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 DU CONTRÔLE PÉRIODIQUE

DATE: Jeudi le 25 octobre 2012

HEURE: 13h45 à 15h45

DUREE: 2H00

NOTE: Toute documentation permise, calculatrice non programmable permise

Ce questionnaire comprend 4 questions pour 20 points

Question 1 (5 points)

Etant expert en systèmes répartis, on vous demande conseil pour l'établissement d'un système de haut-parleurs en réseau pour un domicile. Plusieurs paires de haut-parleurs peuvent se retrouver dans des pièces communicantes (même musique) ou des pièces séparées (musique différente ou non). Une tablette envoie la musique vers les paires de haut-parleurs. Un flux de musique stéréophonique demande 45000 échantillons de 16 bits par seconde pour chacun de deux côtés (gauche et droit). Le réseau sans fil utilise un seul canal de 12Mbit/s, partagé entre tous les haut-parleurs.

a) Quel est le nombre maximal de haut-parleurs qu'il est possible de supporter en utilisant des connexions TCP, de la tablette vers chaque paire? Des envois UDP en multi-diffusion? Pour des fins de simplification, on suppose que peu de paquets sont perdus et on néglige la taille des entêtes et les latences de communication, assumant que toute la bande passante est disponible pour le flux de musique. **(2 points)**

Chaque paire de haut-parleurs requiert 45000/sec x 16 bits x 2 côtés = 1440000 bits/s. Sur un réseau de 12Mbits/s, on peut avoir 12000000 bits/sec / 1440000 bits/sec /paire = 8.33 paires. On peut donc avoir 8 paires sous TCP. Sous UDP, plusieurs paires peuvent utiliser le même

flux de musique sans vraiment de limite. Par contre, il n'y a toujours que 8 flux de musique différents qui pourraient être envoyés, chacun potentiellement reçu en multi-diffusion et joué par plusieurs paires de haut-parleurs.

b) Le logiciel a été configuré pour lier ensemble les haut-parleurs de pièces communicantes et doit ainsi s'assurer qu'ils jouent toujours la même musique à tout instant (possiblement avec un très court délai configurable pour compenser le délai de propagation du son). Lorsque l'utilisateur décide subitement de changer de musique, il faut arrêter la chanson courante immédiatement, même s'il reste encore de l'information pour de nombreuses secondes en queue dans les tampons de la paire de haut-parleurs, avant de démarrer la nouvelle chanson. En fonctionnant avec UDP, si une base ne reçoit pas certains paquets à temps, elle ne jouera pas de son pour une courte période, ce qui n'est pas trop dérangeant si la pièce comporte plusieurs paires. Par contre, il ne faut surtout pas que deux paires dans la même pièce jouent en même temps des chansons différentes parce qu'une aurait reçu la commande de changer de chanson et l'autre pas, en raison de paquets perdus. Proposez un protocole permettant d'éviter cela? **(2 points)**

Ce problème revient à l'envoi d'un message atomique. On peut envoyer un premier message à chaque haut-parleur pour lui demander d'arrêter la musique sous peu et de répondre par un accusé de réception. Une fois que chacun a confirmé qu'il avait compris et allait arrêter la chanson en cours, on peut envoyer le message de confirmation qui initie la nouvelle chanson. De cette manière, on s'assure de ne jamais être dans la situation où deux chansons différentes sont jouées dans la même pièce.

c) Dans un scénario étendu où plusieurs utilisateurs pourraient mettre en queue des chansons à jouer, il faudrait un serveur qui maintient ces queues pour chaque groupe de paires de haut-parleurs et s'assure qu'elles soient cohérentes. Lorsque deux utilisateurs ajoutent une chanson presqu'en même temps dans une même queue, l'ordre résultant pour l'ajout dans la queue importe peu. Par contre, si un même utilisateur ajoute rapidement deux chansons dans une queue, il faudrait qu'elles apparaissent dans leur ordre d'entrée, même si par exemple le paquet contenant la première demande est perdu. Comment appelle-t-on une telle contrainte d'ordonnancement? Comment peut-on satisfaire une telle contrainte si le protocole sous-jacent utilisé est UDP? **(1 point)**

Il s'agit d'obtenir un ordre causal, par client. Une manière triviale de faire ceci est d'avoir un ordonnancement total avec chaque requête vers le serveur central qui est bloquante. Le client ne peut alors faire une seconde demande avant que la première n'ait été confirmée. Une implantation plus complexe mais plus performante (non bloquante) est d'avoir chaque client qui numérote ses requêtes. Le serveur sert les requêtes dans l'ordre et attend/redemande la ou les requêtes non reçues si une requête qui saute des numéros dans l'ordre est reçue.

