ECOLE POLYTECHNIQUE DE MONTREAL

Département de génie informatique et génie logiciel

Cours INF4410: Systèmes répartis et infonuagique (Automne 2014)
3 crédits (3-1.5-4.5)

CORRIGÉ DE L'EXAMEN FINAL

DATE: Vendredi le 19 décembre 2014

HEURE: 13h30 à 16h00

DUREE: 2H30

NOTE: Toute documentation permise, calculatrice non programmable permise

Ce questionnaire comprend 4 questions pour 20 points

Question 1 (5 points)

a) Un ordinateur A envoie un message à B à 13h30m10.250, heure de A, pour lui dire qu'il est 13h30m10.250. L'ordinateur B reçoit ce message à 13h30m08.350, heure de B. Quel ajustement devrait appliquer B pour se synchroniser sur A? Enoncez vos hypothèses. Peut-on calculer l'incertitude sur cet ajustement? Si oui, quel est-il? **(2 points)**

Il n'y a pas moyen a priori de savoir quel est le délai sur le réseau. On peut donc assumer qu'il est négligeable. Ceci ferait qu'à 13h30m08.350 heure de B, on sait qu'il est près de 13h30m10.250 heure de A, et qu'un ajustement de 13h30m08.350 - 13h30m10.250 = -1.9s serait requis. Nous n'avons pas de donnée sur l'incertitude de cet ajustement. On pourrait faire des hypothèses sur le délai minimal et maximal requis pour l'envoi du message, surtout si on a plus d'information sur le type de réseau qui relie les deux ordinateurs. On pourrait alors utiliser la valeur moyenne de délai réseau et la soustraire de l'ajustement calculé. Par exemple, pour un délai minimal de .1ms et maximal de 10ms, on aurait une moyenne de 5.05ms, ce qui donnerait un ajustement de -1.90505s. On pourrait alors utiliser +/-5.05ms comme incertitude si ces valeurs minimales et maximales sont fiables.

b) Trois processus utilisent des horloges logiques qui sont incrémentées à chaque événement, incluant les envois et réceptions de messages entre processus. Les événements vus par chacun sont listés ci-après. On veut fournir des clichés cohérents de l'état global du système. L'état global est déterminé par la liste des événements inclus, et donc pour chaque processus par la partition incluse de leur liste des événements. La liste complète des événements pour chaque processus, incluant tous les messages échangés, est disponible. Il faut donc choisir un point de coupure dans la liste de chacun des trois processus de sorte que le cliché résultant de l'état global soit cohérent. On vous propose deux clichés possibles (points de coupure pour chaque processus): (p1: entre les événements 2 et 3, p2: entre les événements 2 et 3, p3: entre les événements 2 et 3) et (p1: entre les événements 3 et 4, p2: entre les événements 3 et 4, p3: entre les événements 3 et 4). Pour chacun de ces deux clichés, dites s'il est cohérent ou non et expliquez pourquoi. **(2 points)**

p1: 1 message A vers p2; 2 message B de p3; 3 message C vers p3; 4 message D de p2; 5 message E vers p2, 6 message F de p2; 7 message G de p3

p2: 1 message A de p1; 2 message D vers p1; 3 message H vers p3; 4 message F vers p1; 5 message I de p3; 6 message E de p1

p3: 1 message B vers p1; 2 message C de p1; 3 message I vers p2; 4 message H de p2; 5 message G vers p1

Un cliché est cohérent si pour chaque événement inclus, tous les événements qui le précèdent sont aussi inclus. Ceci peut se résoudre graphiquement en vérifiant qu'aucun message n'a son origine après la frontière et sa destination avant la frontière. Une autre méthode est de vérifier le dernier événement inclus de chaque processus (les autres inclus sont plus antérieurs) versus le premier événement non inclus de chaque autre processus (les autres non inclus sont plus postérieurs) en utilisant les vecteurs d'horloges logiques.

p1: <1,0,0> message A vers p2; <2,0,1> message B de p3; <3,0,1> message C vers p3; <4,2,1> message D de p2; <5,2,1> message E vers p2, <6,4,1> message F de p2; <7,4,5> message G de p3

p2: <1,1,0> message A de p1; <1,2,0> message D vers p1; <1,3,0> message H vers p3; <1,4,0> message F vers p1; <3,5,3> message I de p3; <5,6,3> message E de p1

p3: <0,0,1> message B vers p1; <3,0,2> message C de p1; <3,0,3> message I vers p2; <3,3,4> message H de p2; <3,3,5> message G vers p1

