ECOLE POLYTECHNIQUE DE MONTREAL

Département de génie informatique et génie logiciel

Cours INF4410: Systèmes répartis et infonuagique (Automne 2013)
3 crédits (3-1.5-4.5)

CORRIGÉ DU CONTRÔLE PÉRIODIQUE

DATE: Mardi le 29 octobre 2013

HEURE: 9h30 à 11h20

DUREE: 1H50

NOTE: Toute documentation permise, calculatrice non programmable permise

Ce questionnaire comprend 4 questions pour 20 points

Question 1 (5 points)

Un service de forum de discussion réparti permet à des usagers de se connecter à un serveur géographiquement proche et de participer à une discussion en y lisant ou ajoutant des messages. Les serveurs collaborant pour offrir ce service forment un groupe de serveurs qui se diffusent les messages. Il est possible d'ajouter un message sur un nouveau sujet de discussion, de répondre à un message dans une discussion existante, ou de lire les messages disponibles en listant les messages sur de nouveaux sujets ou en listant les réponses à un autre message.

a) Chaque message doit avoir un identifiant unique utilisé afin de demander un message, de lister les réponses à un message, ou pour répondre à un message. De plus, il ne faut pas afficher une réponse à un message avant d'afficher le message auquel il répond. Donnez i) le nom utilisé pour un tel ordonnancement de message de groupe, et proposez une méthode efficace ii) pour créer un identifiant unique pour chaque message, et iii) pour assurer le bon ordonnancement du traitement des messages dans chaque serveur. **(2 points)**

Il s'agit d'un ordonnancement causal. Il est facile d'assigner un identifiant unique en juxtaposant un identificateur unique de noeud (adresse IP ou adresse MAC de la carte réseau) avec un compteur de messages. Ainsi, lorsqu'un message d'un serveur est reçu, on peut

attendre pour les messages de compteur moins élévé pour le même serveur. Toutefois, il faut surtout attendre avant d'afficher un message reçu que le message auquel il répond soit reçu lui aussi.

b) Les messages du forum peuvent être diffusés entre les serveurs par TCP, ou par UDP en multi-diffusion. Chaque envoi de paquet par UDP ou TCP occupe le réseau, qui est ici une ressource partagée, pendant 100 microsecondes, plus le temps de transmission à 100Mbits/s. Un message moyen demande un paquet de 1500 octets tout compris. Par TCP, en plus du paquet du message d'envoi, le récipiendaire envoie un paquet d'accusé de réception de 100 octets. Combien de messages du forum par seconde le réseau peut-il supporter avec UDP et avec TCP s'il y a 4 serveurs? Proposez une organisation efficace pour détecter les messages perdus et les retransmettre en UDP avec multi-diffusion, puisque cette fonctionalité est offerte par TCP mais pas pour UDP. **(2 points)**

*Un paquet demande 100us / 1000000us/s + n octets * 8 bits / octet / 100Mbits/s. Ceci donne donc .00022s pour 1500 octets et .000108s pour 100 octets. Avec UDP, chaque message demande .00022s, pour un taux maximal de 1s/.00022s = 4545.45 messages / seconde. Avec TCP, chaque serveur doit envoyer le messages aux trois autres séparément et recevoir un accusé de réception, ce qui prend 3 * (.00022s + .000108s) = .000984s, pour un taux maximal de 1016.26 messages / seconde. Un serveur saura si un message est manquant s'il reçoit un message de valeur supérieure de compteur, l'accusé de réception n'est donc pas essentiel. Il demeure un problème de latence. Si un serveur A envoie le message numéro 85 et qu'il soit perdu avant d'arriver à B, et qu'ensuite le prochain message, 86, ne soit pas créé avant une heure, un long délai se passera avant que le serveur B ne réalise que le message 85 lui manque. Pour pallier à ce problème, un serveur qui ne reçoit rien d'un autre pour plus que 5 minutes pourrait le contacter afin de vérifier le numéro du dernier message envoyé.*

c) Dans un sujet de discussion du forum, portant sur une question politique délicate, un message incendiaire a été ajouté par un inconnu se faisant passer pour une autre personne, un candidat à une prochaine élection, afin de le mettre dans l'embarras. Expliquez brièvement ce qui est requis pour qu'un tel système de discussion soit sécuritaire à cet égard. **(1 point)**

Il faut que le message et son contenu soit proprement authentifié. Ceci peut se faire avec une signature (somme de contrôle cryptographique pour le contenu du message) ou l'encryption avec la clé privée de l'envoyeur (tous pouvant décrypter avec la clé publique associée de cet envoyeur). L'authentification peut se faire pour la connexion au serveur, pour le contenu de chaque message envoyé au serveur, ou même mieux au niveau du lecteur qui consulte le message.

Question 2 (5 points)

a) Lors du premier travail pratique, vous avez étudié le temps requis pour effectuer un appel à distance avec Java RMI en fonction de la taille des arguments de l'appel. Décrivez comment varie la durée de l'appel en fonction de la taille des arguments. Comment peut-on expliquer cette variation en fonction de la taille des arguments? **(2 points)**

Le temps de traitement et d'envoi possède une composante constante et une composante qui croit avec la taille. On peut donc approximer la fonction entre la taille et le temps par une droite. Toutefois, certains effets non-linéaires peuvent se produire localement en raison de tailles fixes pour des tampons ou pour les paquets sur le réseau. Un octet de plus dans le même paquet requiert moins de temps supplémentaire qu'un octet qui force le débordement dans un deuxième paquet. Le temps de transmission sur le réseau est la composante la plus importante. Pour des tailles inférieures à environ 1K, la dimension d'un paquet, il y a très peu de différence. Ensuite, le temps croit linéairement avec la taille des arguments.

b) Pour les appels à distance, les sémantiques *au moins une fois* de même que *au plus une fois* peuvent être utilisées. Par ailleurs, le protocole UDP et le protocole TCP peuvent être utilisés. Java RMI utilise généralement *au plus une fois* sur TCP alors que les Sun RPC utilisent habituellement *au moins une fois* sur UDP. Montrez dans chaque cas combien de paquets sont échangés pour un appel où aucun paquet n'est perdu, et dans le cas où le paquet de réponse à l'appel à distance est perdu. On suppose que les arguments pour l'appel de même que la réponse sont sans problème plus petits que la taille maximale pour un paquet. Pour chaque scénario, montrez bien la fonction de chaque paquet envoyé et indiquez à chaque fois lorsque la fonction à distance est exécutée. **(2 points)**

Avec RMI, en supposant que la connexion TCP est déjà établie, il y aura le paquet pour la requête, un accusé de réception, le traitement par la fonction à distance, l'envoi de la réponse et un accusé de réception de la réponse. Il est possible que l'accusé de réception de la requête soit jumelé par TCP au paquet contenant la réponse, donnant donc 3 paquets plutôt que 4. Si le paquet de la réponse est perdu, l'accusé de réception de la réponse ne parviendra pas et le serveur renverra sa réponse.

Avec UDP, la requête est envoyée dans un paquet, la fonction à distance est exécutée et le paquet de réponse revient, un total de 2 paquets. Si le paquet de réponse est perdu, la requête sera envoyée à nouveau, la fonction à distance exécutée à nouveau et la réponse reviendra.

c) Le langage C# offre les appels à distance avec l'interface *Remoting*. Une des difficultés avec les appels d'objets à distance est la gestion automatique de la mémoire. En effet, un objet peut être détruit et son espace recyclé seulement lorsqu'il n'est plus utilisé. Il faut donc savoir s'il existe localement des objets rejoignables qui utilisent cet objet, ou s'il en existe dans des applications distantes. Comment peut-on maintenir à jour l'information sur les utilisateurs distants? Comment est-ce que le C# se prémunit contre les utilisateurs distants qui disparaissent (e.g. ordinateur qui saute) sans prévenir qu'ils n'utilisent plus un objet? **(1 point)**

Chaque fois qu'un client utilise une référence réseau, il en avertit le serveur et obtient un bail (droit d'utilisation de durée limitée) sur cet objet. Lorsqu'il cesse d'utiliser l'objet, il peut envoyer un message pour le relâcher, en quelque sorte résilier son bail. S'il veut utiliser l'objet plus longtemps, il peut renouveler le bail. Si le client disparaît sans prévenir, le serveur peut libérer l'objet sans problème à la fin de la période associée au bail.

Question 3 (5 points)

a) Dans une université, le service de fichiers est fourni par trois serveurs CODA. Les trois serveurs redondants contiennent les mêmes fichiers en réplication et la charge est bien répartie entre les trois serveurs. Chaque client en moyenne ouvre 2 fichiers par seconde et ferme un fichier modifié (à envoyer aux serveurs) par 4 secondes. Lors d'une ouverture, le client vérifie si une copie est déjà présente et si elle est à jour (il n'a pas reçu de notification d'invalidation). Sinon, une copie du fichier est prise auprès d'un des serveurs. Lorsqu'un fichier est écrit sur un serveur, le serveur doit envoyer des messages de notification si d'autres clients en ont présentement une copie. Cependant, pour simplifier le problème, on suppose que peu d'usagers accèdent des fichiers différents et les messages de notification ou de validation seront rares et pourront être négligés. On suppose aussi que tous les fichiers ont une longueur inférieure à 64KiO et leur lecture ou écriture constitue un seul accès. Lors d'une ouverture d'un fichier sur un client, une copie du fichier est déjà présente dans 60% des cas et à jour dans 90% de ces cas. Si 200 clients sont actifs en parallèle, combien d'accès en lecture et en écriture par seconde chaque serveur verra-t-il? Si un disque permet d'effectuer un accès en 15ms, combien de disques devrait-on avoir sur chaque serveur, en supposant que la charge est bien répartie entre les disques. **(2 points)**

*Chaque client demande 0.25 écriture par seconde vers chaque serveur. Les lectures représentent .4 + .6 * .1 = .46 des ouvertures, soit 2 * .46 = .92 lectures par seconde sur un des serveurs, ou en moyenne 0.92 / 3 = .3066 par serveur par seconde. Le total est donc de .25 + .3066 = 0.5566 accès par seconde par client. A 200 clients, ceci donne sur chaque serveur 0.5566 * 200 = 111.33 accès/seconde. Un disque permet 1 / .015s = 66.66 accès / seconde. Il faut donc 111.33 accès /seconde / 66.66 accès / seconde / disque = 1.67 disque, donc 2 disques par serveur.*

b) Une compagnie offre des vidéos sur demande par Internet. Elle peut soit avoir des serveurs extrêmement bien connectés, soit utiliser les équipements et ressources des clients pour répartir la charge à l'aide de protocoles comme bittorrent. Un gros film doit sortir et la compagnie s'attend à voir 1000 clients le télécharger la première journée. Le film a une taille de 4GiO. La compagnie obtiendra 2$ par copie mais doit payer $3 par Megabit/s de bande passante pour la journée. Une solution alternative est de permettre à 100 clients de prendre une copie gratuitement, à condition qu'ils agissent comme serveurs (pairs) bittorrent pour les autres clients. On suppose que les téléchargements peuvent être uniformément répartis sur les 24 heures de la journée. Laquelle des deux solutions sera la plus payante pour la compagnie? Justifiez. **(2 points)**

*Dans le premier cas, il faut permettre le téléchargement de 4GiO * 1000 * 8 bits / octet / (24 heures * 3600 secondes / heure) = 397Mbit/s. Le revenu sera donc de $2 * 1000 clients - 397Mbits/s * 3$ / Mbit/s = $809. Dans le second cas, il faudra 4GiO * 100 * 8 bits / octet / (24 heures * 3600 secondes / heure) = 39Mbit/s et le revenu sera de $2 * 900 clients - 39Mbits/s * 3$ / Mbit/s = $1683. La seconde solution sera donc plus avantageuse.*

c) Avec le Global File System, plusieurs noeuds peuvent accéder directement des unités de stockage habituellement connectées par FibreChannel. Ces unités de stockage reçoivent de ces noeuds des commandes simples pour lire et écrire des blocs. Comment est-ce que cela

se compare avec un système de fichiers comme Lustre? Quels sont les avantages de cette approche? **(1 point)**

Avec Lustre, le processeur du serveur de fichiers est directement associé à ses disques. Si le processeur ou l'unité de disques sont défectueux, le serveur n'est plus accessible. Avec GFS, l'unité de disque peut être connectée à plusieurs processeurs qui se coordonnent pour mettre à jour les blocs de disque de manière cohérente afin de modifier le système de fichiers dessus. Si un processeur est en panne, un autre processeur connecté à la même unité de disque peut poursuivre le travail. Les processeurs sont généralement connectés à l'unité de disque via un réseau très rapide de type Fiber Channel.

Question 4 (5 points)

a) Un processus implémente un service de nom de manière récursive. Dans 90% des cas, le processus peut répondre immédiatement après 0.5ms de temps CPU. Autrement, il doit faire une requête à un serveur de nom plus haut dans la hiérarchie, ce qui prend 0.3ms de temps CPU et aussi un temps d'attente pour la réponse. Ce temps d'attente est de 7ms dans 50% des cas, 25ms dans 30% des cas et 75ms dans 20% des cas. Combien de requêtes par seconde ce service peut-il soutenir avec un seul fil d'exécution qui traite les demandes séquentiellement? Combien peut-il en soutenir avec de nombreux fils d'exécution (mais un seul CPU)? Combien de fils d'exécution doit-on avoir au minimum? **(2 points)**

*Le temps d'attente moyen est de .5 * 7ms + .3 * 25ms + .2 * 75ms = 26ms. Pour un traitement en séquentiel, le temps d'une requête sera en moyenne de .9 * .5ms CPU + .1 * (.3ms CPU + 26ms Attente) = .48ms CPU + 2.6ms Attente = 3.08ms. On peut donc soutenir 1000ms / 3.08ms = 324.67 requêtes / seconde. S'il y a toujours des fils d'exécution disponibles, le facteur limitant est le CPU et on peut soutenir 1000ms / .48ms = 2083.3 requêtes / seconde. Il faut 1 fil pour le CPU et 2.6 / .48 = 5.41 pour les requêtes en attente, soit un total de 6.41 donc 7.*

b) Pour une application particulière, vous désirez qu'une mise à jour du service de nom soit faite de manière atomique. Deux clients ne devraient jamais en même temps voir auprès de serveurs de noms différents l'ancienne et la nouvelle adresse associées à un nom. Pour ce faire, vous devez effectuer des requêtes auprès du serveur responsable de ce nom (ajouter, effacer ou modifier un attribut comme l'adresse IP associée à un nom), tout en sachant que cette valeur peut être maintenue en cache pendant un certain temps (spécifé par l'attribut TTL) par les autres serveurs de noms. Indiquez comment vous pourriez obtenir une telle mise à jour atomique dans ce contexte? **(2 points)**

S'il ne s'agissait que de serveurs répliqués, il serait possible d'enlever la valeur actuelle sur chaque serveur dans un premier temps, et d'insérer la nouvelle valeur sur tous les serveurs ensuite, pour obtenir le résultat désiré. La difficulté réside dans le fait que des valeurs sont maintenues en cache par les autres serveurs. Si le temps de validité (TTL) est de 24h, on pourrait enlever la valeur maintenant et ne pas insérer la nouvelle valeur avant 24h, au moment où il n'existerait plus de copie de l'ancienne valeur en cache. Il est aussi possible de diminuer le temps de validité graduellement (le diminuer d'une heure à chaque heure) de

sorte que l'ancienne valeur demeurerait valide dans les cache jusqu'à peu de temps avant le changement, au lieu d'avoir un trou qui pourrait aller jusqu'à 24h selons les disponibilités en cache.

c) Une entreprise veut se doter d'un annuaire comme service de nom pour ses listes d'ordinateurs, ses listes d'usagers, etc. Un premier ingénieur informaticien propose d'utiliser le protocole LDAP et le serveur OpenLDAP. Un autre collègue est d'avis qu'un service de nom n'est rien d'autre qu'une base de donnée et propose de simplement mettre une base de donnée de type SQL avec un service d'accès à distance comme ODBC. Qu'en pensez-vous? Qu'est-ce qu'un service de nom offre qui lui est spécifique? **(1 point)**

Un serveur de nom comme LDAP vient avec un protocole, un API et divers paramètres (cache en mémoire) qui ont été conçus et optimisés pour agir comme serveur de nom. Le déploiement d'un serveur LDAP est donc en général plus simple et plus efficace dans ce contexte. Un service de base de donnée demandera plus de configuration pour arriver à un service qui sera comparable mais possiblement moins efficace. Ceci dit, OpenLDAP peut utiliser une basse de donnée comme service de stockage sur disque pour son annuaire.

Le professeur: Michel Dagenais

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 DU CONTRÔLE PÉRIODIQUE

DATE: Vendredi le 7 novembre 2014

HEURE: 9h30 à 11h20

DUREE: 1H50

NOTE: Toute documentation permise, calculatrice non programmable permise

Ce questionnaire comprend 4 questions pour 20 points

Question 1 (5 points)

a) Les techniciens du département doivent copier une nouvelle version de l'image des postes de laboratoire, à partir d'un serveur, vers chacun des 150 postes. L'image occupe 45×10^9 octets. Ils utilisent le protocole UDP en multi-diffusion et chaque poste envoie un accusé de réception négatif pour chaque paquet manquant. Ainsi, chaque paquet est numéroté et les stations redemandent les paquets manquants lorsqu'ils réalisent qu'il leur manque un paquet dans la séquence. On néglige l'espace requis pour les entêtes et la numérotation des paquets. La probabilité de perdre un paquet, (qu'il ne se rende pas à un poste donné), est proportionnelle à sa longueur $po \times n$ où po est la probabilité par octet et vaut 10^{-10}, et n est la longueur des paquets et doit être compris entre 1 et 10^8. Si la longueur des paquets est de 10^7 octets, combien d'accusés de réception négatifs seront reçus suite à l'envoi initial de l'image, (avant la retransmission des paquets manquants qui pourraient à leur tour être perdus). **(2 points)**

Les 45Go se décomposent en 4500 paquets de 10Mo. La probabilité qu'un client ne reçoive pas un paquet donné est de $10^{-10} \times 10^7 = 10^{-3}$. *Ainsi, sur les 4500 paquets, 4.5 seront manquants pour chacun des 150 postes et un total de* $150 \times 4.5 = 675$ *accusés de réception négatifs seront envoyés.*

b) L'Internet, comme la plupart des réseaux, est considéré non fiable. Des messages peuvent être interceptés, lus, enlevés, ajoutés ou modifiés. Pour pallier à cela, divers mécanismes comme la double encryption, avec des clés privées et publiques, peuvent permettre d'authentifier l'envoyeur et de s'assurer que seul le destinataire prévu peut lire le message. Est-ce suffisant dans tous les cas de s'assurer que si le message se rend, il est authentique et lisible seulement par son destinataire? Sinon, quels sont les problèmes possibles et quelles autres précautions seraient requises? **(2 points)**

Tel qu'énoncé dans la question, la double encryption à clé publique assure que le message lu, s'il n'a pas été modifié, provient bien de l'envoyeur et ne peut être lu que par le destinataire. Dans le cas le plus simple, on suppose que chacun a obtenu par contact sécuritaire la clé publique de l'autre. La clé publique peut aussi venir sous la forme d'un certificat, message émis par une autorité pour laquelle nous avons obtenu directement la clé publique. Le problème est que certaines autorités ne sont pas assez soigneuses et se sont fait voler (ou ont vendu?) leurs clés, permettant à des gens malveillants de créer de faux certificats. Il arrive aussi qu'on ne fasse que lire du site de notre correspondant sa clé publique. Dans un tel cas, une attaque par interposition serait assez facile à opérer.

Donc, le problème posé est, en supposant que la double encryption fonctionne, que doit-on ajouter au message de base pour ne pas se faire jouer de tour. Evidemmment, un attaquant qui coupe le fil réseau peut nous empêcher de recevoir quoi que ce soit. En fait, il faut s'assurer que le message envoyé arrive bien à ce moment et qu'il n'a pas été modifié. Par exemple, quelqu'un qui espionne votre lien de communication et voit que vous avez fait une commande par Internet pourrait rejouer ce message à répétition pour générer plusieurs commandes et vous embêter. Pour éviter cela, il faut que le message vienne avec l'heure et la date ainsi qu'un numéro de séquence et une somme de contrôle. Ainsi, si le message est modifié, la somme de contrôle sera erronnée, si le message est retardé significativement, la date et l'heure le signaleront, et si le message est rejoué, ceci sera facilement détecté avec le numéro de séquence. Celui qui modifie

le paquet encrypté n'est pas capable de changer directement de manière spécifique ces champs ajoutés au message. Dès que le message est modifié, l'incohérence sera quasi toujours visible (avec une probabilité reliée au nombre de bits de la somme de contrôle).

c) Un nouvel atelier de fabrication doit être installé et contiendra une fraiseuse à commande numérique, équipement dispendieux, sujet à l'usure, et avec une longue durée de vie. Cette fraiseuse constitue un petit système réparti avec un ordinateur central et quatre ordinateurs de commande, un pour le moteur de chacun des axes X, Y et Z, ainsi qu'un pour le moteur de la fraiseuse qui fait tourner l'outil de coupe. Un des administrateurs de l'atelier insiste pour que les ordinateurs de commande soient ouverts et leur logiciel à code source libre, alors qu'un deuxième insiste pour que simplement le protocole utilisé pour parler à ceux-ci soit ouvert. Finalement, le troisième préfère que tout soit fermé et secret pour éviter la tentation de jouer dedans et pour minimiser les risques de sécurité informatique. Quelle est l'utilité d'avoir un système ouvert à code source libre? Un système avec un protocole ouvert? **(1 point)**

Au moment de choisir entre deux types d'appareils pour un achat, un système entièrement ouvert à code source libre offre un maximum de flexibilité quand vient le moment de réparer l'appareil ou même de l'améliorer. Il est possible d'effectuer la réparation soi-même ou de choisir un fournisseur parmi plusieurs pour effectuer la réparation ou la modification. Un protocole ouvert permet de changer chacun des ordinateurs de manière modulaire et permet d'acheter l'ordinateur de remplacement en choisissant un fournisseur parmi plusieurs. Autrement, si le système est entièrement fermé, le seul choix est souvent de faire réparer ou modifier l'appareil par son fournisseur original. Celui-ci, en situation de monopole, pourra charger un prix très élevé, puisque rendu là le seul autre choix pour son client est de le faire réparer par lui ou d'acheter un nouvel appareil.

Question 2 (5 points)

a) Vous devez réaliser un service de fichiers simple à l'aide de Java RMI. Le client peut faire les appels `fp->lire(nom)` qui retourne `contenu`, et `fp->ecrire(nom, contenu)`. L'argument `nom` (nom du fichier) est de type `String`, et `contenu` (contenu du fichier) est de type `byte[]`, alors que `fp` est une instance qui offre l'interface `Fichier`. Le nom du fichier est par rapport au répertoire courant du processus qui offre ce service. Ecrivez l'interface `Fichier`, et la classe `FichierServant` (qui fournit les méthodes lire et écrire sur le serveur). La classe `FichierServant` lit ou crée et écrit en conséquence les fichiers demandés dans le répertoire courant. Pour simplifier le problème, vous n'avez pas à gérer les erreurs et les exceptions, montrer les inclusions (import) ou fournir les destructeurs ou constructeurs. Vous ne devez écrire que l'interface `Fichier` et la classe `FichierServant`, sans fournir le code qui utilise ces dernières pour instancier et exporter le service, ni le code pour implémenter le client. **(2 points)**

```
public interface Fichier extends Remote {
  byte[] lire(String nom);
  void ecrire(String nom, byte[] contenu);
```

```
}

public class FichierServant extends UnicastRemoteObject
    implements Fichier{

  public byte[] lire(String nom) throws RemoteException {
    byte[] contenu;
    FileInputStream f = new FileInputStream(nom);
    return f->read(contenu);
  }

  public void ecrire(String nom, byte[] contenu) throws RemoteException
    FileOutputStream f = new FileOutputStream(nom);
    f->write(contenu);
  }
}
```

b) Un des premiers systèmes d'appel à distance largement utilisés est SUN RPC. Il est maintenant disponible avec TCP ou UDP. En supposant qu'avec TCP la connexion est maintenue pour plusieurs appels à distance, les coûts de création et de destruction de la connexion peuvent facilement être amortis sur de nombreux appels. Par contre, pour chaque paquet envoyé, on suppose que TCP envoie un accusé de réception. Un appel RPC simple est effectué entre deux programmes et aucun paquet n'est perdu. Combien de paquets seront transmis si UDP est utilisé? Si TCP est utilisé? Les SUN RPC utilisent une sémantique *au moins une fois*, quel comportement auquel il faut faire attention cela peut-il causer sur UDP? sur TCP? **(2 points)**

Avec UDP, la requête est envoyée dans 1 paquet et la réponse revient aussi dans 1 paquet. Avec TCP, en plus de ces 2 paquets, il y aurait deux accusés de réception. En pratique, l'accusé de réception de la requête pourrait souvent être inclus dans le paquet qui contient la réponse. Si une seconde requête suit peu de temps après, l'accusé de réception de la réponse pourrait même être inclus dans la requête suivante.

Avec la sémantique au moins une fois, si la réponse n'est pas reçue, la requête peut être refaite jusqu'à obtention d'une réponse. Il pourrait donc arriver que la même requête soit soumise plusieurs fois à l'application serveur, par exemple si c'est la réponse qui avait été perdue par le réseau. Pour cette raison, le programmeur d'un service SUN RPC, en raison de UDP, doit établir son API et programmer le serveur de manière à s'assurer que les requêtes dupliquées ne causent pas de problème. Ainsi, le service NFS utilise des commandes d'écrire un contenu à une position donnée, ce qui est idempotent, donnant le même résultat si effectué plusieurs fois; ceci n'aurait pas été le cas pour une écriture qui demanderait d'ajouter à la fin du fichier, par exemple. Avec TCP, le protocole ajoute un numéro de séquence aux paquets et peut ainsi éliminer les doublons. De plus, TCP s'occupe de préserver les paquets envoyés pour lesquels les accusés de réception n'ont pas été reçus et de maintenir les numéros de séquence des octets contenus. Il peut ainsi facilement gérer les demandes de retransmissions sans que le client ou le serveur n'aient besoin de faire quoi que ce soit de particulier. Ainsi, les problèmes de requêtes dupliquées ne peuvent se présenter avec TCP.

c) Quels sont les avantages respectifs des systèmes d'appels à distance CORBA et Java RMI? Expliquez? **(1 point)**

Java RMI est plus simple à mettre en oeuvre car bien intégré au langage et demandant peu d'information additionnelle pour rendre l'accès aux procédures disponible à distance. Toute la gestion des objets passés en argument est aussi passablement automatisée (sérialisation versus par référence, gestion de la mémoire...) et donc simplifiée. CORBA a l'avantage, contrairement à RMI, d'être normalisé pour fonctionner et interopérer entre plusieurs langages informatiques. On peut donc vraiment découpler le client et le serveur au niveau des paramètres d'implémentation. CORBA fournissant un accès de plus bas niveau aux différents mécanismes d'appel à distance et de gestion de la mémoire, il est plus facile d'avoir le plein contrôle afin de mettre au point et d'optimiser les applications.

