TER

Rémi Navarro - 21401257
Edouard Fouassier - 21400750

16 mai 2019

Table des matières

1 Introdution **1**

2 Structures de données **1**

3 Algorithmes **2**

4 Analyse **5**

5 Conclusion **6**

6 Annexes **7**

1 Introdution

Dans le cadre du module TER du S2 Master Informatique à l'UVSQ, nous avons eu l'occasion de réaliser un projet sous la direction de Mr Yann Strozecki et Mael Guiraud. Nous avons choisi, parmi les sujets proposés, le sujet "Algorithme glouton de remplissage" car c'est un sujet qui demande une bonne compréhension de l'algorithmique ce qui nous a beaucoup intéressé.

De nos jours les échanges par les différents réseaux sont centralisés dans des datacenters ou cloud. Pour gagner en efficacité il faut minimisé la latence lors de l'envoie d'un message vers un cloud.

L'objectif de ce projet est de concevoir et comparer des algorithmes gloutons qui permettent de placer au mieux des tâches periodiques avec des contraintes portant sur les paires de tâches. Pour cela nous utilisons un modèle où les tâches sont envoyées periodiquement et le temps entre l'envoie et la reception est fixe. Dans ce modèle il y a deux périodes de taille P, l'envoie d'une tâches est placé sur la première période et la réception sur la seconde après un delai. Il faut donc réussir à placer un maximum de tâches dans la période.

2 Structures de données

Dans un premier temps nous utilisions les structures suivantes :

Une structure "Task" représentant les tâches, composées de 3 entiers : le numero de la tâche, son délai et sa place qui est initialisé a -1, ainsi qu'un tableau de deux entiers, un pour le cycle aller et un pour le cycle retour.

Les taches "Task" étaient liées avec la structure chaine.

Chaine {
    Task t   //une tache
    chaine ↑next   //la tache suivante.
}

La période était stockée dans deux tableaux d'entier, nous écrivions le numéro de la tâche dans la ou les case(s) qu'elle occupait.

Mais comme seul les espaces disponibles de la periode nous interesse, cette structure n'était pas optimale.

Nous sommes donc passé à une structure représentant les espaces libres de la periode sous forme d'une chaine.

	Structure initiale	**Nouvelle structure**
Periode initiale de taille 10	[0,0,0,0,0,0,0,0,0,0]	(0,9)
Placement d'une tache de taille de en 5	[0,0,0,0,0,1,1,0,0,0]	(0,4)→(6,9)

Les taches n'étant plus stockées dans une liste mais dans un tableau, cela a permis de réduire la mémoire utilisée et d'augmenter la taille des tests effectués.

La structure Periode est utilisée pour représenter les intervalles disponibles d'une période.

Periode {
  entier begin   //Le debut de la periode libre
  entier end   //La fin de la periode libre.
  Periode ↑next   //La periode libre suivante.
}

La structure Tasktab représente un tableau de tâches.

Tasktab {
  Task tab   //Le debut de la periode libre
  entier taille   //La fin de la periode libre.
}

3 Algorithmes

L'algorithme "FirstFit" place dans la période les tâches par ordre d'arrivée, au premier endroit disponible (first fit), si la tâche ne peut pas être placée, on passe à la suivante.

Algorithm 1 FirstFit

Require: Tasktab, PeriodeMax
  for chaque Task dans Tasktab **do**
    for $i \leftarrow 0$ to PeriodeMax **do**
      if task entre dans la periode aller et t entre dans la periode retour après le delay **then**
        $task.place \leftarrow i$
      end if
    end for
  end for
  return Tasktab

C'est l'algorithme le plus trivial, il a une complexité faible en O(n*m) avec n le nombre de tâches et m la taille de la période.

L'algorithme "AlgoLourd" calcul pour chaques tâches son nombre de places disponibles puis place celle ayant le plus de contraintes.

Algorithm 2 AlgoLourd

```
Require: Tasktab, PeriodeMax
  min ← PeriodeMax
  taskMin ← 0
  libreMin ← 0
  for chaque Task do
    for chaque Task t dans Tasktab do
      compteur ← 0
      libre ← 0
      for i ← 0 to PeriodeMax do
        if t entre dans la periode aller et t entre dans la periode retour après le delay then
          compteur ← compteur + 1
          libre ← i
        end if
      end for
      if compteur <= compteurMin then
        compteur ← compteurMin
        taskMin ← t
        libreMin ← libre
      end if
    end for
    taskMin.place ← libreMin
  end for
  return Tasktab
```

C'est l'algorithme qui permet de placer le plus de taches parmi nos quatres algorithmes mais il est 30 fois plus long que l'algorithme "FirstFit".
Sa complexité est O($n^2 * m$) avec n le nombre de tâches et m la taille de la période.