Pour le premier cas, il faut comparer <3,0,1>, <1,3,0> et <3,0,3> (exclus) qui ne doivent pas être antérieurs à <2,0,1>, <1,2,0> et <3,0,2> (inclus). Ceci ne fonctionne pas car <3,0,1> est effectivement antérieur à <3,0,2> et le cliché n'est pas cohérent. Dans le second cas, il faut comparer <4,2,1>, <1,4,0> et <3,3,4> (exclus) qui ne doivent pas être antérieurs à <3,0,1>, <1,3,0> et <3,0,3> (inclus). C'est le cas et ce deuxième cliché est cohérent.

c) Un professeur de géologie vous explique que la durée de la rotation de la terre sur son axe varie dans le temps en raison du mouvement dans le noyau liquide de la terre et même des marées. Plus encore, la vitesse de rotation de la terre diminue graduellement (la durée des jours augmente) en raison du frottement associé aux marées. Puisque les journées font généralement 24 heures, les heures 60 minutes et les minutes 60 secondes, comment tient-on compte de ce facteur? Est-ce que la durée des secondes varie d'une journée à l'autre pour correspondre à $24 \times 60 \times 60$ secondes par jour? Ou est-ce que le nombre de secondes par jour varie? Expliquez. **(1 point)**

La durée des journées varie tout de même très peu. Le temps universel coordonné (UTC) ajoute ou retranche une seconde de temps en temps, soit le 30 juin ou le 31 décembre à la fin de la journée, lorsque la différence avec le temps solaire est trop grande. En pratique, une seconde est ajoutée une fois par 3 ans environ. Le logiciel d'ajustement de l'heure sur l'ordinateur peut choisir de faire varier la durée des secondes afin de cacher cette seconde ajoutée qui pourrait confondre certains logiciels (e.g., calendrier).

Question 2 (5 points)

a) Quatre transactions, T, U, V et W s'exécutent concurremment. Le séquencement des opérations est fourni ci-après. Lesquelles des transactions T, U, V et W pourraient être validées si une validation en reculant était utilisée pour vérifier la cohérence des transactions? Une validation en avançant? Justifiez. **(3 points)**

T: Début
U: Début
T: Read(a)
T: Read(b)
T: Read(c)

U: Read(c)
U: Read(d)
V: Début
V: Read(e)
T: Write(b)
T: Write(f)
T: Compléter
W: Début
W: Read(e)
W: Read(f)
U: Write(c)
U: Write(d)
U: Compléter
V: Read(d)
W: Write(a)
W: Write(b)
W: Compléter
V: Read(g)
V: Write(e)
V: Compléter

En reculant, au moment de compléter T, il n'y a pas de problème. Pour compléter U, il faut vérifier ce qu'il a lu (c, d) versus ce qui a été écrit par les transactions précédentes concurrentes (T qui a écrit b, f). Il n'y a pas d'intersection et U peut compléter. Pour W, ses lectures (e, f) sont vérifiées versus les écritures de U (c, d), alors que T était terminé avant que W ne débute. Il n'y a pas de problème là non plus et W peut compléter. Pour V, les lectures (d, e, g) doivent être vérifiées versus les écritures de T, U et W (a, b, c, d, f). Il y a intersection (d) et la transaction V doit être abandonnée. En fait, U était complété avant que V ne lise la valeur de d mais cette information plus détaillée n'est pas utilisée par la méthode de base en reculant, elle est utilisée par la méthode par estampille de temps.