Question 3 (5 points)

a) Un serveur de fichiers sert de nombreux clients. Chaque client lit un fichier de code source (toujours le même), l'édite pendant 5 minutes, le sauve (réécrit son contenu), le compile et l'exécute. Le fichier de code source occupe 16Kio. La compilation demande de lire 128Kio de fichiers (les 16Kio du fichier de code source, et les 112Kio de 8 fichiers d'entête ou de librairies qui sont les mêmes d'une compilation à l'autre) et d'écrire un exécutable de 64Kio. Finalement, l'exécution demande de lire l'exécutable qui vient d'être écrit. Toutes ces opérations de compilation et d'exécution, (hormis l'édition par l'usager), prennent un temps très court (i.e., moins d'une seconde). Le protocole NFS 3 est utilisé pour ce service de fichiers. Il n'y a pas d'autres accès que ceux décrits ici (e.g., personne qui modifie les fichiers d'entête ou de librairies), et beaucoup de mémoire est disponible pour conserver localement une copie des fichiers lus ou écrits récemment par NFS. On peut donc déduire de cette description l'état de chaque fichier accédé: absent, en cache localement et validé depuis plus que 3 secondes, ou depuis moins que 3 secondes. Contrairement aux lectures ou aux écritures qui se font un bloc à la fois, la validation fournit la date de dernière modification du fichier au complet, et une seule requête de validation couvre tous les blocs du fichier. Chaque requête au serveur, lecture de 4Kio, écriture de 4Kio ou validation prend 2ms de temps CPU sur le serveur. En plus, chaque lecture prend 8ms de disque 30% du temps, chaque validation prend 8ms de disque 10% du temps, alors que chaque écriture prend 8ms de disque à chaque fois. Combien de clients ce serveur peut-il supporter au maximum s'il contient un CPU et un disque? **(2 points)**

Un cycle du client dure 5 minutes. Pendant ce cycle, il devra: lire 16Kio, écrire 16Kio, lire 128Kio, écrire 64Kio, lire 64Kio. Toutefois, tous ces fichiers soient ne changent pas (entête et librairies) soient viennent d'être écrits par ce client et n'ont pas besoin d'être lus. Par contre, certaines validations sont requises. La lecture du fichier à éditer ne demande pas de validation puisqu'il vient d'être sauvé / compilé / exécuté moins de 3 secondes auparavant. Au moment de la compilation, le fichier de code source vient encore là tout juste d'être modifié et n'a pas à être validé. Par contre, les 8 fichiers d'entête et de librairies doivent être validés puisque cela fait 5 minutes. L'exécutable vient d'être écrit et n'a donc pas besoin d'être relu ni validé. Le résultat est donc qu'à chaque cycle le client va écrire 16Kio + 64Kio = 80Kio (20 pages) et demander la validation de 8 fichiers. Cela requiert $(20+8) \times 2ms = 56ms$ *de temps CPU et*

$20 \times 8ms + 8 \times .10 \times 8ms = 166.4ms$ *de disque. Le disque sera le facteur limitant et le serveur pourra supporter un maximum de* $5min \times 60s/min \times 1000ms/s/166.4ms = 1802$ *clients.*

b) Un centre de recherche se prépare pour une expérience qui générera un très grand débit de données. On compte 5000 capteurs, et chacun génère 100Mbit/s et est connecté sur une prise réseau 1Gbit/s. Toutes les prises réseau sont commutées et connectées à un ensemble de serveurs. Chaque serveur est connecté à l'aide de deux cartes réseau 10Gbit/s sur deux prises qui peuvent soutenir le même débit. Le bus de données de chaque serveur a une capacité de 16Goctet/s. Chaque ordinateur a 16 disques connectés sur son bus, chaque disque ayant une bande passante de 100Moctets/s en régime continu. L'idée est de bien répartir la charge entre tous les disques sur tous les serveurs, afin de supporter tous ces capteurs avec le nombre minimal possible de serveurs. Quel est ce nombre minimal de serveurs requis? Quel système de service de fichiers pourrait typiquement être utilisé pour une telle organisation? **(2 points)**

Les 5000 capteurs généreront $5000 \times 100Mbits/s/8bits/o = 62.5Go/s$. *Chaque serveur peut passer* $2 \times 10Gbit/s/8bits/o = 2.5Go/s$ *par le réseau, 16Go/s sur son bus de données et écrire* $16 \times 100Mo/s = 1.6Go/s$ *sur ses disques dans le meilleur cas. Le facteur limitant est donc l'écriture sur les disques. Il faudra* $62.5Go/s/1.6Go/s = 39.06$ *serveurs (40) pour soutenir ce débit. Le système de fichiers Lustre est très utilisé pour ce type d'application puisqu'il est optimisé pour écrire les données d'un même fichier sur un grand nombre de disques en parallèle répartis sur plusieurs serveurs. De plus, comme Lustre est à code source ouvert, il est possible de l'adapter et de l'optimiser pour une application exigeante particulière.*

c) Donnez un exemple d'application où BitTorrent est beaucoup plus intéressant que Gnutella et un autre où Gnutella est plus intéressant. Expliquez. **(1 point)**

Pour le transfert avec peu de latence de très gros fichiers vers un très grand nombre de clients, par exemple la sortie du DVD pour la nouvelle version d'Ubuntu, BitTorrent est très intéressant. En effet, dès qu'un morceau du fichier a été envoyé à un client, celui-ci peut commencer à le redistribuer à d'autres clients. Plus encore, le serveur original priorise l'envoi de morceaux peu répandus pour favoriser les gains en parallélisme. Le principal avantage de Gnutella est sa structure complètement distribuée et adaptative. Il n'y a pas de structure ou de configuration initiale requise, les noeuds et hypernoeuds se choisissent eux-mêmes et les connexions se découvrent peu à peu. Gnutella est donc particulièrement indiqué lorsqu'il n'y a pas d'autorité centrale, que ce soit parce que personne ne veut prendre ce rôle, soit parce que l'organisation est perturbée suite à une catastrophe naturelle, soit parce que tout noeud central pourrait être la cible de poursuites.

Question 4 (5 points)

a) En bourse, il existe des règles sur la divulgation d'information. Un employé qui a de l'information privilégiée sur la situation de sa compagnie ne peut pas utiliser cette information pour jouer à la bourse et avoir un avantage sur les non initiés. Lorsqu'un spéculateur fait un gros coup en bourse, une enquête est souvent effectuée pour voir s'il a reçu de l'information privilégiée illégalement. Vous devez concevoir le système de communication par messages d'une compagnie de manière à aider ces enquêtes. En effet, on veut pouvoir facilement prouver qu'il n'y a pas

eu de chaîne de messages liant une information privilégiée et un ordre d'achat. Par exemple, supposons que A reçoit une telle information, envoie ensuite un message à B qui ensuite envoie un message à C. Si C a envoyé un ordre d'achat avant la réception de ce message, il n'y a pas de lien possible, alors que si l'ordre a été envoyé après, il y a un délit potentiel. On suppose que toutes les communications se font de manière électronique par messages, qu'il est possible d'ajouter une étiquette avec de l'information à chaque message, et qu'un journal des messages échangés est sauvegardé, avec les informations pertinentes comme l'étiquette du message et les temps d'envoi et de réception. Toutefois, une estampille de temps sur l'envoi et la réception de chaque message ne suffirait pas dans le journal pour les fins d'enquête, car les horloges des différents ordinateurs ne sont pas synchronisées. Proposez une méthode d'étiquetage ou de gestion des messages qui fera en sorte qu'il soit possible facilement de déterminer si deux événements (information reçue et ordre d'achat) peuvent être liés. **(2 points)**

On pourrait être tenté de suggérer un ordonnancement total avec un serveur central. Toutefois, il y aurait des faux positifs car cela donne seulement la chronologie stricte, alors qu'on veut voir le croisement (sur un noeud) des messages et donc se limiter aux cas où l'envoyeur avait reçu un message de quelqu'un qui avait reçu un message de quelqu'un... qui avait reçu une information privilégiée. Si à chaque envoi de message, une entrée de journal est écrite sur un serveur central décrivant le numéro de séquence sur le noeud, l'envoyeur et le destinataire, on pourrait a posteriori bâtir le graphe des envois et faire les vérifications voulues. Ce ne serait pas nécessairement élégant ni efficace.

La méthode classique pour calculer dynamiquement et valider l'ordonnancement causal est d'avoir un compteur logique d'événement dans chaque ordinateur pour quantifier leur progression logique dans le temps. De plus, on peut maintenir un vecteur de ces compteurs dans chaque ordinateur. Chaque fois qu'un message est envoyé, le compteur local est incrémenté et le vecteur de compteurs de l'ordinateur est envoyé. A chaque fois qu'un message est reçu, le vecteur de compteurs reçu est combiné avec le vecteur maintenu localement en prenant la valeur maximale pour chaque compteur. Ainsi, le vecteur de compteurs indique la valeur logique la plus élevée du compteur de chaque ordinateur pour laquelle de l'information peut avoir été reçue directement ou indirectement. En conséquence, si le vecteur de compteurs de l'ordre d'achat de C a une valeur de compteur qui est plus petite que celle qui correspond dans le vecteur de A pour le message avec l'information privilégiée, il ne peut y avoir eu de délit. Ceci suppose que le système d'exploitation et les intergiciels d'envoi de message sont entièrement contrôlés par les autorités de la compagnie, et qu'il n'y a pas de divulgation par des canaux autres comme les chuchotements près de la machine à café.

b) Un gros fournisseur Internet supporte les requêtes DNS de ses nombreux clients. Chaque client effectue en moyenne 1 requête DNS par seconde. Le serveur DNS peut répondre à i) 95% des requêtes en 1ms de CPU, ii) 4% des requêtes en 2ms de CPU et 10ms de disque, et iii) 1% des requêtes en 4ms CPU, 10ms de disque et 100ms d'attente après le réseau pour faire la requête à un autre serveur dans la hiérarchie. Cet ordinateur possède 8 CPU et 4 disques. Combien de clients peut-il supporter au maximum? Combien de fils d'exécution doit-il utiliser au minimum pour cela? **(2 points)**

Chaque requête demande $.95 \times 1ms + .04 \times 2ms + .01 \times 4ms = 1.07ms$ *de CPU,* $.04 \times 10ms + .01 \times 10ms = 0.5ms$ *de disque en plus de* $.01 \times 100ms = 1ms$ *de temps d'attente. Avec 8 CPU*

on peut supporter $8 \times 1000ms/s/1.07ms/r = 7476r/s$ *soit 7476 clients (à 1 requête par seconde par client). Les 4 disques permettent de supporter* $4 \times 1000ms/s/0.5ms/r = 8000r/s$ *ou 8000 clients. Les CPU sont le facteur limitant et 7476 clients pourront être servis. Il faut 1 fil par CPU et un nombre proportionné pour inclure les autres activités* $(1.07ms + 0.5ms + 1.0ms)/1.07ms \times 8 = 19.2$*, afin que toutes les ressources puissent être actives en parallèle. Elles ne seront pas bloquées en attente d'un fil libre alors que des ressources comme les disques ou les CPU sont libres. Ainsi avec 20 fils d'exécution on peut maintenir tous les CPU occupés même avec des requêtes en attente des disques ou du réseau. Pour être plus sûr, on en met généralement plus, par exemple le double.*

c) Une grosse entreprise veut installer un service de noms pour maintenir et rendre disponible les informations sur ses usagers et ses ordinateurs. Ceci lui permettra d'avoir un grand nombre de postes de travail pratiquement sans information de configuration, et n'importe quel usager pourra ainsi travailler à partir de n'importe quel poste de travail, tout en utilisant son mot de passe usuel et en accédant ses fichiers à partir du serveur. La compagnie hésite entre un service X500, LDAP et Active Directory. Que leur conseillez-vous? Quels sont les avantages et inconvénients de chacun? **(1 point)**

La solution X500 est complexe à mettre en oeuvre et n'a jamais réussi à percer. Il n'y a donc que très peu d'usagers et de fournisseurs et conséquemment peu d'avantages à l'utiliser. A la rigueur, on peut dire que cette solution est bien structurée. LDAP est la solution la plus utilisée sur les systèmes autres que Windows. Le protocole est ouvert et bien documenté et des implémentations de référence libres existent. C'est un système flexible et efficace. C'est la solution la plus souvent recommandée. Active Directory est bien intégré à l'environnement Windows et constitue donc un choix intéressant dans un environnement homogène Windows.

Le professeur: Michel Dagenais

ECOLE POLYTECHNIQUE DE MONTREAL

Département de génie informatique et génie logiciel

Cours INF4410: Systèmes répartis et infonuagique (Automne 2016)
3 crédits (3-1.5-4.5)

CORRIGÉ DU CONTRÔLE PÉRIODIQUE

DATE: Lundi le 31 octobre 2016

HEURE: 13h45 à 15h35

DUREE: 1H50

NOTE: Toute documentation permise, calculatrice non programmable permise

Ce questionnaire comprend 4 questions pour 20 points

Question 1 (5 points)

a) Une banque achète et vend des devises (e.g. dollars US ou Euros) à des taux très concurrentiels. Lorsque les prix sont mis à jour, il est essentiel que ces mises à jour rejoignent toutes les succursales en même temps et que la mise à jour se fasse simultanément. Il ne faut jamais que des prix différents, en raison d'une mise à jour en cours, soient vus dans deux succursales différentes, puisqu'alors des spéculateurs pourraient prendre de l'argent à la banque simplement en profitant de cette différence de prix. Il est aussi important de s'assurer que les messages sont valides (non modifiés) et authentiques (viennent bien de la maison-mère de la banque). Décrivez comment un tel service peut être implanté (messages envoyés et reçus pour une mise à jour, contenu de ces messages en termes de champs ajoutés et de transformation). **(2 points)**

Un message de groupe atomique permet de s'assurer que la mise à jour soit faite partout. On ne peut pas avoir une simultanéité parfaite de la mise à jour, mais on peut s'assurer que les deux prix ne sont jamais visibles simultanément. Une solution simple mais un peu inefficace est de faire un premier message atomique pour arrêter toute transaction (retirer l'ancien prix). Si ce message a fonctionné, un second message atomique transmet le nouveau prix. On peut bâtir là-dessus pour s'assurer plus efficacement que deux succursales ne puissent jamais montrer simultanément des prix différents. Un premier message pourrait prévenir qu'une mise à jour s'en vient et valider que toutes les succursales peuvent désactiver le prix courant. Une fois reçu la confirmation de chaque succursale, le serveur peut confirmer que la mise à jour est en route et faire désactiver le prix courant partout. Lorsque chaque succursale a confirmé avoir retiré le prix courant, le serveur peut envoyer un message (pas nécessairement atomique) à chaque succursale avec le nouveau prix. La première ronde de message, pour s'assurer que chaque succursale est prête, n'est pas vraiment requise. Cependant, elle évite de procéder à un changement si une des succursales n'est pas en mesure de répondre, ce qui bloquerait les autres avec un prix désactivé. Pour valider l'authenticité des messages, on peut utiliser un système d'encodage (par exemple à clés publiques avec double encodage pour assurer l'authenticité de l'envoyeur et la confidentialité du message). L'ajout de champs (avant encryption) avec une date, un numéro de séquence et une somme de contrôle permet de s'assurer que le message n'a pas été retardé, rejoué ou modifié.

b) Un gros fichier doit être transmis d'un serveur vers un grand nombre de clients. Chaque client est rejoignable soit par un réseau sans-fil commun, qui supporte la multi-diffusion et offre un débit de 100Mbit/s, soit par un réseau filaire commuté, dont chaque prise supporte un débit de 1Gbit/s, mais qui ne supporte que les communications point à point. Pour évaluer le temps requis pour les transmissions, on néglige la latence d'envoi sur le réseau, les paquets perdus et les bits requis pour les en-têtes de paquets, et on ne tient compte que du débit. Les choix disponibles sont soit de faire un envoi du fichier à tous les clients simultanément par multi-diffusion sur le réseau sans-fil, soit de faire une chaîne de distribution en arborescence en utilisant le réseau filaire (chaque client qui a une copie du fichier en envoie une copie à un autre client qui ne le possède pas encore). Quel est le temps pour transmettre un fichier de taille k (en bits) à n clients dans chacun des deux cas? Pour quelles valeurs de n et de k est-ce que chaque solution (sans-fil versus filaire) est plus rapide? **(2 points)**

Avec le réseau sans-fil, une seule transmission rejoint tout le monde en $t_{sf} = k/100M$. Avec le réseau filaire, après $t = k/1G$, 1 client est rejoint, ensuite 3, 7, 15... Donc $t_f = log_2(n+1) \times$

$k/1G$. Le ratio entre les deux est donc de $t_f/t_{sf} = log_2(n+1) \times k/1G \times 100M/k = log_2(n+1)/10$. Ainsi, pour de petites valeurs de n, le réseau filaire est plus rapide. Les deux seront identiques pour $t_f/t_{sf} = 1 = log_2(n+1)/10$ ou $log_2(n+1) = 10$ soit $n = 1023$. Ainsi, la valeur de k ne fait pas de différence et le réseau filaire est plus efficace lorsque n est inférieur à 512, équivalent lorsque n est entre 512 et 1023, et moins efficace autrement.

c) Quels sont les avantages et inconvénients de CORBA? En quoi pourrait-il être plus intéressant que SOAP? **(1 point)**

CORBA est un système complexe et donc un peu difficile à mettre en oeuvre. Par contre, il offre beaucoup de flexibilité, fonctionne avec de nombreux langages de programmation, et offre une excellente performance. SOAP est plus simple et offre aussi la possibilité de s'interfacer à plusieurs langages. Par contre, en raison de son format texte, les messages sont plus gros et il est moins efficace.

Question 2 (5 points)

a) Un service similaire à celui que vous avez programmé dans votre premier TP offre la lecture de fichiers à distance par Java RMI. Voici l'interface pour ce service qui retourne le contenu du fichier dont le nom est reçu en argument. Fournissez le contenu complet du fichier Server.java qui permettrait d'implémenter ce service en java, incluant la classe `Server`, les déclarations et les initialisations, mais sans les inclusions (`import`). **(2 points)**

```
package ca.polymtl.inf4410.tp1.shared;
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface ServerInterface extends Remote {
byte[] ReadFile(String name) throws RemoteException;
}


public class Server implements ServerInterface {
  public static void main(String[] args) {
    Server server = new Server();
    server.run();
  }

  public Server() { super(); }

  private void run() {
    if (System.getSecurityManager() == null) {
      System.setSecurityManager(new SecurityManager()); }
    try {
      ServerInterface stub = (ServerInterface)
```

```
        UnicastRemoteObject.exportObject(this, 0);
      Registry registry = LocateRegistry.getRegistry();
      registry.rebind("server", stub);
    } catch (ConnectException e) {
      System.err.println("Impossible de se connecter au registre RMI");
    } catch (Exception e) { System.err.println(e.getMessage()); }
  }

  @Override
  public byte[] ReadFile(String name) throws RemoteException {
    return Files.ReadAllBytes(Paths.get(name));
}
```

b) Dans le premier TP, un client, un serveur et un processus `registry` s'exécutaient et utilisaient Java RMI. Dans le serveur, un objet de type `Server` était initialisé puis enregistré auprès du `registry`. Java fonctionne avec un ramasse-miettes (garbage collector) et libère la mémoire associée aux objets qui ne sont plus utilisés. A quel moment est-ce que l'objet de type `Server` du serveur ne sera plus utilisé et le système de support à l'exécution saura qu'il peut récupérer la mémoire associée? Toujours dans le client et le serveur fournis dans le premier TP, on retrouve les deux appels distants suivants, `UnicastRemoteObject.exportObject(this, 0);` pour exporter l'objet de type `Server` vers le registry et `distantServerStub.execute(a, b);` lorsque le client demande au serveur d'effectuer un calcul. Expliquez quelle information est transmise pour chaque appel entre les deux processus en cause, et en particulier pour chaque argument (this, a, b) dites si une copie distincte est effectuée ou si un autre mécanisme est employé. **(2 points)**

Lors de l'appel pour enregistrer l'objet de type `Server` *dans le* `registry`*, le système RMI du serveur mémorise que cet objet est en utilisation car exporté. Lorsque le* `registry` *cessera d'utiliser l'objet de type* `Server` *et qu'il avertira le RMI du serveur que cet objet n'est plus utilisé, s'il n'y a pas d'autres utilisateurs pour l'objet, il pourrait être mis aux rebuts.*

Lors d'un appel de méthode à distance, on envoie une référence à l'objet ciblé, un numéro identifiant la méthode voulue et une référence aux objets en arguments s'ils implémentent l'interface Remote (comme this) ou une copie distincte s'ils implémentent l'interface serializable (a et b).

c) Dans les fichiers XDR pour les SUN RPC, il est possible de spécifier un numéro de version pour un service. Quel usage peut-on en faire? **(1 point)**

Il est possible de fournir une implémentation séparée pour chaque version et ainsi d'avoir différentes versions du même service (avec leur implémentation spécifique) disponibles.

Question 3 (5 points)

a) Lors de la migration d'une machine virtuelle (VM) d'un serveur physique à l'autre, il faut copier toutes les pages qui constituent son image en mémoire virtuelle d'un serveur à l'autre. On

suppose que les même fichiers sont accessibles des deux serveurs et un peu d'état du logiciel de virtualisation (taille négligeable) doit aussi être copié. On vous propose deux techniques possibles pour effectuer la migration tout en minimisant le temps pendant lequel la machine virtuelle est complètement arrêtée. La première méthode consiste en tout copier sans arrêter la VM mais en notant les pages qui ont été modifiées après que cette itération de copie ait commencé. On continue ainsi pour quelques itérations jusqu'à ce que le nombre de pages encore modifiées soit petit. A ce moment, on arrête la VM, copie les pages restantes et redémarre la VM sur le nouveau serveur. La seconde méthode consiste en commencer dès le départ en même temps la copie des pages et l'exécution sur le nouveau serveur. Cependant, lorsque l'exécution de la VM requiert une page qui n'est pas encore copiée, la VM bloque, priorise la copie de la page après laquelle elle attend, et reprend dès que cette page arrive. Avec cette seconde méthode, il n'y a pas d'arrêt complet de la VM mais plusieurs pauses en attente de pages accédées qui n'ont pas encore été copiées. L'image en mémoire virtuelle contient 2 000 000 pages. Les pages sont copiées d'un serveur à l'autre au rythme de 20 000 pages par seconde. Pour la première méthode, la VM en exécution modifie 2 000 pages par seconde et initie l'arrêt et le transfert des pages restantes après trois intérations (la première copie et deux itérations pour les pages modifiées depuis lors). Pour la seconde méthode, la VM accède 4 000 pages par seconde non encore copiées. Pour la première méthode, quel est le temps pendant lequel la VM est arrêtée? Quel est le temps total de migration? Pour la seconde méthode, quel est le nombre de fois et la durée des arrêts en attente d'une page accédée non déjà copiée? Quel est le temps total de migration? **(2 points)**

L'ensemble des pages peut être copié en 2 000 000 / 20 000 = 100s. Pendant ce temps, $100 \times 2000 = 200000$ *pages ont été modifiées, ce qui demande 200 000 / 20 000 = 10s pour la seconde itération. Pendant ce temps,* $10 \times 2000 = 20000$ *pages ont été modifiées, ce qui demande 1s pour la troisième itération. Pendant ce temps, 2000 pages ont été modifiées. La VM est alors arrêtée et il faut 2000 / 20 000 = 0.1s pour finaliser le transfert. La VM est donc arrêtée pendant 0.1s. La migration totale a pris 100s + 10s + 1s + 0.1s = 111.1s. Avec la seconde méthode, chaque page n'est copiée qu'une seule fois, ce qui demande 100s et constitue le temps total de migration. Il n'y a pas d'arrêt complet. Cependant, pendant le transfert, il arrive à* $100 \times 4000 = 400000$ *reprises que la page désirée ne soit pas disponible et qu'il faille attendre le temps du transfert d'une page pour qu'elle parvienne en priorité sur le nouveau serveur. Ce temps est d'environ 1 / 20 000 = 0.00005 si la page est transférée immédiatement. S'il faut attendre après le transfert en cours d'une page, on pourrait ajouter la moitié de cette valeur (entre 0 et 1, soit une moyenne de 0.5 page à attendre pour le transfert déjà en cours, avant de pouvoir transférer la page spécifique après laquelle on attend). Ces 400 000 attentes de 0.00005s totalisent 20s mais n'ont pas un gros impact car ces faibles latences ajoutées sont plus difficilement perceptibles par les clients de la VM que l'arrêt complet pendant 0.1s.*

b) Lors de vos travaux pratiques, vous utilisez un système infonuagique basé sur OpenStack. Au moment de créer une nouvelle instance de machine virtuelle, OpenStack doit choisir sur quel noeud physique la placer. Quelle composante de OpenStack est responsable de choisir le noeud physique et démarrer l'instance virtuelle? Comment se fait le choix du noeud physique? **(1 point)**

C'est le module Nova qui s'occupe d'exécuter les instances de machines virtuelles. Le choix se

fait premièrement en déterminant les noeuds physiques qui remplissent les requis (architecture, zone géographique, quantité de mémoire, nombre de coeurs...) et ensuite en sélectionnant celui qui a le meilleur pointage parmi ceux-ci selon divers critères (charge, nombre d'instances déjà présentes, puissance du noeud...).

c) La fonctionalité KSM (Kernel Same page Merging) a été ajoutée au noyau Linux pour aider la virtualisation. Que fait KSM? Pourquoi est-ce particulièrement intéressant pour la virtualisation? **(1 point)**

KSM vérifie les pages (en mode lecture seulement) dont le contenu est identique en mémoire virtuelle et les consolide en n'en conservant qu'une seule copie. Ainsi, si plusieurs machines virtuelles utilisent les même fichiers, par exemple les exécutables du noyau Linux, de librairies ou d'applications populaires, il est possible de n'avoir qu'une seule copie, partagée, en mémoire physique, comme c'est le cas lorsque tout s'exécute sur un seul ordinateur, sans virtualisation. Ceci permet donc d'atténuer un des surcoûts associés à l'utilisation de machines virtuelles, à savoir la plus grande utilisation de mémoire lorsque des copies redondantes de certains fichiers populaires sont conservées en mémoire par chaque machine virtuelle.

d) Sur Amazon EC2, trois types de services de stockage (disque) sont disponibles, quels sont-ils? Quels sont les composantes de OpenStack qui offrent les services équivalents? **(1 point)**

Sur Amazon EC2, on retrouve le stockage d'instance, qui disparait lorsque l'instance est arrêtée, le stockage de blocs (Elastic Block Store, EBS), qui demeure lorsque l'instance est arrêtée mais qui ne peut être associé qu'à une seule instance à la fois, et le stockage d'objets (Simple Storage Service S3) qui sont accessibles de plusieurs instances à la fois. Sur OpenStack, Cinder offre le stockage d'instance et de blocs, et Swift offre le stockage d'objets.