L'algorithme "AlgoSuperLourd" calcul pour chaques tâches celle qui bloque le plus les autres et on place en priorité les moins contraignantes.

Algorithm 3 AlgoSuperLourd

Require: Tasktab, PeriodeMax
  $cptAvant[tasktab.nbTask]$
  $gene[tasktab.nbTask]$
  for chaque Task **do**
    $cptAvant[] \leftarrow cptplace()$ (cptplace() permet de compter le nombre de places disponibles pour chaque tâche)
    $cptApres[tasktab.nbTask]$
    for chaque Task t dans Tasktab **do**
      $gene[t] \leftarrow 0$
      for $i \leftarrow 0$ to PeriodeMax **do**
        if t entre dans la periode aller et t entre dans la periode retour après le delay **then**
          Place t
        end if
      end for
      $cptApres[] \leftarrow cptplace()$ //cptplace() permet de compter le nombre de places disponibles pour chaque tâche
      $gene[t] \leftarrow \sum_{i \leftarrow 0}^{Tasktab.nbTask} cptAvant[i] - cptApres[i]$
      Retire t de la periode
    end for
    Place les Task dans l'ordre croissant de génance
  end for
  return Tasktab

Dans cet algorithme on utilise la fonction cptplace() qui a une compléxité O(n*m) qui augmente grandement la compléxité de l'algorithme "AlgoSuperLourd".
On obtient donc une complexité O($n^3 * m$), avec n le nombre de tâches et m la taille de la période, ce qui le rend bien moins intéressant que les autres, de plus il place moins de taches.

L'algorithme "AlgoPasilourd" calcul la valeur delay mod(cycle) de chaques tâches, cela permet de les regrouper sur une période et de bien les ordonner sur l'autre.

Algorithm 4 AlgoPasilourd

Require: Tasktab, PeriodeMax, nbTask
  $val[nbTask]$
  $cycle \leftarrow$ cycle des tâches
  for chaque tache t **do**
    $tmpval \leftarrow tasktab.tab[t].delay\%cycle$
    Placement de tmpval dans le tableau val dans l'ordre croissant des tmpval
  end for
  Placement de la tâche t ayant la valeur val[t] la plus grande à l'emplacement $periodeMax - val[t]$ de la période de retour et son correspondant sur la période aller
  $nbPlace \leftarrow 1$
  for le nombre de tâches **do**
    for chaque tâches t par ordre décroissant de val[t] **do**
      if Placement de la tâche t de la tâche t ayant la valeur val[t] la plus grande à l'emplacement $periodeMax - val[t] + nbPlace * cycle$ de la période de retour et son correspondant sur la période aller possible **then**
        for $i \leftarrow 0$ to PeriodeMax **do**
          if task entre dans la periode aller et t entre dans la periode retour après le delay **then**
            $t.place \leftarrow i$
          end if
        end for
        $nbPlace \leftarrow nbPlace + 1$
      end if
    end for
  end for
  Pour toute les tâches qui n'ont pas été placé, on essaye de les placer à la manière d'un FirstFit
  return Tasktab

C'est un algorithme moyen, il est un peu plus long que l'algorithme "FirstFit" et un peu moins éfficace.
Sa complexité est O($n^2 * m$) avec n le nombre de tâches et m la taille de la période.

4 Analyse

Pour chaques tests effectués les paramètres utilisés sont : période de 50000, cycle de taille 1000 et 100 essais.

Nous avons réaliser plusieurs mesures :

— celle du taux de réussite, un algorithme reussi lorsqu'il arrive à placer toutes les tâches données. (cf Figure 1 en Annexe)
— celle du taux de complétion, le taux de completion d'un algorithme est le pourcentage de tâches qu'il a reussi à mettre parmi les tâches données. Sur ce graphique, la ligne représente le taux de complétion moyen, les points le taux de complétion minimum et les losanges le taux de complétion maximum. (cf Figure 2 en Annexe)
— celle du temps moyen. (cf Figure 3 en Annexe)

Sur le graphique du taux de reussite (Figure 1) nous pouvons constater que l'algoLourd est le plus éfficace.
Il parvient à 100% de réussite jusqu'à 32 tâches sur les 50 possibles, son taux de réussite d'écroit fortement au-delà.
Le FirstFit est un peu moins efficace, il conserve un taux de réussite de 100% jusqu'à 28 tâches mais celui decroit doucement puis chute à 32 tâches.
L'algoPasiLourd a une efficacité proche du FirstFit mais légèrement plus faible et il devient totalement inefficace plus vite.
L'algoSuperLourd a une courbe en dessous des autres à partir de 26 tâches.

