TD Tolérance aux fautes
INF722

auteur : Thomas Robert

1 Rapide évaluation de vos connaissances

Voici un QCM dont l'objectif est de vérifier le niveau de votre maîtrise du vocabulaire du domaine. Indiquez la ou les réponses correctes pour chaque question qui suit.

Q.1. **QCM** ($2\frac{1}{2}$ points)

(a) Comment détecte-t-on la défaillance d'une réplique dans le cas de la réplication passive

1. En réalisant un vote sur les résultats produits par les répliques
2. En armant un timer et en vérifiant la réception de la sauvegarde d'état envoyée par le primaire
3. En utilisant, un test de vraisemblance sur la réplique primaire

Réponse : 2.

(b) Une implémentation du principe de recovery blocks repose sur certains des principes/mécanismes suivants, lesquels :

1. le recouvrement avant
2. le N version programming
3. la sauvegarde d'état
4. un mécanisme de vote pour masquer les erreurs

Réponse : 2.3. 1. accepté aussi

(c) On suppose que l'on utilise une bibliothèque dont on ne dispose pas du code source. A l'exécution, une instruction du code de la bibliothèque engendre de manière systématique une exception qui déclenche le signal SIGSEGV. On souhaiterait ajouter un mécanisme au projet pour tolérer cette situation et garantir un service minimum. Quel mécanisme n'améliorera en rien le fonctionnement de l'application ?

1. recouvrement avant
2. recouvrement arrière
3. N version programming

Réponse : 2.

(d) Quel mécanisme de tolérance aux fautes n'augmente pas significativement le temps de réponse de la fonction

1. réplication passive
2. recouvrement arrière

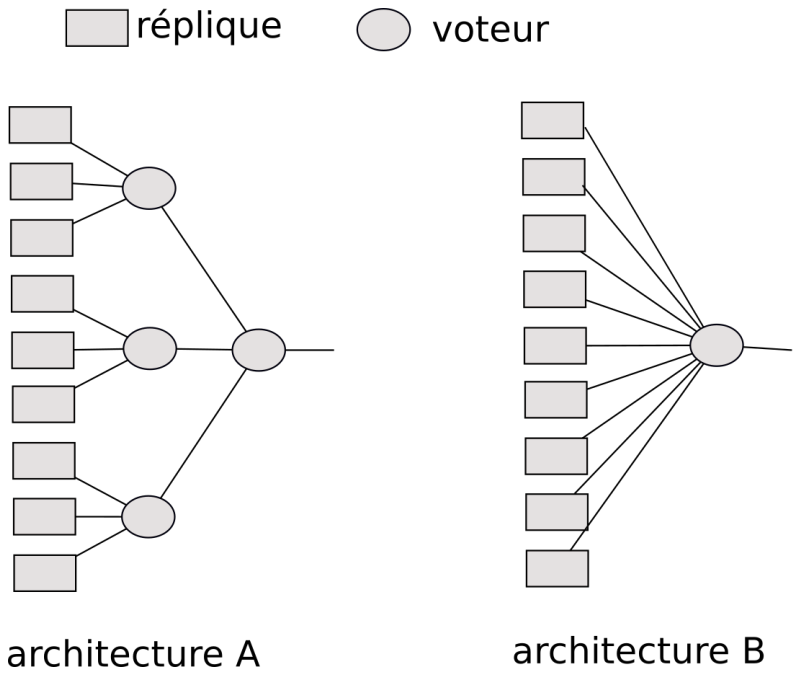

Figure 1: Architectures possibles

3. réplication active
4. recovery block

Réponse : 3.

(e) Le masquage d'erreur consiste à écraser l'état erroné par une valeur par défaut

1. vrai
2. faux

Réponse : 2.