Question 2 (5 points)

a) Tel qu'étudié dans votre premier travail pratique, comment se compare la performance entre un appel de fonction ordinaire (dans le même processus), un appel de fonction à distance

entre deux processus sur un même ordinateur, et un appel à distance entre deux ordinateurs? Donnez une idée du temps requis dans chaque cas et expliquez les étapes requises (système d'exploitation, réseau...) qui expliquent ce délai. **(2 points)**

Un appel de fonction ordinaire requiert quelques dizaines d'instructions (mettre les paramètres sur la pile, sauver les registres, initialiser quelques registres, exécuter le contenu de la fonction, restorer les registres), ce qui prend quelques dizaines de nanosecondes. Un appel de procédure à distance entre deux processus sur un même ordinateur requiert quelque chose comme 4 appels système (écriture dans un tube et lecture du tube pour l'appel et même chose en sens inverse pour la réponse) et deux changements de contexte (processus client vers processus serveur et revenir ensuite vers le processus client). Ceci prend quelques microsecondes. Un appel de procédure à distance entre deux ordinateurs prend ces mêmes appels système, peut ou non demander des changement de contexte mais ajoute la latence des envois de paquets réseau. Selon le réseau utilisé, cela peut facilement prendre près d'une miliseconde.

b) Votre premier travail pratique mettait en oeuvre un client, un serveur et le registre RMI. Expliquez l'interaction entre les différents acteurs (client, serveur et registre RMI) à partir du tout début de l'exécution. Décrivez toutes les communications qui ont lieu entre ces acteurs. Expliquez les ajouts à la table des objets importés et exportés du client et du serveur. **(2 points)**

Le serveur RMI charge les classes Stub et Skel requises disponibles pour tous les intervenants à partir d'un petit serveur http. Ensuite, le serveur enregistre l'objet à exporter dans le serveur rmiregistry. Le client interroge le rmiregistry avec le nom de l'objet qu'il recherche et obtient la référence à cet objet. La classe Stub correspondant au type de cet objet est obtenue du petit serveur http. Le client peut alors communiquer directement avec le serveur. Lors d'un appel, les arguments sont sérialisés et envoyés au serveur avec l'identité de l'objet réseau et de la méthode visés, ainsi qu'un numéro de requête. Le serveur donne cette information au Skel correspondant à l'objet et la méthode visés qui désérialise les arguments, effectue l'appel sur l'objet réel, sérialise les valeurs de retour et réenvoie le résultat avec le numéro de requête correspondant.

Lorsque le serveur enregistre son objet avec le rmiregistry, cet objet est ajouté à la table des objets exportés du serveur avec le rmiregistry comme utilisateur. Lorsque le client reçoit la référence réseau à cet objet en provenance du rmiregistry, il se fait ajouter à la liste des utilisateurs pour cet objet dans la table des objets exportés du serveur. De plus, il crée un proxy pour cet objet qui est inséré dans la table des objets importés du client.

c) En CORBA, il est possible de déclarer une fonction distante *oneway*. Quelles sont les contraintes pour qu'une fonction puisse être *oneway*? Quelles sont les conséquences en fiabilité et performance de cet attribut? **(1 point)**

Cet attribut ne peut s'appliquer qu'à des fonctions qui n'ont aucune valeur de retour et permet d'avoir des appels asynchrones. Le paquet pour l'appel de fonction est envoyé et ceci ne bloque pas car il n'y a pas de réponse ou accusé de réception à attendre. L'avantage est que c'est plus rapide étant non bloquant. Par contre, l'appel n'est pas confirmé et pourrait être perdu sans que le client n'en soit averti.

