M2 COMASIC
Examen de Sureté de Fonctionnement
30/01/2017

Durée de l'examen 3h.

Consignes particulières : les documents sont interdits à l'exception d'une feuille A4 manuscrite recto verso. Les téléphones portables doivent être rangés. Le présent examen est noté sur 20 pts. Le barème est indiqué devant chaque question.

I Principes de la sûreté de fonctionnement (4 pts)

Question 1) (1 pt) Donnez un exemple, soit d'une exigence non fonctionnelle de sureté de fonctionnement, soit une exigence fonctionnelle de sureté de fonctionnement (vous pouvez par exemple vous servir de l'exemple du distributeur de boisson, ou d'un train).

Réponse : une exigence fonctionnelle de sureté peut concerner la sécurité innocuité du distributeur de boisson chaudes. « L'accès physique au gobelet ne doit pas être possible lorsque la boisson est en cours de préparation, le liquide tomber dans le gobelet) ».

Un exemple d'exigence non fonctionnelle est d'exiger que la disponibilité de la machine corresponde à un temps entre défaillance de 6 mois.

Question 2) (1,5 pts) Définissez en expliquant la différence entre les deux concepts : l'élimination des fautes et la tolérance aux fautes

Réponse : la tolérance aux fautes correspond à compenser ou corriger l'impact des fautes e.g. les erreurs ou les défaillances au sein du système. L'élimination de faute consiste à altérer la structure du système ou son environnement pour que les fautes en question n'existent plus. Par exemple, si l'on prend un fonction de calcul dont l'implémentation peut contenir des pointeurs non initialisés qui seront déréférencé.

Illustration des différences (un exemple plus simple aurait pu L'élimination consiste à les initialiser systématiquement à une valeur pouvant être initialisée. Prenons le même modèle de faute, la tolérance pourrait consister à capturer l'erreur issue de ce déréférencement et à rediriger l'exécution du code vers une autre implémentation.

Question 3) (1,5 pts) Définissez brièvement ce qu'est une zone de confinement d'erreur dans une description architecturale d'un système. Décrivez en les principales caractéristiques.

Réponse : Une zone de confinement d'erreur décrit un (ou plusieurs) composant(s) dont l'interface dispose de mécanismes de confinement des erreurs permettant de limiter, ou contrôler la propagation. Ceci correspond donc à l'utilisation de mécanismes de tolérances aux fautes pour masquer ou corriger les erreurs, ou à minima signaler leur occurrence aux composant dépendant de cette zone de confinement. Les caractéristiques principales sont donc la définition d'un comportement attendu et de modes de défaillances connus, le tout garantis par des mécanismes intégrés au composant.

Quelques mots sur le barème ici : 1 pt si la zone de confinement est définie comme un composant dont l'interface limite/ contrôle la propagation des erreurs + 0,5 si vous parlez du signalement (dit dans le cours), ou des modes de défaillances connus.

II Architecture redondantes et modèles de fautes (8 pts)

Question 4) (1 pt) Rappelez ce qu'est une architecture redondante de type réplication passive. (On demande le fonctionnement mais pas l'analyse des états de fiabilité / disponibilité)

Réponse : une réplication passive consiste à disposer de N calculateurs pour assurer l'exécution d'une fonction. Le principe est que l'une des répliques exécute la fonction dont on cherche à augmenter la disponibilité. Ce calculateur est usuellement appelé primaire. La fonction doit exhiber un modèle de défaillance par crash pour que l'architecture fonctionne. En l'absence de crash, le primaire capture périodiquement son état et l'envoie aux autres répliques (dites secondaires ou de sauvegarde). Dès que les secondaires détectent que les mises à jour se sont arrêtées, ils échangent entre eux pour définir un nouveau primaire. Une fois ce dernier défini, il charge le dernier état d'exécution reçu en tant que secondaire et reprend l'exécution à partie de là. Il doit à partir de maintenant assurer le comportement du primaire décrit au début de la réponse.

Une architecture dite de réplication active est constituée d'un voteur, et d'un certain nombre de calculateurs identiques. Cette architecture repose sur le principe suivant :

- Chaque calculateur possède une version logicielle de la fonction qui doit être redondée.
- Chaque version du logiciel est censée s'exécuter en un temps borné connu T
- Les défaillances de chaque calculateur et de son logiciel sont supposées indépendantes entre calculateurs.
- Au bout d'un temps T, le voteur compare les réponses reçues pour déterminer une réponse majoritaire.