Question 4 (5 points)

a) Un serveur de disque reçoit des requêtes à partir de clients. Chaque client requiert en moyenne des données pour 10 megabits/s. Le réseau est entièrement commuté. Le serveur est connecté au réseau par une prise qui fournit 10 gigabits/s. Son bus a une capacité de 16 gigaoctets/s et 6 disques y sont connectés. Les disques actuels fournissent chacun 100 megaoctets/s. Combien de clients est-ce que ce serveur peut supporter maintenant? Combien de clients peut-il supporter si on remplace les disques actuels par des disques SSD qui fournissent 1 gigaoctets/s chacun? **(2 points)**

Chaque client requiert 10 megabits ou 10/8 = 1.25 megaoctets/s. Le serveur peut recevoir 10 gigabits ou 1.25 gigaoctets/s. Le bus a une capacité de 16 gigaoctets/s et les 6 disques $6 \times 100 = 600$ *megaoctets/s. Le facteur limitant est donc les disques qui peuvent soutenir 600 megabits/s / 1.25 megabits/s / client = 480 clients. Avec des disques SSD, on passe à 6 gigaoctets/s et le facteur limitant devient le réseau à 1.25 gigaoctets/s, ce qui donne 1.25 gigaoctets/s / 1.25 megaoctets/s / client = 1000 clients.*

b) Sur un client NFS, le système d'exploitation reçoit 2 lectures et 1 écriture de page de 4Kio par seconde en moyenne. Pour une lecture, la page cherchée ne se trouve pas sur le client dans 25% des cas, se trouve déjà sur le client et a été validée depuis moins de 3 secondes dans 40% des

cas, et se trouve sur le client mais a été validée depuis plus de 3s dans le reste des cas, soit 35%. Lorsqu'une page est présente mais sa validation est trop vieille, une revalidation suffit dans 60% des cas et une relecture doit être faite en plus (car la validation indique que la page a changé) dans 40% des cas. Sur le serveur, une demande de validation demande 2ms de CPU, une lecture demande 4ms de CPU et dans 30% des cas un accès disque de 12ms, et une écriture demande 5ms de CPU et 12ms de disque. Si 25 clients accèdent ce serveur qui comporte 1 CPU et 1 disque, quel sera le pourcentage d'utilisation du CPU? Du disque? **(2 points)**

Un client génère 1 écriture/s sur le serveur. Il génère aussi 2 lectures/s $\times$ (0.25 + 0.35 $\times$ 0.4) = 0.78 lectures/s. Finalement, un client génère 2 lecture/s $\times$ 0.35 = 0.70 validations/s. Ceci génère sur le serveur à chaque seconde pour chaque client: 5ms CPU et 12ms disque (1 écriture), 0.7 $\times$ 2ms = 1.4ms CPU (0.7 validation), et 0.78 $\times$ 4ms = 3.12ms CPU et 0.78 $\times$ 0.3 $\times$ 12ms = 2.808ms disque (0.78 lecture). Le total est donc de 9.52ms CPU et 14.808ms disque. Avec 25 clients, le taux d'utilisation sera de 25 $\times$ 9.52ms / 1000ms/s = 23.8% CPU et 25 $\times$ 14.808ms / 1000ms/s = 37.02% disque.

c) Avec le système de fichiers CEPH, la répartition des groupes de placement (PGID) sur les serveurs de fichiers est déterminée par un algorithme de hachage (CRUSH). Quel est l'intérêt d'un tel algorithme pour déterminer le placement sur un service de fichiers réparti? **(1 point)**

Cet algorithme permet de calculer directement, avec un calcul simple effectué sur le client, le serveur de fichiers sur lequel se trouve le PGID. Il n'est ainsi pas nécessaire de consulter un service quelconque afin d'obtenir cette information. Ceci enlève un goûlot d'étranglement potentiel (serveur de métadonnées pour le placement des PGID) et permet à CEPH de plus facilement se mettre à l'échelle avec un très grand nombre de clients et de serveurs.

Le professeur: Michel Dagenais

ECOLE POLYTECHNIQUE DE MONTREAL

Département de génie informatique et génie logiciel

Cours INF8480: Systèmes répartis et infonuagique (Automne 2018)
3 crédits (3-1.5-4.5)

CORRIGÉ DU CONTRÔLE PÉRIODIQUE

DATE: Lundi le 22 octobre 2018

HEURE: 13h45 à 15h35

DUREE: 1H50

NOTE: Aucune documentation permise sauf un aide-mémoire, préparé par l'étudiant, qui consiste en une feuille de format lettre manuscrite recto verso, calculatrice non programmable permise

Ce questionnaire comprend 4 questions pour 20 points

Question 1 (5 points)

a) Un service de message de groupe relie n ordinateurs. Quel est le nombre de messages envoyés (précisez pour chaque message s'il est en multi-diffusion ou à un seul destinataire) pour réaliser un message de groupe i) non fiable, ii) fiable, iii) atomique, iv) totalement ordonnancé. **(2 points)**

Pour un message de groupe non fiable, un seul message en multi-diffusion suffit. Pour un message de groupe fiable, un message en multi-diffusion rejoint chacun, et ensuite chacun des $n-1$ autres membres du groupe répond avec un message régulier d'accusé de réception. Pour un message atomique, on a un premier message en multi-diffusion pour envoyer l'information, ensuite $n-1$ messages des autres membres du groupe pour accepter l'envoi, et finalement un message de confirmation en multi-diffusion. Pour un message totalement ordonnancé, il faut un message au serveur d'ordonnancement et un message de réponse. Ensuite on envoie un message, étiqueté avec le numéro d'ordre obtenu, en multi-diffusion. Il n'est pas nécessaire d'avoir des accusés de réception de ces messages numérotés. En effet, si un client voit qu'il lui manque un message dans la séquence numérotée, il peut facilement déduire que le message correspondant lui manque.

b) Un client effectue une requête auprès d'un serveur. Le client prépare sa requête en 25ms, la requête transite sur le réseau en 1ms, elle est traitée en 8ms par le serveur qui ne fait que des calculs sur le CPU pendant ce temps et n'utilise aucun périphérique, la réponse transite sur le réseau en 1ms et elle est traitée sur le client en 15ms. Le client est alors prêt pour envoyer sa prochaine requête. Le serveur sert les requêtes avec un seul thread. Combien de requêtes par seconde est-ce qu'un client seul peut faire? Combien de clients ce serveur pourrait-il supporter avant de devenir saturé? **(2 points)**

Avec un seul client, le temps total est de 25ms+1ms+8ms+1ms+15ms = 50ms. Le client peut donc faire 1000ms/s / 50ms/requête = 20 requêtes / s. Le serveur n'est en fait occupé que pendant 8ms. Il peut donc servir 1000ms/s / 8ms/requête = 125 requêtes / s soit 125r/s / 20r/s = 6.25 donc 6 clients avant de devenir saturé.

c) Vous êtes en train de concevoir un service de multi-diffusion fiable. Vous avez le choix entre des accusés de réception positifs (accuser réception de chaque paquet reçu) ou négatifs (envoyer un message pour signifier chaque paquet manquant). Le meilleur choix dépend de la probabilité p qu'un client ne reçoive pas un paquet. Pour quelles valeurs de p est-ce que chacune des deux options sera préférable? **(1 point)**

Puisqu'on veut minimiser le nombre total de paquets envoyés, on prendra des accusés négatifs, si les paquets perdus sont peu probables ($0 \leq p \leq 0.5$), et des accusés positifs, si les paquets sont plus souvent perdus que non ($0.5 \leq p \leq 1$). On suppose ici que les accusés de réception étant très petits, il sont rarement perdus eux-mêmes.

Question 2 (5 points)

a) Un service d'interrogation de la base de donnée des employés utilise gRPC avec les définitions données plus bas. Un employé est embauché et ses coordonnées sont: id = 12, nom = "Haddock", prenom = "Archibald", adresse = "Chateau de Moulinsart", numero = 7654321. Quelle est la longueur du message de type EmployeFiche correspondant, étant donné l'encodage utilisé par protobuf dans gRPC? L'entreprise grossit et commence à embaucher des employés à l'international. Il faut donc ajouter le pays pour ceux dont l'adresse n'est pas au Canada. Comment peut-on ajouter ceci aux définitions actuelles? Est-ce que cela va demander de mettre à jour tous les logiciels client et serveur qui utilisaient ces définitions? **(2 points)**

```
service EmployeInfo {
  rpc EmployeEmbauche (EmployeFiche) returns (EmployeId);
  rpc EmployeNumero (EmployeId) returns (NumeroTelephone);
}
message EmployeId {
  int32 id = 1;
}
message NumeroTelephone {
  int32 numero = 1;
}
message EmployeFiche {
  int32 id = 1;
  string nom = 2;
  string prenom = 3;
  string adresse = 4;
  int32 numero = 5;
}
```

Pour tous les champs, le numéro de champ et le type peuvent entrer dans un seul octet. En effet, le type prend 3 bits et 4 bits restent disponibles dans le varint de 1 octet pour le numéro de champ; le numéro de champ le plus grand est de 5, ce qui prend 3 bits alors que 4 sont disponibles. Pour id, le nombre 12 requiert 1 octet, pour numero le nombre 7654321 requiert 23 bits, à 7 bits disponibles par octet dans un varint, cela veut dire 4 octets. Pour nom, la longueur prend 1 octet et les caractères 7 octets. Pour prenom, la longueur prend 1 octet et les caractères 9 octets. Pour adresse, la longueur prend 1 octet et les caractères 21 octets. Le total est donc de (1+1)+(1+1+7)+(1+1+9)+(1+1+21)+(1+4) = 50. Il suffirait d'ajouter un champ optionel, `string pays = 6 [default = "Canada"]` *à EmployeFiche. Ainsi, une nouvelle application qui cherche le pays trouvera Canada si rien n'est spécifié, même si l'autre partie n'a pas été mise à jour. Concrètement, les serveurs et les clients qui peuvent avoir à traiter des cas d'employés n'ayant pas le Canada comme pays doivent être mis à jour. Les autres n'ont pas à le faire, étant donné que les champs sont auto-identifiés avec gRPC et que la valeur par défaut de Canada est ajoutée.*

b) Une application de serveur de dessin est créée. Deux processus, P1 et P2, interagissent via Java RMI en utilisant les interfaces *Shape* et *ShapeList* définies ci-après. La méthode *allShapes* retourne un vecteur de *Shape*. Le processus P1 crée 4 formes, S1, S2, S3 et S4, du type *ShapeServant* qui est défini comme suit *ShapeServant extends UnicastRemoteObject implements Shape*. Ensuite P1 crée un dessin, SL1, du type *ShapeListServant* qui est défini comme suit *ShapeListServant extends UnicastRemoteObject implements ShapeList*. P1 ajoute S1, S2, S3 et S4 dans le dessin SL1 avec la méthode *addShape*. Avec le service *Naming*, P2 obtient une référence réseau (un proxy) à l'objet SL1, dénotée rrSL1. P2 crée alors un dessin, SL2, du type *ShapeListServant*. Il obtient avec la méthode *allShapes* de rrSL1 un vecteur et ajoute les 4 *Shape* du vecteur à l'objet SL2 avec la méthode `addShape`. Avec le service *Naming*, P1 obtient une référence réseau à SL2 dénotée rrSL2. Il obtient avec la méthode *allShapes* de rrSL2 un vecteur. Suite à ces opérations, dites quels objets réels et quelles références réseau sont contenus dans P1 et P2 pour les formes et les dessins (par exemple S1 pour l'objet réel et rrS1 pour une référence réseau à S1). **(2 points)**

```
public interface Shape extends Remote {
  GraphicalObject getAllState() throws RemoteException;
}

public interface ShapeList extends Remote {
  void addShape(Shape s) throws RemoteException;
  Vector allShapes() throws RemoteException;
}
```

Le processus P2 contient SL2, rrSL1, rrS1, rrS2, rrS3, rrS4. Le processus P1 contient S1, S2, S3, S4, SL1, rrSL2. Lorsque des références réseau à S1, S2, S3 et S4 sont reçues sur P1 par RMI suite à l'appel de méthode `allShapes` *de rrSL2 , elles sont automatiquement reconverties en des pointeurs aux objets réels, puisque ceux-ci sont des objets exportés par P1.*

c) Dans le cadre du premier travail pratique, la consigne suivante était spécifiée: "comme le serveur sera en mesure d'accepter des appels de plusieurs clients de façon simultanée, vous devez prendre les précautions nécessaires pour assurer la cohérence des données dans les structures partagées au sein du serveur". Expliquez comment il fallait implémenter ces précautions dans votre programme. **(1 point)**

Puisque plusieurs requêtes concurrentes de différents clients peuvent être traitées en parallèle sur différents threads, les méthodes qui accèdent les structures de données de l'objet doivent être protégées, par exemple avec l'attribut synchronized qui couple une prise de verrou à l'appel de la méthode. Autrement, si deux appels sont reçus en même temps par RMI de deux clients pour prendre le verrou sur un fichier, Les deux pourraient accéder en même temps la structure de donnée qui dit si le fichier est réservé pour un client, conclure que le fichier est libre, et le donner en même temps aux deux clients.

Question 3 (5 points)

a) Une machine virtuelle exécute une tâche urgente qui nécessite 1000 secondes de CPU sur un coeur (disponible à 100%) et le transfert par réseau de 200GiO de données (.2GiO par seconde de CPU à 100%); le transfert peut s'effectuer en parallèle avec les calculs sur le CPU. Cette machine virtuelle dispose sur le noeud physique actuel de 50% d'un coeur et d'une bande passante pour le réseau de 0.5GiO/s. Elle pourrait migrer vers un noeud physique moins occupé où elle bénéficierait d'un coeur à 100% et de 1.0GiO/s de bande passante de réseau. Toutefois, il faut qu'il reste de la bande passante disponible pour que la migration s'effectue (la migration est en plus basse priorité que l'aplication). De plus, la machine virtuelle subirait un délai associé à la migration, pendant l'arrêt total afin de finaliser la migration. L'image de la machine virtuelle à migrer occupe 8GiO. La migration prend un volume de bande passante correspondant mais ne consomme pratiquement aucun CPU. La machine virtuelle, pendant son exécution, modifie 0.2GiO de son image par seconde de CPU à 100%. La copie de l'image à migrer se fait en plusieurs itérations, jusqu'à ce que les modifications à transférer constituent moins de 0.5GiO; à ce moment, la machine est arrêtée et les dernières modifications sont transférées. Combien de temps prends la migration? Est-ce que la tâche urgente sera terminée plus tôt avec la migration? En combien de temps? **(2 points)**

Sur le noeud actuel, la tâche urgente prendra 1000s / 0.5 = 2000s de CPU. En parallèle, les entrées-sorties sur le réseau nécessitent 200GiO / 0.5GiO/s = 400s. Le CPU est le facteur limitant et cela requiert 2000s. Sur le nouvel ordinateur, le temps requis serait de 1000s, puisque le coeur est disponible à 100%. La migration demande de transférer 8GiO. L'application prend 0.1GiO/s à 50% de CPU. Il reste donc 0.4GiO/s pour transférer les 8GiO, ce qui demande 20s. Pendant 20s, 20s x .1GiO/s (à 50% de CPU) = 2GiO seront modifiés. Ceci prendra 2GiO/0.4GiO/s = 5s pour le transfert en seconde itération. L'itération suivante demandera 1.25s pour 0.5GiO. Il y aura ensuite 0.125GiO modifiés et la machine virtuelle sera arrêtée pendant 0.3125s afin de transférer ce restant, avant de démarrer sur le nouveau noeud. Le temps total pris par la migration est de 20s + 5s + 1.25s + .3125s = 26.5625s. Pendant 26.25s, la machine virtuelle s'est exécutée à 50% de CPU, il lui reste donc 1000s - 26.25s/2 = 986.875s à exécuter sur le nouveau noeud, pour un temps total de 26.5625s + 986.875s = 1013.4375s. La tâche urgente sera effectivement terminée bien plus vite avec la migration.

b) Trois machines virtuelles, A, B et C, s'exécutent sur un même noeud physique. Le noeud physique contient 4 coeurs et 4 disques. Chaque disque supporte 100 opérations d'entrée/sortie (IOP) par seconde. Chaque machine virtuelle sert des requêtes et répartit sa charge entre 4 coeurs virtuels et 4 disques virtuels. Les requêtes à la machine A prennent 50ms et 4 IOP, celle à la machine B 10ms et 8 IOP et celles à la machine C 100ms et 2 IOP. L'opérateur de la machine A a payé pour avoir une priorité absolue (même performance que si seul sur le noeud physique), celui de la machine B a payé pour une certaine priorité, et celui de la machine C a payé le minimum (la machine ne roule que si A et B ne font rien). Si A et B ne reçoivent aucune requête, combien de requêtes par seconde C peut-elle soutenir? Si A reçoit 40 requêtes par seconde et B 30 requêtes par seconde, combien de requêtes par seconde C peut-elle soutenir? **(2 points)**

Lorsque C est seule, elle peut soutenir (4 CPU x 1000ms/s) / 100ms/r = 40r/s au niveau du CPU et (4 disques x 100IOP/s) / 2 IOP/r = 200r/s pour les entrées-sorties. Elle peut donc effectuer

au maximum 40r/s. Lorsque A et B sont actives, il restera 4 CPU x 1000ms/s - 40r/s x 50ms/r - 30r/s x 10ms/r = 1700ms/s ainsi que 4 disques x 100IOP/s - 40r/s x 4 IOP/r - 30r/s x 8 IOP = 0 IOP/s. Tout les disques sont occupés à 100% et C ne pourra pas servir une seule requête!

c) OpenStack (avec virtualisation par KVM) et Kubernetes sont deux technologies très utilisées pour déployer des applications parallèles réparties. Peut-on rouler efficacement OpenStack au-dessus de Kubernetes? Kubernetes au-dessus de OpenStack? Kubernetes sur Kubernetes? OpenStack sur OpenStack? **(1 point)**

La virtualisation fonctionne assez bien à 1 niveau, en raison du support matériel dans les processeurs Intel et AMD. A deux niveaux, i.e. OpenStack sur OpenStack, le support matériel n'aide pas pour le second niveau et le surcoût est plus important, ce qui rend cela nettement moins efficace. Kubernetes est basé sur les conteneurs, ce qui n'implique pas de surcoût important, même lorsque 2 niveaux sont imbriqués. Toutes les solutions avec un seul (ou aucun) niveau de OpenStack sont donc efficaces.

Question 4 (5 points)

a) Un service de fichiers CODA est répliqué sur 3 serveurs (i.e. chaque fichier se retrouve en 3 copies). Chaque serveur possède 4 disques. Chaque disque peut effectuer 100 accès (lecture ou écriture) par seconde. Les clients, lors des ouvertures ou fermetures de fichiers, font des accès en lecture ou en écriture au serveur. Quel est le nombre maximal de lectures (s'il n'y a que des lectures) par seconde que pourrait soutenir ce service répliqué sur 3 serveurs, en supposant que la charge est répartie uniformément sur les serveurs et les disques, et que les disques constituent le facteur limitant? Quel est le nombre maximal d'écritures (s'il n'y a que des écritures)? Si on change pour un système avec 3 serveurs mais sans réplication, avec la charge uniformément répartie entre les 3, que devient le nombre maximal de lectures? Le nombre maximal d'écritures? **(2 points)**

En lecture, les serveurs opèrent en parallèle. Avec 3 serveurs de 4 disques à 100 accès/s, on a une capacité maximale de $3 \times 4 \times 100 = 1200$ *lectures par seconde. En écriture, chaque écriture doit être faite sur les 3 serveurs. On peut donc soutenir 3 fois moins d'écritures que de lectures soit 400 écritures par seconde. Si on enlève la réplication, la lecture reste la même, à 1200/s et les écritures deviennent semblables à 1200/s aussi.*

b) Une expérience a lieu au laboratoire du CERN. Pendant l'expérience, de nombreux noeuds reçoivent des données venant de nombreux capteurs et stockent cette information sur un système de fichiers réparti avec réplication en 3 copies. Tous les fichiers sont ouverts avant le début de la capture des données et chaque noeud sait donc déjà sur quels serveurs de fichiers il doit envoyer ses 3 copies. L'information générée sur chaque noeud est de 1GiO/s, et doit être stockée de manière répliquée sur les serveurs. Le service de fichiers est réparti sur 20 serveurs. Chaque serveur a une connexion au réseau de 20GiO/s et contient 4 contrôleurs de disque. Chaque contrôleur de disque est alimenté par un canal PCIe de 5GiO/s et est connecté à 8 disques. Chaque disque est capable de soutenir des écritures à un rythme de 2GiO/s. On suppose que la charge est parfaitement répartie entre les serveurs, les contrôleurs et les disques. Combien de noeuds est-ce que cette infrastructure est capable de supporter? **(2 points)**

Chaque noeud doit envoyer 3 x 1GiO/s = 3GiO/s. Sur chaque serveur, le réseau a une bande passante de 20GiO/s, et les contrôleurs de disques ont une bande passante de 4 x 5GiO/s = 20GiO/s. Pour chaque contrôleur, les disques une bande passante de 8 x 2GiO/s = 16GiO/s. Le facteur limitant est donc le contrôleur à 5GiO/s. Globalement, ce sont donc le réseau et les 4 contrôleurs qui l'un comme l'autre limitent la bande passante d'un serveur à 20GiO/s. Avec 20 serveurs, on peut soutenir 20 x 20GiO/s = 400GiO/s soit 400GiO/s / 3GiO/s = 133 clients.

c) Pour chacun de ces systèmes de fichiers répartis, NFS, AFS, CODA et CEPH, dites i) s'ils permettent la réplication en écriture, ii) s'ils permettent la mise à l'échelle à un vraiment très grand nombre de clients. **(1 point)**

CODA et CEPH permettent la réplication en écriture alors que NFS et AFS ne le permettent pas. Seul CEPH permet la grande mise à l'échelle en calculant la position du fichier de manière autonome avec un code de hachage, plutôt que d'avoir à consulter un serveur central pour cette méta-information.

Le professeur: Michel Dagenais

ECOLE POLYTECHNIQUE DE MONTREAL

Département de génie informatique et génie logiciel

Cours INF4410: Systèmes répartis et infonuagique (Hiver 2017)
3 crédits (3-1.5-4.5)

CORRIGÉ DU CONTRÔLE PÉRIODIQUE

DATE: Lundi le 27 février 2017

HEURE: 10h30 à 12h20

DUREE: 1H50

NOTE: Toute documentation permise, calculatrice non programmable permise

Ce questionnaire comprend 4 questions pour 20 points

Question 1 (5 points)

a) Un groupe de N ordinateurs s'échangent des messages de groupe, en utilisant un ordonnancement causal. Chaque membre a un numéro (0 à N-1) qui lui est associé dans le groupe. Ainsi, chaque ordinateur maintient un vecteur de N entrées qui contient, pour chaque membre du groupe, le numéro du dernier message reçu de ce membre et livré aux applications. Chaque ordinateur *i* numérote séquentiellement les messages qu'il envoie au groupe et place dans le vecteur à sa position (*i*) le numéro du dernier message créé. Lorsqu'un message est envoyé au groupe par l'ordinateur *i*, l'ordinateur *i* joint au message son vecteur de N entrées qui inclut le numéro du présent message en position *i*. Le vecteur est utilisé par chaque ordinateur du groupe qui reçoit le message afin d'ordonnancer causalement les messages avant de les livrer aux applications. Les messages suivants, dans un groupe de 4 ordinateurs, sont envoyés et reçus par les différents ordinateurs dans l'ordre spécifié. Pour chaque message, indiquez le vecteur qui sera associé et son contenu, et pour chaque noeud indiquez dans quel ordre les messages seront livrés aux applications. Tous les ordinateurs sont initialemnent rendus à envoyer leur premier message (1_i où *i* est le numéro de l'ordinateur qui envoie le message). **(2 points)**

Ordinateur 0: envoi de 1_0, réception de 1_1, envoi de 2_0, réception de 2_1, envoi de 3_0, réception de 3_1
Ordinateur 1: envoi de 1_1, réception de 1_0, envoi de 2_1, réception de 2_0, envoi de 3_1, réception de 3_0
Ordinateur 2: réception de 1_0, réception de 2_0, réception de 2_1, réception de 3_0, réception de 3_1, réception de 1_1
Ordinateur 3: réception de 1_1, réception de 2_0, réception de 2_1, réception de 1_0, réception de 3_0, réception de 3_1

Lors de chaque envoi, le contenu du vecteur est indiqué. Lors de chaque réception, le vecteur reçu est indiqué. Si le vecteur reçu est inférieur ou égal au vecteur d'envoi pour toutes les positions, sauf celle de l'ordinateur envoyeur où il est supérieur de 1, tous les messages "connus" lorsque ce message avait été envoyé ont été reçus et celui-ci peut être livré aux applications. Sinon, le message reçu est retenu jusqu'à ce que les messages préalables manquants soient reçus. Lorsque le message est livré aux applications, le vecteur de ce message est fusionné au vecteur courant d'envoi (la valeur la plus grande est retenue pour chaque position) ce qui tient compte de la valeur pour l'ordinateur envoyeur qui est supérieure de 1.

Ordinateur 0: envoi de 1_0 *(1,0,0,0), réception de* 1_1 *(0,1,0,0) et livraison (1,1,0,0), envoi de* 2_0 *(2,1,0,0), réception de* 2_1 *(1,2,0,0) et livraison (2,2,0,0), envoi de* 3_0 *(3,2,0,0), réception de* 3_1 *(2,3,0,0) et livraison (3,3,0,0)*
Ordinateur 1: envoi de 1_1 *(0,1,0,0), réception de* 1_0 *(1,0,0,0) et livraison (1,1,0,0), envoi de* 2_1 *(1,2,0,0), réception de* 2_0 *(2,1,0,0) et livraison (2,2,0,0), envoi de* 3_1 *(2,3,0,0), réception de* 3_0 *(3,2,0,0) et livraison (3,3,0,0)*
Ordinateur 2: réception de 1_0 *(1,0,0,0) et livraison (1,0,0,0), réception de* 2_0 *(2,1,0,0), réception de* 2_1 *(1,2,0,0), réception de* 3_0 *(3,2,0,0), réception de* 3_1 *(2,3,0,0), réception de* 1_1 *(0,1,0,0) et livraison (1,1,0,0) qui permet la livraison de* 2_0 *(2,1,0,0) donnant (2,1,0,0), la livraison de* 2_1

(1,2,0,0) donnant (2,2,0,0), la livraison de 3_0 (3,2,0,0) donnant (3,2,0,0), et la livraison de 3_1 (2,3,0,0) donnant (3,3,0,0)
Ordinateur 3: réception de 1_1 (0,1,0,0) et livraison (0,1,0,0), réception de 2_0 (2,1,0,0), réception de 2_1 (1,2,0,0), réception de 1_0 (1,0,0,0) et livraison (1,1,0,0), qui permet la livraison de 2_0 (2,1,0,0) donnant (2,1,0,0), et la livraison de 2_1 (1,2,0,0) donnant (2,2,0,0), réception de 3_0 (3,2,0,0) et livraison (3,2,0,0), réception de 3_1 (2,3,0,0) et livraison (3,3,0,0)

b) Un système envoie des messages par multi-diffusion. Vous hésitez entre l'utilisation d'accusés de réception (positifs) pour les paquets reçus, ou l'utilisation de messages signalant chaque paquet perdu et donc manquant (accusés de réception négatifs). Dans un cas comme dans l'autre, il y aura un accusé par paquet (présent ou manquant selon le cas). Les paquets peuvent avoir une longueur choisie entre 1024 octets et 1048576 octets. La probabilité qu'un message ne soit pas reçu dans ce système (soit perdu) est donnée par: p = longueur en octets / 1200000. Pour quelles longueurs de paquets sera-t-il plus avantageux d'avoir des accusés de réception positifs? Négatifs? **(2 points)**

L'une ou l'autre solution permettra d'avoir éventuellement tous les paquets à destination. On ne peut que minimiser l'envoi des accusés de réception. Si la probabilité de perdre un paquet est inférieure à 0.5, il y aura moins de paquets manquants que présents et des accusés négatifs seront préférables. Si la probabilité est supérieure à 0.5, il sera plus avantageux de signaler les paquets reçus puisqu'ils seront moins nombreux que ceux manquants. Ceci veut donc dire que des accusés négatifs sont plus intéressants si la longueur est inférieure à 600000 octets et positifs autrements.

c) Il existe différents modèles de panne qu'un système peut avoir à détecter, afin d'avertir l'utilisateur et de ne pas fournir des résultats corrompus. Expliquez comment on peut détecter une panne par omission? Une panne de mauvaise réponse aléatoire? Donnez un exemple pour chaque cas. **(1 point)**

Une panne par omission se détecte à l'aide de l'expiration d'un délai. Par exemple, à chaque envoi de paquet, une minuterie peut être activée. Lorsqu'un accusé de réception est reçu, la minuterie est désactivée. Si la minuterie arrive à échéance, elle génère une interruption et le système sait alors que l'accusé de réception n'est pas arrivé et que quelque chose s'est perdu dans le système. Une panne par réponse aléatoire est plus difficile à détecter. Idéalement, il faut avoir une certaine redondance qui permet, avec une bonne certitude, de détecter tout problème. Par exemple, ce peut être plusieurs capteurs identiques en redondance, ou une somme de contrôle sur le contenu des blocs lus d'un disque optique. Autrement, il est peut-être possible d'essayer de détecter une certaine incohérence dans les résultats reçus pour suspecter un problème.