2 Exercices Architecture complexes

Un ingénieur souhaite utiliser un mécanisme de réplication active reposant sur 9 répliques pour améliorer la fiabilité de composants peu fiables. Il doit décider si il utilise l'architecture A ou B (cf figure 1). Dans un premier temps on considère que le composant "voteur" stocke les réponses de toutes les répliques qui lui sont connectées avant d'émettre son résultat. Le résultat émis correspond à la valeur majoritairement reçue (ou la première valeur reçue si elles ont toutes la même fréquence, e.g. 1 par exemple).

Q.2. **Architecture A** (1 point)

Combien de défaillances en valeur peut-on tolérer dans l'architecture A dans le pire cas ?

Réponse : Au maximum 3 car avec 4 défaillances réparties comme suit : 2 sur les répliques du premier voteur et 2 sur les répliques du deuxième voteur, l'architecture produirait une valeur incorrecte. En effet dans ce cas des défaillances en valeur identiques entraînent un vote majoritaire en faveur d'une valeur incorrecte (et minoritaire par rapport aux résultats des différentes répliques exécutées).

Q.3. **Architecture B** (1 point)
Combien de défaillance en valeur peut-on tolérer dans l'architecture B dans le pire cas ?

Réponse : 4 car dans ce cas elles restent minoritaires lors du vote global ....

3 Réplication Active avec multiples fautes

Nous considérons le cas où nous avons une architecture implémentant la stratégie de réplication active à 5 répliques avec un voteur ne pouvant pas être affecté par des fautes.

Chaque réplique peut être affecté soit par des défaillances par omissions, ou des défaillances byzantines. On suppose que pour chaque sollicitation on sache détecter une omission (i.e. un processus correct ne peut pas être déclaré défaillant car il est trop lent).

Supposons que l'on modélise l'état de chaque réplique r_j sur une séquence de requêtes $(req_i)_{0<i\leq k}$ par une séquence de variables $(r_j(i))_{0<i\leq k}$. Chaque variable peut prendre trois valeurs différentes $\{vrai, faux, abs\}$ correspondant respectivement à la production d'un la valeur "vrai" en sortie, "faux" en sortie, ou à l'absence de valeur produite. On part du principe que la défaillance byzantine est permanente par nature, alors que la défaillance par omission est transitoire. Nous supposerons que le voteur délivre une réponse dès qu'il est capable d'identifier une unique valeur majoritaire parmi les valeurs reçues.

Q.4. **Voteur simple** (1 point)
Définissez la formule caractérisant les états où un voteur simple ne peut pas prendre de décision

Réponse : Deux réponses étaient possibles : l'énumération des cas ou une formule du premier ordre. Voici l'énumération des cas décrite via 2 sous formules F, et G. La formule complète étant $F \vee G$. F représente tous les états avec exactement une valeur absente et

G avec 3 ou 5.