Question 3 (5 points)

a) Un serveur NFS 3 avec un CPU et un disque 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 dans le client dans 9 cas sur 10. Parmi les blocs en cache, 50% ont été validés depuis moins de 3 secondes. Dans la moitié des cas, lors d'une demande de validation, le bloc est modifié et il faut le relire. L'écriture d'un bloc sur le serveur prend 10ms de disque et 1ms de CPU, la lecture d'un bloc prend 1ms CPU dans 80% des cas, et 1ms CPU plus 10ms de disque dans 20% des cas. Une validation prend 1ms CPU. Quel est le nombre de clients que peut soutenir le serveur s'il traite chaque requête séquentiellement avec un seul fil d'exécution? Que devient ce nombre si le serveur utilise plusieurs fils d'exécution pour traiter simultanément plusieurs requêtes (le CPU et le disque peuvent alors fonctionner en parallèle)? **(2 points)**

Pour chaque client, on a à chaque seconde 1 écriture et 10 lectures. De ces lectures, 1 va directement au serveur et 9 sont en cache dont 50% sont déjà validées et ne vont pas au serveur; l'autre 50%, 4.5 lectures, demandent une validation et la moitié, 2.25 lectures, demandent en plus une lecture sur le serveur. Le serveur voit donc 1 écriture, 1 + 2.25 = 3.25 lectures, et 4.5 validations. Le coût en CPU de ces accès est de 1 x 1ms + 3.25 x 1ms + 4.5 x 1ms = 8.75ms. Le coût en disque est de 1 x 10ms + 3.25 x .2 x 10ms = 16.5ms. Avec un traitement séquentiel sur un seul fil, une requête prend 8.75ms + 16.5ms = 25.25ms / client-seconde, soit 39.6 (39) clients pouvant être servis.

Avec plusieurs fils, les accès disque pourront se faire en parallèle avec le traitement d'autres requêtes sur le CPU. Le CPU peut servir 1000ms/8.75ms / client = 114.28 (114) clients alors que le disque peut en servir 1000ms/16.5ms / client = 60.6 (60) clients et constitue le facteur limitant. Il est donc possible de servir 60 clients.

b) Un programme parallèle s'exécute sur 1024 noeuds. Chaque noeud possède un numéro d'ordre, de 0 à 1023. Le noeud 0 est le coordonnateur et s'occupe d'ouvrir le fichier de résultats dans lequel chaque noeud écrira 10GiO de résultats à la position qui lui est réservée. Le noeud n écrit 10GiO de la position n x 10GiO à la position (n + 1) x 10GiO - 1. Un système de fichiers Lustre est utilisé avec un serveur de métadonnées et 8 serveurs d'objets de stockage. Chaque client est connecté via un réseau 1Gbit/s à un commutateur et chaque serveur est connecté via un réseau 10Gbit/s. Si chaque disque permet d'écrire au rythme de 100MiO/s, combien de disques devrait avoir chaque serveur d'objets de stockage pour une meilleure performance et combien de temps prendra l'écriture de tous ces résultats? **(2 points)**

En supposant que la charge est bien répartie sur les disques, il ne sert à rien d'avoir plus de disques parallèles que ce que l'interface réseau de chaque serveur d'objets de stockage peut recevoir, soit 10Gbit/s / (100 x 1048576 octets / s x 8 bits / octet) = 11.92, soit 12. Les clients génèrent 1024 x 10GiO et les 8 serveurs permettent de recevoir 8 x 10Gbit/s, ce qui prend (1024 x 10 x 2^{30} octets x 8 bits / octet) / (8 x 10Gbit/s) = 1099.51 secondes.

c) Il existe plusieurs systèmes de fichiers poste-à-poste. Expliquez la différence de principe de fonctionnement entre Napster et Gnutella et donnez les avantages et inconvénients de

chacun. **(1 point)**

Napster est plus simple, il fonctionne sur la base d'un serveur central qui sait où se trouvent tous les fichiers disponibles. Tout est directement disponible. Gnutella est basé sur un réseau de postes qui échangent de l'information sur les fichiers qu'ils connaissent. Les recherches peuvent demander de passer par plusieurs noeuds et sont alors plus longues. L'avantage de Gnutella est qu'il n'y a pas de serveur central facile à espionner, filtrer ou démanteler, par exemple pour contourner la censure sous un régime dictatorial.