Question 2 (5 points)

a) Dans le cadre du premier TP, vous avez étudié la performance des appels, en fonction de la taille des arguments passés, pour 3 contextes différents: appel normal, appel RMI local et appel RMI distant. Décrivez comment le temps varie dans chaque cas et comment se comparent les résultats entre ces 3 contextes différents. Expliquez ces différences. **(2 points)**

Le temps pour exécuter un appel normal est minime, puisqu'il suffit de sauver quelques registres et sauter à la nouvelle adresse. Si l'argument est passé par pointeur, sa taille a peu d'impact sur la durée de l'appel. Pour un appel RMI local, il faut sérialiser les arguments et écrire dans un tube par des appels système, le processus serveur doit lire du tube et désérialiser les arguments, faire le travail demandé et retourner la réponse en la sérialisant et l'écrivant dans le tube, et finalement le processus appelant doit lire la réponse du tube et la désérialiser. Tout ceci se fait assez rapidement, puisque tout est en mémoire, mais cela demeure beaucoup plus lent que l'appel normal. De plus, le temps croît linéairement avec la taille de l'argument, possiblement avec des sauts lorsque la taille dépasse la taille de la mémoire cache. Pour un appel distant, le transfert par réseau s'ajoute avec toute la latence que cela comporte et l'effet des paquets. La durée varie donc peu lorsque des octets s'ajoutent dans un même paquet, puis un incrément plus important survient s'il faut un paquet de plus pour contenir l'argument.

b) Un serveur reçoit 1000 requêtes par seconde et utilise la sémantique au plus une fois. Chaque requête prend 20ms du client au serveur, 0.5ms dans le serveur, et la réponse prend 20ms pour revenir au client. Le client envoie un accusé de réception au serveur 100ms après avoir reçu la réponse et cet accusé prend 20ms pour aller au serveur qui le traite en temps négligeable. En régime permanent, avec un flot constant de requêtes au serveur, combien de résultats en moyenne sont conservés par le serveur, au cas où une retransmission serait requise, en attente de l'accusé de réception? Si pour .1% des requêtes, l'accusé de réception n'est jamais envoyé par le client, car il est tombé en panne, combien de résultats seront en attente d'accusé de réception après une journée d'opération par le serveur? **(2 points)**

*Une fois la requête reçue par le serveur et traitée, le résultat est sauvegardé en attente de l'accusé de réception, ce qui prend: 20ms pour revenir au client, 100ms pour que le client produise son accusé de réception et 20ms pour retourner au serveur, soit 140ms. Pendant ce temps, puisque nous avons 1s/1000 requêtes = 1ms/requête, nous allons recevoir 140 requêtes, soit autant de résultats conservés simultanément sur le serveur. Si .1% des requêtes restent prises dans le système, cela fait .001 * 1000 requêtes / s = 1 requête / s. Nous avons donc en 24 heures 24 * 60 * 60 = 86400 résultats qui traînent par erreur dans le système, en plus des 140 pour les requêtes actives.*

c) Quels sont les avantages du système d'appels à distance *gRPC* avec les *Protocol buffers* proposé par Google? **(1 point)**

Le système gRPC présente plusieurs avantages. Il est simple à utiliser, supporte plusieurs langages, est très efficace étant binaire avec compression des entiers, et les messages sont auto-décrits. Ceci permet de traiter des messages dont le type n'est pas connu à l'avance et surtout d'assurer une certaine compatibilité vers l'avant et vers l'arrière (vieux serveur qui reçoit et ignore des nouveaux champs inconnus, ou nouveau serveur qui prend des valeurs par défaut pour des nouveaux champs non fournis par le vieux client).

Question 3 (5 points)

a) Vous devez effectuer un calcul de rendu d'image par lancer de rayon, à partir d'une scène 3D, pour un travail en infographie. Vous décidez d'utiliser une grappe de calcul dans le nuage pour

ce faire et devez décider du nombre d'instances à activer afin d'aller le plus vite possible, étant donné que votre échéance vient rapidement. Un serveur doit tout d'abord envoyer séquentiellement à chaque instance la définition de l'espace 3D (éléments de la scène, mouvements, position de l'observateur), ce qui prend 5s pour chaque envoi. Par la suite, le calcul des 200000 images qui constitueront un film est réparti uniformément entre les différentes instances. Le calcul de chaque image sur une instance prend 4s. Ceci terminé, le serveur doit recevoir séquentiellement toutes les images calculées sur les instances, ce qui prend 1ms par image. Quel est le temps total pour compléter ce travail et obtenir toutes les images sur le serveur si la grappe contient n instances? Quelle est la valeur de n qui permet de minimiser ce temps? **(2 points)**

Le temps requis est de $n \times 5s + 200000 \times 4s/n + 200000 \times .001s$. La valeur optimale de n est obtenue lorsque la dérivée est nulle soit $5s - 200000 \times 4/n^2 = 0$ ou $5 \times n^2 = 800000$ et $n = 400$. La valeur optimale est donc de 400 instances.

b) Pour exécuter une machine virtuelle, trois approches sont possibles: la virtualisation complète, la paravirtualisation, ou les conteneurs. Quels sont les avantages et inconvénients de chaque approche? **(1 point)**

La virtualisation complète permet d'exécuter n'importe quelle image et présente donc le moins de contraintes. Par contre, le surcoût est le plus élevé des trois méthodes. La paravirtualisation demande une configuration particulière du noyau du système d'exploitation, qui est compatible avec le gestionnaire de machines (para)virtuelles utilisé. Le surcoût pour la (para)virtualisation est moins élevé que pour la virtualisation complète. Les conteneurs doivent utiliser le système d'exploitation de l'hôte. Il n'y a donc aucune flexibilité à ce niveau. Toutefois, il n'y a pas vraiment de surcoût par rapport à une exécution native sur une machine physique.

c) Sur Android, les applications peuvent être écrites en C/C++ et être compilées ou être écrites en Java. Quels sont les avantages de chaque approche? **(1 point)**

Une application native, compilée, offre normalement la meilleure performance. Elle doit toutefois être disponible pour le bon matériel et la bonne version du système d'exploitation et des librairies au niveau binaire. Une application en Java sera livrée sous forme de bytecode qui est indépendant du matériel. Il faut que la version des librairies installées soit compatible au niveau source, ce qui est beaucoup moins contraignant. L'application pourrait toutefois être un peu moins performante.

d) Quelle est l'utilité de pouvoir migrer des machines virtuelles d'un noeud physique à l'autre? Pourquoi décide-t-on de migrer une machine virtuelle? **(1 point)**

L'utilité de migrer des machines virtualles (VM) est de pouvoir facilement s'adapter aux circonstances. Si une VM a besoin de plus de ressources, il est possible de la migrer vers un noeud plus puissant. Si le taux d'utilisation des noeuds physiques dans un centre de données diminue, il est possible de consolider les VM sur un plus petit nombre de noeuds et d'en éteindre quelques-uns. Si un ordinateur requiert de l'entretien, il est possible de migrer vers d'autres noeuds les VM qu'il supporte. D'autres raisons peuvent aussi motiver une migration comme mettre ensemble deux VM qui communiquent beaucoup ensemble, rapprocher une VM du réseau avec lequel elle interagit, favoriser l'utilisation des noeuds physiques où l'électricité coûte moins cher ou est plus verte...

Question 4 (5 points)

a) Un réseau de clients est servi par 3 serveurs CODA répliqués. Chaque client ouvre en moyenne 5 fichiers par seconde et en ferme autant. Lors de l'ouverture, le fichier n'est pas présent localement dans 25% des cas et doit être lu à partir d'un serveur. Lors de la fermeture, le fichier a été modifié dans 10% des cas et doit alors être écrit sur chacun des 3 serveurs. Chaque requête de lecture prend 5ms de CPU sur un serveur et en plus, dans 30% des cas, une lecture du disque de 30ms. Chaque écriture prend sur chaque serveur 10ms de CPU et 40ms de temps du disque. Si chaque serveur possède 2 CPU et 2 disques, que les requêtes sont bien réparties entre les serveurs, les CPU et les disques, et que le service utilise plusieurs threads afin de servir en parallèle les requêtes, quel est le nombre maximal de clients possible avant que le service ne sature? Combien de thread devrait-on rouler sur chaque serveur? **(2 points)**

*Pour un client, on a 5 ouvertures / s générant 5 / s * .25 = 1.25 lecture / s. On a aussi 5 fermetures / s soit 5 / s * .1 = 0.5 écriture / s. Sur un des serveurs, cela donne 1.25 / 3 lecture / s (lectures réparties entre les 3 serveurs) soit 1.25 / 3 * 5ms = 2.08ms CPU et 1.25 / 3 * 0.3 * 30ms = 3.75ms disque. Cela donne aussi sur chaque serveur 0.5 écritures / s soit 0.5 * 10ms = 5ms CPU et 0.5 * 40ms = 20ms disque. Le total pour un serveur est donc 7.08 ms CPU et 23.75 ms disque. Le goûlot d'étranglement est au niveau des disques. Avec 2 disques, on peut supporter 2 * 1000ms / 23.75ms = 84.21 donc 84 clients. Il faut un thread par ressource, CPU ou disque, soit un total de 4 threads.*

b) Dans Glusterfs, plusieurs types de services de fichiers sont offerts. Supposons que nous ayons 4 fichiers à stocker sur un service de fichiers Glusterfs offert par 4 serveurs qui contribuent chacun une brique. Montrez quel fichier (ou morceau de fichier) pourrait se trouver sur chaque serveur pour chacune des configurations suivantes: i) serveurs répartis, ii) serveurs répliqués, iii) serveurs répartis et répliqués, iv) serveurs agrégés (striped). **(2 points)**

Pour le service réparti i), chaque fichier pourrait se retrouver sur un serveur différent puisqu'ils se répartissent la charge. Pour le service répliqué ii) une copie de chacun des 4 fichiers pourrait se retrouver sur chacun des 4 serveurs de manière à avoir une redondance quadruple. Dans le service réparti et répliqué, en supposant 2 serveurs répartis de 2 serveurs répliqués, on pourrait avoir 2 fichiers qui se retrouvent tous deux sur 2 des serveurs, et les 2 autres fichiers qui se retrouvent répliqués sur les 2 autres serveurs. Pour des serveurs agrégés, un même fichier très gros pourrait se retrouver avec 1/3 de son contenu sur chacun de 3 des serveurs et les 3 autres fichiers sur le quatrième serveur.

c) Lorsque le client d'un service de fichiers effectue une lecture, cette lecture peut être servie dans la cache d'E/S du client, sinon dans la cache d'E/S du serveur, ou sinon à partir du disque. Comment se compare la vitesse dans chaque cas? Est-ce qu'il y a un inconvénient à prendre la contenu à partir de la cache du client? De la cache du serveur? **(1 point)**

La cache du client sera la plus rapide, à la vitesse de la mémoire qui contient la cache d'E/S. La cache d'E/S sur le serveur est accédée à travers le réseau. Le délai de réseau ajoute donc au temps d'accès. Pour un accès servi à partir du disque du serveur, il faut attendre après le réseau et après le disque, ce qui ajoute autant au délai. Il n'y a pas de problème à lire de la cache d'E/S du serveur, puisqu'elle est nécessairement cohérente avec le contenu du disque (aussi à

jour ou plus à jour). La cache du client peut ne pas être au courant des dernières mises à jour et un problème de cohérence se pose. Sur NFS, on revalide le contenu dans la cache avant de l'accéder si la dernière validation date de plus de 3 secondes. Sur d'autres systèmes, on compte sur les notifications de changement en provenance du serveur pour être averti de tout bloc dont le contenu devient caduc.

Le professeur: Michel Dagenais

ECOLE POLYTECHNIQUE DE MONTREAL

Département de génie informatique et génie logiciel

Cours INF8480: Systèmes répartis et infonuagique (Hiver 2018)
3 crédits (3-1.5-4.5)

CORRIGÉ DU CONTRÔLE PÉRIODIQUE

DATE: Vendredi le 23 février 2018

HEURE: 9h30 à 11h20

DUREE: 1H50

NOTE: Aucune documentation permise sauf un aide-mémoire, préparé par l'étudiant, qui consiste en une feuille de format lettre manuscrite recto verso, calculatrice non programmable permise

Ce questionnaire comprend 4 questions pour 20 points

Question 1 (5 points)

a) Un serveur reçoit des requêtes qui se font en 7 étapes: 1) ouverture d'une connexion TCP, réception d'une commande et envoi de la réponse en 10ms de CPU, 2) attente de 20ms, 3) réception d'une seconde commande qui prend aussi 10ms de CPU à traiter, 4) attente de 20ms, 5) réception d'une troisième commande qui prend aussi 10ms de CPU à traiter, 6) attente de 20ms, 7) fermeture de la connexion TCP, ce qui indique la fin de la requête. Si le serveur s'exécute avec un seul thread d'exécution qui traite séquentiellement les requêtes, combien de requêtes par seconde peut-il traiter? Si le serveur exécute de nombreux threads qui se partagent les requêtes, mais roulent sur un seul coeur de CPU, combien de requêtes par seconde peut-il traiter? **(2 points)**

Chaque requête prend un total de 10+20+10+20+10+20=90ms, un seul thread peut donc en traiter 1000ms/90ms = 11.11 par seconde. Toutefois, puisque chaque requête ne prend que 30ms de CPU, avec plusieurs threads on peut en traiter 1000ms/30ms = 33.33 par seconde.

b) Un serveur doit transmettre son contenu pour le répliquer sur 2 autres serveurs, soit 1000000 paquets de 4096 octets. La transmission d'un petit paquet comme un accusé de réception occupe le réseau pendant 0.1ms alors qu'un paquet de 4096 octets l'occupe pendant 0.5ms. On suppose que le nombre de paquets perdus sur ce réseau est négligeable. i) Quel est le temps requis pour un envoi séparé vers chacun des 2 autres serveurs avec un accusé de réception pour chaque paquet envoyé? Quel est le temps requis pour un envoi simultané en multi-diffusion aux 2 serveurs avec un accusé de réception négatif à la toute fin (sur réception de la fin de fichier, le récepteur envoie la liste des paquets manquants)? **(2 points)**

Dans le premier cas, il faut l'envoi de 1000000 paquets pour le contenu, à 0.5ms chacun, et de 1000000 paquets d'accusé de réception, à 0.1ms chacun, vers chacun des deux serveurs. Le total est donc de $2 \times 1000000 \times (0.5ms + 0.1ms) = 1200s$. *Dans le second cas, on envoie 1000000 paquets à 0.5ms chacun en multi-diffusion aux deux serveurs et il n'y a pas d'accusés de réception négatif si aucun paquet n'est perdu. Le temps est donc de* $1000000 \times 0.5ms = 500s$.

c) Lors d'un envoi de message de groupe atomique, un noeud a reçu le message et est en attente de la confirmation que tous l'ont reçu avant de livrer le message à l'application. Que pourra faire ce noeud après un certain délai i) si le message de confirmation du noeud d'envoi a été perdu? ii) Si le noeud d'envoi a flanché dans le milieu de l'envoi de ses messages de confirmation? **(1 point)**

Au bout d'un certain temps, le noeud va interroger le noeud d'envoi pour savoir ce qui en est. Si le message de confirmation s'était perdu, la réponse qui ne lui était pas parvenue sera retransmise et à sa réception il pourra livrer le message de groupe atomique à l'application. Si le noeud d'envoi a flanché, il ne répondra pas à cette interrogation. Le noeud en réception ne peut donc savoir s'il faut livrer le message atomique à l'application ou au contraire le détruire. Il faut que le système ait une manière de récupérer de cette erreur. Par exemple, le noeud d'envoi pourrait avoir écrit dans un journal le fait qu'il commençait à envoyer des confirmations. Après avoir flanché puis redémarré, il pourra relire ce journal et savoir quoi répondre au client récepteur qui l'interroge. Le noeud en réception peut aussi interroger les autres membres du groupe. Si un

des autres membres a reçu une confirmation, le noeud en réception peut se baser là-dessus pour livrer le message à l'application.

Question 2 (5 points)

a) Un client accède un service de fichiers directement par le biais d'une librairie générée par gRPC. La fonction RPC pour lire le contenu d'un fichier est `FileRead` tel que montré ci-après. i) Si on effectue un appel avec comme arguments 1000 pour le `fileId` et la valeur par défaut pour le `fileOffset`, et que le contenu retourné est le contenu complet du fichier qui contient 300 octets, expliquez combien d'octets sont requis pour encoder ces 2 messages (FileSegment et FileContent). ii) On ajoute un argument optionnel, `optional uint64 length = 3 [default = 0];`, au message FileSegment et le serveur est mis à jour en conséquence pour lire tout le fichier si la longueur est 0, et se limiter à la longueur spécifiée autrement. Est-ce que cela va continuer à fonctionner de manière transparente pour un ancien client qui ne spécifie pas de longueur? Expliquez. **(2 points)**

```
message FileSegment {
  required uint64 fileId = 1;
  optional uint64 fileOffset = 2 [default = 0];
}
message FileContent {
  required bytes content = 1;
}
service FileService {
rpc FileRead (FileSegment) returns (FileContent) {}
...
}
```

Pour fileId, le numéro de champ et le type peuvent entrer dans un seul octet. La valeur 1000 requiert 2 octets avec les varint utilisés. Pour fileOffset, aucune valeur n'est spécifiée et le champ peut être omis. Un total de 3 octets est donc requis. Pour le contenu retourné, nous avons content avec 1 octet pour le champ/type, 2 octets pour la longueur (300) en varint, et 300 octets pour le contenu lui-même. La longueur totale pour FileContent est donc de 303. Le grand total est de 306 octets. Si un argument optionnel est ajouté à la fonction, cela fonctionnera correctement si la valeur par défaut correspond au comportement de la version précédente de la fonction. C'est le cas ici puisque la valeur par défaut est de 0 (lire tout le fichier comme avant cette option). Ainsi, un ancien client qui ne spécifie pas le nouveau champ verra exactement le même résultat qu'avant le changement sur le serveur.

b) Une application de dessin répartie est programmée avec Java RMI. Deux organisations possibles sont évaluées. Dans l'organisation A, l'objet DessinA est un objet réseau qui contient un vecteur d'objets réguliers FormeA qui sont des formes géométriques. Dans la seconde organisation, B, l'objet DessinB est un objet réseau qui contient un vecteur d'objets réseau FormeB qui sont des

formes géométriques. Un client doit lire le contenu d'un dessin afin de l'afficher (appel pour obtenir le vecteur de formes du dessin, puis appel pour chaque forme pour lire son contenu afin de l'afficher). Si le dessin D1 contient 10 formes géométriques, F1 à F10, répondez à i) et ii) pour chacune des deux organisations, A et B. i) Combien d'appels réseau seront requis pour ainsi afficher le dessin? ii) Quel sera le contenu de la table des objets exportés et de la table des objets importés dans le client et dans le serveur? **(2 points)**

Avec l'organisation A, un appel réseau est requis afin d'obtenir le vecteur de formes. On suppose que le proxy vers D1 est déjà dans le client, puisque la manière de l'obtenir n'est pas spécifiée. Le vecteur et les formes qu'il contient sont des objets locaux et leur accès ne demande pas d'appel réseau. La table des objets exportés du serveur contient D1 et la table des objets importés du client contient le proxy de D1. Avec l'organisation B, il faut 1 appel pour obtenir le vecteur de formes et un appel réseau pour obtenir le contenu de chaque forme, soit un total de 11. La table des objets exportés du serveur contient D1 et F1 à F10 alors que la table des objets importés du client contient les proxy de D1 et F1 à F10.

c) Dans votre travail pratique, expliquez ce que fait la fonction `lock(nom, clientid, checksum)` selon les différents cas possibles. **(1 point)**

Cette fonction verrouille le fichier nom *sur le serveur pour le client* clientid, *s'il existe et s'il n'est pas déjà verrouillé par un autre client. De plus, si la version la plus récente n'est pas présente sur le client (la somme de contrôle ne correspond pas à celle envoyée,* checksum*), le fichier est copié dans le répertoire local sur le client.*

Question 3 (5 points)

a) Sur OpenStack, le service Nova s'occupe de faire exécuter les machines virtuelles sur les noeuds disponibles. Lorsqu'un client demande de démarrer une nouvelle machine virtuelle, comment Nova décide sur quel noeud démarrer la nouvelle machine virtuelle? **(2 points)**

Nova regarde la liste des noeuds présents et fonctionnels. Parmi ceux-ci, il détermine le sous-ensemble éligible des noeuds qui satisfont aux contraintes de la machine virtuelle à créer (répertoire d'instructions, mémoire, capacité...). Parmi les noeuds éligibles, il choisit le noeud de plus forte priorité selon différents critères qui peuvent être configurés (charge, nombre de VM déjà présentes, puissance du noeud...).

b) Une machine virtuelle contient une image de 2000000 pages de 4096 octets. On veut migrer cette machine virtuelle d'un noeud à un autre. L'envoi d'une page de 4096 octets sur le réseau prend 0.1ms. Quel sera le temps requis pour la migration si i) la machine virtuelle est peu occupée et modifie 10 pages par seconde? ii) Si la machine virtuelle est très occupée et modifie 10000 pages par seconde? **(2 points)**

La première ronde de copie prend $2000000 \times 0.1ms = 200s$. *Si la machine virtuelle modifie 10 pages par secondes, 2000 pages seront modifiées pendant ce temps, ce qui prendra 0.2s à transmettre. Pendant cette seconde copie, 2 pages seront modifiées, ce qui prend .2ms à transmettre et il n'y aura vraisemblablement plus de pages modifiées rendu là. De toutes manières, lorsque le nombre de pages restant est très faible, on arrête la machine virtuelle pendant la dernière ronde*

de copie. La migration prendra donc environ 200s+.2s+.2ms=200.2002s et l'arrêt de la machine virtuelle sera très bref. Si la machine virtuelle modifie 10000 pages par seconde, pendant la première copie de 200s, il y aura $200s \times 10000p/s = 2000000$ *pages modifiées. On constate donc qu'il n'y a aucun progrès puisque les pages sont modifiées aussi vite qu'elles sont copiées. La migration ne terminera jamais dans ces conditions. Elle ne pourra pas se faire sans arrêter la machine virtuelle durant toute la copie.*

c) Kubernetes et Docker sont deux systèmes pour exécuter des conteneurs. Expliquez le rôle de chacun dans une solution infonuagique basée sur les conteneurs. **(1 point)**

Docker est un format d'image et un environnement pour exécuter un conteneur. Sa force est de simplifier la construction des images et de faciliter l'interface pour démarrer, exécuter et arrêter une telle image sur différents systèmes (incluant Linux et Windows). Kubernetes est un système d'orchestration des conteneurs. On peut spéficier comment déployer les conteneurs, incluant la réplication et la répartition de requêtes. Les conteneurs déployés seront souvent des images Docker avec l'environnement d'exécution associé.

Question 4 (5 points)

a) Un serveur NFS dessert de nombreux clients. Les clients effectuent en moyenne 1 écriture et 10 lectures par seconde sur des blocs de fichiers venant de ce serveur. Les blocs accédés en lecture se trouvent en cache sur le client dans 90% des cas. Parmi les blocs en cache, 40% ont été validés depuis moins de 3 secondes. Les autres blocs en cache demandent une validation auprès du serveur. Parmi ces blocs qui demandent une validation, 30% ont été modifiés et nécessitent une lecture en plus, alors que 70% sont valides. L'écriture d'un bloc sur le serveur prend 15ms de disque. La lecture d'un bloc du serveur prend 10ms de disque dans 20% des cas, et est servie à partir de la cache d'entrée-sortie en temps négligeable dans 80% des cas. Une validation d'un bloc du serveur prend 10ms de disque dans 10% des cas, et est servie à partir de la cache d'entrée-sortie en temps négligeable dans 90% des cas. Quel est le nombre de clients maximal que peut soutenir le serveur sans être saturé, s'il contient 8 disques et que les requêtes sont réparties uniformément entre les disques? **(2 points)**

Pour chaque lecture d'un client, on a .1 lecture sur le serveur (non présent en cache), $.9 \times .4 = .36$ *rien (bloc en cache et récent),* $.9 \times .6 \times .7 = .378$ *revalidation sur le serveur (copie en cache validée),* $.9 \times .6 \times .3 = .162$ *revalidation sur le serveur et en plus lecture sur le serveur (copie en cache non validée). Pour 10 lectures sur le client, on a donc* $10 \times (.1 + .162) = 2.62$ *lectures et* $10 \times (.378 + .162) = 5.4$ *revalidations sur le serveur à chaque seconde, en plus de 1 écriture par seconde.*

Le disque sur le serveur subit donc pour un client la charge suivante par seconde $1\ ecriture \times 15ms + 2.62\ lectures \times .2 \times 10ms + 5.4\ revalidations \times 0.1 \times 10ms = 25.64ms$. *Ainsi, sur 8 disques on peut soutenir* $8 \times 1000ms/25.64ms = 312.01$ *clients.*

b) Un service de fichiers CODA est répliqué sur 4 serveurs (i.e. chaque fichier se retrouve en 4 copies). Chaque serveur possède 6 disques. Chaque disque peut effectuer 100 accès (lecture ou écriture) par seconde. Les clients, lors des ouvertures ou fermetures de fichiers, font des accès en lecture ou en écriture au serveur. Quel est le nombre maximal de lectures (s'il n'y a que des

lectures) par seconde que pourrait soutenir ce service répliqué sur 4 serveurs, en supposant que la charge est répartie uniformément sur les serveurs et les disques, et que les disques constituent le facteur limitant? Quel est le nombre maximal d'écritures (s'il n'y a que des écritures)? **(2 points)**

En lecture, les serveurs opèrent en parallèle. Avec 4 serveurs de 6 disques à 100 accès/s, on a une capacité maximale de $4 \times 6 \times 100 = 2400$ *lectures par seconde. En écriture, chaque écriture doit être faite sur les 4 serveurs. On peut donc soutenir 4 fois moins d'écritures que de lectures soit 600 écritures pas seconde.*

c) Quel est le principe de fonctionnement de BitTorrent? En quoi est-ce optimisé pour les gros fichiers? **(1 point)**

Avec BitTorrent, les fichiers sont offerts en morceaux. De plus, les morceaux les plus rares sont offerts en priorité. Ceci permet de rapidement mettre en circulation tous les morceaux d'un fichier et de paralléliser le plus rapidement possible les transferts, avec de nombreux clients qui participent au transfert. Ceci est particulièrement intéressant pour les gros fichiers. En effet, pour un gros fichier, s'il faut attendre une copie complète avant qu'un client puisse participer à la retransmission, cela prend beaucoup de temps.

