Master 1 - Informatique
Systèmes Distribués

Éric Leclercq

Révision : 18 septembre 2020

Résumé

Ce document contient l'ensemble des exercices de travaux dirigés du module de systèmes distribués. Tous les exercices ne seront pas forcement traités durant les séances de travaux dirigés. Le fascicule de TD en ligne sera mis à jour assez régulièrement ainsi que le GitHub qui contient des squelettes de programme (https://github.com/EricLeclercq).

Table des matières

1 Notion Générales

Exercice 1. Lois d'Amdahl et de Gustafson

1. Représenter sur un graphique l'évolution du speedup $S(n)$ en fonction du nombre de processeurs pour différentes valeurs de f (en pourcentage : 0,5,10, 15,20).
2. Représenter sur un graphique l'évolution du scaling speedup $S_s(n)$ en fonction de la portion séquentielle (au moins pour les valeurs 0.2, 0.4, 0.6, 1.0) pour $n = 8$, $n = 16$, $n = 32$, $n = 64$.
3. Pour un programme donné, comment exploiter ces deux informations pour avoir une idée de sa qualité parallèle ?

Exercice 2. Architecture et limite de la lumière

On considère un système comportant 1 000 unité de calcul chacune ayant une puissance de 20 MIPS (20 millions d'instructions par seconde).

— Pour un processeur multicœurs quelle est la durée d'exécution d'une instruction ?
— On suppose que le processeur mesure 1,5cm, quelle est la durée pour parcourir cette distance à la vitesse de la lumière ?
— Qu'en est il pour 1 000 ordinateurs personnels actuels ayant une puissance de 6 000 GigaFLOPS (Floatting Point Operations Per Second) ? Consulter ensuite le top 500 [1], regarder la liste des super calculateurs, leur puissance, le type de processeur ainsi que le pays.

Exercice 3. Protocoles et transparence

1. Faire un schéma du fonctionnement du protocole HTTP incluant la partie serveur, les ressources nécessaires et la partie client.
2. Étudier les différents types de transparence par rapport au protocole.
3. Reprendre les deux questions ci-dessus pour le protocole NFS.

Exercice 4. Prise en charge des clients dans le modèle Client serveur

Rappeler les 3 stratégies vues en cours pour prendre en charge les demandes des clients dans une le modèle client serveur, pour chacune, écrire en pseudo code l'algorithme qui permet la communication du client et du serveur.

2 Modèles d'interactions

Exercice 5. Pattern bag of tasks et équilibrage de charge

1. Avec le langage Java créer une méthode qui renvoie `true` si un nombre passé en paramètre est premier, `false` sinon. Utiliser une algorithme qui teste les diviseurs jusqu'à la racine carrée du nombre donné. Énumérer les nombres premiers jusqu'à N, déterminer N pour avoir une exécution séquentielle de plusieurs minutes.
2. Utiliser plusieurs threads pour répartir le travail en appliquant le pattern *bag of task* : chaque thread vient chercher une tâche (un nombre à tester) dans un objet de la classe `BagOfTasks` en invoquant la méthode `getNext()`. Lorsqu'un thread a terminé le test de primalité d'une valeur, il va en rechercher une autre. Le distributeur

1. http://top500.org

de tâche peut retourner `-1` lorsque le travail global est terminé, c'est-à-dire lorsque les N nombres ont été testés. Cette méthode à l'avantage de répartir équitablement le travail entre tous les threads.

(a) Réaliser le programme.

(b) Mesurer le temps d'exécution pour un thread.

(c) En fonction du type de processeur du nombre de cœurs réels, lancer le programme avec n thread, mesurer le temps d'exécution et évaluer le speedup.

(d) Mesurer le speedup avec $n + 1$ threads (si n est le nombre de cœurs réels du processeur).

3. Généraliser le pattern *bag of task* pour l'utiliser avec plusieurs machines multi-cœurs [2]. Dans ce programme, utilisez des socket TCP ou UDP et ne pas utiliser Java RMI.

(a) Réaliser le programme avec des sockets UDP et des sockets TCP.

(b) Le tester avec une machine cliente et un serveur *bag of task* pour mesurer le temps d'éxecution et en déduire le coût induit par les communications et ainsi estimer la fraction de code non parallélisable.

(c) En utilisant ssh lancer le programme sur plusieurs machines. Évaluer le speedup réel et le speedup théorique. Évaluer à nouveau le speedup. Que peut-on dire sur l'impact du temps d'acheminement des valeurs entre les différentes machines.

(d) Comparer l'implémentation UDP et TCP.

(e) Quel sont les inconvénients de l'implémentation en TCP ? Utiliser les sélecteurs d'entrée-sortie pour améliorer cette version.

Exercice 6. Pattern bag of tasks généralisé

Cet exercice est la suite du précédent. Au lieux de distribuer une valeur, le *bag of task* distribue un objet qui possède une méthode `run()`. Ainsi, des clients peuvent soumettre n'importe qu'elle tâche dans le *bag of task* et un certain nombre de *workers* viennent chercher des objets tâches et exécutent la méthode `run()`.