Sur le graphique suivant (Figure 2) nous pouvons constaté que l'allure est la même.
L'algoLourd est le plus efficace et parvient à placer 78% des tâches quand il y a autant de tâches que de places. Ses taux de complétion minimum et maximum sont les moins extrêmes.
L'algoSuperLourd est le plus mauvais avec un taux de complétion maximum élevé mais un taux minimum très faible.

Le dernier graphe (Figure 3) apporte plus d'information, il représente le temps moyen de chaque algorithme.
Nous pouvons voir qu'il y a un algorithme ayant un temps bien supérieur aux autres, l'algoSuperLourd, il est jusqu'à 3000 fois plus long.
L'algorithme FirstFit et l'algoPasilourd ont un temps similaire (ligne la plus basse) et l'algoLourd est un peu plus long, 30 fois dans le pire des cas.

5 Conclusion

Après ces différentes analyse nous pouvons determiné que, parmis ces algorithmes, si nous voulons placer le plus de tâches l'algoLourd est le plus éfficace, il parvient a garder un taux de réussite totale plus longtemps et après cela sont taux de complétion est plus élevé.
Si nous voulons un algorithme rapide, le FirstFit est recommandé, il est plus rapide et reste très éfficace.
Si nous voulons trouver un juste milieu le FirstFit est le meilleur car il est rapide et a une éfficacité proche de l'algoLourd.

6 Annexes

Pour chaques graphes les paramètres utilisés sont : période de 50000, cycle de taille 1000 et 100 essais.

