Compte-rendu de TP

Julien GIRAUD (11704709)

TP1

Question 1.1

- La foncion `main`
  - crée un thread qui exécutera la fonction `spawnedFunc`,
  - envoie à ce thread la valeur `196`,
  - affiche un message.
- Le `thread`
  - reçoit une valeur de type `int` dont la valeur est `196`,
  - affiche cette valeur dans un message.

Les fonctions importantes sont `send` et `receive`.

Question 1.2

La communication mise en place entre ces deux processus est

- **bidirectionnelle**,
- **asynchrone** comme on peut s'y attendre avec un programme multithread,
- mais surtout les threads utilisent de l'**attente passive**.

Question 2.1

- La fonction `main`
  - crée 10 threads dont elle mémorise les identifiants dans un tableau, ces threads exécuteront la fonction `spawnedFunc`,
  - envoie à chacun des threads fils l'identifiant de leur "thread fils suivant" pour mettre en place la topologie en anneau, en supposant que l'identifiant envoyé est le bon car le code qui récupère l'identifiant est à compléter *question 2*,
  - attend que tous les threads fils envoient un message pour indiquer qu'ils ont fini leur traitement.
- La fonction `spawnedFunc`
  - reçoit un objet de type Noeud,
  - extrait l'identifiant du thread de son "prochain voisin" à partir de cet objet,
  - envoie un message au thread parent pour indiquer qu'elle a terminé son traitement.

Les fonctions importantes sont toujours `send` et `receive` ainsi que `receiveAllFinalization` et `cast(T)` qui permet de transmettre tout type d'objet aux threads.

Question 2.3

Soit `n` le nombre de threads de notre processus, l'anneau contient `n-1` threads car celui de la fonction `main` n'y est pas.

Après plusieurs tests je constate que `32 678` est le nombre maximum de threads qui peuvent fonctionner sur un processus de ma machine, ce résultat a été obtenu par dichotomie.

La taille maximale de l'anneau sur ma machine est donc `32 677`.

Question 2.4

Chacun des `n` noeuds va exécuter `2` fois le contenu de la boucle for qui permet de compter les noeuds. Il y a donc `2n` messages échangés.

Question 2.5 - Implémentation de l'horloge scalaire

L'initialisation se fait au lancement des threads avec le code suivant.

```
int scalHorloge = 1;
```

L'incrémentation et la mise à jour de l'horloge se font avec le code suivant après chaque receive.

```
// Le receive vient du noeud parent
scalHorloge++;

// Le receive vient d'un noeud voisin, le voisin nous a donc communiqué son
horloge
scalHorloge = max(++scalHorloge, ++precScalHorloge);
```

L'envoi de l'horloge se fait à chaque fois que le noeud communique avec son voisin.

Question 2.6 - Implémentation de l'horloge vectorielle

L'initialisation se fait au lancement des threads avec le code suivant.

```
int[] vectHorloge = new int[n];
for (int i = 0; i < n; i++) {
    vectHorloge[i] = 0;
}
vectHorloge[myId] = 1;
```

L'incrémentation et la mise à jour de l'horloge se font avec le code suivant après chaque receive.

```
// Le receive vient du noeud parent
vectHorloge[myId]++;

// Le receive vient d'un noeud voisin, le voisin nous a donc communiqué son
horloge
for (int i = 0; i < precVectHorloge.length; i++) {
    if (precVectHorloge[i] > vectHorloge[i]) {
        vectHorloge[i] = precVectHorloge[i];
    }
}
vectHorloge[myId]++;
```

TP2

Question 1.2

Soit `n` le nombre de nœud dans l'anneau.
Il y a un anneau par permutation possible, on a donc à faire à une factorielle.

Si `n = 1` alors il y a `2` éléments, disons `{a, b}`. Les permutations sont `[ab, ba]`, il y a donc `2` anneaux.