$$
\begin{array}{rl}
 & ((r_1(i) = abs) \wedge (r_2(i) = r_3) \wedge \neg(r_2(i) = r_4(i)) \wedge \\
 & (r_4(i) = r_5(i)) \wedge \neg(r_2(i) = abs) \wedge \neg(r_4(i) = abs)) \\
\vee & ((r_1(i) = abs) \wedge (r_2(i) = r_4) \wedge \neg(r_2(i) = r_3(i)) \wedge \\
 & (r_3(i) = r_5(i)) \wedge \neg(r_2(i) = abs) \wedge \neg(r_3(i) = abs)) \\
\vee & ((r_1(i) = abs) \wedge (r_2(i) = r_5) \wedge \neg(r_2(i) = r_3(i)) \wedge \\
 & (r_3(i) = r_4(i)) \wedge \neg(r_2(i) = abs) \wedge \neg(r_3(i) = abs)) \\
\vee & ((r_2(i) = abs) \wedge (r_1(i) = r_3) \wedge \neg(r_1(i) = r_4(i)) \wedge \\
 & (r_4(i) = r_5(i)) \wedge \neg(r_1(i) = abs) \wedge \neg(r_4(i) = abs)) \\
\vee & ((r_2(i) = abs) \wedge (r_1(i) = r_4) \wedge \neg(r_1(i) = r_3(i)) \wedge \\
 & (r_3(i) = r_5(i)) \wedge \neg(r_1(i) = abs) \wedge \neg(r_3(i) = abs)) \\
\vee & ((r_2(i) = abs) \wedge (r_1(i) = r_5) \wedge \neg(r_1(i) = r_3(i)) \wedge \\
 & (r_3(i) = r_4(i)) \wedge \neg(r_1(i) = abs) \wedge \neg(r_3(i) = abs)) \\
\vee & ((r_3(i) = abs) \wedge (r_1(i) = r_2) \wedge \neg(r_1(i) = r_4(i)) \wedge \\
 & (r_4(i) = r_5(i)) \wedge \neg(r_1(i) = abs) \wedge \neg(r_4(i) = abs)) \\
F = \vee & ((r_3(i) = abs) \wedge (r_1(i) = r_4) \wedge \neg(r_1(i) = r_2(i)) \wedge \\
 & (r_2(i) = r_5(i)) \wedge \neg(r_1(i) = abs) \wedge \neg(r_2(i) = abs)) \\
\vee & ((r_3(i) = abs) \wedge (r_1(i) = r_5) \wedge \neg(r_1(i) = r_2(i)) \wedge \\
 & (r_2(i) = r_4(i)) \wedge \neg(r_1(i) = abs) \wedge \neg(r_2(i) = abs)) \\
\vee & ((r_4(i) = abs) \wedge (r_1(i) = r_2) \wedge \neg(r_1(i) = r_3(i)) \wedge \\
 & (r_3(i) = r_5(i)) \wedge \neg(r_1(i) = abs) \wedge \neg(r_3(i) = abs)) \\
\vee & ((r_4(i) = abs) \wedge (r_1(i) = r_3) \wedge \neg(r_1(i) = r_2(i)) \wedge \\
 & (r_2(i) = r_5(i)) \wedge \neg(r_1(i) = abs) \wedge \neg(r_2(i) = abs)) \\
\vee & ((r_4(i) = abs) \wedge (r_1(i) = r_5) \wedge \neg(r_1(i) = r_2(i)) \wedge \\
 & (r_2(i) = r_3(i)) \wedge \neg(r_1(i) = abs) \wedge \neg(r_2(i) = abs)) \\
\vee & ((r_5(i) = abs) \wedge (r_1(i) = r_2) \wedge \neg(r_1(i) = r_3(i)) \wedge \\
 & (r_3(i) = r_4(i)) \wedge \neg(r_1(i) = abs) \wedge \neg(r_2(i) = abs)) \\
\vee & ((r_5(i) = abs) \wedge (r_1(i) = r_3) \wedge \neg(r_1(i) = r_2(i)) \wedge \\
 & (r_2(i) = r_4(i)) \wedge \neg(r_1(i) = abs) \wedge \neg(r_2(i) = abs)) \\
\vee & ((r_5(i) = abs) \wedge (r_1(i) = r_4) \wedge \neg(r_1(i) = r_2(i)) \wedge \\
 & (r_2(i) = r_3(i)) \wedge \neg(r_1(i) = abs) \wedge \neg(r_2(i) = abs))
\end{array}
$$

