POLYTECHNIQUE MONTRÉAL

Département de génie informatique et génie logiciel

Cours INF8480: Systèmes répartis et infonuagique (Hiver 2020)
3 crédits (3-1.5-4.5)

CORRIGÉ DU CONTRÔLE PÉRIODIQUE

DATE: Vendredi le 28 février 2020

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 client envoie une requête de 4000 octets à un serveur et reçoit une réponse de 10 000 octets. Le client prend 200us pour préparer sa requête, le serveur 800us pour y répondre, et le client 100us pour traiter la réponse. La transmission de la requête et de la réponse se font sur un réseau qui a une latence de 100us et un débit de 10 000 000 octets / seconde. Combien de temps au total prend une requête? Combien de requêtes par seconde le client peut-il donc faire à ce serveur avec un seul thread d'exécution? Le client et le serveur ne font rien d'autre et ne sont donc pas interrompus par d'autres tâches. **(2 points)**

Le débit de 10 000 000 o/s représente 10 o/us. L'envoi sur le réseau prend 100 us + 4000 o / 10 o/us = 500 us. La réponse prendra 100 us + 10000 o / 10 o/us = 1100 us. Le total avec le traitement est donc 200 us + 500 us + 800 us + 1100 us + 100 us = 2700 us. On peut donc servir 1 000 000 us/s / 2700 us/r = 370 requêtes par seconde.

b) Une machine virtuelle tourne sur un noeud physique A. L'image de cette machine virtuelle contient 8 000 000 pages. On veut migrer cette machine virtuelle vers un noeud B à travers un lien réseau qui permet d'envoyer 25 000 pages par seconde. Au premier tour, on copie l'ensemble des pages, aux tours subséquents, on copie les pages modifiées depuis le tour précédent. Pendant son exécution, la machine virtuelle modifie 1000 pages par seconde. La migration se fait d'abord en copiant les pages sans arrêter l'exécution, puis, lorsqu'il reste peu de pages modifiées, l'exécution est arrêtée le temps de copier les pages restantes. Cette phase d'arrêt ne doit pas durer plus de 0.25 seconde. Combien de temps durera la migration au total? Combien de temps est-ce que l'exécution sera arrêtée? **(2 points)**

Le critère d'arrêt est de ne requérir que .25s pour copier les dernières pages, soit à 25 000 pages/s: .25s x 25 000 pages/s = 6250 pages. Au premier tour, il faut 8 000 000 pages / 25 000 pages/s = 320s. Pendant ce temps, 320s x 1000 p/s = 320 000 pages seront modifiées. Au second tour, il faut 320 000 pages / 25 000 pages/s = 12.8s pendant lesquelles 12 800 pages seront modifiées. Au troisième tour, il faut 12 800 pages / 25 000 pages/s = .512s pendant lesquelles 512 pages seront modifiées. On peut alors arrêter l'exécution, transférer ces pages en 512 pages / 25 000 pages / s = .02s. L'arrêt n'est donc que de 0.02s mais la migration totale a duré 320s + 12.8s + .512s + .02s = 333.3s.

c) Un serveur prend 100us pour traiter chaque requête. Le serveur utilise un seul thread, sur un coeur dédié, pour traiter les requêtes, et les requêtes sont mises en queue d'attente lorsque le serveur est déjà occupé avec une requête. Si le serveur reçoit en moyenne 9500 requêtes par seconde, quel est le nombre moyen de requêtes en queue sur le serveur? **(1 point)**

Le serveur peut traiter 1 000 000 us/s / 100 us/r = 10000 r/s. Le taux d'utilisation du service est donc de 9500 / 10000 = 0.95. On peut en déduire que le nombre moyen de requêtes en queue est de 0.95 / (1 - 0.95) = 19.