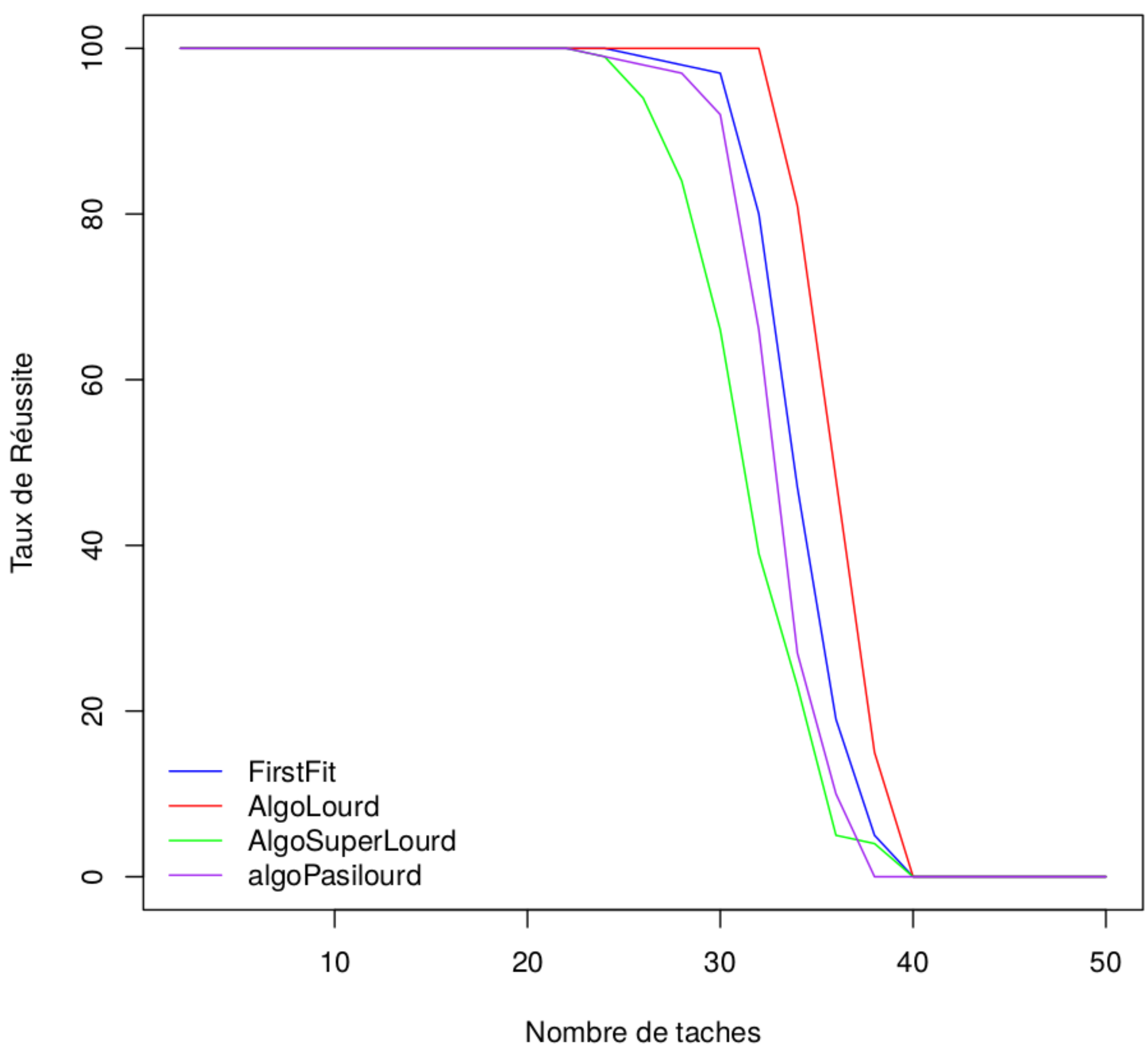

Figure 1 – Taux de réussite des algorithmes sur 100 éssais en fonction du nombre de tâches

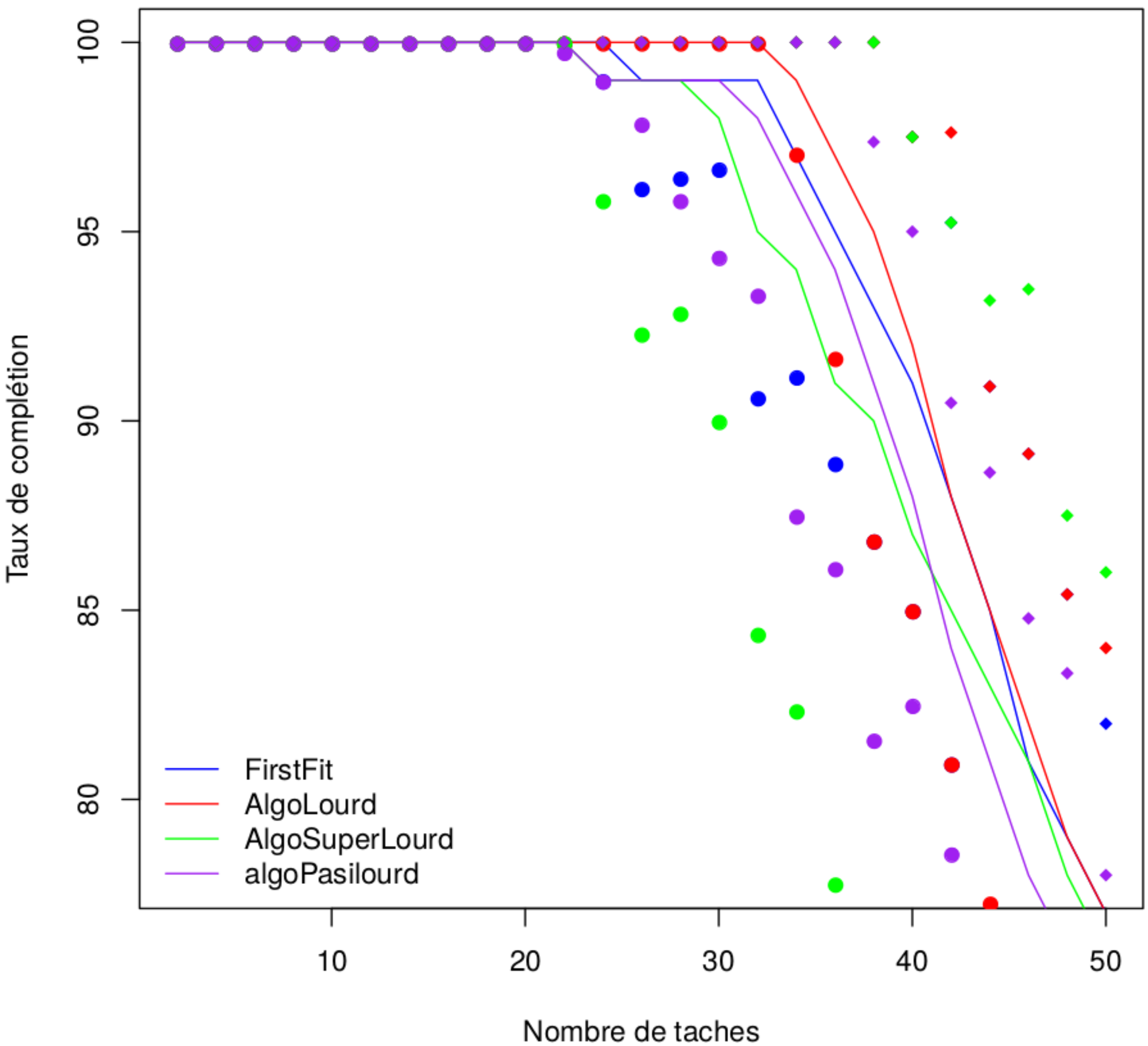

FIGURE 2 – Taux de complétion des algorithmes sur 100 éssais en fonction du nombre de tâches