En avançant, il faut vérifier ce qui est écrit par T (b, f) versus ce que U et V ont lu à ce moment (c, d, e). Il n'y a pas d'intersection et T peut compléter. Au moment de compléter U, ses écritures (c, d) sont comparées aux lectures à ce moment de V et W (e, f); les lectures à venir de V et W ne sont pas encore connues à ce moment et n'ont de toutes manières pas d'impact sur la cohérence. Il n'y a pas d'intersection et la transaction U est validée. Pour W, ses écritures (a, b) sont comparées aux lectures de V à ce moment (d, e). Il n'y a pas d'intersection et W peut compléter. La dernière transaction, V, complète nécessairement et ainsi les quatre transactions sont validées.

b) Pour les transactions réparties, un protocole à deux phases est utilisé. Lors de la première phase, on demande à chaque serveur de préparer la transaction et de confirmer qu'il est prêt à commettre. Lors de la seconde phase, on confirme à chaque serveur et au client que la transaction a été acceptée par tous et est commise. A quel moment au plus tôt peut-on envoyer le message au client que la transaction est confirmée? Au plus tard? A quel moment au plus tôt peut-on envoyer le message à chaque serveur impliqué dans la transaction que celle-ci est confirmée? Au plus tard? **(1 point)**

La confirmation peut être envoyée au client au plus tôt après avoir reçu de chaque serveur la confirmation qu'il est prêt à commettre. Il n'y a pas de borne pour le plus tard, mais il n'y a pas intérêt à faire attendre le client non plus. La situation est exactement la même pour confirmer la transaction aux serveurs. Le plus tôt est après avoir reçu de chaque serveur la confirmation qu'il est prêt à commettre et il n'y a pas de borne pour le plus tard. Là encore, il est désavantageux d'attendre trop longtemps puisque cela retarde la possibilité pour chaque serveur de clore la transaction, et de l'enlever de sa liste de transactions actives et éventuellement de son journal.

c) Pour un système transactionnel, vous hésitez entre un système de maintien de la cohérence avec des verrous ou un système de validation de la cohérence avec la méthode en reculant. Comment est-ce que ces deux systèmes se compareront en termes de vitesse et de temps d'attente pour accéder aux ressources? Est-ce que d'autres facteurs sont à considérer? Le système le plus lent présente-t-il d'autres avantages? **(1 point)**

Les verrous imposent des opérations supplémentaires pendant la transaction (prise des verrous) et peuvent causer de l'attente pour l'obtention de ces verrous. L'utilisation des verrous est donc a priori plus lente que la validation optimiste de la concurrence par la méthode en reculant. Par contre, avec la validation optimiste de la concurrence, un certain nombre de transactions pourront devoir être abandonnées. Ceci peut enlever tout l'avantage procuré par l'exécution plus rapide sans verrou. Plus encore, dans certaines applications, le fait de devoir invalider une transaction demandée par un client peut être un désagrément majeur. Pour cette raison, la méthode avec les verrous demeure la plus intéressante dans un grand nombre d'applications.

Question 3 (5 points)

a) On vous demande de choisir entre deux configurations pour le prochain serveur de fichiers de votre entreprise. La première configuration consiste en un ordinateur avec 4 disques en miroir. Chaque disque contient l'ensemble des données et un seul disque suffit donc. Le second système est constitué de deux ordinateurs, qui sont deux serveurs redondants, chacun étant connecté à deux disques en miroir et un seul disque suffit donc pour un serveur. La probabilité qu'un ordinateur soit opérationnel (hormis les disques) est de 0.95. La probabilité qu'un disque soit opérationnel est de 0.85. Quelle est la probabilité que le service soit disponible, pour chacune des deux configurations? **(2 points)**

Dans la première configuration, le système de disque est disponible sauf si les 4 sont indisponibles $1 - (1 - .85)^4 = 0.99949375$. *Le service sera disponible si les disques et l'ordinateur le sont,* $0.99949375 \times 0.95 = 0.949519063$. *Dans le second cas, sur un serveur, les disques seront disponibles sauf si les 2 sont indisponibles* $1 - (1 - .85)^2 = 0.9775$. *Un serveur sera disponible si l'ordinateur et les disques le sont* $0.9775 \times 0.95 = 0.928625$. *Le service sera disponible sauf si les deux serveurs redondants sont indisponibles* $1 - (1 - 0.928625)^2 = 0.994905609$. *Cette deuxième configuration est donc nettement mieux pour réduire le temps d'indisponibilité. Elle est toutefois légèrement plus coûteuse puisqu'elle utilise deux ordinateurs plutôt qu'un, tout en conservant le même nombre de disques.*