$$
\begin{array}{rl}
 & ((r_1(i) = abs) \wedge \neg(r_2(i) = abs) \wedge \neg(r_2(i) = r_3(i)) \wedge \neg(r_3(i) = abs) \wedge (r_4(i) = abs) \wedge (r_5(i) = abs)) \\
\vee & ((r_1(i) = abs) \wedge \neg(r_2(i) = abs) \wedge \neg(r_2(i) = r_4(i)) \wedge \neg(r_4(i) = abs) \wedge (r_3(i) = abs) \wedge (r_5(i) = abs)) \\
\vee & ((r_1(i) = abs) \wedge \neg(r_2(i) = abs) \wedge \neg(r_2(i) = r_5(i)) \wedge \neg(r_5(i) = abs) \wedge (r_3(i) = abs) \wedge (r_4(i) = abs)) \\
\vee & ((r_1(i) = abs) \wedge (r_2(i) = abs) \wedge \neg(r_3(i) = abs) \wedge \neg(r_3(i) = r_4(i)) \wedge \neg(r_4(i) = abs) \wedge \wedge(r_5(i) = abs)) \\
\vee & ((r_1(i) = abs) \wedge (r_2(i) = abs) \wedge \neg(r_3(i) = abs) \wedge \neg(r_3(i) = r_5(i)) \wedge \neg(r_5(i) = abs) \wedge \wedge(r_4(i) = abs)) \\
\vee & ((r_1(i) = abs) \wedge (r_2(i) = abs) \wedge (r_3(i) = abs) \wedge \neg(r_4(i) = abs) \wedge \neg(r_4(i) = r_5(i)) \wedge \wedge\neg(r_5(i) = abs)) \\
\vee & (\neg(r_1(i) = abs) \wedge \neg(r_2(i) = abs) \wedge \neg(r_1(i) = r_2(i)) \wedge (r_3(i) = abs) \wedge (r_4(i) = abs) \wedge (r_5(i) = abs)) \\
\vee & (\neg(r_1(i) = abs) \wedge (r_2(i) = abs) \wedge \neg(r_3(i) = abs) \wedge \neg(r_1(i) = r_3(i)) \wedge (r_4(i) = abs) \wedge (r_5(i) = abs)) \\
\vee & (\neg(r_1(i) = abs) \wedge (r_2(i) = abs) \wedge (r_3(i) = abs) \wedge \neg(r_4(i) = abs) \wedge \neg(r_1(i) = r_4(i)) \wedge (r_5(i) = abs)) \\
\vee & (\neg(r_1(i) = abs) \wedge (r_2(i) = abs) \wedge (r_3(i) = abs) \wedge (r_4(i) = abs) \wedge \neg(r_5(i) = abs) \wedge \neg(r_1(i) = r_5(i))) \\
\vee & ((r_1(i) = abs) \wedge (r_2(i) = abs) \wedge (r_3(i) = abs) \wedge (r_4(i) = abs) \wedge (r_5(i) = abs))
\end{array}
$$

Une solution alternative consistait à utiliser une formule du premier ordre avec quantificateur. (Je ne donne que l'équivalent pour la formule F en logique du premier ordre)

$$
\begin{array}{ll}
\exists, a, b, c, d, e \in \{1, 2, 3, 4, 5\} & (a \neq b) \wedge (a \neq c) \wedge (a \neq d) \wedge (a \neq e) \\
 & \wedge(b \neq c) \wedge (b \neq d) \wedge (b \neq e) \wedge (c \neq d) \wedge (c \neq e) \wedge (d \neq e) \\
 & \wedge(r_a(i) = abs) \wedge (r_b(i) = r_c(i)) \wedge (r_d(i) = r_e(i)) \wedge \neg(r_c(i) = r_d(i))
\end{array}
$$

Vous noterez qu'assurer des indices distincts est aussi important que la fin de la formule.

Q.5. **Etats de fiabilité** (1 point)
En supposant que l'on dispose d'une variable comptant le nombre de noeuds byzantin et défaillants par omission (ensemble supposés disjoints dans le pire car). Définissez la formule des états de fiabilité à un instant t (i.e. la formule ?

Réponse : Soit n_b le nombre de byzantin et n_o le nombre de défaillants par omission, un état est garanti fiable si $(5 - n_o) > 2 * n_b$.

4 Tolérance aux fautes sur les données

Un burst est une suite de bits altérés de manière aléatoire (pas nécessairement une inversion de valeur mais on remplace une séquence de bit par une autre). Le problème ici est que l'on ne connaît pas a priori la nature de l'altération réalisée (inversion ou mise à une valeur prédéfinie), ni la position à laquelle l'altération commence par apport à une séquence de données codées par bloc et stockée de manière contiguë en mémoire.