Question 2 (5 points)

a) Sur un réseau qui supporte la multi-diffusion, un message de groupe atomique est envoyé d'un membre aux autres. Le groupe compte N membres. Si sur ce réseau 1 message sur 4 envoyé par chaque noeud est perdu (i.e. pour cet échange, le premier message envoyé par un noeud donné est perdu, le second, troisième et quatrième passent, le cinquième est perdu...), combien de messages envoyés au total seront requis pour compléter ce message atomique. **(2 points)**

L'envoyeur envoie un message par multi-diffusion et attend les confirmations. Le message étant perdu, il ne reçoit rien et renvoie son message. Les N-1 accusés de réception des autres membres du groupe sont perdus. L'envoyeur envoie un troisième message, les N-1 accusés de réception parviennent alors à l'envoyeur. Il envoie en conséquence son message de confirmation. Le total est donc de 1 + 1 + (N - 1) + 1 + (N - 1) + 1 = 2N + 2.

b) Un message doit être envoyé avec gRPC utilisant l'encodage protobuf. Ce message suit la structure Recette et a pour contenu `{{"eau",80}, {"levure",5}, {"sucre",5}, {"farine",250}, {"sel",2}, {"yogourt",15}}`. Quel est le nombre d'octets requis pour encoder le message suivant sur un ordinateur 32 bits? 64 bits? **(2 points)**

```
message Ingredient {
  string nom = 1;
  int32 quantite = 2;
}

message Recette { repeated Ingredient = 1; }
```

Les ingrédients prennent 1 octet pour le champ/type "nom", 1 octet pour la longueur du "nom" plus 1 octet par caractère dans le nom, 1 octet pour le champ/type "quantite" et 1 octet pour la quantité, sauf pour la "farine" qui prend 1 octet supplémentaire puisque 250 n'entre pas sur 7 bits. Le total pour les 6 ingrédients est donc de 6 champs x 4 octets + 1 octet (pour la continuation de 250, quantité de farine) = 25 octets, plus le nombre total d'octets dans les noms 3 + 6 + 5 + 6 + 3 + 7 = 30, pour un total de 25 + 30 = 55 octets. Pour la structure Recette qui comprend 6 fois le champ répété Ingredient, selon la version de protobuf, on pourrait avoir 1 octet pour le champ/type de chaque Ingredient dans Recette, ce qui ajoute 6 octets, pour un total de 55 + 6 = 61 octets. Le fait d'avoir un ordinateur 32 ou 64 bits ne change rien avec les varints.

c) Quelle est la différence entre la sémantique "peut-être" et la sémantique "au plus une fois" pour les appels de procédure à distance? **(1 point)**

Pour la sémantique peut-être, le message n'est envoyé qu'une seule fois sans demander d'accusé de réception. Il peut ou non se rendre. Rarement, le réseau pourrait même répéter le paquet. La sémantique au plus une fois renvoie la requête jusqu'à obtention d'une réponse. Pour ne pas que le service puisse voir plus d'une fois la même requête, lors d'une seconde réception de la même requête, on suppose que la réponse envoyée n'a pas été reçue et on renvoie la réponse qui avait été mémorisée, sans traiter la requête une seconde fois. Cette sémantique est en fait "exactement une fois", puisqu'on renvoie la requête jusqu'à obtention d'une réponse, sauf si le service est perpétuellement en panne.

Question 3 (5 points)

a) Trois serveurs CODA redondants sont pris pour servir des clients. Tous les fichiers lus ou écrits par les clients se retrouvent déjà dans leur cache CODA, et n'ont pas besoin d'être revalidés car le client se fie sur le service de notification des serveurs. Des étudiants sont en train de travailler sur un intéressant projet de programmation. Ils modifient et sauvent le code source, de 64KiO, et ensuite créent une nouvelle version de l'exécutable compilé, de 128KiO. Ceci se répète sur chaque client à chaque 5 minutes, le temps de trouver et corriger le prochain bogue. A chaque sauvegarde du fichier source, ou création du fichier exécutable par le compilateur, l'application ferme le fichier et son contenu est propagé en séquence, bloc par bloc, aux trois serveurs redondants. Le serveur peut lire ou écrire un bloc de 8KiO en 2ms de coeur de CPU et 8ms de disque. Combien de clients ces trois serveurs redondants peuvent-ils supporter, s'ils contiennent chacun 32 coeurs de CPU et 16 disques? **(2 points)**