b) Les interblocages posent un problème sérieux dans les systèmes de base de données répartis. i) Comment peut-on détecter un interblocage dans un tel système? ii) Est-il possible de le faire sans simultanément arrêter tous les processus impliqués? iii) Doit-on constamment vérifier la présence d'interblocages, ou peut-on le faire seulement à la demande (selon quel critère)? iv) A défaut de faire une détection précise des interblocages, quel mécanisme peut-on utiliser pour s'assurer de ne pas laisser des transactions bloquées indéfiniment? **(2 points)**

i) Pour détecter les interblocages, il est possible de construire un graphe de dépendance entre les verrous, ceux qui les possèdent et ceux qui les attendent, et de vérifier l'existence d'un cycle dans le graphe. ii) A défaut de tout arrêter pour avoir un cliché exact du graphe réparti de dépendance, il est possible de le construire de manière incrémentale un graphe approximatif en interrogeant les processus impliqués les uns après les autres. Si un interblocage est présent, les dépendances en cause sont figées et ceci apparaîtra dans le graphe obtenu, même s'il est construit incrémentalement. Cependant, si une transaction est annulée au milieu de la construction de notre graphe, il est possible que nous détections de manière erronnée un interblocage. iii) La vérification des interblocages ne se fait pas sans arrêt puisque ce serait trop coûteux. Elle peut se faire à intervalle régulier ou seulement lorsqu'il est détecté qu'une transaction reste anormalement longtemps en attente d'un verrou. iv) Dans la plupart des cas, la détection des interblocages n'est pas faite directement. Chaque transaction est simplement annulée si elle reste bloquée trop longtemps.

c) Vous avez besoin d'un service d'exclusion mutuelle pour votre système réparti. On vous propose un système symétrique réparti. Pour obtenir un verrou, un système demande la permission de chaque autre système. Celui qui détient le verrou attend d'en avoir terminé avant de donner la permission. Celui qui est en demande de verrou attend d'avoir obtenu et fini du verrou avant de donner la permission. Est-ce que ce système est sûr, vivace et respecte l'ordre? Expliquez. Est-ce que ce système est efficace? **(1 point)**

Ce système est sûr puisque la permission de chacun ne peut être obtenue tant qu'un autre processus détient le verrou. Ce système respecte l'ordre puisque chaque demande est traitée dans l'ordre par chaque processus. Le système est vivace puisque chacun passe dans l'ordre et que le système ne peut normalement bloquer; ainsi chacun verra éventuellement sa demande satisfaite. Ce système n'est pas efficace puisqu'il ne fonctionne que si les n processus répartis sont opérationnels, et puisque l'obtention de chaque verrou demande au moins n messages, contre 2 pour un serveur central.

Question 4 (5 points)

a) Dans le travail pratique 3, à la question 3, on vous demande d'écrire une section pour votre gabarit heat qui définit une alarme qui est déclenchée lorsque le taux moyen d'utilisation du CPU d'une machine est supérieur à 90% sur une période de 1 minute. Ecrivez une version modifiée qui déclenche une alarme de la même manière, lorsque le taux moyen d'utilisation du CPU d'une machine est supérieur à CPU_limit_high sur une période de CPU_limit_period, où CPU_limit_high et CPU_limit_period sont des paramètres d'entrée pour le gabarit. Donnez la portion de gabarit à insérer dans la section *resources:* ainsi que la portion à ajouter dans la section *parameters:*. **(2 points)**

Il faut ajouter les sections de gabarit suivantes dans `parameters:` et `resources:` respectivement.

```
parameters:
  CPU_limit_high:
    type: number
    label: CPU usage
    description: CPU usage percentage high threshold value
    constraints:
      - range: { min: 0, max: 100 }
  CPU_limit_period:
    type: number
    label: CPU usage period
    description: CPU usage period in seconds

resources:
  cpu_alarm_high:
    type: OS::Ceilometer::Alarm
    properties:
      description: Augmentation de 1 si cpu > CPU_limit_high
      meter_name: cpu_util
      statistic: avg
      period: {get_param: CPU_limit_period}
      evaluation_periods: 1
      threshold: {get_param: CPU_limit_high}
      alarm_actions:
        - {get_attr: [web_server_scaleup_policy, alarm_url]}
      matching_metadata: {'metadata.user_metadata.stack':
          {get_param: "OS::stack_id"}}
      comparison_operator: gt
```

b) Quels sont les principaux avantages de Protocol Buffers de Google, comparé à un système de RPC comme les Sun RPC? **(1 point)**