Nous considérons deux modèles de défaillance différents : crash (ou défaillance silencieuse) et byzantin.

Le premier entraine l'arrêt du calculateur : la défaillance ne propage pas de valeur incorrecte. Dans le cas byzantin la défaillance peut propager une valeur incorrecte mais avant tout la manière dont les nœuds défaillent peut être coordonnée.

Question 6) (0,5 pts) Indiquez le nombre de calculateurs devant être considérés pour tolérer deux défaillances (on suppose le voteur totalement fiable) pour chaque modèle, justifiez ?

Réponse :

Pour le modèle par crash, le système est fonctionnel dès qu'une réplique est non défaillante. En effet, seul les répliques fonctionnelles transmettent une valeur au voteur. Toute réponse reçue sera donc identique. Une unique réponse est donc suffisante. Il en découle que pour tolérer 2 défaillances, il faut 3 répliques.

Pour le modèle byzantin, le système est fonctionnel seulement si une majorité de nœuds sont non défaillants lors de chaque vote. Il en découle que pour tolérer 2 défaillances, il est nécessaire d'avoir 5 répliques.

Il s'avère que le voteur reçoit des valeurs pouvant subir des légère variations dues au calcul numérique. Ainsi par rapport au résultat théorique D chaque réplique peut retourner une valeur X telle que | X- D| < err. (|.| désigne la valeur absolue)

Question 7) (1 pt) indiquez comment procéder coté voteur pour déterminer si les résultats retournées sont valides (sachant que D est inconnue). Indiquez quelle serait l'erreur numérique finale produite par l'architecture de réplication active à 3 répliques ?

Réponse :

Attention, il y a avait une ambiguité sur la présence ou non de défaillance et si oui leur nombre possible (si le raisonnement est justifié, l'une des trois réponse possible est acceptée).

Le raisonnement de base est le même. Le résultat retourné par une réplique Ri sera notée Xi. Il en découle que pour R1, R2, |X1-D|<err et |X2-D|<err. La distance de deux valeurs correcte est au plus de 2*err. On n'a pas accès à mieux. La notion de vote majoritaire doit être raffinée dans ce cadre là car a priori aucun ensemble de réplique ne fournira un résultat identique. La première solution consiste à donner une note à chaque réponse. Cette note vaut initialement 0 et est augmentée pour chaque réponse à distante inférieure à 2*err. La valeur de note la plus élevée sera considérée comme résultat du vote. SI K>1 valeurs ont une note identique. Il y a deux choix : on en prend une au hasard (cas 1) ou on réalise un vote. (cas 2)

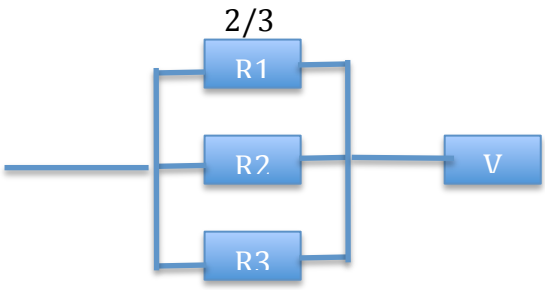
- Il en découle que sans défaillance, la réponse choisie sera forcément à distance err du bon résultat (puisque toutes les valeurs ont la même note).
- En présence d'une unique défaillance, on a au plus une erreur. Dans le cas 1 on borne l'erreur à 2 *err. Car il est possible que la réplique défaillante et une réplique correcte aient une note de 3. Dans le cas 2, la moyenne des résultats respectif des ces répliques ayant une note de 3 correspond à une erreur de 2err pour la première et err pour la seconde ce qui donne 3/2 err.
- Pour 2 défaillances ou plus, l'erreur est non maitrisée car TMR est défaillant. Quel que soit le modèle de vote.

On notera que dans la version généralisée à N répliques, l'erreur tend vers 2 quelque soit la méthode choisie tant que TMR n'est pas défaillant (minorité de processus défaillants).

Remarque : la réponse est longue mais une version simplifiée sans tous les cas rapportait le point complet.