Si `n = 2` alors il y a `3` éléments, disons `{a, b, c}`. Les permutations sont `[abc, acb, bac, bca, cab, cba]`, il y a donc `6` anneaux.

De façon générale il y a `(n-1)!` anneaux.

Question 2.1 - Hypothèses

- Chaque nœud a un identifiant unique et sait que les identifiants sont uniques.
- Chaque nœud connaît son voisin.
- Le nombre de nœuds dans le système est inconnu de chaque nœud.

Question 2.3

Chaque thread

- compte le nombre de messages envoyé ou reçu,
- envoie l'information via l'objet `CancelMessage`.

La fonction `receiveAllFinalization`

- effectue un *reduce* des CancelMessage (au sens MapReduce),
- transmet le résultat du *reduce* de l'exécution au `main`.

La fonction `main` se charge du traitement.

```
$ filename=ex2q1_4; dmd -of=$filename $filename.d; ./$filename

Meilleur cas : 77 messages, 3.85 message/thread
Pire cas : 127 messages, 6.35 message/thread
Nombre moyen de messages : 92, 4.629 messages/thread
```

Question 2.4

Pour obtenir le meilleur cas on suppose que

- les messages sont envoyés et traités à la suite comme en séquentiel,
- seul le noeud de plus grand identifiant se porte volontaire.

Dans ce cas l'élection de leader nécessite `n` messages soit un message par thread.

Pour obtenir le pire cas on suppose que

- les messages sont envoyés tous en même temps,
- les messages sont traités à la suite comme en séquentiel dans l'ordre de leur identifiant,
- tous les noeud se portent volontaires,
- les nœuds sont voisins dans l'ordre croissant de leur identifiant, par exemple `n = 3` donnerait `1 → 2 → 3 → 1`.

Dans ce cas l'élection de leader nécessite n^2 messages soit `n` messages par thread.

Simulation à l'appui :

```
1 → 2 → 3 → 1

Étape 1 :
1 dit à 2 qu'il se présente
2 dit à 3 qu'il se présente
3 dit à 1 qu'il se présente

Étape 2 :
1 reçoit la candidature de 3, il dit à 2 que 3 se présente
2 reçoit la candidature de 1, il dit à 3 que 2 se présente
3 reçoit la candidature de 2, il dit à 1 que 3 se présente

Étape 3 :
1 reçoit la candidature de 3, il dit à 2 que 3 se présente
2 reçoit la candidature de 3, il dit à 3 que 3 se présente
3 reçoit la candidature de 2, il dit à 1 que 3 se présente

Étape 4 :
1 reçoit la candidature de 3, 3 est leader
2 reçoit la candidature de 3, 3 est leader
3 reçoit la candidature de 3, 3 est leader
```

Question 2.5

Changements par rapport à la *question 2.4*.