1. Quelles classes sont nécessaires ?
2. Quel mécanisme de Java est essentiel pour implanter un *bag of task* générique ?
3. Comment retourner les valeurs à l'issue de l'exécution du `run()`, les transmettre au client ?
4. Implanter le programme

3 Petits outils utiles pour les TD et les TP

3.1 Observer les exécutions, mesurer le temps

Pour observer les exécutions sur une machine vous pouvez utiliser la commande `htop`. La figure 1 présente les informations affichées par la commande `htop`. Les premières lignes donnent l'activité des cœurs du CPU, les barres vertes corresponde à une exécution de programme utilisateur, les barres rouge à du code exécuté en mode noyau. Les trois lignes de droite donnent les information sur le nombre de processus et de thread et le nombre de processus s'exécutant dans les cœurs. La ligne suivante donne la charge moyenne à Pour

2. en distribué, ce pattern se nomme Manager/Worker

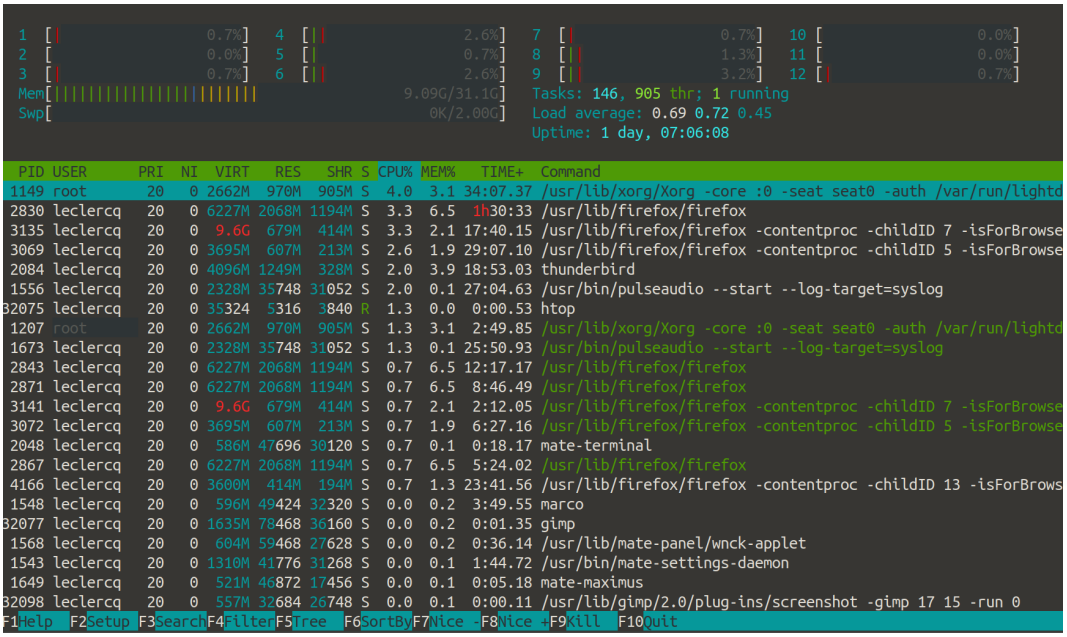

FIGURE 1 – Informations affichées par htop

pouvoir interpréter correctement les informations sur le nombre de cœurs, il faut connaître les spécification du processeur. Avec `cat /proc/cpuinfo` vous obtiendrez la description du ou des processeurs installés, utiliser ensuite la référence sur un moteur de recherche pour accéder à la fiche technique du site du constructeur.

Pour mesurer le temps d'exécution d'un programme utiliser la commande `time` de la façon suivante : `time commande` ou `time java monprog`

3.2 Se connecter avec SSH en utilisant les clés

Avec ssh et son implémentation OpenSSH, il est possible de se connecter sur des machines en utilisant la clé publique déposée dans le répertoire `.ssh` sous votre home. La clé privé permet de prouver votre identité et donc d'éviter à avoir à saisir un mot de passe. De plus ssh permet d'exécuter directement une commande à la connexion, il suffit de la passer en paramètre. Il est donc possible et plutôt facile de lancer une commande sur un ensemble de machines et ainsi d'administrer un cluster ou de lancer des programmes qui vont s'exécuter en parallèle et coopérer.

ENCADRÉ 1 – Lancer des exécutions sur plusieurs machines

Créer les clés SSH sur une de machines :

```
ssh-keygen -t ras
# ne pas mettre de passphrase
# les clés sont générées dans le répertoire .ssh de votre home
# la clé publique a l'extension .pub
cat id_ras.pub > authorized_keys
# le fichier authorized_keys contient la liste des clés
# autorisées a se connecter sans password
# tester avec la commande suite :
ssh MI104-04
# à la première connexion, il faut valider la source
```

Pour lancer des exécutions sur différentes machines on utilise un script shell qui exploite un fichier texte contenant l'ensemble des machines sur lesquelles on veut effectuer la commande passée en paramètre

```
for i in `cat myhost.txt`
do
 ssh $i -c "ma␣commande"
done
```

Le fichier `myhost.txt` contenant la liste des machines est de la forme :

```
MI104-01
MI104-02
MI104-03
```