Le professeur: Michel Dagenais

ECOLE POLYTECHNIQUE DE MONTREAL

Département de génie informatique et génie logiciel

Cours INF4402: Systèmes répartis sur l'Internet (Automne 2012)
3 crédits (3-1.5-4.5)

CORRIGÉ DE L'EXAMEN FINAL

DATE: Dimanche le 16 décembre 2012

HEURE: 9h30 à 12h00

DUREE: 2H30

NOTE: Toute documentation permise, calculatrice non programmable permise

Ce questionnaire comprend 4 questions pour 20 points

Question 1 (5 points)

a) Un client envoie un message au serveur pour lui demander l'heure exacte. Ce message est envoyé à 10h00m00.000s et la réponse reçue à 10h00m00.542s, heure du client. La réponse indique que l'heure exacte du serveur au moment où il a répondu était de 10h00m05.041s. En utilisant l'algorithme de Christian, calculez le décalage à appliquer à l'heure du client et donnez l'intervalle d'incertitude sur cette valeur de décalage? Si le serveur avait fourni l'information additionnelle suivante, réception de la demande à 10h00m04.539 et envoi de la réponse à 10h00m05.041s, en utilisant l'algorithme de NTP qui tire parti de cette information, que deviendrait le calcul de décalage et l'intervalle d'incertitude associé? **(2 points)**

Avec la méthode de Christian, il suffit de calculer le délai de réponse vu du client, .542s, et d'assumer que cela représente le délai symétrique associé au réseau (.271s dans chaque direction). La nouvelle heure est donc 10h00m05.041 (heure du serveur) + .271s (délai réseau), duquel on soustrait 10h00m00.542s (heure de réception du client) pour calculer le décalage du client. Le décalage est donc de 4.77s +/- .271s.

Avec la méthode NTP, nous possédons de l'information supplémentaire. Le calcul donne 4.519s +/- .02s, ce qui est cohérent avec le premier calcul mais beaucoup plus précis.

a = 10h00m04.539s - 10h00mm00.000s = 00m04.539s
b = 10h00m05.041s - 10h00m00.542s = 00m04.499s
Ajustement = (a+b)/2 = (4.539s + 4.499s)/2 = 4.519 s
Précision = (a-b)/2 = (4.539s - 4.499s)/2 = 0.02s

b) Afin de détecter un interblocage réparti dans un système de transactions réparti, il faudrait idéalement prendre un cliché cohérent de tous les liens de dépendance entre les verrous et les transactions. Ceci n'est géneralement pas fait, les dépendances de chaque serveur sont prises en séquence avec un délai faible plutôt que d'essayer d'avoir simultanément tout l'état de tous les serveurs. Pourquoi? Quelles sont les conséquences d'avoir un graphe de dépendance global construit à partir des informations non simultanées (cohérentes) des différents serveurs? Donnez un exemple concret de situation problématique qui peut se présenter. **(2 points)**

Obtenir un cliché cohérent de l'état global serait très coûteux et pourrait demander de tout arrêter les processus impliqués le temps de récolter toute l'information. C'est pour cette raison qu'une approximation est faite en lisant toutes les informations rapidement mais sans garantie de cohérence.

Une première situation qui peut arriver est qu'un interblocage se constitue pendant le processus de collecte de donnée. Juste après que ses informations aient été lues, le premier processus demanderait un verrou détenu par le dernier processus lu. Ceci ne pose pas vraiment de problème car l'interblocage demeurera et sera facilement détecté au prochain tour. L'autre possibilité est que le premier processus relâche un verrou juste après que ses informations aient été lues. Ceci peut paraître difficile a priori qu'un processus qui semble impliqué dans un interblocage puisse relâcher un verrou; un processus ne relâche normalement pas un verrou avant de compléter et cela serait difficile s'il y a des dépendances circulaires. Cepen-

dant, il peut arriver que la transaction du premier processus soit annulée avant que d'autres liens de dépendances qui formeraient un cycle ne s'ajoutent.

c) Dans le travail pratique 2, chaque client numérotait ses demandes afin de permettre aux serveurs de maintenir un ordonnancement cohérent entre les requêtes (pour la même donnée d'un même client). Discutez brièvement des divers problèmes de sécurité de cette application et de solutions possibles. Par exemple, un client malicieux pourrait-il bloquer le système en envoyant par exprès des requêtes avec des numéros d'ordre manquants (e.g., requête 5, ensuite 7...)? **(1 point)**

Il faudrait premièrement être capable d'authentifier les clients auprès des serveurs. Autrement, n'importe qui peut se faire passer pour n'importe quel client et modifier n'importe quelle donnée. Puisque chaque serveur n'ordonnance que les requêtes par client, un client qui saute un numéro ne pourra voir ses requêtes servies tant que le numéro manquant ne sera pas vu par le serveur. Cela n'empêchera toutefois pas les autres clients d'être servis. Le client malicieux pourrait cependant remplir la queue du serveur de requêtes que le serveur retient. Si le serveur a été bien pensé, il aura une taille limite pour la queue de chaque client après laquelle il ignore toute nouvelle requête.

S'il n'y a pas d'authentification, le client malicieux pourrait se faire passer pour un autre client et envoyer des requêtes avec des numéros plus élevés (n+2, n+3...) que la requête courante (n) du client qu'il personnifie. Il pourrait ainsi remplir la queue de cet autre client. Le client réel va tout de même éventuellement envoyer la prochaine requête en séquence (n+1). Le serveur devrait idéalement accepter cette requête même si sa queue est pleine, puisque cela permettra de le débloquer. Ensuite, si le client réel envoie d'autres requêtes (n+2) le serveur sera perplexe car il croit avoir déjà reçu cette requête et devrait générer un message d'erreur. S'il refuse la seconde requête au même numéro, le client malicieux aura réussi à empêcher le client réel de se faire servir.

Question 2 (5 points)

Trois transactions, T, U et V, s'exécutent concurremment. Le séquencement des opérations est le suivant:

T: Début
U: Début
V: Début
T: Read(x)
U: Read(z)
T: Write(x,1)
T: Compléter
U: Read(x)
V: Read(w)
U: Read(y)
U: Write(y,2)

U: Compléter
V: Read(x)
V: Read(y)
V: Write(z,3)
V: Compléter

a) Lesquelles des transactions T, U et V pourraient être validées si une validation en reculant était utilisée pour vérifier la cohérence des transactions? Justifiez. **(1 point)**

La première transaction, T, est nécessairement validée. Pour U, il faut vérifier l'intersection entre ce que T a écrit (x) et ce que U a lu (x, y, z) ou écrit. Il y a intersection et il faut annuler U. Pour V, il ne reste plus que T à vérifier. T a écrit x alors que V a lu w, x et y. Là encore, l'intersection (x) fait qu'il faut annuler V.

b) Lesquelles des transactions T, U et V pourraient être validées si une validation en avançant était utilisée pour vérifier la cohérence des transactions? Justifiez. **(1 point)**

Au moment de compléter T, il faut vérifier ce qui est écrit par T (x) versus ce que U a lu (z, car à ce moment U n'a pas encore lu x). Il n'y a pas de conflit et T est validé. La transaction U écrit (y) alors que V a lu (w), elle peut donc compléter. Rendu à compléter V, il ne reste plus d'autre transaction et donc pas de problème.

c) Lesquelles des transactions T, U et V pourraient être validées si une validation par compteur de temps était utilisée pour vérifier la cohérence des transactions? Justifiez. Les écritures sont faites dans des valeurs tentatives jusqu'à ce que la transaction associée ne soit complétée, les lectures sont faites dans les valeurs tentatives. **(1 point)**

La transaction T se voit assigner le temps t1, U t2 et V t3 ($t0 < t1 < t2 < t3$).

T: Début, t1
U: Début, t2
V: Début, t3
T: Read(x), écrit en t0, lu en t1
U: Read(z), écrit en t0, lu en t2
T: Write(x,1), version tentative pour t1
T: Compléter, écrire x en t1, pas de conflit, validé
U: Read(x), écrit en t1, lu en t2
V: Read(w), écrit en t0, lu en t3
U: Read(y), écrit en t0, lu en t2
U: Write(y,2), version tentative pour t2
U: Compléter, écrire y en t2, pas de conflit, validé
V: Read(x), écrit en t1, lu en t3
V: Read(y), écrit en t2, lu en t3
V: Write(z,3), version tentative pour t3
V: Compléter, écrire z en t3, pas de conflit, validé

d) Vous devez concevoir un système d'achat de billets de spectacles où le client peut choisir plusieurs sièges et ensuite confirmer son choix et son achat par son paiement. De nombreux clients peuvent utiliser le système d'achat en même temps pour un même spectacle. Vous hésitez entre un contrôle optimiste de la concurrence et un contrôle strict de la concurrence. Quelles seront les conséquences de ce choix sur l'utilisation du site par un client (performance, déroulement de la procédure d'achat...)? Donnez un exemple concret de problème qui pourrait se poser. **(2 points)**

Un contrôle strict fait qu'il faut bloquer au fur et à mesure les sièges tentativement réservés par un client. Ceci demande de prendre un verrou à chaque nouveau siège ajouté au panier d'achat du client. Ceci est assez coûteux car la prise de verrou est une opération atomique sur un serveur central. De plus, on ne peut utiliser un contrôle strict sans mettre des restrictions importantes, faute de quoi le système sera très vulnérable aux abus. Par exemple, si un client réserve tentativement tous les billets de la salle, il pourrait bloquer tous les autres acheteurs. Pour cette raison, il faut imposer des limites possiblement sur le nombre de billets mais surtout sur le délai maximal pendant lequel une réservation tentative est conservée. Ainsi, un client peut se faire avertir qu'il a 2 minutes pour procéder au paiement, à defaut de quoi les sièges qu'il a placés dans son panier d'achat seront relâchés pour les autres clients.

Avec un contrôle optimiste, le client peut prendre tout le temps qu'il veut mais il peut éventuellement se faire avertir, en temps réel ou au moment de procéder au paiement, que les sièges qu'il avait identifiés viennent d'être achetés par un client plus rapide que lui. Cette procédure est plus efficace sur le serveur, en l'absence d'acquisition de verrous au fur et à mesure de la transaction. Par contre, le client pourrait être fort indisposé de de faire dire au dernier moment que quelqu'un a pris les sièges convoités avant lui.

Question 3 (5 points)

a) Un grand mélomane s'est procuré un système redondant pour jouer de la musique dans son salon. Il possède donc deux paires de haut-parleurs (HP_A et HP_B), chacune connectée à un ordinateur client (CL_A pour HP_A et CL_B pour HP_B) relié à un réseau sans fil différent ($WIFI_A$ pour CL_A et $WIFI_B$ pour CL_B). Il possède en outre deux serveurs redondants (SRV_1 et SRV_2) pour sa musique, chacun avec 4 disques en RAID 5 et deux cartes réseau sans fil. Chaque client peut aussi bien se connecter sur SRV_1 ou sur SRV_2. Les unités RAID 5 permettent de tolérer la panne d'un disque sur les quatre. Les probabilités d'être en panne à un moment donné sont de 0.00001 pour une paire de haut-parleurs, 0.00002 pour un client, 0.00003 pour un réseau sans fil, 0.00004 pour un serveur (électronique et logiciels mais sans les disques) et de .001 pour un disque. Quelle est la probabilité que ce système redondant soit non-disponible, empêchant d'écouter de la musique avec une paire de haut-parleurs à partir d'un serveur? **(2 points)**

Le RAID 5 sera disponible si les 4 disques sont disponibles ou si un seul disque est en panne (n'importe lequel des 4) et 3 disques fonctionnels. La probabilité est donc de $P_{RAID5} = (1 - 0.001)^4 + 4 \times (1 - 0.001)^3 \times 0.001 = 0.999994008$. *Un client musique est disponible si les haut-parleurs, le réseau et le client sont disponibles soit* $P_{CL} = (1 - .00001) \times (1 - .00002) \times (1 - .00003) = 0.999940001$. *Un serveur est disponible si le serveur et le RAID5*

sont disponibles $P_{SRV} = (1 - .00004) * P_{RAID5} = 0.999954008$. *Le service est disponible si au moins un des serveurs est disponible* $P_{Serveurs} = 1 - (1 - P_{SRV})^2) = 0.999999998$ *et au moins un des clients est disponible* $P_{Clients} = 1 - (1 - P_{CL})^2 = 0.999999996$. *Le service sera non-disponible sauf si un client et un serveur sont disponibles soit* $P = 1 - (P_{Serveurs} \times P_{Clients}) = 0.000000006$

b) Une entreprise avec 15 sites veut offrir des forums à ses employés, un peu comme le système Usenet qui utilisait le protocole NNTP. Sur chaque site, un serveur permettra aux employés de consulter une copie des messages et de contribuer de nouveaux messages sur les forums. Afin de propager chaque message sur chaque site, deux organisations sont envisagées. La première organisation se base sur un serveur central dans un des 15 sites, la maison mère, avec lequel chacun des serveurs sur les 14 autres sites communiqueront pour obtenir les nouveaux messages produits sur les autres sites et contribuer les nouveaux messages générés sur ce site. La seconde organisation se base sur un arbre avec 8 sites classés niveau 3, 4 sites classés niveau 2, 2 sites classés niveau 1 et un serveur racine de niveau 0 sur le site de la maison mère. Dans cette structure arborescente, chaque serveur de niveau n est alimenté par un parent de niveau n - 1 et alimente deux enfants de niveau n + 1; le niveau 4 n'a pas d'enfant et le niveau 0 pas de parent. Chaque serveur obtient de son parent les nouveaux messages produits dans les autres branches et lui contribue les nouveaux messages générés dans sa branche. Pour chaque nouveau message créé, comparez le nombre de messages échangés entre les serveurs pour chacune de ces deux organisations. **(2 points)**

Dans le premier scénario, lorsqu'un message est créé sur un site satellite, il faut 1 message pour l'envoyer au serveur central et 13 messages pour envoyer le message aux 13 autres sites satellites, pour un total de 14 messages (1 envoi de l'origine, 1 réception et 13 envois du central, 1 réception de chaque autre satellite). Avec le second scénario, chaque site sauf celui qui génère le nouveau message reçoit un message, pour un total de 14 aussi. Cependant, la répartition des messages est différente. Chaque noeud interne reçoit 1 message et en envoie 2 alors que les feuilles n'en reçoivent que 1 (sauf celui qui génère le nouveau message qui en envoie 1) et la racine en reçoit 1 et en envoie 1. Dans ce second scénario, la charge est beaucoup mieux répartie et risque de beaucoup mieux tolérer la mise à l'échelle. Il est peu probable que le multicast soit bien supporté sur un tel réseau externe entre différents sites. Avec le multicast, la première solution pourrait se faire avec 2 messages alors que la seconde demanderait entre 7 et 10 messsages dépendant de l'origine du message dans l'arbre.

c) Dans les bases de données conventionnelles, les modifications associées à une transaction sont premièrement écrites dans le journal, ensuite (si la transaction est confirmée) écrites dans la base de données, et finalement une entrée de confirmation de la transaction est ajoutée au journal. On vous propose plutôt une base de données structurée en journal où les modifications sont simplement écrites dans le journal de même que la confirmation de la transaction. Est-ce suffisant pour permettre de récupérer en cas de redémarrage, tout comme avec les bases de données conventionnelles? Quels sont les avantages et inconvénients d'une telle organisation pour les accès en lecture et en écriture à la base de données? **(1 point)**

Le journal peut effectivement suffire à assurer le recouvrement en cas redémarrage. Une telle écriture séquentielle est aussi très efficace car elle demande très peu de mouvement de la tête de lecture/écriture du disque. Par contre, l'accès efficace en lecture aux données peut

être problématique. La bonne valeur pour une variable (e.g. rangée dans une table de la base de donnée) est la dernière écrite dans le journal. Toutefois, il est possible de maintenir des index du contenu du journal. La mise à jour des index est moins coûteuse car elle peut être faite de manière asynchrone. En effet, il est toujours possible de reconstruire le fichier dérivé qu'est l'index; un arrêt imprévu qui cause une corruption de l'index ne constitue pas une perte d'information.

Question 4 (5 points)

a) Dans votre travail pratique 3, lorsqu'un client envoie un message à une salle de discussion, il envoie un message séparé à chacun des membres de cette salle. Lorsque le nombre de participants est très élévé, ceci peut poser des problèmes de performance. Proposez une amélioration qui permettrait de réduire le nombre total de messages transmis. **(2 points)**

Puisque le même message est envoyé à plusieurs destinataires possiblement en transitant par les mêmes noeuds intermédiaires, il serait plus économique d'envoyer le message une seule fois mais avec une liste de destinataires. Chaque noeud sur le chemin vers les destinataires ne recevrait le message qu'une seule fois. Le multicast a peu de chance d'être disponible sur un tel réseau externe et demanderait de révéler en partie la topologie des communications, ce qu'un réseau poste-à-poste cherche généralement à éviter. De la même manière, prendre un serveur central pour distribuer plus efficacement les messages d'une discussion irait à l'encontre de l'architecture distribuée.

b) Quelles sont les principales caractéristiques du Google Build System? En quoi chacune de ces caractéristiques permet-elle d'être plus rapide et efficace qu'un système de compilation conventionnel où chaque usager ou groupe recompile son logiciel avec des outils comme make (ou parallel make sur une grappe d'ordinateurs)? **(1 point)**

Le GBS conserve tous les fichiers sur les serveurs, ce qui évite des aller-retour inutiles vers la station client. Il vérifie le contenu de chaque fichier (e.g. avec une somme de contrôle) et sait ainsi reconnaître les doublons qu'il peut ne stocker qu'une seule fois. A chaque compilation, il note les options et le fichier d'entrée et vérifie si cette compilation a déjà été effectuée. De cette manière, chaque fichier n'est compilé qu'une seule fois (avec les mêmes options). Finalement, s'il voit que le fichier compilé n'est pas différent de la version précédente (e.g. fichier source dont seulement les commentaires ont été modifiés), il conserve l'ancien fichier compilé et évite ainsi possiblement du travail additionnel qui découlerait d'un nouveau fichier objet. De cette manière, Google augmente la performance autant en mettant plusieurs ordinateurs en parallèle qu'en évitant de refaire plus d'une fois chaque compilation.

c) Avec le format d'encodage sur le réseau des données CORBA CDR, l'envoyeur et le récepteur connaissent à l'avance l'ordre et le type des données échangées. En quoi cela diffère-t-il de ce qui arrive dans les Protocol Buffers de Google? Quels sont les avantages respectifs de chacune de ces manières d'encoder les données sur le réseau. **(1 point)**

L'encodage CDR n'a pas besoin d'encoder la position de ses champs ou leur type. Il peut donc être plus compact et plus rapide à encoder et décoder. Avec Protocol Buffers, le

fait d'identifier le champ et le type permet de tolérer des variations de versions de protocole en autant qu'un numéro de champ ne soit jamais réutilisé pour un même type d'objet. L'information de type peut aussi être utilisée pour varier le nombre d'octets alloués pour la valeur, ce qui peut finalement produire des données encore plus compactes. En effet, pour un entier 32 bits, il arrive souvent que la valeur utilisée soit assez petite et entre facilement dans un seul octet. Dans ce cas, CDR envoie toujours 4 octets alors que Protocol Buffers pourrait prendre moins de place avec un octet pour le numéro de champ, un octet pour le type (entier d'un octet), et l'octet contenant l'entier.

d) Dans votre travail pratique 3, à quoi correspond l'étape d'initialisation (bootstrap) dans Kademlia et pourquoi est-elle nécessaire? Concrètement, dans un tel système poste-à-poste décentralisé, comment est-ce qu'un client qui veut se connecter au réseau peut-il obtenir l'adresse d'un client existant? **(1 point)**

Chaque noeud dans le réseau poste-à-poste maintient des connexions avec quelques proches voisins. Il doit trouver ces voisins les plus proches. Pour cela, il doit entrer en communication avec au moins un noeud déjà connecté au réseau qu'il veut joindre. Ce noeud déjà connecté lui servira de porte d'entrée pour rechercher ses proches voisins. La manière d'amorcer le système est d'avoir une liste de postes connus pour être souvent connectés à ce réseau. Cette liste peut être obtenue de différentes manières, par exemple livrée avec le logiciel, disponible sur un site Web... Les premiers noeuds qui initient le réseau poste-à-poste doivent aussi bénéficier d'information pour les aider à se rejoindre. Parfois, quelques noeuds de départ ont des adresses statiques connues des autres.

Le professeur: Michel Dagenais

ECOLE POLYTECHNIQUE DE MONTREAL

Département de génie informatique et génie logiciel

Cours INF4410: Systèmes répartis et infonuagique (Automne 2013)
3 crédits (3-1.5-4.5)

CORRIGÉ DE L'EXAMEN FINAL

DATE: Samedi le 7 décembre 2013

HEURE: 9h30 à 12h00

DUREE: 2H30

NOTE: Toute documentation permise, calculatrice non programmable permise

Ce questionnaire comprend 4 questions pour 20 points

Question 1 (5 points)

a) 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. Associez un vecteur de compteurs d'événements à chacun (<p1,p2,p3>), fournissant ainsi ces mêmes trois listes, mais avec des vecteurs de compteurs plutôt que des compteurs. A l'aide de ces vecteurs, que pouvez-vous dire de l'ordre relatif (avant, concurrent, après) des événements 2 de p1 versus 2 de p3, et 4 de p1 versus 4 de p2. Justifiez. **(2 points)**

p1: 1 lecture; 2 message A vers p2; 3 écriture; 4 écriture; 5 message B de p3
p2: 1 message A de p1; 2 écriture; 3 message C vers p3; 4 lecture; 5 message D de p3
p3: 1 lecture; 2 message C de p2; 3 écriture; 4 message B vers p1; 5message D vers p2

Pour construire les vecteurs de compteurs d'événements, il faut initialiser les compteurs à 0 et joindre les contenus des vecteurs des envoyeurs lors de la réception d'un message, ceci donne ce qui suit.

p1: <1,0,0> lecture; <2,0,0> message A vers p2; <3,0,0> écriture; <4,0,0> écriture; <5,3,4> message B de p3
p2: <2,1,0> message A de p1; <2,2,0> écriture; <2,3,0> message C vers p3; <2,4,0> lecture; <2,5,5> message D de p3
p3: <0,0,1> lecture; <2,3,2> message C de p2; <2,3,3> écriture; <2,3,4> message B vers p1; <2,3,5> message D vers p2

En comparant p1:3, vecteur de <2,0,0>, avec p3:2, vecteur <2,3,2>, puisque toutes les entrées du second sont supérieures ou égales au premier, ce dernier est postérieur au premier. Dans l'autre cas, p1:4, vecteur <4,0,0>, versus p2:4, vecteur <2,4,0>, aucun vecteur n'est strictement supérieur ou égal à l'autre. On ne peut donc rien conclure et on doit considérer les événements comme concurrents.

b) Un ordinateur A envoie un message à B à 9h00m10.200s pour obtenir le temps et reçoit une réponse à 9h00m10.600s. L'ordinateur B reçoit la requête de A à 9h00m05.300s à son heure, envoie une requête à C à 9h00m05.400s, reçoit la réponse de C à 9h00m05.500s et envoie sa réponse à A à 9h00m05.550s. Le serveur C reçoit la demande de B à 9h00m00.300s et lui répond à 9h00m00.350s, heure de C. La réponse finale à A contient toutes ces valeurs de temps ainsi que leur origine. A peut donc appliquer l'algorithme utilisé par NTP pour synchroniser son horloge avec celle de C, celle-ci étant très précise. Quel est le décalage à appliquer sur A? Quel est l'intervalle d'incertitude associé? **(2 points)**

Nous pouvons appliquer de manière imbriquée l'algorithme de NTP. En regardant B par rapport à C, on a:

a = 9h00m00.300s - 9h00m05.400s = -5.100s
b = 9h00m00.350s - 9h00m05.500s = -5.150s
Ajustement = (a+b)/2 = (-5.100s - 5.150s)/2 = -5.125 s

Précision = (a-b)/2 = (-5.100s + 5.150s)/2 = 0.025s

Nous pouvons alors convertir les temps sur B en temps de C: 9h00m05.300 - 5.125 = 9h00m00.175 et 9h00m05.550 - 5.125 = 9h00m00.425. Le calcul de A par rapport à B (synchronisé à C) donne donc:

a = 9h00m00.175s - 9h00m10.200s = -10.025s
b = 9h00m00.425s - 9h00m10.600s = -10.175s
Ajustement = (a+b)/2 = (-10.025s - 10.175s)/2 = -10.100s
Précision = (a-b)/2 = (-10.025s + 10.175s)/2 = -0.075s

Le décalage à appliquer est ainsi de -10.1s et l'incertitude est la somme des deux et est donc de +/- (.075+.025) soit +/- .1s. Nous pouvons aussi calculer le décalage de A à B sans convertir le temps de B:

a = 9h00m05.300s - 9h00m10.200s = -4.9s
b = 9h00m05.550s - 9h00m10.600s = -5.05s
Ajustement = (a+b)/2 = (-4.9s - 5.05s)/2 = -4.975s
Précision = (a-b)/2 = (-4.9s + 5.05s)/2 = -0.075s

Il suffit alors d'additionner les deux décalages à appliquer (C à B et B à A), ce qui donne -5.125 - 4.975 = -10.1s ainsi que les imprécisions .075 + .025 = .1s, ce qui donne la même chose que le résultat précédent. Une troisième manière de résoudre le problème est de simplement soustraire le temps passé dans B à servir d'intermédiaire entre A et C et faire comme si A avait envoyé sa demande 9h00m05.400s - 9h00m05.300s = .1s plus tard et reçu la réponse 9h00m05.550s - 9h00m05.500s = .05s plus tôt. Ceci donnerait:

a = 9h00m00.300s - (9h00m10.200s + .100s) = -10s
b = 9h00m00.350s - (9h00m10.600s - .050s) = -10.200s
Ajustement = (a+b)/2 = (-10s - 10.200s)/2 = -10.1s
Précision = (a-b)/2 = (-10s + 9.8s)/2 = -.100s

c) Lorsque la méthode de l'élection hiérarchique est utilisée, afin de trouver le serveur de plus haute priorité disponible pour être élu coordonnateur, on dit que le nombre de messages envoyés peut être de l'ordre de n^2 dans le pire cas. Décrivez un cas où cela se produirait? **(1 point)**

Si le réseau est lent et que chaque ordinateur déclenche une élection en même temps, chacun aura le temps d'envoyer un message à chaque autre ordinateur de plus haute priorité avant d'obtenir une réponse. Ceci représente n-1 + n-2 + ... 1 message, soit $n^2/2 - n$ messages. Ensuite, chaque ordinateur, ne recevant pas encore de message de réponse, peut vouloir se déclarer élu, avec n - 1 messages (ou un envoi à tous) pour se proclamer coordonnateur, ce qui représente $n \times (n-1) = n^2 - n$ messages.