Question 4 (5 points)

a) L'Internet compte environ 1 500 000 000 utilisateurs qui font une requête DNS en moyenne à chaque 10 secondes. Les 13 serveurs de DNS au plus haut niveau sont capables de servir chacun un maximum de 500 000 requêtes par seconde. Quelle fraction minimale des requêtes doivent être servies à un niveau plus bas pour éviter de surcharger les serveurs du plus haut niveau? Il est arrivé dans le passé qu'une attaque réussisse à rendre indisponibles les 13 serveurs de plus haut niveau, quel est l'impact sur les utilisateurs si ceci dure quelques heures? Quelques jours? **(2 points)**

En supposant que les requêtes sont bien réparties, les serveurs de premier niveau peuvent servir ensemble 13 x 500000 = 6500000 requêtes / seconde. Les utilisateurs génèrent 1500000000 utilisateurs x 1/10 requête / utilisateur / seconde = 150000000 requêtes / seconde. Le ratio est donc de 6500000 / 150000000 = 4.33% au maximum qui peuvent être servies au plus haut niveau, donc au moins 95.6% qui doivent être servies à un niveau plus bas. Les durées de validité des informations sont souvent de 24 heures avec les DNS. Une panne de quelques heures ne nuit donc que pour quelques adresses plus rares qui ne sont pas en cache. Une panne de plus de 24 heures fera que les caches ne contiendront plus d'entrées considérées valides et l'impact deviendra très important.

b) Comment doit-on restreindre les droits d'accès (lecture ou écriture) sur un serveur LDAP comme celui utilisé à l'Ecole Polytechnique pour stocker l'information sur les utilisateurs (nom, adresse courriel, nom d'usager, mot de passe, matricule)? Le protocole DNS dans sa version originale ne contient aucun mécanisme de sécurité pour authentifier le serveur auprès du client. Est-ce que cela peut être un problème? Pour un client, obtenir la bonne association nom versus adresse IP est-il important au niveau de la sécurité? **(2 points)**

L'usager peut changer ses informations secondaires (e.g. numéro de téléphone) ou son mot de passe mais pas son nom, matricule ou autres paramètres. Un usager peut consulter certaines informations sur les autres usagers (nom, numéro de téléphone...) si ceux-ci accceptent de rendre cette information disponible. L'administrateur peut changer toutes les valeurs mais ne peut normalement pas lire les mots de passe puisque ceux-ci sont déjà irrémédiablement encryptés. Il ne peut que les changer. Il est important de pouvoir se fier sur les adresses IP obtenues, car autrement nous pourrions nous retrouver sur un site frauduleux qui nous demande nos coordonnées bancaires plutôt que sur le site de notre banque.

c) Quelles sont les principales différences entre les services de noms DNS et LDAP? Quels sont les usages typiques de chacun? **(1 point)**

Le DNS a été prévu pour contenir des associations simples entre des noms et des adresses numériques. Ceci a été étendu pour aussi lister les serveurs de courriel pour pour chaque ordinateur et même permettre les requêtes inverses (nom associé à une adresse). DNS est strictement utilisé pour les serveurs de nom sur l'Internet. LDAP est beaucoup plus général, il permet de stocker de multiples attributs dans une structure arborescente. En outre, LDAP permet des requêtes asynchrones, et des recherche basées sur différents critères à un niveau ou à plusieurs niveaux dans l'arborescence. LDAP est donc utilisé pour les répertoires d'entreprise plutôt que simplement comme serveur de nom pour les adresses ip. Ces répertoires d'entreprises contiennent toutes les informations sur les usagers mais aussi possiblement sur les ordinateurs, services de fichiers, équipements réseau...

Le professeur: Michel Dagenais