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