Le modèle de reliability bloc diagram est une représentation graphique des conditions qui définissent l'état de fiabilité du système : la formule indiquant sous quelles conditions une architecture reste opérationnelle étant donné que certains de ses composants sont défaillants. Il en existe trois variantes de base : série, parallèle et k parmi n. Vous indiquerez pour le cas « k parmi n » les valeurs de k et n choisies. (cette expression indique la condition de correction k nombre corrects requis sur nombre total de répliques). Nous supposerons dans un premier temps qu'un bloc permet d'identifier : le fonctionnement d'un calculateur avec le logiciel

Question 8) (1pt) Donnez le schéma correspondant à une réplication active avec 3 répliques



Réponse : le voteur est optionnel mais l'architecture doit être du 2 parmi 3. Noté que le voteur est optionnel car on a fait l'hypothèse dans la question précédente que les voteurs étaient parfaits.

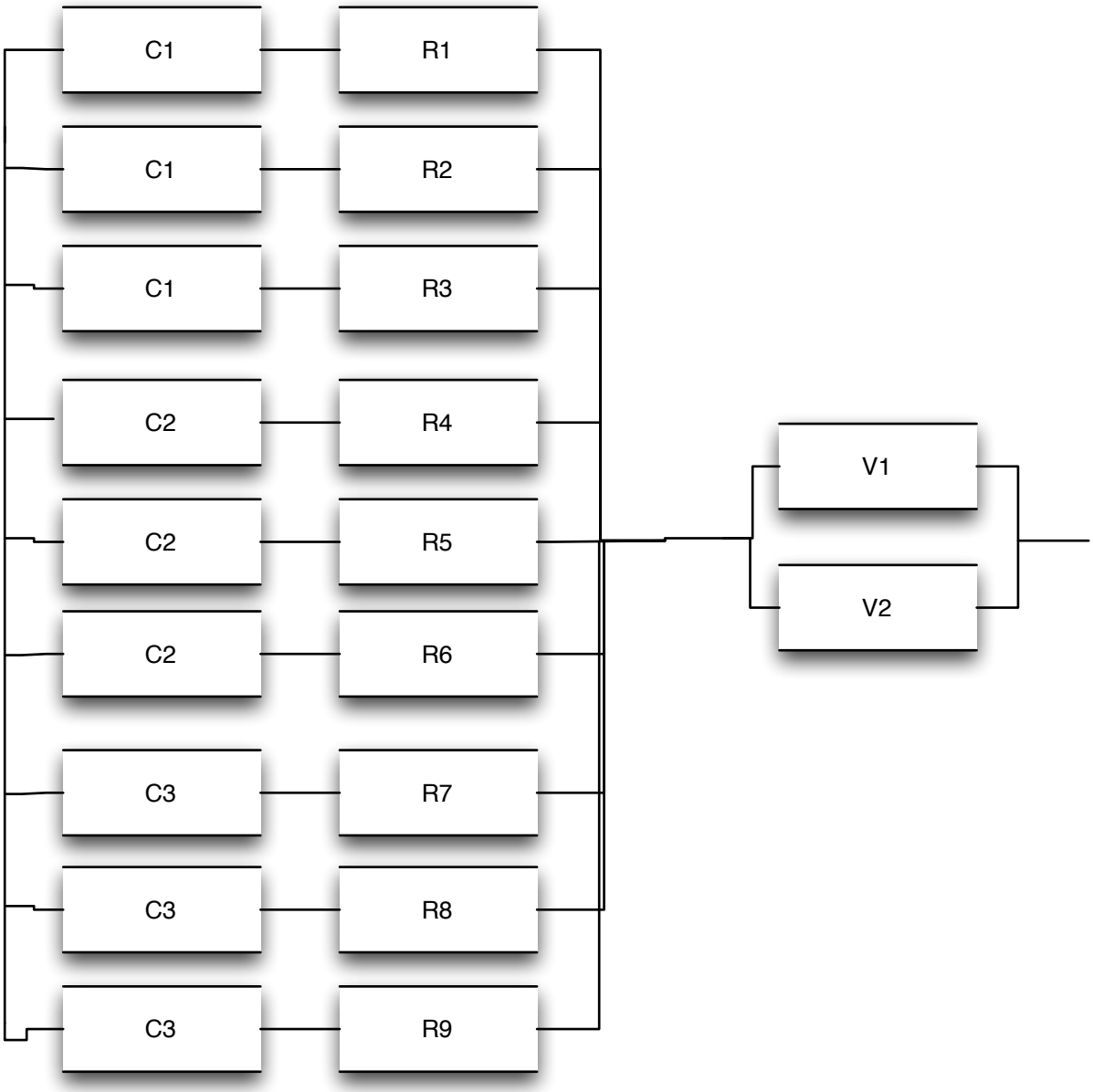
On suppose maintenant que l'on dispose calculateurs pouvant chacun exécuter trois répliques et un voteur (si nécessaire). On suppose de plus que la défaillance provient du logiciel seul et ne peut se propager aux répliques exécutées sur le même calculateur.

