherche sur le prétraitement des données pour la méthode exacte

Cette tâche est la partie principale du projet. On va travail cette fois sur la méthode exacte. L'idée est de faire des prétraitements des données d'entrée du programme exact, pour que ce dernier peut trouver plus facilement la solution optimale avec ces données prétraitées. Il s'agit du travail collaboratif avec d'autres chercheurs.

6.1.2 Planning

Cette partie concerne le planning des tâches. Cependant en tant que projet de type recherche, ce n'est pas évident d'évaluer la durée des tâches. Ici cen'est un planning en général, ceci peut évoluer suivant le déroulement du projet.

Diagramme Gantt

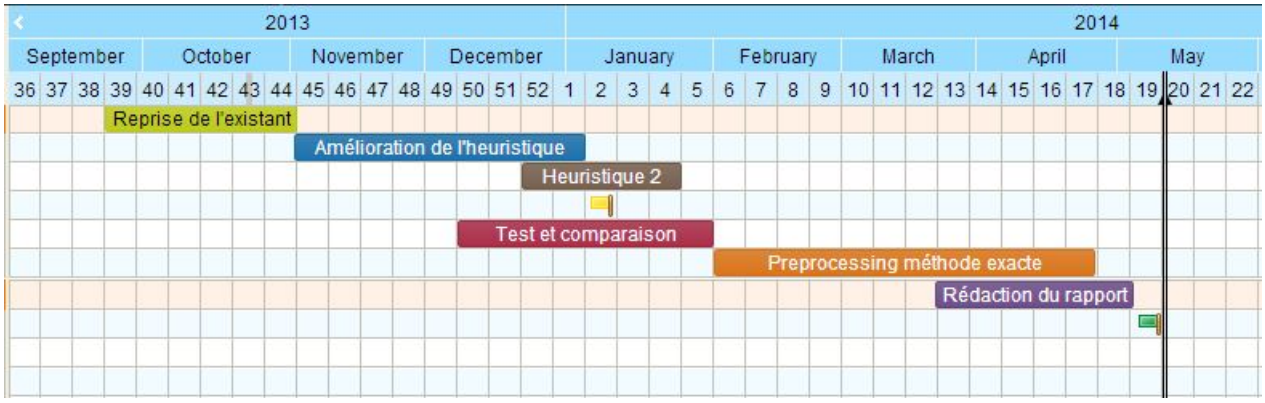

FIGURE 6.1 – Diagramme de Gantt

6.2 Gestion de versions

Le système de gestion de versions Git a été mis en place pour sauvegarder tous les programmes et les données de tests. A la fin du projet, il y a 137 commits soumis au total. La version finale des programmes peut être trouvée sur la plateforme Redmine de l'école. Un paquet qui contient toutes les données du projet est aussi livré.

Difficultés rencontrées

7.1 Compréhension du modèle mathématique

Le travail au début du projet est de reprendre les éléments existants surtout le modèle mathématique qui est composé par beaucoup de formules. La création de ces formules étant déjà un gros travail, c'est surtout pas facile de comprendre l'idée derrière. Heureusement avec l'aide de mon encadrant, j'ai enfin franchi cet obstacle et assuré l'avancement du projet.

7.2 Solveur Cplex

Pendant la réalisation de ce projet, nous avons été perturbé par le solveur Cplex. En tant que logiciel commercial, nous n'avons pas de moyen pour connaître le fonctionnement intérieur du Cplex, ceci a posé certains problèmes car nous ne pouvons pas prévoir son comportement.

Par exemple Cplex a mis en place des stratégies heuristiques pendant sa résolution, malgré qu'il nous avait donné l'impression qu'il applique une méthode exacte pendant la rechercher de la solution optimale. Il existe de nombreux paramètres pour ajuster la stratégie du Cplex. En tant que logiciel assez compliqué, le réglage de ces paramètres fait déjà un gros travail indépendant donc nous ne sommes pas entrés dedans à cause de la limite du temps de ce projet.