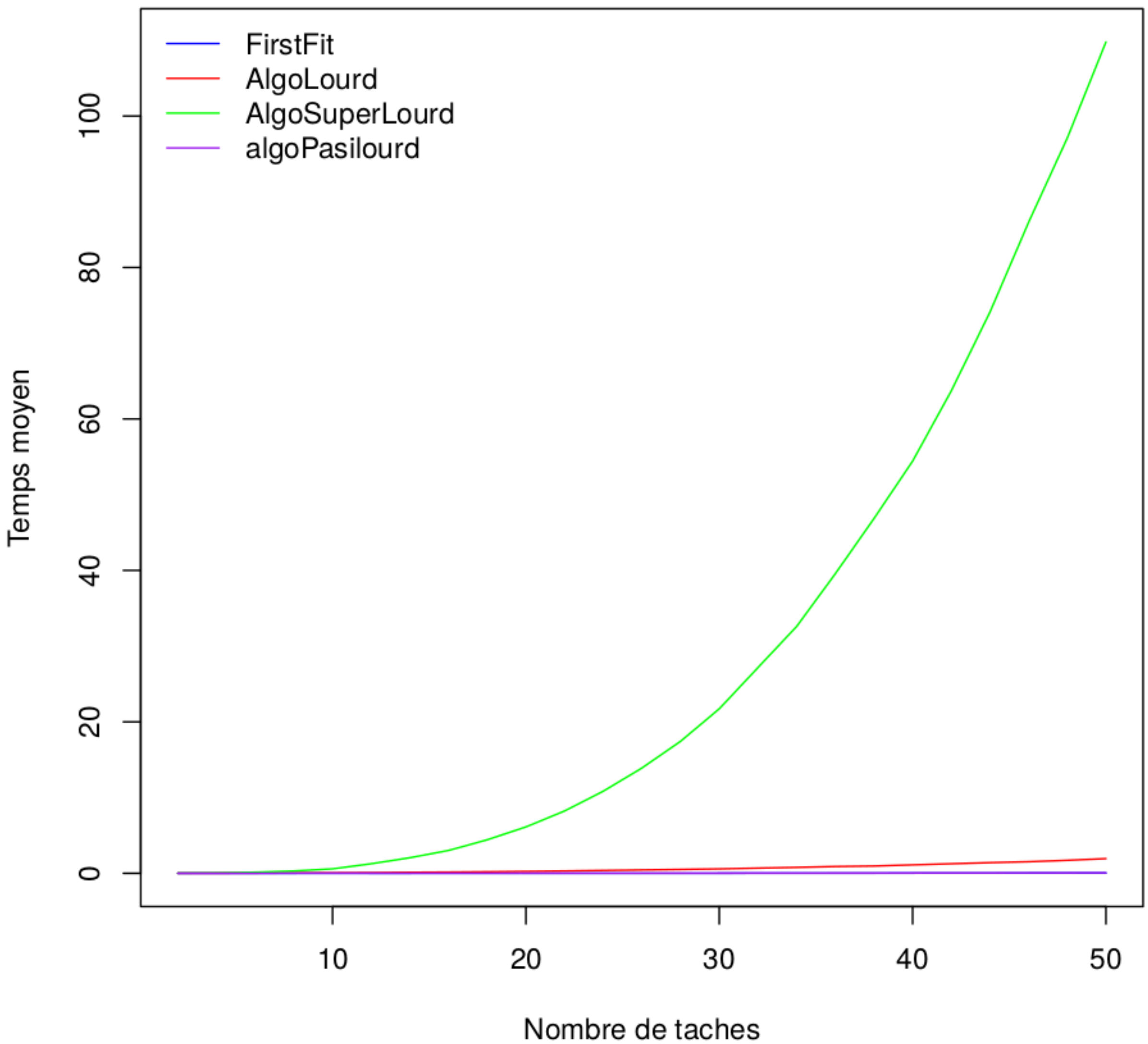

FIGURE 3 – Temps moyen d'éxécution des algorithmes sur 100 éssais en fonction du nombre de tâches