Question 9) (1 pt) Quelle est la démarche la plus fiable : réaliser un unique vote parmi toutes les répliques ou 3 votes locaux aux calculateur et un vote global entre réponses de calculateur (on suppose toujours les composants de vote parfait).

Réponse : La première solution est la plus fiable. En effet, si l'on considère 3 calculateur avec 3 répliques sur chaque, on obtient que dans le premier cas 5 répliques défaillantes sont nécessaire pour rendre le système défaillance, alors dans le second cas 4 suffisent.

On souhaite désormais séparer l'impact du logiciel et du matériel, ainsi que l'impact du voteur. On supposera que le voteur ne peut défaillir que par crash, et que l'implémentation logicielle de la fonction, et chaque calculateur peut défaillir selon un modèle byzantin. Cependant, ces défaillances sont indépendantes dans leur occurrence (i.e. chaque nœuds défaille indépendamment mais une fois défaillant, il peuvent se synchroniser pour produire des données fausses).

Question 10) (3,5 pts) Donnez le reliability bloc diagram de l'architecture ou le voteur est dupliqué (deux voteurs), et où l'on dispose de 3 calculateurs exécutant chacun 3 répliques du logiciel. On supposera que le voteur de l'architecture vote sur la réponse majoritaire parmi l'ensemble des répliques logicielles exécutées.



Notez que C1..3 désignent les calculateurs, R1..9 les répliques logicielles, et V1 V2 les voteurs. On notera l'architecture 5/9 de deux blocs en série un calculateur + une réplique logicielle. Les voteurs et les calculateurs sont liés. Ce système tolère jusqu'à 4 répliques logicielles défaillante et 1 voteur défaillant, ou 1 calculateur défaillant, une réplique défaillante sur un autre calculateur et un voteur défaillant.

La réponse n°1 ou n°2 seule donne 2 pts, la réponse plus le diagramme donne tous les points.

III Mécanismes de tolérance aux fautes (3 pts)

Un ingénieur propose d'implémenter un mécanisme de capture d'état d'exécution de processus Unix (contexte d'exécution d'un programme). Ce mécanisme serait couplé

avec un détecteur d'erreur pour pouvoir tolérer fautes liées à des erreurs de développement se manifestant de manière aléatoire. En pratique, il propose donc de déployer un mécanisme de tolérance aux fautes utilisant :

- une sauvegarde régulière d'état associée à un mécanisme de détection d'erreur
- si une erreur est détectée elle déclenche la restauration du dernier état sauvegardé en cas d'erreur.

Question 11) (0,5 pts) Un tel mécanisme porte un nom, donnez le.

Réponse : C'est un recouvrement arrière.

Question 12) (1 pt) Quelle caractéristique doit vérifier la faute dont ce mécanisme est censé corriger les erreurs ?

Réponse : la faute ne doit pas avoir une logique d'activation déterministe lié à l'état d'exécution du processus (sinon le chargement de l'état réactive la faute).

Un recovery block est une manière de munir un code, réalisant un calcul, de capacité de tolérance aux fautes. Le principe de fonctionnement d'un recovery block est d'utiliser ce que l'on appelle un test de vraisemblance pour savoir si le résultat produit est correct. De plus, on dispose d'un certain nombre de versions différentes de logiciels permettant de réaliser la même tâche. On suppose que les calculs sont réalisés sur un volume de données important.

Question 13) (1,5 pt) Dans certaines versions du recovery block, le mécanisme capture une sauvegarde de l'état avant exécution des fonctions de calcul. Donnez une situation dans laquelle ce choix se justifie. Proposez une méthode pour éviter de devoir nécessairement réécrire l'intégralité de l'état du système à chaque chargement.

