DEA d'Informatique - Systèmes et Communication. Année 1999-2000
INPG-UJF Grenoble

Algorithmique et Techniques de Base des Systèmes Répartis
Examen

13 décembre 1999
Durée: 3 heures, documents autorisés

---

*L'examen comporte 4 problèmes indépendants. Lire l'ensemble des énoncés avant de commencer à répondre. La longueur des énoncés n'est pas signe de difficulté mais elle est nécessaire à une bonne spécification des problèmes. La **clarté**, la **précision** et la **concision** des réponses, ainsi que leur **présentation matérielle**, seront des éléments importants d'appréciation.*

Problème 1 (5 points). Les questions sont indépendantes ; néanmoins la réflexion sur la question 2 peut aider à aborder la question 3.

Si chaque site d'un système réparti dispose d'une horloge donnant un temps physique universel, et que ces horloges sont parfaitement synchronisées, on dispose d'un moyen simple d'ordonner les événements de ce système, puisqu'il suffit de les estampiller par l'heure physique.

On suppose maintenant que l'on dispose de telles horloges physiques (soit H_i l'horloge du site i), mais que la synchronisation entre les sites n'est qu'approximative. Plus précisément, on suppose que la dérive maximale entre deux sites quelconques (différence des heures données par les horloges physiques) est toujours inférieure à une valeur connue d. D'autre part, on suppose que l'on connaît une borne supérieure max et aussi une borne inférieure min pour le temps de transmission d'un message entre deux sites quelconques du système.

Question 1. À quelle condition les horloges H_i possèdent-elles la propriété faible de validité ($e \rightarrow e' \Rightarrow H(e) < H(e')$, où H est la date donnée par la valeur locale de H_i) ? Justifier votre réponse.

Question 2. Dans un système comportant n sites, on peut caractériser la dépendance causale au moyen d'horloges vectorielles V_i à n composants (autrement dit, ces horloges ont la propriété forte de validité : $e \rightarrow e' \Leftrightarrow V(e) < V(e')$, où V est la date donnée par la valeur locale de V_i). Montrer, par un raisonnement informel, qu'il n'est pas possible, dans le cas général, de détecter la dépendance causale dans un système à n sites avec une horloge comportant moins de n composants.

Question 3. On considère maintenant un système comportant m serveurs et n clients, avec $m << n$. On sait que les clients ne communiquent jamais entre eux ; les seules communications possibles sont entre un client et un serveur, ou entre deux serveurs. En tenant compte de cette restriction sur les communications, est-il possible d'améliorer la méthode de datation par horloges vectorielles pour la rendre plus efficace, en permettant toujours la détection de la dépendance causale ?

Problème 2 (3 points). Les deux questions sont indépendantes

Question 1. Si l'on dispose d'horloges physiques parfaitement synchronisées et que l'on connaît une borne supérieure D pour le temps de transmission d'un message, l'algorithme de Chandy-Lamport pour l'enregistrement d'un état global prend une forme très simple : à un instant t, le processus initiateur ordonne à tout processus d'enregistrer son état à un moment $t_e > t + D$, en diffusant un message estampillé par t. Comment est enregistré dans ce cas l'état des canaux de communication ?

Question 2. On suppose maintenant que le système est asynchrone. On considère d'abord deux processus qui doivent enregistrer leur état en vue d'une reprise. On suppose que p_1 envoie des messages à p_2, et que p_2 n'envoie jamais de messages à p_1. Montrer que, dans ce cas, on peut rendre indépendants les instants d'enregistrement de p_1 et p_2, tout en garantissant de toujours pouvoir reconstituer un état global cohérent. Plus précisément, si on numérote par ordre chronologique les enregistrements des états de p_1 et p_2, donner une caractérisation simple d'un état global cohérent en fonction de ces numéros d'enregistrement. Comment peut-on généraliser cette propriété au cas de n processus ?

Problème 3 (5 points)

Les questions sont indépendantes.