Question 2 (5 points)

Trois transactions, T, U et V, s'exécutent concurremment. Le séquencement des opérations est le suivant:

T: Début
U: Début
T: Read(a)
U: Read(a)
T: Read(b)
T: Write(b,1)
T: Compléter
V: Début
V: Read(b)
U: Read(b)
U: Read(c)
U: Write(b,2)
U: Write(c,3)
U: Compléter
V: Read(c)
V: Write(c)
V: Compléter

a) Lesquelles des transactions T, U et V pourraient s'effectuer ainsi, si un contrôle de la concurrence par prise de verrou (partagé pour la lecture et exclusif pour l'écriture) est utilisé? Justifiez. **(2 points)**

T acquière un verrou en lecture sur a, ensuite partagé avec U. T acquière un verrou en lecture sur b et le promeut en verrou d'écriture et peut compléter, relâchant ses verrous. V prend un verrou de lecture sur b, ensuite le partage avec U. U acquière un verrou sur c et veut ensuite promouvoir son verrou sur b pour l'écriture mais ne peut le faire à cause de V. S'il abandonne, cela laisse libre cours à V. S'il attend, U et V se bloqueront mutuellement car V voudra le verrou sur c.

b) Lesquelles des transactions T, U et V pourraient être validées si une validation en reculant était utilisée pour vérifier la cohérence des transactions? Une validation en avançant? Justifiez. **(2 points)**

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 (a, b, c) versus ce qui a été écrit par les transactions précédentes (T qui a écrit b). Il y a intersection et donc potentiellement conflit. U est abandonné. Ceci laisse le libre champ à V, puisque V a commencé après la fin de T et n'est donc pas concurrent avec T.

En avançant, il faut vérifier ce qui est écrit par T (b) versus ce que U a lu à ce moment (a). Il n'y a pas d'intersection et T peut compléter. Au moment de compléter U, ses écritures (b, c) intersectent les lectures de V (b) et U doit être abandonné, laissant le champ libre à V.

c) Un client effectue une série d'opérations dans le cadre d'une transaction avec plusieurs sous-transactions imbriquées. Indiquez lesquelles des variables a à e seront modifiées par la transaction ainsi que leur nouvelle valeur. **(1 point)**

```
Début T1
  Write(a, 2)
  Début T1.1
    Write(b, 4)
    Début T1.1.1
      Write(c, 6)
      Commettre T1.1.1
    Write(d,8)
    Abandonner T1.1
  Write(b,10)
  Début T1.2
    Write(e,3)
    Début T1.2.1
      Write(e,5)
      Abandonner T1.2.1
    Commettre T1.2
  Commettre T1
```

Après avoir enlevé les transactions abandonnées (et celles imbriquées). il reste:

```
Début T1
  Write(a, 2)
  Write(b,10)
  Début T1.2
    Write(e,3)
    Commettre T1.2
  Commettre T1
```

Les variables affectées sont donc a=2, b=10, et e=3.

Question 3 (5 points)

a) Un jeune finissant veut offrir un nouveau service internet et doit se monter une infrastructure au plus bas coût possible. Son infrastructure est constituée de deux serveurs de base de données redondants et de 20 serveurs Web. Chaque serveur de base de données est constitué d'un ordinateur, et de deux disques redondants en miroir. Chaque serveur Web est constitué d'un ordinateur et d'un disque. Son fournisseur Internet s'occupe de répartir les requêtes entre ses serveurs Web. Le matériel utilisé a été recyclé et est en mauvais état. A chaque instant, chaque disque a une probabilité d'être en panne de 0.2 et chaque ordinateur (hormis

son ou ses disques) a une probabilité de 0.1. Quelle est la probabilité que tout (incluant chaque élément redondant) soit fonctionnel en même temps, (aucun disque ou ordinateur en panne)? Quelle est la probabilité que le service soit complètement non disponible?

(2 points)

Il y a 22 ordinateurs, chacun opérationnel avec une probabilité de .9, et 24 disques, chacun opérationnel avec une probabilité de .8. La probabilité que tout soit fonctionnel est de $.9^{22} \times .8^{24} = 0.000465$. *Chaque serveur de base de données sera opérationnel si l'ordinateur est opérationnel ainsi que les disques. Les disques le sont sauf si les deux sont en panne, une probabilité de* $1 - .2 \times .2 = .96$. *Chaque serveur a donc une probabilité de* $0.96 \times 0.9 = 0.864$ *d'être opérationnel. Le service sera offert sauf si les deux serveurs de base de données sont en panne* $1 - (1 - 0.864)^2 = 0.9815$. *Ensuite, chaque serveur Web est opérationnel si son disque et ordinateur le sont, soit une probabilité de* $0.8 \times 0.9 = 0.72$. *Le service Web sera opérationnel sauf si tous les serveurs Web sont en panne, une probabilité de* $1-(1-0.72)^{20} = 8.7 \times 10^{-12}$. *Le service sera non disponible sauf si les serveurs Web et les serveurs de base de données sont disponibles* $1 - ((1 - 8.7 \times 10^{-12}) \times 0.9815) = 0.0185$.

b) Une transaction répartie effectue les opérations suivantes sur un des serveurs impliqués. Expliquez qu'est-ce qui devra être écrit dans le jounal qui permet la récupération en cas de panne, et à quel moment ceci devra être fait au plus tôt et au plus tard (i.e. après quelle opération)? **(2 points)**

1: lire a
2: écrire 40, b
3: lire c
4: écrire 60, d
5: préparer à commettre
6: commettre

Les lectures n'ont pas à être indiquées dans le journal. Les valeurs écrites pour b et d peuvent être écrites dès que reçues ou au plus tard juste avant d'écrire l'instruction de préparation dans le journal. L'opération de se préparer à commettre doit être écrite lorsque reçue, avant de répondre pour confirmer que le serveur est prêt à commettre. L'opération commettre doit être ajoutée dans le journal après avoir été reçue. Il est avantageux (mais pas strictement requis) de le faire rapidement, pour éviter d'avoir à revérifier son statut en cas de panne avant de l'avoir écrit.

c) Dans le cadre du travail pratique 2, un répartiteur demandait à des serveurs de compter les incidences des mots dans des morceaux de texte. Lorsqu'un serveur faisait défaut et ne fournissait pas la réponse, le répartiteur pouvait demander à un autre serveur de refaire le travail, offrant ainsi une tolérance aux pannes. Cependant, le répartiteur demeure un maillon faible. Proposez une architecture qui permette d'améliorer la résilience du répartiteur. **(1 point)**

Le répartiteur pourrait écrire les résultats obtenus sur disque et les recouvrer après un redémarrage. Il pourrait y avoir deux répartiteurs redondants, tous deux actifs, qui se répartissent les serveurs et qui s'informent mutuellement des progrès réalisés.

Question 4 (5 points)

a) Une entreprise opère 9 ordinateurs identiques, 3 pour chacun de trois départements très semblables. Chacun des 3 ordinateurs pour un département est affecté à un service particulier (serveur LDAP, serveur de fichiers, serveur de compilation) et ces trois serveurs ont un taux d'occupation moyen de 10%, 20% et 30% respectivement. Un administrateur de système, inspiré par le cours INF4410, propose de consolider ces serveurs en les virtualisant, ce qui permettrait de réduire le nombre d'ordinateurs requis. Le surcoût de la virtualisation est de 20%, ce qui prenait 1 seconde en prendrait 1.2. Deux solutions sont envisagées, conserver un ordinateur par département qui roule les trois machines virtuelles de ce département, ou avoir un nuage de 3 ordinateurs physiques sur lesquels seraient répartis les 9 machines virtuelles. Que deviendrait le taux d'utilisation moyen dans chaque cas, en supposant que la charge sur chaque serveur était assez uniforme dans le temps? Quelle solution vous semble la plus intéressante? Pourquoi? **(2 points)**

Sur 100 secondes, les trois serveurs en prenaient respectivement 10, 20 et 30, soit un total de 60. Avec le surcoût de la virtualisation, ceci deviendrait $60 \times 1.2 = 72$*, soit un taux d'occupation de 72%. Il n'y a pas de différence de taux d'utilisation entre les deux cas, trois fois plus de charge sur trois ordinateurs au lieu d'un seul. La solution du nuage est plus intéressante car elle permet une certaine tolérance aux pannes. Par contre, elle peut être un peu plus difficile à mettre en oeuvre, et ne sépare pas tout à fait aussi bien les activités des trois départements au niveau de la sécurité informatique.*

b) Sur le nuage de la compagnie Amazon, un service de répartiteur existe qui envoie les requêtes reçues à tour de rôle à un des serveurs disponibles. Le répartiteur reçoit de l'information sur les différents serveurs (taux d'utilisation des serveurs, et temps de réponse aux requêtes). Le répartiteur peut aussi décider d'instancier des serveurs supplémentaires ou de retirer des serveurs instanciés. Quel critère est-ce que le répartiteur utilise pour choisir le prochain serveur auquel envoyer une requête reçue? Quel critère utilise-t-il pour décider d'activer une instance supplémentaire de serveur? Pour retirer une instance de serveur? **(2 points)**

Le répartiteur envoie la requête reçue au serveur qui répond le plus rapidement, ou les distribue à tour de rôle lorsque la différence de temps n'est pas importante. Lorsque le taux d'utilisation des serveurs est trop élevé, de nouvelles instances sont ajoutées. Lorsque le taux d'utilisation est trop faible, des instances sont retirées.

c) Les solutions de virtualisation comme KVM offrent la virtualisation complète ou la paravirtualisation. Quels sont les avantages et limitations de ces deux alternatives? Dans quelle situation choisirait-on de préférence chacune? **(1 point)**

La paravirtualisation est plus rapide puisque l'opération demandée est directement déléguée à la machine physique, plutôt que d'émuler le comportement d'un dispositif physique particulier, comme une carte réseau, avant de déléguer le travail à la machine physique. Toutefois, la paravirtualisation n'est possible que lorsqu'on peut facilement configurer de cette manière la machine virtuelle. Lorsqu'on a plein contrôle sur la configuration de la machine virtuelle, la paravirtualisation est préférable puisqu'elle est plus rapide et ne requiert pas de support matériel. Par contre, s'il n'est pas possible de modifier la machine virtualisée, par exemple

parce qu'elle est fournie par un client qui ne sait pas que son ordinateur est virtualisé ou qui n'est pas intéressé ou capable de modifier sa configuration, alors la virtualisation complète sera nécessaire.

Le professeur: Michel Dagenais

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

ECOLE POLYTECHNIQUE DE MONTREAL

Département de génie informatique et génie logiciel

Cours INF4410: Systèmes répartis et infonuagique (Automne 2016)
3 crédits (3-1.5-4.5)

CORRIGÉ DE L'EXAMEN FINAL

DATE: Vendredi le 16 décembre 2016

HEURE: 9h30 à 12h00

DUREE: 2H30

NOTE: Toute documentation permise, calculatrice non programmable permise

Ce questionnaire comprend 4 questions pour 20 points

Question 1 (5 points)

a) Sur les ordinateurs Linux, le fichier `resolv.conf` permet de lister plusieurs serveurs de nom (DNS), qui seront interrogés dans l'ordre selon lequel ils apparaissent dans le fichier, jusqu'à ce qu'une réponse soit obtenue. Ceci offre donc une tolérance aux pannes de serveur DNS par redondance, lorsqu'une traduction de nom à adresse IP est requise. Si 5 serveurs sont listés dans le fichier de configuration, combien de serveurs en panne par omission ce mécanisme peut-il tolérer avant de cesser de fonctionner? Si vous étiez dans un environnement hostile, où certains serveurs compromis fourniraient de mauvaises réponses, comment pourriez-vous modifier ce mécanisme pour aussi tolérer des pannes byzantines? Dans ce cas, combien de serveurs en panne (arrêtés ou compromis), pourriez-vous tolérer tout en maintenant le service opérationnel? **(2 points)**

Avec des pannes par omission, il suffit d'un seul serveur opérationnel pour fonctionner, soit jusqu'à 4 serveurs en panne qui peuvent être tolérés. Pour les pannes byzantines, il faudrait comparer les réponses et retenir celle qui apparaît le plus souvent. Dans un tel cas, jusqu'à 2 serveurs en panne pourraient être tolérés (3 bonnes réponses contre 2 mauvaises); avec 3 serveurs compromis, la mauvaise réponse pourrait l'emporter.

b) Un serveur de nom contient un CPU à un coeur ainsi qu'un disque. Chaque requête prend 150us (us pour microseconde) de temps CPU pour traiter la requête au complet lorsque la réponse est présente en mémoire centrale. Lorsque la réponse n'est pas présente en mémoire centrale, ce qui arrive pour 10% des requêtes reçues, il faut ajouter un 150us supplémentaire de temps CPU et une lecture au disque qui prend 500us (disque rapide de type SSD). Finalement, lorsque la réponse n'est pas en mémoire ni sur disque, ce qui arrive pour 1% des requêtes reçues, il faut ajouter à ces deux temps le temps pour une demande récursive au serveur plus haut dans la hiérarchie, ce qui ajoute 200us de temps CPU et 20ms (ms pour miliseconde) d'attente. Si le processus pour ce service traite séquentiellement les requêtes avec un seul fil d'exécution (thread), combien de requêtes par seconde peut-il servir? Si plusieurs fils d'exécution (thread) sont utilisés de manière à maximiser la performance, combien de requêtes par seconde peut-il servir et combien de fils d'exécution doit-on utiliser? **(2 points)**

*Ce service requiert en moyenne par requête: pour le CPU 150us + 150us * 0.1 + 200us * 0.01 = 167us, pour le disque 500us * 0.1 = 50us, pour l'attente 20000us * .01 = 200us, pour un total de 417us. Avec un seul fil d'exécution, ce système peut servir 1s / 417us / requête = 2398.08 requêtes / seconde. Le goûlot d'étranglement ici est le CPU plutôt que le disque. Avec assez de fils d'exécution, le système pourrait servir 1s / 167us / requête = 5988.02 requêtes / seconde. Il faut s'assurer d'avoir assez de fils pour couvrir les besoins du CPU (1) et le reste de la requête en proportion du temps pris par chaque composante. On regarde donc le total pris pour la requête sur le temps pris par le CPU. Ceci demanderait 1 fil * 417us / 167us = 2.497 fils, donc 3 fils d'exécution.*

c) Il y a eu, au cours des dernières années, quelques épisodes où des attaques en déni de service ont été lancées contre les serveurs de nom (DNS) principaux sur l'Internet. Dans un tel cas, il peut arriver que les serveurs racines, complètement submergés, soient pratiquement inopérants. Cependant, la plupart des utilisateurs n'ont rien remarqué et ils ont pu continuer à utiliser la plupart des sites populaires. Toutefois, certains sites moins populaires, mais aussi quelques sites

populaires qui font un usage plus dynamique des serveurs de nom, n'étaient effectivement pas accessibles en raison de l'attaque. Comment expliquez-vous cela? **(1 point)**

Le service de nom (DNS) fonctionne efficacement en grande partie en raison des caches maintenues dans les serveurs à différents niveaux. Généralement, les réponses fournies par les serveurs de nom en autorité viennent avec une échéance de 24 heures. Ainsi, les adresses populaires sont en cache et peuvent fonctionner pour 24 heures, même si le serveur de nom au plus haut niveau n'est pas disponible. Si une attaque devait réussir à se poursuivre pour plus de 24 heures, ceci pourrait avoir des conséquences plus importantes. Lorsqu'un site est moins populaire et que son adresse ne se trouve pas en cache du serveur interrogé, il peut effectivement y avoir un problème. Une autre catégorie de site à risque est celle des sites qui utilisent le service de nom de manière très dynamique, par exemple pour répartir la charge entre plusieurs serveurs. Souvent les adresses retournées auront une échéance très courte, par exemple de quelques minutes, et les valeurs en cache deviennent rapidement invalides.

Question 2 (5 points)

a) A l'occasion du nouvel an, vous désirez inviter à la maison les parents de votre conjoint. Cependant, ceux-ci, des ingénieurs, accordent une très grande importance à la précision du temps. A 10h05m00s, heure de votre domicile, vous envoyez un drone avec caméra lire le temps sur l'horloge atomique des parents qui n'habitent pas très loin. Rendu là, le drone filme l'horloge qui affiche 10h10m05s, se pose 5 minutes pour reposer ses moteurs, et filme à nouveau l'heure sur l'horloge atomique qui est maintenant 10h15m05s. Le drone revient à votre domicile à 10h14m00s, heure de votre domicile. Vous pouvez alors visionner le film et connaître les deux temps qui ont été filmés, à l'arrivée et au départ du drone de chez les parents. Quel ajustement apporterez-vous à l'horloge de votre domicile pour la synchroniser avec l'horloge atomique des parents suite à ces lectures? Quelle est l'incertitude sur cet ajustement? Si vous connaissez la direction et la force du vent ainsi que la localisation du domicile des parents, est-ce que cela vous permettrait d'obtenir un meilleur ajustement du temps? **(2 points)**

Nous pouvons ainsi calculer a = t(i-2) - t(i-3) = 10h10m05s - 10h05m00s = 5m05s et b = t(i-1) - t(i) = 10h15m05s - 10h14m00s = 1m05s. L'ajustement est donc de (a+b)/2 = (5m05s + 1m05s) / 2 = 3m05s (avancer l'heure d'autant). L'incertitude est de (a-b) = (5m05s - 1m05s) = 4m00s, soit plus ou moins 2 minutes. S'il n'y a pas de vent et que les temps d'aller et de retour sont strictement égaux, l'ajustement sera parfait. Puisque le drone est parti de 10h05 à 10h14, soit 9 minutes, et qu'il est resté là-bas 5 minutes, le temps de trajet total est de 4 minutes, soit 2 minutes dans chaque direction. Ainsi, puisque le drone part à 10h05 et lorsqu'il arrive 2 minutes plus tard (à 10h07 de votre heure) il voit 10h10m05s, on comprend que l'ajustement à appliquer soit effectivement de 3m05s. En connaissant la position des deux endroits, le temps de trajet total du drone et la vitesse du vent, on pourrait calculer la vitesse effective que le drone a eu dans chaque direction, en tenant compte du vent, et savoir quelle est l'asymétrie du temps d'aller par rapport à celui de retour. En tenant compte ainsi du temps de trajet réel, on peut faire un ajustement plus précis. Ainsi, si après 1 minute de trajet (10h06 de votre heure) on lit 10h10m05, cela signifierait que le bon ajustement serait plutôt de 4m05.

b) Le 31 décembre prochain, une seconde sera ajoutée en fin de journée de sorte qu'il sera éventuellement 23h59m60s, et que seulement une seconde plus tard il sera 0h00m00s. Pourquoi l'organisme en charge de maintenir le temps fait-il une telle modification qui pourrait perturber le décompte de nombreux fêtards lors de la veille du jour de l'an? La compagnie Google, soucieuse d'aplanir le problème, offre un service public de serveurs de temps (NTP) et compte cacher cette seconde supplémentaire, qui pourrait causer des ennuis à plusieurs logiciels, en ralentissant le temps d'une seconde sur une période allant de 10 heures avant et 10 heures après la fin de la journée du 31 décembre. Vous désirez profiter de ce ralentissement du temps pour améliorer votre record de 20s pour courir un 100 mètres. Que deviendra ce temps de 20s lorsque mesuré en utilisant le temps ralenti par Google? **(2 points)**

*L'organisme en charge de coordonner le temps veut aligner notre heure sur le mouvement de la Terre. Cependant, la vitesse de rotation de la terre varie un peu dans le temps en fonction notamment des marées et des mouvements dans le noyau de la Terre. Il faut donc ajouter une seconde de temps en temps pour maintenir le synchronisme avec le cycle planétaire. En ralentissant le temps pour enlever 1 seconde sur 20 heures, ceci donne pour 20s: 20s * (20h * 60m/h * 60s/m) / (20h * 60m/h * 60s/m + 1s) = 19.999722226s.*

c) Lors du travail pratique 2, un répartiteur envoyait des tâches à plusieurs serveurs. Il pouvait arriver que certains serveurs soient tués au beau milieu de l'exécution d'une tâche. Comment est-ce que le répartiteur pouvait s'apercevoir d'une telle panne et comment pouvait-il réagir afin de tolérer cette panne? **(1 point)**

Un serveur ainsi arrêté, tout comme un réseau qui tombe en panne, n'envoie pas de message pour prévenir que la réponse attendue n'arrivera pas. Il faut que le coordonnateur se mette une minuterie à chaque requête envoyée, de manière à conclure à une panne si la réponse n'arrive pas avant un temps raisonnable. A ce moment, il peut conclure que le serveur qui n'a pas encore répondu est en panne et redonner cette tâche à un serveur fonctionnel. Le coordonnateur pourrait aussi communiquer de temps en temps avec les différents serveurs pour suivre les progrès et découvrir plus tôt si un serveur tombe en panne.

Question 3 (5 points)

a) Lesquelles des transactions T, U et V pourraient être validées si une validation en reculant était utilisée pour vérifier la cohérence des transactions? Une validation en avançant? Justifiez. **(2 points)**

T: Début
T: Read(a)
T: Write(b,1)
U: Début
U: Read(a)
U: Write(c,2)
V: Début
T: Compléter
U: Read(b)

V: Read(a)
V: Read(b)
V: Read(c)
V: Write(c,3)
V: Compléter
U: Compléter

Pour la validation en reculant, au moment où T termine, il ne peut y avoir de problème et T est accepté. Pour compléter V, on examine ce qui est lu (a, b, c) versus ce qui a été écrit par T (b) et V ne peut compléter. Rendu à compléter U, ce qui est lu (a, b) intersecte avec T (b) et U doit aussi être abandonné. Pour la validation en avançant, au moment où T termine, ce que T a écrit (b) n'intersecte pas avec ce que U a lu (a) ni V (rien encore) et T peut compléter. Au moment de compléter V, les écritures (c) n'intersectent pas avec les lectures de U (a,b) et V peut compléter. La dernière transaction, U, peut nécessairement compléter.

b) Une transaction répartie implique un coordonnateur, un client et plusieurs serveurs. Dans un premier cas, i), le client a ouvert une transaction (e.g., acheter des billets de spectacle pour sa famille), spécifié plusieurs opérations pour la transaction et demandé de compléter la transaction. Il est maintenant en attente de la confirmation que la transaction est acceptée. Il attend un certain temps et ne reçoit toujours pas de réponse. Peut-il en conclure que la transaction est effectuée ou non? Peut-il demander à ce point d'annuler la transation? Que peut-il faire? Dans un second cas, ii), le coordonnateur d'une transaction reçoit la demande du client de compléter la transaction et envoie à chaque serveur impliqué une demande à savoir s'ils sont prêts à commettre la transaction. Il reçoit des réponses (affirmatives) de tous les serveurs sauf 1 (dont le réseau est possiblement en panne). Peut-il relancer le serveur qui ne répond pas? Peut-il attendre indéfiniment? Que peut-il faire? **(2 points)**

Dans le premier cas, i), le client ne peut rien conclure. Peut-être que le coordonnateur ou le réseau est tombé avant que sa demande de compléter ne soit arrivée, ou après que la transaction soit complétée mais avant que la confirmation ne lui soit parvenue. Sa seule possibilité est de recontacter le coordonnateur jusqu'à ce qu'il obtienne une réponse, une fois que celui-ci aura recouvré ses données. Dans le second cas, ii), le coordonnateur peut encore sans problème abandonner la transaction à ce point-ci, puisqu'aucun message de confirmation n'a encore été envoyé. Peut-être que le serveur qui ne répond pas n'a pas reçu sa demande s'il est prêt, ou peut-être qu'une réponse positive a été envoyée mais ne lui est pas parvenue. Il peut relancer le serveur mais ne peut attendre indéfiniment, puisque pendant ce temps il fait aussi attendre le client et utilise des ressources sur les autres serveurs. Si la réponse tarde trop, il annule la transaction tout simplement et envoie des messages à cet effet.

c) Sachant que vous avez complété le travail pratique 3 et êtes un expert avec OpenStack et Heat, on vous consulte car le gabarit suivant comporte un problème. Dites quelle ligne devrait être corrigée et comment. **(1 point)**

```
resources:
   web_nodes:
       type: OS::Heat::ResourceGroup
```

```
        properties:
            count: 2
            resource_def:
            #Définition des serveur.

    pool:
        type: OS::Neutron::Pool
        #Définition du Pool

    lbalancer:
        type: OS::Neutron::LoadBalancer
        properties:
            protocol_port: 8000
            members: {get_resource: web_nodes}
            pool_id: {get_resource: pool}
```

La ligne "members: get_resource: web_nodes" devrait plutôt être "members: get_attr: [web_nodes, refs]".

