Université Joseph Fourier
DESS Génie Informatique - année 2000-2001

Schéma de corrigé pour l'examen 2000

Problème 1.

Problème 2.

Question 1.

Pour préserver l'invariant indiqué, il faut :

a) qu'une écriture soit propagée à l'ensemble des serveurs corrects (non en panne).
b) que quand un serveur en panne est remis en service, qu'il retrouve l'état résultant de toutes les écritures qui ont eu lieu pendant qu'il était en panne (et qu'aucune opération ne puisse être faite sur lui tant que cet état n'a pas été restauré).

La propriété a) exige que l'opération d'écriture multiple (réalisée par *exec*) ait la propriété d'atomicité (au sens des transactions) : une opération *exec* doit être réalisée sur tous les serveurs corrects de la liste, ou sur aucun de ces serveurs. La propriété n'est pas exigée dans le sens de la lecture puisque si l'invariant est respecté, alors il suffit de lire sur n'importe quel serveur ; néanmoins il est préférable de la respecter aussi si les pannes sont très fréquentes car en lisant tous les serveurs on augmente la probabilité d'en trouver un qui n'est pas en panne. Si on veut en outre réaliser des opérations concurrentes, la primitive *exec* doit aussi assurer la non-interférence (deux opérations concurrentes doivent être sérialisées, dans un ordre quelconque).

L'opération de lecture peut se se dérouler de plusieurs façons ;

si on privilégie la rapidité, et si les pannes sont peu fréquentes, alors on essaie de lire un seul serveur

liste(S) = un serveur quelconque (par ex. le plus proche)
exec(liste(S),x,nil,liste(res)) // liste(res) comporte une seule valeur res
si res≠erreur, res est le résultat cherché ; sinon, il faut recomencer avec un autre serveur.

Si la probabilité de panne n'est pas négligeable, on peut choisir de lire plusieurs serveurs ou tous.

liste(S) = les serveurs choisis
exec(liste(S),x,nil,liste(res))
si une valeur de liste(res) ≠ erreur, c'est le résultat cherché ;
sinon (et si la liste comportait tous les serveurs), échec – recommencer plus tard.

L'opération d'écriture s'exécute ainsi :

liste(S) = tous les serveurs
exec(liste(S),x,v,liste(res))
si au moins un res de la liste ≠ erreur, l'opération a réussi ;
sinon elle a échoué, recommencer plus tard

Question 2.

a) Il est possible en effet d'avoir des numéros différents. Supposons 3 serveurs S1, S3, S3, avec r=2, w=2. On écrit sur les serveurs S1 et S2, mais non sur S3. Plus tard, on lit sur S1 et S3. Les conditions sont bien respectées sur r et w. Le numéro de version de S1 est plus récent que celui de S3, il est donc choisi, et la lecture donne un résultat correct. Le système fonctionne tant qu'il n'a pas plus d'un serveur en panne.
b) Si r+w>N, alors l'intersection d'un ensemble de r serveurs et d'un ensemble de w serveurs n'est jamais vide. Donc il y a toujours au moins un serveur qui contient les données à jour : c'est celui qui a le numéro de version le plus récent. Voir le cas de a).
c) On modifie les algorithmes de la question en assurant que la liste(S) contienne r serveurs corrects dans le cas d'une lecture, et w dans le cas d'une écriture. La primitive *exec* doit avoir des propriétés transactionnelles, c'est-à-dire qu'il faut pouvoir revenir en arrière (*abort*) si on trouve moins de w serveurs corrects dans le cas d'une écriture alors qu'on a commencé à écrire : dans ce cas on revient donc dans l'état initial, et l'écriture a échoué. Idem si on trouve moins de r serveurs corrects dans le cas d'une lecture, sauf qu'ici on n'a pas besoin d'*abort* puisqu'on n'a rien modifié.
d) Ce protocole évite d'avoir à écrire sur tous les serveurs. On peut pondérer vitesse et sûrété par divers choix de r et de w.
e) r=1, w=N correspond au choix (écrire sur tous, lire sur 1). C'est contraignant car il faut que tous les serveurs soient en état de marche pour pouvoir écrire, mais à l'inverse la lecture est rapide. C'est un bon choix s'il y a peu d'écritures. A l'inverse, r=N, w=1 oblige de lire sur tous mais les écritures sont rapides. Utile si les écrituress sont fréquentes et les pannes peu probables.

Question 3. Le serveur qui se réinsère doit reconstituer son état, comme il a été dit en 1. Dans le cas de la question 1, il lui suffit de lire un serveur en état de marche ; dans le cas de la question 2, il doit lire r serveurs et prendre le résultat le plus récent.

Question 4. La solution « pondérée » revient à donner un poids plus grand à certains serveurs considérés comme importants. On peut par exemple imposer qu'une écriture ne puisse avoir lieu que si un certain serveur est en état de marche, en donnant à ce serveur un poids tel que ce serveur doit obligatoirement être en marche pour atteindre la valeur fixée pour w.