Dans ce problème, on examine quelques aspects de la tolérance aux fautes. Les hypothèses communes à l'ensemble des questions sont les suivantes : le système de communication est fiable (tout message arrive à destination) ; les serveurs peuvent avoir des défaillances, mais il s'agit de pannes franches : soit le serveur fonctionne, et donne un résultat correct, soit il est en panne, et ne fait rien.

Question 1. On considère les deux applications suivantes :

- un serveur de calcul ; on lui soumet un ensemble de données, sur lesquelles il réalise un calcul (usage intensif du processeur) pour fournir des résultats.
- un serveur de données ; on lui soumet une requête pour rechercher des informations parmi celles qu'il conserve, le résultat étant l'information trouvée (ou l'indication que l'information est introuvable).

Expliquer en détail pour chacune de ces applications comment la rendre tolérante aux fautes en utilisant les deux techniques vues en cours (serveur primaire - serveur de secours, et duplication active). Indiquer dans chaque cas les avantages et inconvénients de chaque méthode, et en déduire la technique la plus appropriée pour chacune des applications.

Question 2. On veut réaliser un serveur fiable en utilisant la technique de la duplication active. Pour cela, on met en place deux serveurs identiques. Chaque client envoie ses requêtes aux deux serveurs. Il attend la première réponse, et ignore l'autre.

Existe-t-il des contraintes sur l'ordre de réception, sur les deux serveurs, des requêtes émises par les clients ? votre réponse dépend-elle du fait que l'exécution des requêtes modifie ou non l'état du serveur ?

Lorsqu'un des serveurs tombe en panne, quelles opérations doit-on effectuer lors de sa remise en route ?

Question 3. On veut réaliser un serveur fiable en utilisant une variante de la duplication active, réalisée comme suit : un client envoie sa requête au serveur 1, qui attribue un numéro de séquence à cette requête, et la transmet au serveur 2. Chaque serveur exécute les requêtes dans l'ordre des numéros de séquence et renvoie les réponses aux clients. Chaque client conserve la première réponse et ignore l'autre. Les numéros de séquence sont les entiers successifs : 1, 2, 3, etc.

Expliquer l'intérêt de cette technique.

Expliquer quelles sont les opérations à réaliser (côté clients et côté serveurs) en cas de panne d'un des deux serveurs (examiner séparément la panne du serveur 1 et la panne du serveur 2).

Problème 4 (7 points)

N.B. Rédiger la solution de ce problème sur une feuille séparée

Ce problème porte sur la réalisation d'un service réparti de base de données, qui doit permettre de consulter et modifier des données à distance. Les clients de ce service utilisent les primitives :

Valeur Consulter (*Clé c*) : retourne la valeur (de type *Valeur*) associée à la clé *c*,

void Modifier (*Clé c, Valeur v*) : associe la valeur *v* à la clé *c*.

Ce service est initialement réalisé comme un simple système client-serveur, dont l'algorithme est fourni ci-après. Dans la question 1, il est demandé de modifier ce système pour le rendre plus efficace au moyen de caches. Dans la question 2, il est demandé de le faire évoluer vers un système d'agents mobiles.

Pour réaliser ce service, on dispose de deux fonctions d'envoi et de réception de messages :

void EnvoyerMsg (*Port p, Msg m*) : envoie un message *m* à un port *p*.

Msg RecevoirMsg (*Port p*) : recoit un message sur un port *p* (fonction bloquante).

On suppose que ces opérations sont fiables (tout message parvient intact à destination) et que les identifications des ports sont globales et uniques.

Réalisation initiale

Le service de base de données est initialement réalisé comme un système en mode client-serveur pur : les données sont gérées sur une machine (le serveur de la base de données) et toute interaction entre les machines clientes et le serveur est réalisée par appel à distance. Pour les communications, chaque client a un port *portclient* (chaque client connaît le sien) pour recevoir les réponses et le serveur a un port *portserveur* pour recevoir les requêtes (connu de tous les clients).

On suppose que la base de données sur le serveur est utilisable localement avec les fonctions :

Valeur ConsulterLocalement (*Clé c*) : retourne la valeur associée à la clé *c*,

void ModifierLocalement (*Clé c, Valeur v*) : associe la valeur *v* à la clé *c*.