Protocol buffers est un protocole simple pour l'envoi de messages utilisé pour les appels de procédure à distance chez Google. Il peut fonctionner avec plusieurs langages et se distingue par le fait que le contenu des messages est compact et auto-descriptif. En effet, un message est une structure de donnée composée d'une séquence de champs. Chaque champ est numéroté et typé. On peut donc recevoir un message et décoder ses champs (séquence de: numéro de champ, type et valeur) sans en connaître la déclaration complète au préalable. Ceci contraste avec SUN RPC ou CORBA où un message ne peut être interprété sans avoir la déclaration exacte de sa structure, ce qui est très contraignant si on veut faire évoluer un système sans mettre à jour simultanément tous les clients et serveurs.

Les entiers utilisent le nombre d'octets minimal, soit 1 octet pour un nombre dont la valeur est inférieure à 128. Les entiers étant très fréquents et leur valeur étant souvent assez petite, ceci fait que les messages sont souvent plus compacts que ceux de SUN RPC ou CORBA en dépit du fait qu'ils viennent avec un nombre variable de champs et avec l'information

de type. Le nombre variable de champs permet par exemple à un client de lire un message d'une version plus nouvelle du protocole pour un service, en ignorant simplement les champs ajoutés. Il permet aussi de lire un message d'une version plus ancienne du protocole, pour laquelle il manque des champs, à condition que celui qui a défini le format du message ait spécifié des valeurs par défaut.

c) Quelle est l'utilité pour les compagnies impliquées dans le consortium OpenStack.org de développer un logiciel comme OpenStack, alors qu'Amazon offre déjà un tel service de nuage élastique? **(1 point)**

Amazon offre la location de machines virtuelles en utilisant une infrastructure semblable à OpenStack. Cependant, le logiciel utilisé par Amazon lui est propre et n'est pas disponible pour les autres. Différentes compagnies collaborent donc pour développer un logiciel, OpenStack, qui leur permettra de développer un service comparable à celui d'Amazon, soit pour leurs besoins internes, soit pour leurs clients, possiblement en compétition avec Amazon. De plus, les membres de OpenStack ont décidé d'utiliser un modèle à code source libre afin de permettre à chaque utilisateur du logiciel d'avoir toute la liberté voulue pour modifier ou utiliser ce logiciel.

d) Vous voulez rapidement mettre sur pied un service Web de type commerce en ligne, qui peut s'adapter à la charge et qui soit tolérant aux pannes, sans avoir à implémenter ou gérer ces fonctions, à s'occuper des copies de sécurité des données (backup) ou à gérer et entretenir le matériel. Si vous décidez d'utiliser le nuage d'Amazon à cette fin, quels services spécifiques offerts par Amazon utiliserez-vous? Décrivez rapidement l'architecture de votre système avec les serveurs et répartiteurs réseau associés. **(1 point)**

Le service de machines virtuelles d'Amazon permet de ne pas avoir à gérer de matériel et d'avoir accès à un nombre variable d'instances pour s'adapter à la charge. Le répartiteur des requêtes sert à équilibrer la charge entre les différentes instances et peut aussi redémarrer les instances bloquées et augmenter ou diminuer le nombre des instances en fonction de la charge observée. Finalement, il est possible de ne mettre sur les instances que la partie interaction et présentation, sans stocker de données qui nécessiteraient des copies de sécurité. A la place, toutes les informations sur le stock et les commandes peuvent être maintenues dans une base de données, service qui est offert par Amazon. Toute la gestion des mises à jour du logiciel de base de données ou des copies de sécurité pour assurer la tolérance aux pannes est alors prise en charge par Amazon. Ainsi, le répartiteur reçoit les requêtes et les envoie à l'une des instances démarrées pour satisfaire à la charge. L'instance choisie s'occupe du dialogue avec l'utilisateur et prend toutes ses données (items à vendre, description, prix, contenu du panier d'achat...) et envoie toutes ses mises à jour (nouvelle commande par un usager...) au serveur de données.

Le professeur: Michel Dagenais