Question 4 (5 points)

a) Le système informatique d'une centrale solaire commande l'inclinaison des panneaux solaires et doit offrir une très grande fiabilité. Un arrêt peut causer une diminution importante de la production électrique voire même concentrer les rayons du soleil au mauvais endroit et faire griller les maisons voisines. Le système est composé de 3 ordinateurs en redondance (Triple Modular Redundancy) qui reçoivent les mêmes entrées et dont les sorties sont passées par un voteur. Le voteur est très fiable et produit une sortie tant que deux des trois ordinateurs sont en bon ordre (donnent le même résultat). Chaque ordinateur a une probabilité de défaillance (donner un mauvais résultat) de 0.1. Quelle est la probabilité de défaillance de ce système de trois ordinateurs redondants? On vous propose de changer le voteur pour que, s'il obtient 3 réponses différentes, il propagera en sortie la sortie d'un des ordinateurs pré-sélectionné (e.g., l'ordinateur #1, car il est plus neuf). Est-ce que cela augmente la disponibilité du système? Est-ce qu'il y a des inconvénients à faire cela? **(2 points)**

Dans le premier cas, le système sera opérationnel tant qu'il y a au moins 2 ordinateurs fonctionnels (2 ou 3 sur 3). La probabilité d'avoir les 3 ordinateurs fonctionnels est de $0.9^3 = 0.729$. *Celle d'avoir 2 ordinateurs sur 3 est de* $3!/(2! \times 1!)0.9^2 \times 0.1 = 0.243$. *Le total est donc de* $0.729 + 0.243 = 0.972$. *Une autre manière de le calculer est que le tout est fonctionnel lorsqu'il y a 0 ou 1 ordinateur en panne, ce qui donne* $0.9^3 + 3!/(1! \times 2!)0.1 \times 0.9^2 = 0.972$. *Dans le cas où on modifie le voteur pour prendre une chance et utiliser l'ordinateur #1 en dernier recours si les 3 réponses sont différentes, on ajoute un fonctionnement correct dans le cas où l'ordinateur #1 est fonctionnel et les deux autres sont incorrects. La probabilité de ceci est de* $0.9 \times 0.1 \times 0.1 = 0.009$. *Nous n'utilisons pas le terme* $n!/(k! \times (n-k)!)$ *car nous ne voulons pas toutes les combinaisons de 1 ordinateur fonctionnel et 2 non fonctionnels mais seulement le cas où l'ordinateur #1 est fonctionnel et les deux autres non fonctionnels. La disponibilité dans*

ce cas devient $0.972 + 0.009 = 0.981$. *Cependant, dans le cas où les 3 ordinateurs donnent une réponse différente, il peut être plus intéressant de déclarer le système défectueux et effectuer un arrêt sécuritaire (e.g., pointer les miroirs vers le ciel ou vers le sol) plutôt que d'aller chercher quelques minces chances d'avoir le système #1 qui est peut-être fonctionnel.*

b) Un dicton dit que les configurations de serveurs actif-passif sont plus simples à gérer mais moins performantes. Sur le système de fichiers CODA, chaque fichier est stocké sur plusieurs serveurs (typiquement 3). Lorsqu'un client veut lire un fichier, il lui suffit de le lire sur un seul serveur. Pour écrire un fichier, il doit l'écrire sur tous les serveurs, ou du moins ceux qui sont rejoignables puisque ce système est tolérant aux pannes. Est-ce que CODA utilise une configuration actif-passif ou actif-actif? Est-ce que le dicton s'applique ici? Donnez un exemple concret de performance différente et de cas complexe à gérer par rapport à un serveur seul? **(2 points)**

Il s'agit d'un système actif-actif puisque tous les serveurs sont actifs de manière symétrique. Avec 3 serveurs en redondance, les lectures peuvent être réparties entre les serveurs et on peut donc servir 3 fois plus de requêtes, ce qui en fait un système 3 fois plus performant qu'un serveur seul ou actif-passif. Si le réseau est partitionné et que deux clients modifient en même temps le même fichier sur deux serveurs différents (chacun dans sa partition et ne réussissant pas à passer la modification aux autres serveurs dans l'autre partition), on se retrouvera avec des nouvelles versions incohérentes et il faudra résoudre le conflit (fusionner les modifications ou annoncer à un des clients que sa mise à jour sera abandonnée). Le dicton s'applique puisque les serveurs actif-actif sont plus performants mais moins faciles à gérer que les serveurs actif-passif.

c) Dans le processus de développement classique, chaque développeur d'un gros projet extrait du système de gestion de code source (e.g., SVN ou Git) les fichiers du programme et en obtient une copie sur son poste. Ensuite, celui-ci peut modifier quelques fichiers et effectuer la compilation sur son poste. Une compagnie qui dispose d'une grappe de serveurs de compilation pourrait permettre aux développeurs d'envoyer la demande de compilation sur ces serveurs (envoyer les fichiers source et recevoir les fichiers compilés). Avec le Google Build System, la situation est différente. Expliquez ces différences et en quoi permettent-elles de sauver du temps de compilation. **(1 point)**

Avec le Google Build System, les fichiers résident déjà sur les serveurs et les fichiers compilés seront aussi créés directement sur les serveurs, ce qui sauve beaucoup de transferts de fichiers potentiellement volumineux. Le principal attrait du Google Build System est sa cache. Si un fichier a déjà été compilé par un développeur, toute compilation subséquente avec le même fichier et les mêmes options est évitée car on utilise le fichier déjà compilé. Plus encore, si le fichier compilé n'a pas changé par rapport à la version antérieure (e.g., seulement des commentaires ont été édités dans le code source), les étapes subséquentes qui en dépendent, comme l'édition de lien, peuvent être évitées aussi.

Le professeur: Michel Dagenais

ECOLE POLYTECHNIQUE DE MONTREAL

Département de génie informatique et génie logiciel

Cours INF8480: Systèmes répartis et infonuagique (Automne 2018)
3 crédits (3-1.5-4.5)

CORRIGÉ DE L'EXAMEN FINAL

DATE: Vendredi le 14 décembre 2018

HEURE: 13h30 à 16h00

DUREE: 2H30

NOTE: Aucune documentation permise sauf un aide-memoire, préparé par l'étudiant, qui consiste en une feuille de format lettre manuscrite recto verso, calculatrice non programmable permise

Ce questionnaire comprend 5 questions pour 20 points

Question 1 (4 points)

a) Un ordinateur contient un processeur à 8 coeurs, 4 disques de type SSD et 4 disques conventionnels. Il sert des requêtes DNS. Pour toutes les requêtes reçues, une recherche est effectuée en mémoire vive, ce qui prend 1ms de CPU. Ceci permet de servir 60% des requêtes. Pour les autres requêtes, dont la réponse ne se trouve pas en mémoire vive, une recherche est effectuée sur les disques SSD, ce qui prend 2ms de disque SSD. Ceci permet de servir la moitié des requêtes qui n'ont pas eu réponse en mémoire vive. Les requêtes restantes, dont la réponse n'a pas été trouvée en mémoire vive ni sur les disques SSD, seront servies à partir des disques conventionnels en 20ms de disque conventionnel. i) Quel est le nombre maximal de requêtes par seconde qui pourraient être servies si le logiciel traite séquentiellement les requêtes avec un seul thread? ii) Quel est le nombre maximal de requêtes servies si le logiciel utilise un grand nombre de threads et que les requêtes sont réparties uniformément sur les coeurs et les disques? Quel est le nombre minimal de thread requis pour maximiser le nombre de requêtes servies par seconde? **(2 points)**

Une requête demande en moyenne 1ms CPU, 0.4 x 2ms = 0.8ms de disque SSD et 0.5 x 0.4 x 20ms = 4ms de disque conventionnel, pour un total de i) 5.8ms ou 1000ms/s / 5.8ms/r = 172r/s. Avec de nombreux threads, les 8 coeurs pourraient traiter 8 x 1000ms/s / 1ms/r = 8000r/s, les disques SSD 4 x 1000ms/s / 0.8ms/r = 5000r/s, et les disques conventionnels 4 x 1000ms/s / 4ms/r = 1000r/s. Ce sont donc les disques conventionnels qui sont le facteur limitant et le nombre maximal de requêtes qui peuvent être servies est de ii) 1000r/s. Puisque chaque requête dure en moyenne 5.8ms sur un thread, cela donne 5.8ms-t/r x 1000r/s = 5800ms-t/s ou 5.8s de thread par seconde. Il faut donc au moins 5.8, soit iii) 6 threads, pour soutenir ce débit maximal. Ceci est cohérent avec la règle conservatrice qui suggère 1 thread par ressource (coeur ou disque) et assure qu'avec 8 + 4 + 4 = 16 nous aurions assez de threads.

b) Un ordinateur doit traduire l'adresse symbolique www.polenord.com en adresse IP numérique. Il a accès à 2 serveurs DNS redondants dans son réseau local, et aux 13 serveurs DNS racine redondants. Il doit commencer par interroger les serveurs DNS du réseau local pour trouver l'adresse des serveurs de nom pour le domaine polenord.com. Ces 2 serveurs ont exactement le même contenu et ont 60% de chances de contenir l'adresse recherchée. Toutefois, ils sont peu fiables et chaque serveur a 45% de chances d'être fonctionnel. Si ceux-ci sont en panne, ou n'ont pas ces adresses dans leur cache, l'ordinateur doit interroger les 13 serveurs racine. Ces 13 serveurs ont chacun l'information recherchée. Toutefois, ils sont eux aussi très peu fiables en ce moment en raison d'une attaque en déni de service, et chaque serveur a une probabilité de 10% d'être fonctionnel. Ensuite, ayant l'adresse des serveurs de nom pour le domaine polenord.com, l'ordinateur pourra obtenir l'adresse du site www.polenord.com. i) Quelle est la probabilité que l'ordinateur puisse obtenir l'adresse des serveurs de noms de polenord.com auprès des 2 serveurs DNS de son réseau local? ii) Quelle est la probabilité que les 13 serveurs DNS racine ne soient pas disponibles? iii) Quelle est la probabilité que l'ordinateur ne puisse obtenir l'information à propos des serveurs DNS pour le domaine polenord.com? **(2 points)**

Les 2 serveurs DNS locaux seront non disponibles avec une probabilité de $(1-.45)^2 = 0.3025$. *S'ils sont disponibles,* $1-0.3025 = 0.6975$, *ils auront la réponse dans 60% des cas, soit i)* $0.6975 \times 0.6 = 0.4185$. *Les 13 serveurs racines seront indisponibles si les 13 sont simultanément non fonctionnels ii)* $(1-0.1)^{13} = 0.254186583$. *L'ordinateur n'obtiendra pas sa réponse si elle*

n'est pas disponible sur le réseau local $1 - 0.4185 = 0.5815$ *et les serveurs racine ne sont pas disponibles* 0.254186583, *ce qui donne iii)* $0.5815 \times 0.254186583 = 0.147809498$.

Question 2 (5 points)

a) Un ordinateur A envoie un message à B à 13h00m10.200s pour obtenir le temps et reçoit une réponse à 13h00m10.600s, ces deux temps étant mesurés avec l'horloge de A. L'ordinateur B reçoit la requête de A à 13h00m05.300s et retourne sa réponse à A à 13h00m05.550s, ces deux temps étant mesurés avec l'horloge de B. Quel est le décalage à appliquer sur A? Quel est l'intervalle d'incertitude associé? **(2 points)**

a = 13h00m05.300s - 13h00m10.200s = -4.900s
b = 13h00m05.550s - 13h00m10.600s = -5.050s
Ajustement = (a+b)/2 = (-4.900s - 5.050s)/2 = -4.975s
Précision = (a-b)/2 = (-4.900s + 5.050s)/2 = 0.075s

Le décalage à appliquer à A est de -4.975s et l'incertitude est de +/- 0.075s.

b) Dans le nord du Canada, un serveur central d'exclusion mutuelle maintient les verrous qui servent à protéger les K fichiers qui représentent la liste de cadeaux de chaque personne. Ce serveur de verrous doit servir jusqu'à N threads répartis dans différents noeuds qui doivent accéder à ces fichiers. Chaque thread acquiert un après l'autre un certain nombre de verrous, suivant une discipline de verrouillage strict à deux phases, puis les relâche tous après la fin de la transaction. Lorsqu'un thread demande un verrou, cette opération est bloquante, le thread ne continue qu'après avoir obtenu le verrou. On suppose que les demandes de verrou sont toujours ordonnées correctement et qu'il n'y a pas d'interblocage. i) Combien de messages sont requis pour l'acquisition d'un verrou par un thread auprès du serveur central d'exclusion? ii) Si une case mémoire est requise dans le serveur pour chaque verrou en utilisation (numéro du thread qui détient le verrou) et pour chaque demande de verrou en attente (numéro du thread bloqué en attente pour ce verrou dans une liste), quel est le nombre maximal de cases mémoire qui peut être requis dans le serveur dans le pire cas? **(2 points)**

i) L'acquisition d'un verrou demande un message au serveur central par le thread et un message de réponse du serveur central vers le thread. Des accusés de réception pourraient être utilisés afin de ne pas bloquer indéfiniment en cas de perte d'un des deux messages sur le réseau. Il serait aussi possible de ne pas utiliser systématiquement d'accusé de réception mais plutôt que le thread envoie un message de vérification que sa demande a bien été reçue et est en attente si le délai est trop long. ii) Chaque verrou ne peut être détenu que par un seul thread. Chaque thread ne peut être en attente que d'un seul verrou. Ainsi, le nombre de cases mémoire pour les verrous en utilisation est au maximum de K et le nombre de cases pour les verrous en attente est au maximum de N (en fait N-1 puisqu'autrement on est en interblocage si tous les threads sont bloqués). Le nombre maximal de cases mémoire occupées pour cela est donc au total de K+N.

c) Un groupe d'ordinateurs utilise l'élection hiérarchique. Lorsqu'un des ordinateurs du groupe ne réussit pas à contacter le serveur, il déclenche le processus d'élection. Quelles sont les étapes qu'il va effectuer et quels messages va-t-il envoyer à cet effet? **(1 point)**

L'ordinateur qui ne réussit pas à contacter le serveur courant déclenche une élection. Il commence par envoyer un message d'élection au serveur le plus prioritaire. S'il ne reçoit pas de réponse après un certain temps (et possiblement plus d'un essai), il contacte le second serveur le plus prioritaire, et continue ainsi jusqu'à ce qu'il reçoive une réponse ou qu'il ne reste plus de serveur plus prioritaire que lui. Si aucun serveur plus prioritaire ne lui a répondu, il s'autoproclame le serveur et envoie un message à cet effet à tous.

Question 3 (4 points)

a) 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? **(2 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)
W: Début
W: Read(e)
T: Compléter
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 (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 T (b, f) et U (c, d). Il y a intersection (f) et la transaction W doit être abandonnée. Pour V, les lectures (d, e, g) doivent être vérifiées versus les écritures de T (b, f) et U (c, d), mais pas de W qui a été abandonné. Il y a intersection (d) et la transaction V doit être

abandonnée. En fait, T était complété avant que W ne lise f, et U était complété avant que V ne lise d, mais cette information plus détaillée n'est pas utilisée par la méthode de base en reculant. Elle serait utilisée par la méthode avec les estampilles de temps.

En avançant, il faut vérifier ce qui est écrit par T (b, f) versus ce que U (c, d), V (e) et W (e) ont lu à ce moment. 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 (e) et W (e, f); les lectures à venir de V et W ne sont pas encore connues à ce moment et n'ont donc 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 (d, e) à ce moment. 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) Trois transactions concurrentes, T, U et V veulent effectuer les opérations suivantes. Les opérations pour chacune des transactions sont effectuées en séquence, dans l'ordre. Cependant, l'ordre entre les opérations des trois transactions peut varier, puisque chaque transaction est effectuée par un thread parallèle concurrent. La cohérence des opérations est assurée par des verrous ordinaires (i.e. un verrou est pris sur une variable par la transaction au premier accès à cette variable, indifféremment pour une lecture ou une écriture). Est-ce qu'un interblocage pourrait se produire? Si oui, donnez une séquence d'opérations qui mènerait à un interblocage. **(2 points)**

	Transaction T	Transaction U	Transaction V
0	Début	Début	Début
1	Read(a)	Read(c)	Read(d)
2	Read(b)	Read(d)	Read(e)
3	Read(f)	Write(c)	Read(f)
4	Write(c)	Write(c)	Write(e)
5	Write(f)	Write(d)	Compléter
6	Compléter	Compléter	

Oui, un interblocage est possible. Si T effectue ses opérations 0 à 3, il acquiert les verrous pour a, b et f. Ensuite, U avec ses opérations 0 à 1 pourrait acquérir c. Finalement, V pourrait effectuer ses opérations 0 à 2 et acquérir d et e. A ce moment, T voudra acquérir c (détenu par U), U voudra acquérir d (détenu par V) et V voudra acquérir f, détenu par T et nous avons un interblocage.

Question 4 (4 points)

a) Deux scénarios sont comparés pour offrir un service de base de données. Dans le premier scénario a), 2 serveurs redondants viennent chacun avec une unité de disque RAID (4 disques dont au moins 3 sur 4 doivent être fonctionnels). Dans le second scénario b), 3 serveurs redondants viennent chacun avec 3 disques (qui doivent les 3 être fonctionnels). La probabilité d'être fonctionnel est 0.85 pour un serveur (hormis les disques) et de 0.9 pour un disque. Quel scénario sera le plus fiable? **(2 points)**

L'unité RAID a une probabilité de fonctionner de $0.9^4 = 0.6561$ *(4 disques fonctionnels) et* $4!/((4-3)!3!) \times 0.9^3 \times (1-0.9)^{4-3} = 0.2916$ *(exactement 3 disques fonctionnels) pour un total de* $0.2916 + 0.6561 = 0.9477$. *Pour le scénario a) un serveur est opérationnel si le serveur est fonctionnel de même que son unité RAID, ce qui donne* $0.9477 \times 0.85 = 0.805545$. *Le service sera disponible sauf si les 2 serveurs sont en panne, ce qui donne* $1-(1-0.805545)^2 = 0.962187253$. *Pour le scénario b), le serveur est fonctionnel si tout est fonctionnel (3 disques et le serveur)* $0.9^3 \times 0.85 = 0.61965$. *Le service sera foncitonnel sauf si les 3 serveurs sont en panne, ce qui donne* $1-(1-0.61965)^3 = 0.94497624$. *Ainsi, le scénario a) est un peu plus fiable.*

b) Dans le travail pratique 3, un gabarit `Heat`, `autoscaling.yaml`, vous était fourni en exemple. Quel est le principe d'un service qui se met à l'échelle automatiquement sur `OpenStack` avec `Heat`? La section suivante est extraite de ce fichier. Expliquez à quoi sert cette section. **(2 points)**

```
cpu_alarm_high:
  type: OS::Ceilometer::Alarm
  properties:
    description:
    meter_name: cpu_util
    statistic: avg
    period: 60
    evaluation_periods: 1
    threshold: 50
    alarm_actions:
      - {get_attr: [web_server_scaleup_policy, alarm_url]}
    matching_metadata: {'metadata.user_metadata.stack':
                          {get_param: "OS::stack_id"}}
    comparison_operator: gt
```

La mise à l'échelle se base sur un répartiteur de requêtes, et un bassin de machines virtuelles identiques pouvant servir les requêtes. Lorsque la charge est trop élevée, de nouvelles instances de machines virtuelles sont ajoutées au bassin. Lorsque la charge est trop faible, des machines virtuelles sont retirées. Ainsi, le nombre de machines virtuelles disponibles s'ajuste en fonction de la charge. Le gabarit `Heat` *permet de spéficier les différentes ressources requises pour implémenter un tel service, ainsi que leurs paramètres. La section de gararit* `Heat` *fournie sert à instancier une ressource de type* `OS::Ceilometer::Alarm`. *Cette ressource monitore la métrique* `cpu_util` *qui est le taux d'utilisation du CPU. Si le taux d'utilisation du CPU monte au-dessus de 50% pendant 60 secondes, l'alarme exécute une action qui active la ressource* `web_server_scaleup_policy` *pour ajouter des machines virtuelles.*

Question 5 (3 points)

Vous devez planifier un nouveau centre de données et comparer différents scénarios. Vous avez déjà prévu examiner la performance des systèmes (et les revenus qu'on peut en tirer), le coût des ordina-

teurs, le coût du bâtiment ainsi que les coûts d'opération et de renouvellement des équipements, afin de déterminer le projet le plus rentable sur la durée de vie anticipée. Devez-vous comme ingénieur aussi vérifier les aspects du développement durable de ce projet? i) Quelles sont les lois applicables? ii) Quelles sont les différentes phases du cycle de vie? iii) Quels sont les quatre différents types d'impact sur l'environnement? **(3 points)**

L'ingénieur doit tenir compte des conséquences de l'exécution de ses travaux sur l'environnement dans une perspective de développement durable. i) Ceci est prescrit par la loi canadienne de 2008 sur le développement durable, la loi québécoise de 2006 sur le développement durable, et le code de déontologie de l'Ordre des Ingénieurs du Québec. La procédure d'évaluation environnementale du BAPE ne s'applique normalement pas à un centre de données. ii) Les phases du cycle de vie sont la fabrication (construction du site et fabrication des équipements), le transport (des matériaux pour la construction et des équipements), l'opération du site (entretien, alimentation électrique...) et finalement la fin de vie (la démolition ou conversion du bâtiment, la restauration du site, et le recyclage ou la mise aux rebuts des matériaux et équipements). iii) Pour toutes les activités qui se retrouvent dans ces différentes phases, il faut examiner l'impact sur la santé humaine, sur l'écologie, sur les changements climatiques et sur l'appauvrissement des ressources. Pour un centre de données typique, une grande partie de l'impact est reliée à la consommation d'énergie pendant l'opération de l'équipement informatique et de la climatisation. Un autre impact non négligeable est la fabrication des équipement informatiques et électriques, en particulier le raffinage de l'or et du cuivre requis pour ces équipements. Etant donné l'importance de la consommation électrique, une source d'énergie propre, et un climat froid qui ne requiert pas de climatisation, peuvent diminuer considérablement l'impact négatif d'un centre de données.

Le professeur: Michel Dagenais

ECOLE POLYTECHNIQUE DE MONTREAL

Département de génie informatique et génie logiciel

Cours INF4410: Systèmes répartis et infonuagique (Hiver 2017)
3 crédits (3-1.5-4.5)

CORRIGÉ DE L'EXAMEN FINAL

DATE: Jeudi le 20 avril 2017

HEURE: 9h30 à 12h00

DUREE: 2H30

NOTE: Toute documentation permise, calculatrice non programmable permise

Ce questionnaire comprend 4 questions pour 20 points

Question 1 (5 points)

a) Le serveur DNS d'un fournisseur de service Internet peut servir des requêtes DNS soit de manière itérative ou de manière récursive. De manière itérative, le serveur traite la requête avec son CPU pendant 2ms, lit en plus son disque dans 20% des cas pendant 20ms, et retourne dans 40% des cas la réponse demandée et dans 60% des cas une redirection vers un serveur plus haut dans la hiérarchie. En recevant une redirection, le client doit aller chercher la réponse sur un autre serveur, ce qui lui prend 10ms. Si le serveur est configuré pour fonctionner de manière récursive, il prendra aussi 2ms de CPU, et en plus 20ms de disque dans 20% des cas. Ensuite, il retournera la réponse demandée dans 90% des cas et dans 10% des cas devra faire une requête et attendre la réponse avant de la retourner, ce qui lui demande en plus 1ms de CPU et 20ms d'attente. Quel sera le délai moyen vu par un client pour obtenir la réponse demandée dans chaque cas (récursif ou itératif)? Quel est le nombre de requêtes par seconde que peut soutenir le serveur, s'il n'utilise qu'un seul thread, de manière itérative? De manière récursive? **(2 points)**

De manière itérative, une requête prend $2ms + .2 \times 20ms = 6ms$*. A cela, il faut ajouter 10ms dans 60% des cas soit 6ms, pour un total de 12ms. De manière récursive, le délai est de* $2ms + .2 \times 20ms + .1 \times (1ms + 20ms) = 8.1ms$*. La manière récursive est donc plus intéressante de ce point de vue, puisqu'elle fait augmenter le taux de succès en cache du serveur. Dans le premier cas, le serveur peut traiter une requête en 6ms, puisqu'après le client est redirigé vers un autre serveur et le serveur du fournisseur Internet est libre pour la prochaine requête, soit* $1000ms/s/6ms/r = 166$ *requêtes par seconde. Dans le second cas, il peut fournir* $1000ms/s/8.1ms/r = 123$ *requêtes / seconde.*

b) Sur un serveur DNS en autorité, la base de donnée entre entièrement en mémoire et l'ingénieur responsable a réussi à faire traiter chaque requête en 3us (1us recevoir le paquet UDP de requête, 1us chercher la réponse en mémoire, 1us renvoyer la réponse). Chaque paquet UDP (requête ou réponse) occupe en moyenne 100 octets. Pour simplifier le calcul, on suppose que les transferts peuvent se faire à la bande passante indiquée, on néglige les autres effets comme la latence d'envoi pour les paquets. Le serveur est connecté à deux réseaux qui peuvent chacun soutenir 100 mégabits/s par seconde simultanément dans chaque direction. Un usager malicieux a réussi à prendre le contrôle d'un certain nombre de processeurs dans des caméras IP et les utilise pour effectuer une attaque en déni de service sur le serveur DNS. Chaque caméra IP est capable de générer 100000 paquets de requête par seconde et est connectée par un réseau à 1mégabit/s. Les requêtes sont émises avec de fausses adresses de retour et ne reviennent jamais aux caméras. Combien de caméras sont requises pour saturer ce serveur DNS, en tenant compte de la capacité de leur processeur et de leur réseau respectifs? **(2 points)**

Le CPU du serveur peut traiter $1000000us/s/3us/r = 333333r/s$*. Le réseau peut laisser passer* $2 \times 100000000bits/s/(100octets/p \times 8bits/octet) = 250000paquets/s$*. Le facteur limitant est le réseau et le serveur peut donc traiter 250 000 requêtes par seconde. Chaque caméra peut envoyer* $1000000bits/s/(100octets/p \times 8bits/octet) = 1250paquets/s$*. Le serveur sera donc saturé à partir de* $250000/1250 = 200$ *caméras malicieuses.*

c) Pour les requêtes de recherche avec le protocole LDAP, il est possible de spécifier une limite sur la longueur de la réponse ainsi que sur le traitement requis. Pourquoi de tels paramètres ont-ils été incorporés à ce protocole? Dans quelle situation est-ce utile? **(1 point)**

Les répertoires LDAP peuvent être très gros et les requêtes assez complexes (filtres compliqués). Le client peut donc vouloir spécifier des limites pour ne pas recevoir trop d'information ni surcharger le serveur. Le serveur peut aussi à l'interne imposer des quotas par requête, selon le niveau de privilège du client qui fait la requête. Par exemple, un client pourrait demander la liste des usagers, pensant qu'ils ne sont qu'une dizaine, et se retrouver avec une réponse qui en contient des dizaines de milliers.

Question 2 (5 points)

a) Un client A effectue deux requêtes auprès d'un serveur de temps B. La première requête part de A à 9h00m40.000 et arrive à B à 9h00m20.010, puis la réponse part de B à 9h00m20.060 et arrive à A à 9h00m40.100. La seconde requête part de A à 9h01m50.000 et arrive à B à 9h01m30.030, puis la réponse part de B à 9h01m30.130 et arrive à A à 9h01m50.140. Quelles sont les valeurs de décalage et d'incertitude calculées pour chaque requête? Laquelle valeur (i.e. de quelle requête) devrait-on utiliser? Peut-on combiner les informations des requêtes pour avoir un meilleur estimé du décalage? **(2 points)**

Pour la première requête on a $a = 9h00m20.010 - 9h00m40.000 = -19.990$ *et* $b = 9h00m20.060 - 9h00m40.100 = -20.040$ *donc le décalage est de (a+b)/2 +/- (a-b)/2 soit -20.015s +/- .025s ou de -19.99s à -20.04s. Pour la seconde requête on a* $a = 9h01m30.030 - 9h01m50.000 = -19.97$ *et* $b = 9h01m30.130 - 9h01m50.140 = -20.010$ *donc le décalage est de -19.99 +/- .02 ou de -19.97 à -20.01. La deuxième requête donne un résultat plus précis et pourrait être retenue. En combinant les deux valeurs, on voit que l'intersection des intervalles donne de -19.99 à -20.01 ou -20s +/- .01s, ce qui est encore plus précis. Il faut bien comprendre ici que les mesures de temps sont très précises. L'incertitude est sur le délai réseau dans chaque direction. Dans ce cas-ci, par chance, le délai a été très court dans une direction pour un échange et très court dans l'autre direction pour l'autre échange. En recoupant les informations des deux intervalles possibles, on parvient à profiter de toute cette information et raffiner nos bornes. Une autre manière de considérer le problème est d'utiliser l'envoi de la première requête et la réception associée à la seconde requête, ce qui donnerait* $a = 9h00m20.010 - 9h00m40.000 = -19.990$ *et* $b = 9h01m30.130 - 9h01m50.140 = -20.010$*, donc un décalage de -20s +/- .01s.*

b) Dans le cadre du travail pratique 2, vous avez implémenté un service de calcul sécurisé (pas de réponse malicieuse) et un service non sécurisé (réponses possiblement malicieuses). Quelle était votre stratégie pour détecter et laisser tomber les réponses malicieuses? Quel était le surcoût de cette détection (temps CPU pour calcul sécurisé versus calcul non-sécurisé, mais avec très peu de réponses malicieuses finalement)? Quelle était la probabilité de ne pas détecter une réponse malicieuse si les réponses malicieuses sont générées indépendamment sur chaque serveur de calcul? Si elles sont générées de manière concertée? **(2 points)**

Le surcoût de la détection est de faire chaque calcul deux fois, donc le double du temps CPU requis autrement. Si les réponses malicieuses sont générées de manière indépendante et sont réparties uniformément dans l'intervalle, la chance que la seconde réponse malicieuse donne le même résultat que la première réponse malicieuse est de par exemple $1/4000$ *si la réponse peut aller de 0 à 3999. La probabilité d'avoir une réponse malicieuse non détectée est donc la probabilité d'avoir deux réponses malicieuses fois la probabilité que la réponse malicieuse ne*

soit pas détectée. Par exemple, si le taux de réponse malicieuse est de 80% sur chaque serveur de calcul, ce serait $.8 \times .8 \times (1/4000)$*. Si les réponses malicieuses sont concertées, la probabilité sera celle d'avoir deux réponses malicieuses, soit dans l'exemple* $.8 \times .8$*.*

c) Nous avons vu deux algorithmes qui peuvent être utilisés pour une élection dans un système réparti, l'élection hiérarchique et l'algorithme de Paxos. L'élection hiérarchique est a priori beaucoup plus simple. Quels sont donc les avantages de l'algorithme de Paxos? Donnez un exemple où seul l'algorithme de Paxos fonctionnerait correctement? **(1 point)**

L'élection hiérarchique est basée sur un délai et est donc synchrone. Par exemple, en cas de long délai à répondre, ou en cas de partition du réseau, on peut se retrouver avec deux élus simultanément pendant une certaine période. L'algorithme de Paxos est plus complexe mais beaucoup plus robuste. Son fonctionnement correct ne sera pas affecté par un partitionnement ou des délais trop longs. Il est aussi plus général puisqu'il fournit un consensus, dont l'élection est un cas particulier.

