TER

Rémi Navarro - 21401257
Edouard Fouassier - 21400750

5 mai 2019

Table des matières

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 une sujet qui demande une bonne compréhension de l'algorithmique ce qui nous a beaucoup intéressé.

De nos jour 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 envoyés periodiquement et le temps entre l'envoie et la reception est fixe. Dans ce modèle il y a deux periodes de taille P, l'envoie du tâches est placé sur la première periode et la réception sur la seconde après un delai. Il faut donc réussir a placer un maximum de tâches dans la periode.

2 Structures de données

Dans un premier temps nous utilisions les structures suivantes :

Task {
    entier num //Le numero de la tache pour l'identifier
    entier delay //Le delai avant le retour sur la dexime periode
    entier cycle[2] //Le temps occupé sur la première et deuxième periode
    entier place //La place dans la periode, son début sur la première periode,
                  si la tache n'est pas placé cette valeur est à -1..
}

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 tache 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 a 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 periode.

```
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

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

Algorithm 2 AlgoLourd

Require: Tasktab, PeriodeMax
  $min \leftarrow PeriodeMax$
  $taskMin \leftarrow 0$
  $libreMin \leftarrow 0$
  for chaque Task **do**
    for chaque Task t dans Tasktab **do**
      $compteur \leftarrow 0$
      $libre \leftarrow 0$
      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**
          $compteur \leftarrow compteur + 1$
          $libre \leftarrow i$
        end if
      end for
      if compteur <= compteurMin **then**
        $compteur \leftarrow compteurMin$
        $taskMin \leftarrow t$
        $libreMin \leftarrow libre$
      end if
    end for
    $taskMin.place \leftarrow libreMin$
  end for
  return Tasktab

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**
      $gene[t] \leftarrow 0$
      Place t dans la période au premier endroit disponible
      $cptApres[] \leftarrow cptplace()$ //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

4 Conclusion

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 une sujet qui demande une bonne compréhension de l'algorithmique ce qui nous a beaucoup intéressé.

De nos jour 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 envoyés periodiquement et le temps entre l'envoie et la reception est fixe. Dans ce modèle il y a deux periodes de taille P, l'envoie du tâches est placé sur la première periode et la réception sur la seconde après un delai. Il faut donc réussir a placer un maximum de tâches dans la periode.