Réponse : la restauration de l'état d'exécution se justifie si le code possède un effet de bord sur un certain ensemble de variables globales (ou partagées entre les différents appels de fonctions). En gros, cela se justifie si l'exécution de la fonction « écrase » des données importantes. Pour éviter une réécriture systématique de tous l'état du système, il faut se consacrer avant à sauvegarder ce qui est susceptible de changer. Par exemple, si les paramètres ne sont que consultés, inutile de les sauvegarder pour les recharger. Une solution pour éviter de tout sauvegarder à chaque fois, c'est de détecter ce qui est modifié et de restaurer uniquement ces valeurs.

IV Analyse quantitative (6 pts)

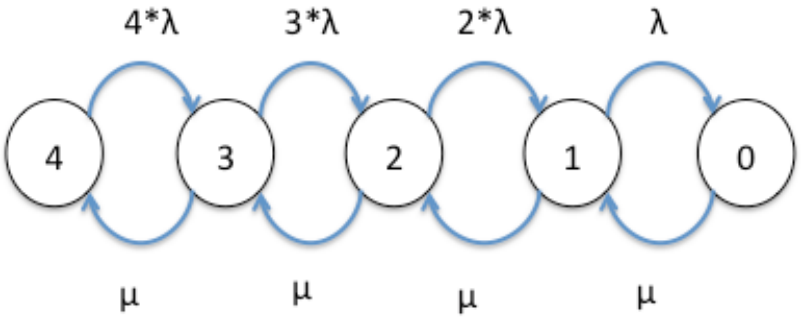
Il est dit qu'une architecture de réplication passive permet d'augmenter la disponibilité d'une fonction de calcul si le modèle de défaillance par crash est considéré.

Question 14) (1pt) Peut on dire la même chose pour une réplication active pour le modèle de défaillance byzantin, justifiez ?

Réponse : c'est faux pour que l'architecture puisse fonctionner, 2 répliques sur 3 doivent être disponibles, il en découle qu'avec la même disponibilité pour un machine donnée, la réplication active exige que 2 machines fonctionnent correctement en même temps. Ceci a tendance à être plus dur que d'avoir 1 machine disponible à un instant donné.

Nous utilisons la chaine de Markov à temps continu page 4 pour modéliser le temps avant défaillance d'une réplication passive. Nous utilisons le paramètre λ pour identifier le taux de défaillance, et le paramètre μ pour le taux de réparation.

Question 15) (2 pt) Que pouvez vous dire sur ce modèle concernant les hypothèses de défaillance des différentes répliques et le processus de réparation ? (On suppose que l'on dispose de 4 répliques initialement).



Réponse : Ce modèle nous indique que chaque réplique non défaillante possède a priori le même taux de défaillance lambda. De plus, il semblerait que l'on ait un taux constant de réparation d'une réplique : mu.

Nous souhaitons modélisez le fait que jusqu'à 2 répliques défaillances peuvent se réparer en même temps avec le même taux moyen de réparation que dans la question précédente.

Question 15) (1,5 pts) modifiez le modèle pour capturer cette situation.

Réponse : La réponse la plus simple consiste à doubler le taux mu pour les transitions ayant pour point de départ un état inférieur ou égal à 2. Il y avait il semblerait une ambiguité sur les termes « même taux moyen». D'autre réponse ont été jugées acceptable en particulier les version ajoutant des transitions de i vers i+2 pour i<3 (notez que dans ce cas les données étaient incomplète pour définir les taux de ces arcs. La première réponse doit être jugée comme la réponse attendue). .

On suppose que l'on réplique un service implémenté en logiciel qui s'exécute en temps constant. Lors d'une exécution, chaque réplique a la même probabilité de défaillir f lors de l'exécution d'une instance du service (lié au matériel). Les phénomènes de défaillance sont indépendants les un des autres. On suppose que l'architecture choisie est une architecture de réplication passive.

Question 16) (1,5 pt) Pensez vous que si l'on prend des composants ayant une chance de défaillir de ½ lors d'une exécution, alors la réplication passive peut améliorer la disponibilité du résultat ? Si oui de combien l'améliore-t-on pour 3 répliques par rapport à une seule.

Oui cela l'améliore, et en pratique on améliore la disponibilité par 3. En effet, chaque défaillance est indépendante des autres. De plus, les répliques ne peuvent défaillir que

lorsque le programme s'exécute. Il en découle que l'on calcule l'espérance d'une somme de variable aléatoire indépendante (somme des moyenne du temps avant défaillance). Et donc le temps moyen se voit triplé.