Les algorithmes du programme client (qui fournit les primitives *Consulter* et *Modifier*) et du programme serveur, qui traite les requêtes dans cette configuration, sont donnés ci-dessous.

Programme client :

```
Valeur Consulter (Clé c) {
    m.fonction = consulter ;
    m.clé = c ;
    m.port = portclient ;
    EnvoyerMsg (portserveur, m) ;
    m = RecevoirMsg (portclient) ;
    return m.valeur ;
}
void Modifier (Clé c, Valeur v) {
    m.fonction = modifier ;
    m.clé = c ;
    m.valeur = v ;
    EnvoyerMsg (portserveur, m) ;
}
```

Programme serveur :

```
while (TRUE) {
        m = RecevoirMsg (portserveur) ;
        switch (m.fonction) {
                case consulter :
                    m1.valeur =
                      ConsulterLocalement (m.clé) ;
                    EnvoyerMsg (m.port, m1) ;
                    break ;
        case modifier :
                    ModifierLocalement (m.clé,
                      m.valeur) ;
                    break ;
        }
    }
```

Dans la suite, pour l'écriture des algorithmes demandés, vous n'avez pas à déclarer vos variables et vous noterez simplement, comme ci-dessus, *msg.x* un champ *x* géré dans un message *msg*, et de même pour toute structure de donnée (y compris les ports).

Question 1

Dans une seconde étape, on désire permettre aux machines clientes de mettre localement des données en cache afin d'améliorer les temps d'accès. On se limite à mettre en cache ces données en lecture seulement. Chaque client gère une table utilisable avec les mêmes fonctions que sur le serveur : *ConsulterLocalement* et *ModifierLocalement* (*ConsulterLocalement* retourne *null* si la donnée n'est pas dans la table). Ces fonctions sont déjà synchronisées.

Pour assurer la cohérence des données en cache, il faut invalider au bon moment les valeurs dans les tables des clients. Pour ce faire, sur chaque machine cliente, un processus exécute une fonction appelée *Cohérence* qui est chargée de recevoir les messages d'invalidation sur un port appelé *portcohérence* (il y en a un par client). On invalide une valeur dans une table locale (cache) en l'affectant à *null* (avec *ModifierLocalement*).

Pour pouvoir invalider les clients, le serveur gère une liste de clients par clé de la base que l'on notera *listeclients*[*clé*]. On peut utiliser cette liste des deux façons suivantes :

void Add (*Liste l, Elément e*) : fonction qui ajoute un élément dans la liste.

foreach *e* ***in*** *l* { ... } : construction qui exécute le code entre { } pour tous les éléments *e* de la liste *l*.

On suppose que les invalidations sont asynchrones : le serveur n'attend pas de réponse aux messages d'invalidation envoyés aux clients.

Modifier les algorithmes précédents (et ajouter la fonction *Cohérence*) afin qu'ils gèrent des données en cache en lecture (de façon cohérente).

Question 2

Dans une dernière étape, on désire permettre aux machines clientes d'envoyer un agent mobile sur la machine serveur pour réaliser une liste d'opérations sur le serveur. La liste est manipulable avec les opérations *add* et ***foreach*** précédentes et chaque entrée indique le nom de la fonction à appeler (*consulter* ou *modifier*) ainsi que les valeurs des paramètres.

Le système à agents mobiles réalise la migration faible et fournit une primitive *move* (*port*) qui déplace l'agent vers la machine désignée par le port de communication. Le point d'entrée de l'agent est une procédure *main*(). On suppose que toutes les variables utilisées sont locales à la procédure courante, mise à part une zone mémoire appelée *Global* dans laquelle on peut gérer une variable *x* (*Global.x*) si nécessaire. La liste des opérations à exécuter sur le serveur est une donnée de *Global*.

Ecrire le programme d'un agent mobile qui traite de cette façon une liste d'opérations sur la base de données. Quel est l'intérêt de ce mode de fonctionnement ?