Il est possible d'améliorer les capacités de tolérance aux faute d'un code sur une donnée correspondant à un message de taille fixe N en tirant parti du fait que l'on ait une séquence de P blocs à stockée de manière fiable, $M_1, ..., M_P$. Il suffit de :

- découper chaque bloc M_i du code en un nombre identique de sous blocs, par exemple k, $M_i(1)...M_i(k)$.
- Puis réaliser le stockage comme suit :
  Pour j de 1 à k, répéter :
  Pour i de 1 à P répéter :
  ajouter $M_i(j)$ en mémoire

Q.6. **Entrelacement pour tolérance aux "burst"** (1 points)
On suppose que l'on utilise un codage capable de tolérer sur chaque bloc 9 bits de valeur erronée avant entrelacement (potentiellement altérés), et que chaque bloc a une taille de N=64 bits. Peut-on tolérer 10 bits consécutifs altérés à partir du premier bit stocké pour le premier bloc stocké si on prend k=8, et P =2? **Justifiez votre réponse**

Réponse : Dans ce cas si 10 bits sont altérés de manière consécutive, cela affecte au plus 8 bits dans un message et donc cela sera toléré car pour tout série de 10 bits consécutifs au plus 8 peuvent appartenir au même message si l'on juxtapose les sous bloc de 8 bits de deux messages différents.

Q.7. **Entrelacement le cas extrême** (2 $\frac{1}{2}$ points)
En supposant que la taille de chaque bloc est un multiple de k, N=k*s (i.e. on découpe le bloc en k sous-blocs de taille identique s), et que le codage utilisé pour chaque bloc de taille N possède (avant découpe et entrelacement) une capacité à tolérer r bits altérés.

(a) Illustrez par un schéma représentant la séquence de bits stockés le scénario où :
- les bits sont altérés dès le premier bit stocké, q bits sont altérés.
- les paramètres s,k,r,P ont les valeurs suivantes : s=16, k=4, r= 20, P=2.

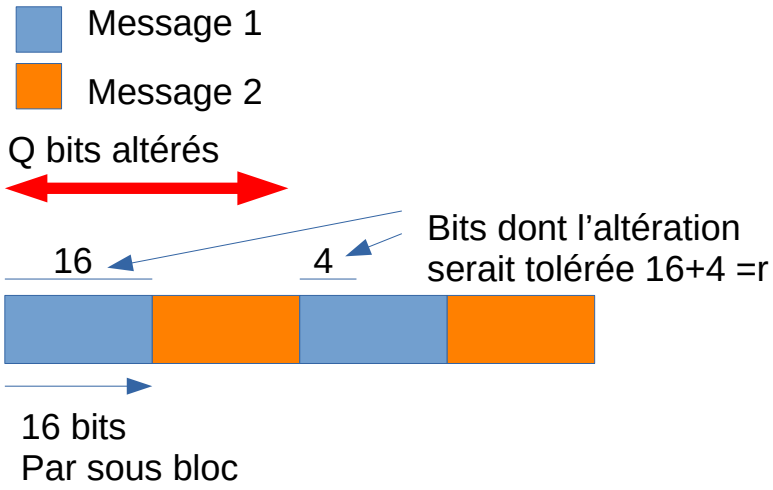

Figure 2: Réponse question schéma

Réponse :

(b) Quelle valeur peut prendre q au maximum dans ce scénario sans empêcher la correction des bits altérés ?

Réponse : $q = 16 + 16 + 4 = 36$

(c) Donnez la formule générale qui permet de connaître la valeur maximale de q pouvant être tolérée en fonction de tout ou partie des paramètres suivants : s,k,r,P ? (vous pouvez utilisez les notations suivantes $\lfloor f \rfloor$ pour désigner la partie entière de la fraction f).

Réponse : $q = \lfloor \frac{r}{s} \rfloor * s * P + r - \lfloor \frac{r}{s} \rfloor * s$