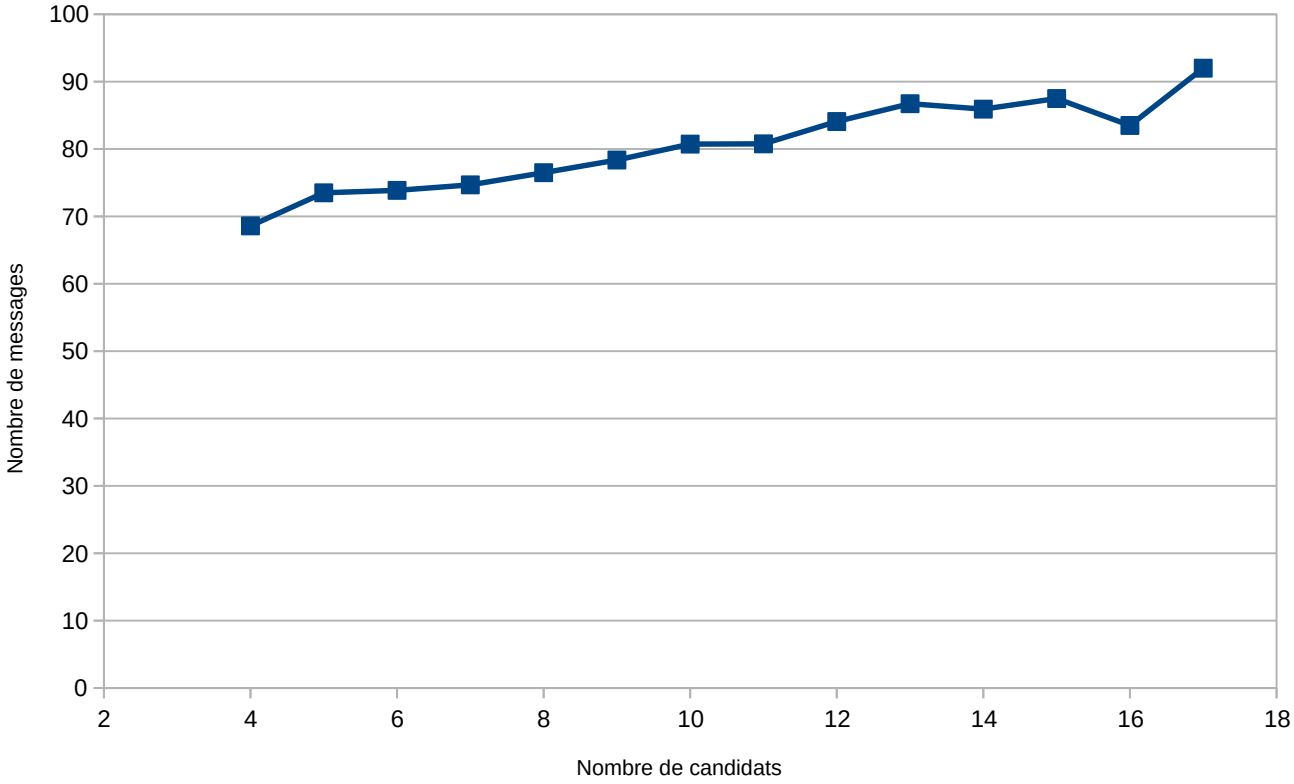
- La fonction `main` ne déclenhe plus les élections.
- Les threads initialisent un booléen `candidat` aléatoirement.
- Les threads n'exécutent le code pour se porter candidat que si `candidat` est à `true`.

Question 2.6

```
$ filename=ex2q5_6; dmd -of=$filename $filename.d; ./$filename

Candid. Avg     Min     Max
4       68.6    56      76
5       73.5    65      82
6       73.8636 61      87
7       74.6829 59      93
8       76.4906 59      111
9       78.3699 64      98
10      80.7283 64      111
11      80.7692 63      104
12      84.0833 69      114
13      86.7273 74      113
14      85.9231 70      107
15      87.5    81      95
16      83.5    82      85
17      92      92      92

Meilleur cas : 49 messages, 2.45 message/thread
Pire cas : 117 messages, 5.85 message/thread
Nombre moyen de messages : 80, 4.0165 messages/thread
Nombre moyen de candidats par tour : 9, 0.496 candidat/thread
```



Le nombre moyen de messages échangés semble croitre de façon linéaire en fonction du nombre de nœuds candidats.

Question 3.1

Se référer à la fonction `getRandomIds`.

Question 3.2

1. Élir un leader avec la méthode du TP2.
2. Le leader fait passer un compteur à tous les noeuds de façon à retrouver `n` comme au TP1, chaque noeud mémorise la valeur du compteur au moment où il l'a reçu puis l'incrémente avant de l'envoyer au noeud suivant, cette valeur servira d'index pour récupérer le nouvel identifiant.
3. Le leader initialise un nouveau tableau d'identifiants aléatoires dans `[0, n]`, il l'envoi à tous les noeuds et chaque noeud met à jour son identifiant à l'aide de l'index.