Question 3 (5 points)

a) Lesquelles des transactions T, U et V pourraient être validées si une validation en reculant était utilisée pour vérifier la cohérence des transactions? Une validation en avançant? **(2 points)**

T: Début
U: Début
V: Début
T: Read(e)
T: Read(a)
T: Read(d)
U: Read(a)
U: Read(e)
T: Write(a,21)
T: Write(b,12)
T: Compléter
U: Read(b)
V: Read(b)
U: Write(c,3)
U: Write(f,8)
U: Compléter
V: Read(c)
V: Write(d,12)
V: Write(e,1)
V: Compléter

Pour la validation en reculant, la première transaction à terminer, T, est toujours validée. Au moment de terminer U, T qui précède a écrit (a, b), alors que U a lu (a, b, e). Ainsi U ne peut être validée. Rendu à V, seulement T a complété en écrivant (a,b), alors que V a lu (b, c). V ne

peut compléter. En avançant, la transaction T regarde ce qu'elle a écrit (a, b) versus ce que U et V ont lu (a, e). T ne peut compléter car elle met en péril U. Rendu à compléter U, ses écritures sont (c, f) alors que les lectures de V jusqu'ici sont (b). Ainsi, U peut compléter. V, qui est la dernière du groupe, peut naturellement compléter.

b) Supposons que les opérations listées en a) pour les transactions T, U et V sont en fait des transactions réparties. Un premier serveur X contient les variables a et b, un second serveur Y contient les variables c et d, alors que le troisième serveur Z contient les variables e et f. Le système utilise un protocole de fin de transaction atomique à deux phases. En supposant que les trois transactions puissent être validées avec les opérations telles que listées, que retrouverait-on dans le journal de chacun des serveurs (X, Y et Z)? Expliquez qu'est-ce qui est écrit à quel moment. **(2 points)**

Au moment de compléter T, les variables (a, b) sont écrites et se trouvent sur X. Le coordonnateur écrit (prepare T), envoie un message de préparation à X qui écrira ces nouvelles valeurs (T: a=21, b=12) ainsi que le fait qu'il accepte (prepare T). En recevant l'acceptation de X, le coordonnateur écrit (commit T) et envoie le message à X qui écrit (commit T). Au moment de compléter U, les variables (c, f) sont écrites et se trouvent sur Y et Z. Le coordonnateur écrit (prepare U) et envoie un message de préparation à Y et Z. Le serveur Y écrit la nouvelle valeur (U: c=3) ainsi que le fait qu'il accepte (prepare U) et Z fera de même, (U: f=8; Prepare U). Après avoir reçu les deux acceptations, le coordonnateur écrit (commit U) et envoie le message à Y et Z qui écrivent chacun dans leur journal (commit U). Finalement, pour V, les écritures sont (d, e) et se trouvent sur Y et Z. Le coordonnateur écrit (prepare V) et envoie un message de préparation à Y et Z. Le serveur Y écrit la nouvelle valeur (V: d=12) ainsi que le fait qu'il accepte (prepare V) et Z fera de même, (V: e=1; Prepare V). Après avoir reçu les deux acceptations, le coordonnateur écrit (commit V) et envoie le message à Y et Z qui écrivent chacun dans leur journal (commit V).

c) Pour les transactions locales, les algorithmes de contrôle optimiste de la concurrence, validation en avançant et validation en reculant, peuvent être intéressantes. Sont-elles applicables pour des transactions réparties? Expliquez. **(1 point)**

Pour les transactions réparties, le verrouillage strict à deux phases ou le contrôle par les estampilles de temps peuvent être utilisés. Le contrôle optimiste de la concurrence par validation de la cohérence (en avançant ou en reculant) n'est pas utilisé car il demande de tout figer pendant la vérification de la cohérence, ce qui n'est généralement pas acceptable dans un système réparti.

Question 4 (5 points)

a) Un système transactionnel en ligne, pour des ventes aux enchères, utilise de la redondance à plusieurs niveaux. Il est connecté à 2 fournisseurs Internet et peut fonctionner tant que l'un des deux est opérationnel (i.e. chaque serveur est connecté aux deux réseaux). Il y a ensuite 3 serveurs de façade alors qu'il suffit d'un seul pour fonctionner. Les serveurs de façade font des requêtes à 2 serveurs de base de données redondants alors qu'un seul suffit. Deux unités de disque RAID sont utilisées. Chaque unité de disque RAID fonctionne si au moins 3 de ses 4 disques sont fonctionnels. Deux configurations sont possibles. Dans la première, chaque unité RAID est connectée à un seul des deux serveurs de base de données; un serveur ne fonctionnera

que si son unité RAID fonctionne. Dans la seconde configuration, chaque serveur de base de données peut accéder aux deux unités RAID; un serveur ne fonctionnera que si au moins une des 2 unités est fonctionnelle. Si la probabilité de panne est de .1 pour un fournisseur Internet, .2 pour un serveur de façade, .15 pour un serveur de base de données et .25 pour un disque, et que la probabilité de panne est négligeable pour les autres composantes, quelle sera la probabilité que le service soit disponible aux clients pour chacune des deux configurations? **(2 points)**

Pour que le service fonctionne, il faut que simultanément fonctionnent le service Internet, le service façade et le service de base de données. Pour la première configuration, il faut associer la panne de l'unité RAID connectée à la panne du serveur de base de donnée. Dans la seconde configuration, le service de disque est dissocié et doit simplement lui aussi être disponible. Pour le service Internet, il y a panne si les deux fournisseurs sont en panne, une probabilité de $.1^2 = .01$. *La probabilité de fonctionner est donc de* $1 - .01 = 0.99$. *Pour les trois serveurs de façade, il y a panne si les trois serveurs sont en panne, une probabilité de* $.2^3 = .008$ *ou* $1 - .008 = .992$ *de fonctionner. Pour les unités RAID, la probabilité de fonctionner est celle d'avoir 0 ou 1 disque en panne, ou 4 ou 3 disques fonctionnels, soit* $(1-0.25)^4 + 4!/(3! \times (4-3)!) \times (1-0.25)^3 \times 0.25 = 0.73828125$. *Dans la première configuration, un serveur de base de donnée fonctionne si lui et son unité fonctionnent, soit* $(1 - 0.15) \times 0.73828125 = 0.627539063$. *Le service fonctionnera sauf si les deux sont en panne* $(1-0.627539063)^2 = 0.13872715$, *soit* $1-0.13872715 = 0.86127285$. *Ceci donne donc un service fonctionnel si toutes ces composantes sont fonctionnelles soit* $0.99 \times .992 \times 0.86127285 = 0.845838841$. *Dans la seconde configuration, le service de base de donnée a une probabilité de panne de* $0.15^2 = 0.0225$ *et de fonctionner de* $1 - 0.0225 = 0.9775$. *Le service de disque a une probabilité de panne de* $(1 - 0.73828125)^2 = 0.068496704$ *et de fonctionner de* $1 - 0.068496704 = 0.931503296$. *Le service sera fonctionnel si toutes ces composantes sont fonctionnelles soit* $0.99 \times 0.992 \times 0.9775 \times 0.931503296 = 0.894227515$. *La seconde configuration est donc plus robuste que la première.*

b) Dans le cadre du travail pratique 3, vous avez utilisé des gabarits Heat afin de spécifier le comportement d'un service. Que doit-on mettre dans le gabarit afin de répartir la charge entre plusieurs serveurs? Que doit-on ajouter au gabarit Heat afin d'ajuster selon la charge le nombre de serveurs entre lesquels le travail est réparti? Donnez le type des principales ressources requises et leur fonction. **(2 points)**

Pour répartir la charge, il faut un groupe de machines, OS::Neutron::Pool, une adresse IP associée au groupe, OS::Neutron::FloatingIP, un moniteur qui vérifie les noeuds fonctionnels, OS::Neutron::HealthMonitor, et finalement le répartiteur de charge, OS::Neutron::LoadBalancer. Pour ajuster le nombre de serveurs selon la charge, il faut des conditions de déclenchement, OS::Ceilometer::Alarm associés, qui activent des politiques pour augmenter ou diminuer le nombre de noeuds, OS::Heat::ScalingPolicy, et finalement le groupe avec mise à l'échelle automatique, OS::Heat::AutoScalingGroup. La mise à l'échelle du groupe est une activité séparée mais complémentaire à ce que fait le répartiteur de charge.

c) Le système Google Wide Profiling permet de comparer la performance fournie par les différentes versions d'un logiciel, par différents types de matériel, ou même de calculer le coût global (temps CPU) pour une fonction donnée. Comment cela fonctionne-t-il? Est-ce que le surcoût est important pour obtenir ces informations? Sont-elles fiables? Expliquez. **(1 point)**

Le système GWP échantillonne les contenus de la pile à intervalle régulier sur un échantillon de noeuds. Cette information permet de voir combien consomme en moyenne chaque fonction ou ligne de code. Ceci peut être corrélé avec la version des logiciels et avec le type de matériel. On peut donc savoir le coût global de chaque fonction ou l'efficacité relative de chaque version ou type de matériel (prend plus de temps pour effectuer le même travail en moyenne). Le surcoût est facilement contrôlé en ajustant la fréquence d'échantillonnage et le nombre de noeuds faisant partie de l'échantillon. Ceci nous donne donc un compromis entre le surcôut et la vitesse à laquelle on peut avoir assez de données pour obtenir une bonne précision. Avec de l'échantillonnage, il y a toujours une certaine imprécision, surtout pour détecter des événements rares. Pour les événements fréquents, qui sont souvent les plus importants, la moyenne sera faite sur un grand nombre d'échantillons et les données seront passablement fiables.

Le professeur: Michel Dagenais

ECOLE POLYTECHNIQUE DE MONTREAL

Département de génie informatique et génie logiciel

Cours INF8480: Systèmes répartis et infonuagique (Hiver 2018)
3 crédits (3-1.5-4.5)

CORRIGÉ DE L'EXAMEN FINAL

DATE: Mercredi le 25 avril 2018

HEURE: 13h30 à 16h00

DUREE: 2H30

NOTE: Aucune documentation permise sauf un aide-memoire, préparé par l'étudiant, qui consiste en une feuille de format lettre manuscrite recto verso, calculatrice non programmable permise

Ce questionnaire comprend 4 questions pour 20 points

Question 1 (5 points)

a) Un serveur DNS sert des requêtes de manière récursive. Lorsqu'une requête est reçue d'un client, elle requiert 0.5ms de CPU. Dans 30% des cas, la réponse n'est pas en mémoire centrale et un accès au disque est requis en plus, ce qui ajoute 15ms en attente du disque. Finalement, pour 1/3 des cas où un accès disque est requis, la réponse n'est pas sur disque non plus et un accès récursif est requis, ce qui ajoute 200ms en moyenne d'attente après la requête par réseau au serveur de plus haut niveau. Si ce serveur exécute toutes les requêtes séquentiellement avec un seul thread, combien de requêtes par seconde pourra-t-il traiter au maximum? Si ce serveur contient un CPU à 4 coeurs et 8 disques, et que de nombreux threads sont disponibles pour servir les requêtes en parallèle, combien de requêtes pourra-t-il traiter au maximum? Quel est le nombre de threads requis pour atteindre ce maximum? On suppose que la charge entre les disques est bien répartie, tout comme la charge entre les coeurs. **(2 points)**

Pour une requête moyenne, on a 0.5ms de CPU, 0.3 x 15ms = 4.5ms d'attente après le disque et 1/3 x 0.3 x 200ms = 20ms d'attente après le serveur de plus haut niveau, pour un total de 25ms. Le serveur pourra donc traiter séquentiellement 1000ms/s / 25ms/r = 40 requêtes / s. Chaque coeur du CPU peut traiter 1000ms/s / 0.5ms/r = 2000r/s, soit 8000r/s pour les 4 coeurs. Chaque disque peut traiter 1000ms/s / 4.5ms/r = 222.22r/s, soit 1777.77r/s pour les 8 disques. Avec 1777.77r/s, chaque requête occupant un thread pendant 25ms ou .025s, ceci donne un total de 1777.77r x .025s = 44.44 secondes de temps de thread à chaque seconde, ce qui requiert au minimum 45 threads en parallèle.

b) Un client interroge 10 serveurs DNS redondants et retient la réponse la plus fréquente. Combien de serveurs en panne ce client pourra-t-il tolérer avant de ne plus avoir la bonne réponse si les pannes sont par omission? Par réponse aléatoire? Par réponse byzantine? **(2 points)**

Par omission, le client peut fonctionner même si 1 seul serveur est fonctionnel (et 9 sont en panne). Par réponse aléatoire, le client peut avec un peu de chance fonctionner si au moins 2 serveurs sont fonctionnels (et 8 en panne). En présence de réponses byzantines, le client doit avoir plus de la moitié des serveurs fonctionnels, soit 6 (et un maximum de 4 serveurs en panne).

c) Comment se comparent les services de nom (Directory service) par LDAP et X.500? Lequel est le plus simple? Le plus utilisé? **(1 point)**

Le système X.500 était très compliqué, en ayant une architecture se voulant capable de s'adapter à un grand nombre de situations. Il y a eu très peu de tels serveurs en opération et il y en a encore moins aujourd'hui. Le système LDAP s'inspire de X.500 tout en conservant une organisation beaucoup plus simple. C'est le service de répertoire le plus utilisé, sauf dans les environnements qui utilisent exclusivement le système d'exploitation Windows de Microsoft où Active Directory est généralement utilisé.

Question 2 (5 points)

a) Un client interroge un serveur pour synchroniser son horloge avec la méthode de Christian. Il effectue 3 requêtes en espérant avoir plus de précision et obtient les résultats suivants. Envoi de la requête 1 à 13h00m00.000 et réception à 13h00m01.000 de la réponse disant que

l'heure du serveur était 13h00m05.000. Envoi de la requête 2 à 13h00m02.000 et réception à 13h00m03.000 de la réponse disant que l'heure du serveur était 13h00m07.000. Envoi de la requête 3 à 13h00m04.000 et réception à 13h00m04.010 de la réponse disant que l'heure du serveur était 13h00m09.000. A l'aide de ces données, calculez la valeur de décalage à appliquer à l'horloge du client qui minimise l'incertitude sur le décalage, et donnez cette incertitude. **(2 points)**

L'incertitude est déterminée par le temps d'aller-retour de la requête. La requête 3 est celle avec le temps d'aller-retour le plus court, 13h00m.04.010 - 13h00m04.000 = 0.010. Le nouveau temps à appliquer au client est donc de 13h00m09.000 (heure du serveur) + 0.010 / 2 (aller-retour / 2) = 13h00m09.005 +/- .005. Le décalage à appliquer à ce moment sur l'heure du client qui était de 13h00m04.010 est donc de 13h00m09.005 - 13h00m04.010 = 4.995s +/- .005.

b) Dans le cadre du travail pratique 2, vous avez implémenté un service réparti de calcul. Comment pouviez-vous tolérer une panne par omission de réponse? Comment pouviez-vous tolérer une panne de mauvaise réponse? Pour quel type de panne ou quelle combinaison de pannes simultanées est-ce que votre système ne réussirait plus à fonctionner correctement? **(2 points)**

Les pannes par omission sont tolérées grâce à un temps d'expiration. Si ce temps est dépassé, on suppose que le serveur est en panne et on soumet la même requête à un autre serveur. Ceci ajoute un peu de latence mais n'empêchera pas le calcul d'être finalement obtenu. Si le système n'est pas sécurisé et que des réponses incorrectes peuvent être reçues, le répartiteur considère qu'une réponse est correcte seulement si deux serveurs donnent la même réponse. Autrement, il faut resoumettre la requête à d'autres serveurs jusqu'à obtention de deux réponses identiques. Si deux serveurs malicieux s'entendent pour retourner la même mauvaise réponse, le système ne fonctionnera pas correctement. Le système comporte aussi plusieurs composantes qui ne sont pas redondantes, par exemple le repartiteur ou le réseau.

c) Nommez 3 algorithmes d'élection différents et expliquez brièvement les avantages et désavantages de chacun (simplicité, efficacité, robustesse). **(1 point)**

L'élection en anneau est simple mais exige que tous les participants soient fonctionnels, ce qui est peu robuste. Le délai associé à faire le tour de l'anneau à 3 reprises est aussi un peu long. L'élection hiérarchique est relativement simple et est assez efficace dans le cas courant où un ordinateur de haut niveau de priorité peut rapidement se déclarer le plus fort, et conséquemment élu. Cependant, l'élection hiérarchique ne fonctionne pas bien si les délais sont trop longs ou si le réseau est partitionné. Le consensus de Paxos est l'algorithme le plus complexe. Il est très robuste et fonctionne (ne donne pas de résultat incorrect) même en cas de partition de réseau. Dans le cas simple, il est raisonnablement efficace.

Question 3 (5 points)

a) Lesquelles des transactions T, U et V pourraient être validées si une validation en reculant était utilisée pour vérifier la cohérence des transactions? Une validation en avançant? **(2 points)**

T: Début

U: Début
V: Début
U: Read(w)
T: Read(x)
U: Read(z)
T: Write(x,10)
T: Compléter
U: Read(x)
V: Read(w)
U: Read(y)
U: Write(y,22)
U: Compléter
V: Read(z)
V: Read(y)
V: Write(z,9)
V: Compléter

Pour la validation en reculant, la première transaction à terminer, T, est toujours validée. Au moment de terminer U, T qui précède a écrit (x), alors que U a lu (w, z, x, y). Ainsi U ne peut être validée. Rendu à V, seulement T a complété en écrivant (x), alors que V a lu (w, z, y). V peut compléter. En avançant, la transaction T regarde ce qu'elle a écrit (x) versus ce que U et V ont lu (w,z). T peut compléter. Rendu à compléter U, ses écritures sont (y) alors que les lectures de V jusqu'ici sont (w). Ainsi, U peut compléter. V, qui est la dernière du groupe, peut naturellement compléter.

b) Une transaction répartie T effectue les opérations suivantes. Les variables a, b et c sont sur le serveur X, les variables d, e et f sur le serveur Y, et les variables g, h et i sur le serveur Z. Le système utilise un protocole de fin de transaction atomique à deux phases. Que retrouvera-t-on dans le journal de chacun des serveurs, X, Y et Z? Quelles requêtes et réponses est-ce que le coordonnateur échange avec les trois serveurs pour le protocole de fin de transaction? Chaque requête ou réponse à chacun des serveurs pour le protocole de fin de transaction arrive à quel moment par rapport aux écritures dans le journal de ce serveur? **(2 points)**

T: Début; Read(a); Read(b); Read(c); Read(d); Read(f); Read(h); Read(i); Write(c,0);
Write(b,1); Write(d,2); Read(e); Write(f,3); Write(h,4); Write(i,5); Read(e); Write(c,6);
Write(e,7);

Dans le journal de X, on aura: P0: Write(c,0); P1: Write(b,1); P2: Write(c,6); P3: Préparer T(P0, P1, P2); P4: Compléter T,P3. Selon les optimisations faites, il est possible que l'entrée P0 soit absente ou qu'elle ne soit pas listée dans la préparation P3, étant donné que cette valeur est écrasée par une autre écriture de c dans la même transaction. Dans le journal de Y on aura: P0: Write(d,2); P1: Write(f,3); P2: Write(e,7); P3: Préparer T(P0, P1, P2); P4: Compléter T,P3. Dans le journal de Z, on aura: P0: Write(h,4); P1: Write(i,5); P2: Préparer T(P0, P1); P3: Compléter T,P2. Chaque serveur, au moment où on lui demande s'il est prêt pour la transaction T, écrit toutes les informations à propos de T, incluant l'entrée pour "Préparer", dans son journal.

Il peut alors répondre qu'il est prêt et accepte la transaction. Le coordonnateur, ayant reçu l'acceptation de chaque serveur, peut alors envoyer la confirmation à chaque serveur qui pourra en conséquence ajouter l'entrée "Compléter".

c) Quelle est la différence entre une analyse économique classique et une analyse complète du cycle de vie? **(1 point)**

Dans une analyse économique classique, seule la rentabilité est évaluée en utilisant les chiffres des coûts directs. Plusieurs aspects plus difficiles à mesurer, dont l'impact sur l'environnement ou l'appauvrissement des ressources, ne sont pas tenus en compte autrement que via des taxes directes qui seraient imposées à cet effet. Avec l'analyse du cycle de vie, toutes les phases sont considérées, de la construction et la fabrication des intrants à l'opération et au démantèlement éventuel avec le recyclage ou la mise aux rebuts. Certains éléments comme l'impact sur l'environnement et la santé humaine sont quantifiés, afin de comparer entre eux différents scénarios. Ces impacts ne sont pas nécessairement convertis en valeur monétaire, puisque cette conversion n'est pas facile à réaliser et dépend de nombreux facteurs et hypothèses. L'analyse complète du cycle de vie est donc beaucoup plus complète que l'analyse économique classique.

Question 4 (5 points)

a) Un laboratoire informatique contient 26 postes de travail, reliés à 2 serveurs redondants. Chaque serveur contient à son tour deux disques redondants. Il est prévu pour accueillir 25 équipes. Il faut donc qu'au moins 25 postes de travail soient opérationnels. Il faut aussi qu'au moins 1 des 2 serveurs soit opérationnel. Pour qu'un serveur soit opérationnel, il faut que son électronique soit opérationnelle et qu'au moins 1 de ses 2 disques le soit. La probabilité de fonctionner à un moment donné est de 0.95 pour un poste de travail, 0.8 pour l'électronique d'un serveur et 0.7 pour un disque. Quelle est la probabilité que le laboratoire soit opérationnel pour accueillir 25 équipes? **(2 points)**

Pour que le service soit disponible à 25 équipes, il faut que 25 ou 26 postes soient opérationnels et au moins 1 serveur. Pour qu'un serveur soit opérationnel, il faut que son électronique fonctionne et au moins 1 disque. La probabilité que les 26 postes soient fonctionnels est de $0.95^{26} = 0.26352$. *La probabilité qu'exactement 25 sur 26 postes soient fonctionnels est de* $26!/((26-25)! \times 25!) \times 0.95^{25} \times (1-0.95)^{(26-25)} = 0.3606$. *Le total est* $0.26352 + 0.3606 = 0.62412$. *Les disques d'un serveur fonctionnent sauf si les 2 sont en panne, ce qui donne* $1-(1-0.7)^2 = 0.91$. *Le serveur fonctionne si son électronique et ses disques fonctionnent, soit* $0.8 \times 0.91 = 0.728$. *Les serveurs fonctionneront sauf si les 2 sont en panne, soit* $1-(1-0.728)^2 = 0.926$. *Le laboratoire sera opérationnel pour 25 équipes si les postes et les serveurs le sont, soit* $0.62412 \times 0.926 = 0.57794$.

b) Pour l'établissement d'un nouveau centre de données, on doit effectuer une analyse complète qui inclut l'analyse du cycle de vie. Quelles sont les différentes phases à considérer dans une analyse de cycle de vie? Quels sont les quatre différents types d'impact sur l'environnement à considérer? Quels sont les éléments d'un centre de données qui ont généralement le plus d'impact et pendant quelle phase? Est-ce que la source d'énergie et le climat ont un effet sur l'impact d'un centre de données? Expliquez. **(2 points)**

Il faut tenir en compte toutes les phases du projet, de sa construction et la fabrication des équipements qui s'y trouvent à l'opération et à la fin de vie. La fin de vie inclut la démolition ou la conversion du bâtiment, et le recyclage ou la mise aux rebuts des matériaux et équipements. Pour toutes les activités qui se retrouvent dans ces différentes phases, il faut examiner l'impact sur la santé humaine, sur l'écologie, sur les changements climatiques et sur l'appauvrissement des ressources. Pour un centre de données typique, une grande partie de l'impact est reliée à la consommation d'énergie pendant l'opération pour l'équipement informatique et pour la climatisation. Un autre impact non négligeable est la fabrication des équipement informatiques et électriques, en particulier le raffinage de l'or et du cuivre requis pour ces équipements. Etant donné l'importance de la consommation electrique, une partie étant reliée à la climatisation, une source d'énergie propre, et un climat froid qui ne requiert pas de climatisation, peuvent diminuer considérablement l'impact négatif d'un centre de données.

c) Dans le gabarit Heat du service Web avec répartition de charge du travail pratique 3, vous avez défini des propriétés permettant d'instancier plusieurs ressources. Expliquez brièvement le rôle de chacune des ressources suivantes dans le travail pratique, OS::Neutron::HealthMonitor, OS::Neutron::Pool, OS::Neutron::LoadBalancer **(1 point)**

Pour répartir la charge, il faut un groupe de machines, OS::Neutron::Pool (un bassin de machines virtuelles identiques pour servir des requêtes), un moniteur qui vérifie les noeuds fonctionnels, OS::Neutron::HealthMonitor (un moniteur qui interroge les serveurs à intervalle régulier pour détecter ceux qui ne sont pas fonctionnels). Finalement on retrouve le répartiteur de charge, OS::Neutron::LoadBalancer (en sachant quel serveur est fonctionnel et en mesurant le temps de réponse, les requêtes sont réparties de manière à partager équitablement la charge et minimiser le temps de réponse).

Le professeur: Michel Dagenais