Conception des systèmes répartis

Mastère RTSA
Durée : 2 heures — Documents autorisés : Précis et notes de cours

Septembre 2005

1 Causalité et flux

Trois usagers, localisés sur 3 machines distinctes, dialoguent à distance par des flux audio. Lorsqu'un usager parle, son flux audio est transmis (diffusé) aux 2 autres participants. Le chronogramme de la figure 1 montre le déroulement d'un échange entre les trois interlocuteurs. Lors de cet échange, U_3 se contente d'écouter. Les événements de début (respectivement de fin) de flux sont notés d_i (respectivement f_i).

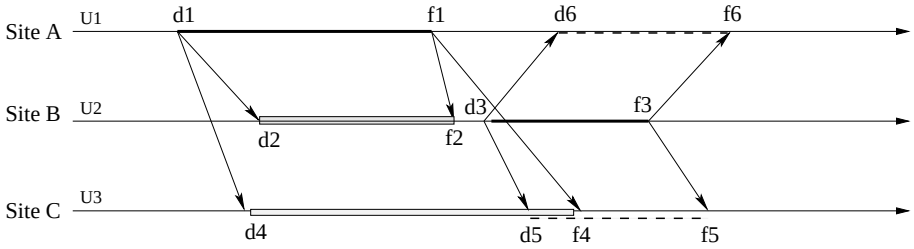

FIG. 1 – Echange de flux audio

Questions (2 points par question)

1. Donner un exemple de relation causale (notée $\prec$) entre deux événements de début d'émission de flux distincts, un exemple entre deux événements de fin de réception de flux distincts.
2. Pour U_3, le flux provenant de U_2 vient interrompre le flux provenant de U_1. Quelle relation causale aurait due être garantie lors de la réception sur C pour que cette anomalie ne se produise pas ?
3. Un protocole garantissant l'ordre causal aurait pu éviter cette anomalie. Expliquer pourquoi ?
4. Quel protocole pourrait-on proposer pour éviter que deux usagers parlent en même temps ?

2 Contrôle de charge par agent mobile

On envisage de réaliser le contrôle de la charge d'un ensemble de sites (serveurs) grâce à un agent mobile se déplaçant aléatoirement de sites en sites. On souhaite par exemple garantir une bonne répartition de données diverses ou objets (fichiers MP3, vidéo, etc) enregistrés sur ces serveurs.

Hypothèses générales

– Chaque serveur gère la création de nouveaux objets, assure leur consultation et leur destruction. On ne se préoccupe pas ici de la localisation des objets à partir d'un identifiant. On suppose que ces objets peuvent être déplacés pour des raisons d'équilibrage de charge sans pour autant remettre en cause la possibilité pour les clients de localiser et d'accéder à ces objets.

- On suppose qu'un serveur ne connaît que quelques serveurs voisins immédiats avec lesquels il peut communiquer. Autrement dit, chaque serveur possède un *voisinage* composé des serveurs avec lesquels il peut échanger des données. Si le serveur S_j appartient au voisinage du serveur S_i, les relations $voisin(i,j)$ et $voisin(j,i)$ sont vraies (symétrie)[1].
- On suppose que cette relation de voisinage définit un réseau de communication connexe. De cette façon, l'agent mobile qui se déplace de serveur en serveur peut atteindre tous les serveurs.
- Pour simplifier l'algorithme d'équilibrage, on suppose que toutes les objets sont équivalents en terme de charge : tout fichier représente un charge unitaire.
- Lorsque l'agent mobile visite un serveur S, il peut l'interroger pour connaître la charge courante c du serveur S visité et décider ou non de demander au serveur de transmettre un ou plusieurs objets vers un ou plusieurs voisins. On suppose que les serveurs acceptent toujours d'exécuter de telles requêtes issues de l'agent visiteur.

Première stratégie

On utilise donc un seul agent mobile qui migre de serveur en serveur en choisissant aléatoirement le serveur suivant qu'il visitera parmi les serveurs du voisinage courant. Si le nombre global d'objets reste constant pendant suffisamment longtemps, l'objectif de l'agent est de parvenir à assurer que l'état global des serveurs est stable et vérifie :

$$\textbf{stable} \;\; \forall i,j \;:\; voisins(i,j) \;\Rightarrow\; |c_i - c_j| \leq 1 \tag{1}$$

Pour assurer la convergence vers l'état stable (1), on propose le comportement suivant de l'agent sur chaque site qu'il visite :

```
agent Visiteur() {
  int cpred  = 0; serveur spred;  // contexte transporté de serveur en serveur
  void visiter() {
      int diff = 0;
     int cloc = get_load();
      if (cpred < cloc + 1) { // envoyer des objets vers le serveur précédent
           diff = (cloc - cpred) / 2 ;
           ejecter(diff, spred);
     }
     cpred = cloc ;
  }
  void run() {
      cpred = get_load();
      loop {  spred = get_ici(); migrer();  visiter();  }
  }
}
```

Les appels en italiques représentent des appels auprès du serveur visité par l'agent :
- *get_ici()* retourne le nom du serveur visité ;
- *get_charge()* retourne la charge courante du serveur visité ;
- *ejecter(int p,serveur s)* demande au serveur visité de déplacer p objets sur le serveur s.
- *migrer()* provoque la migration de l'agent appelant vers un serveur voisin choisi aléatoirement.

[1] Attention, la relation n'est pas transitive $voisin(i,j) \;\wedge\; voisin(j,k)$ ne garantit pas $voisin(i,k)$.

Questions (2 points par question)

5. Vérifier en expliquant pourquoi que l'état stable (1) est bien atteint grâce au comportement de cet agent ;
6. Montrer par un contre exemple minimal que le comportement de l'agent ne garantit pas que l'état stable final vérifie la propriété plus forte :

$$\textbf{stable} \ \ \forall i, j \ : \ |c_i - c_j| \leq 1$$

7. Lorsqu'un état stable est atteint, les visites de l'agent n'ont aucun effet. Le retrait d'un serveur est-il un événement qui entraîne la disparition de l'état stable ou reste-t-on dans un état stable ? Autrement dit, l'agent continue-t-il à exécuter des visites sans effet ou doit-il faire à nouveau converger vers un nouvel état stable ?

Seconde stratégie

L'agent calcule une moyenne approchée au fur et à mesure de ses visites des serveurs. Le calcul de cette moyenne correspond la la formule dans laquelle a_i désigne le nombre de voisins du site i et p désigne le nombre de visites exécutées par l'agent depuis sa création[2] :

$$m = \sum_{i=1}^{i=p} \frac{c_i}{a_i} \ / \ \sum_{i=1}^{i=p} \frac{1}{a_i}$$

Cette fois-ci, on veut que l'agent fasse converger le système vers l'état stable suivant ;

$$\textbf{stable} \ \ \forall i \ : \ (c_i - m) \leq 1 \tag{2}$$

Questions (2 points par question)

8. Programmer le comportement de l'agent pour qu'il calcule la valeur de la moyenne au fur et à mesure de ses visites selon la formule proposée.
9. Compléter la programmation du comportement de l'agent pour que le système atteigne finalement l'état stable (2).
10. En supposant que l'estimation de la moyenne m est correcte, montrer que l'état stable (2) ne garantit pas :

$$\textbf{stable} \ \ \forall i \ : \ 0 \leq (c_i - m) \leq 1$$

[2]On admettra la validité de cette formule pour estimer la moyenne