7.3 Complexité du problème

Une autre difficulté vient de la complexité du problème que nous traitons. Pendant la deuxième partie du projet qui concerne le Preprocessing, nous avons effectué une vingtaines de tests pour analyser la performance des coupes. Mais à cause de la complexité du problème, un test a souvent besoin d'une ou deux journée pour se terminer. En plus nous ne pouvons que lancer les tests en ordre séquentielle sur la même machine pour garder la cohérence entre les tests. C'est un peu perturbant sur le bon avancement du projet. Heureusement, tout le travail important du projet a été fini avant la date de fin.

Conclusion

Ce projet contient un travail très rich qui m'a beaucoup apris. D'abord ce projet m'a donné une compréhension plus profonde sur les méthodes de résolution : l'heuristique et la méthode exacte. J'ai pu connaître la procédure complète de la résolution d'un problème NP-difficile par programmation mathématique, y compris la modélisation mathématique et son implémentation en C++ sous Cplex.

Le travail sur le Preprocessing m'a donné une idée sur la façon d'effectuer un travail de recherche scientifique ainsi que l'importance de la campagne de tests. J'ai aussi noté l'importance de faire la gestion du projet : la documentation permet de faciliter la reprise du projet par d'autres collaborateurs ; la sauvegarde de toutes les données, surtout pour les programmes et les données de tests, permet d'autres personnes de reproduire le résultat, ce qui est très important pour assurer la justesse des travaux.

Remerciement

Je tiens à remercier dans un premier temps, mon encadrant Vincent T'Kindt, pour ses conseils et sa patience et la centaine de mails échangés entre nous. Il répondait très vite mes questions et nous discutions fréquemment pendant le projet. Tout ça m'a beaucoup aidé contre les difficultés et m'a fait maîtriser la méthodologie pour effectuer une recherche scientifique.

Je souhaite aussi de remercier sincèrement mes parents qui m'ont toujours aimé et soutenu pendant toutes mes études. Je remercie à la fin, tous mes professeurs et mes amis, en France ou en Chine, qui m'ont aidé et m'accompagné. Vous faite une partie importante de ma vie, de mes études et de la source de mon bonheur.

Le problème de Consolidation de Serveurs (hors-ligne)

Département Informatique
5e année
2013 - 2014

Rapport PFE

Résumé : Ce rapport a été rédigé pour un projet de fin d'études à l'école d'ingénieurs Polytech'Tours, département informatique. Le sujet qui est plutôt de nature de recherche scientifique, consiste à résoudre un problème d'ordonnancement NP-complet rencontré par les fournisseurs d'infrastructure comme Amazon Cloud. On cherche à ordonnancer des machines virtuelles (tâches des clients) sur des machines physiques de façon optimale pour minimiser le coût d'utilisation. Ce projet consiste à la réalisation des méthodes de résolution ainsi que la recherche sur la technique de *Preprocessing*. Beaucoup de tests et analyses ont aussi été réalisés.

Mots clefs : Consolidation de serveurs, Méthode de résolution, Heuristique, Preprocessing, Solveur Cplex

Abstract: This report is created for a graduation project at Engineering School Polytech'Tours, Computer Science department. The subject of the project which is a scientific research, consists of the resolution of a scheduling problem encountered by infrastructure service providers like Amazon Cloud. We try to find the best scheduling of virtual machines (clients' tasks) on physical machines in purpose of minimising the total cost. This report involves the modeling of the problem, the heuristic and exact approachs, the *Preprocessing* technique and a lot of tests and analysises.

Keywords: Server consolidation, Resolution method, Heuristic, Preprocessing, Cplex solver

---

Encadrants
Vincent T'Kindt
vincent.tkindt@univ-tours.fr

Étudiants
Lei SHANG
lei.shang@etu.univ-tours.fr

Université François-Rabelais, Tours