Chaque client envoie 8 blocs de 8KiO pour le source et 16 blocs de 8KiO pour l'exécutable, à chaque 5 minutes, soit 24 blocs / (5m x 60s/m) = .08 bloc/s. Chaque serveur est occupé par 1 bloc pendant 2ms de CPU et 8 ms de disque. Il peut donc servir 32 x 1000 ms/s / 2 ms/r = 16000 requêtes de bloc par seconde avec son CPU et 16 x 1000 ms/s / 8 ms/b = 2000 b/s avec ses disques. Les disques seront donc le facteur limitant. Puisque chaque client demande .08 b/s, le serveur pourra satisfaire 2000 b/s / .08 b/s = 25000 clients. Puisque les 3 serveurs redondants font le même travail en même temps, cela n'ajoute pas de capacité par rapport au calcul fait pour un seul serveur.

b) Le nouvel épisode de Pollux à Polytechnique vient de sortir et est offert par BitTorrent. Une première copie, qui contient les 1000 blocs de 1MiO, est disponible sur un serveur de départ. Les 8000 étudiants de Polytechnique en veulent une copie le plus tôt possible dès sa sortie. Tous sont connectés sur un réseau très rapide qui n'est pas un facteur limitant. Le serveur de départ et les ordinateurs des 8000 étudiants permettent tous de recevoir et de transmettre simultanément 1MiO par seconde dans chaque direction. Quel sera le temps requis avant que tous en obtiennent une copie? **(2 points)**

Pour que les 8000 étudiants aient une copie complète, il faudra transférer 8000 x 1000 blocs = 8 000 000 blocs de 1MiO. Pour la première seconde, 1 seul transfert est possible (serveur de départ), pour la deuxième seconde 2, ensuite 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8000 (on ne peut pas avoir plus de 8000 ordinateurs étudiants qui reçoivent des blocs en même temps). A chaque seconde suivante on peut encore avoir 8000 transferts au maximum. Le total pour les 13 premières secondes est de 8191 blocs. Ensuite, on peut transférer 8000 blocs à chaque seconde au maximum. Pour transférer les 8 000 000 blocs il faudra donc au minimum 13s + (8000000 b - 8191 b) / 8000 b/s = 1011.97s, soit 1012 secondes. On suppose ici qu'il est toujours possible pour un ordinateur étudiant de trouver un bloc qu'il n'a pas encore, ce qui devrait souvent être le cas avec BitTorrent qui favorise la propagation des blocs plus rares. Cela reste néanmoins une borne inférieure.

Une stratégie de diffusion systématique et relativement efficace est de commencer par envoyer un bloc différent à chacun de 1000 ordinateurs étudiants, ce qui prend 1000s. Ensuite, chacun des 1000 ordinateurs envoie son bloc à un autre ordinateur qui n'a encore rien, en 1s. Ces 2000 ordinateurs envoient ensuite chacun leur bloc à 2000 autres ordinateurs qui n'ont encore rien en

1s. Finalment, ces 4000 ordinateurs envoient chacun leur bloc aux 4000 autres ordinateurs en 1s. Ainsi, après 1003 secondes, chacun des 8000 ordinateurs étudiants a un bloc, et chaque bloc du fichier original se retrouve sur 8 ordinateurs étudiants différents. Dans les 999 secondes suivantes, chaque ordinateur, dans un groupe de 1000 qui contiennent collectivement les 1000 blocs du fichier, peut envoyer son bloc aux 999 autres ordinateurs du groupe. Après ces échanges, chaque ordinateur aura une copie complète du fichier, ce qui aura pris un total de 1000s + 3s + 999s = 2002s. On voit facilement que cette solution est une borne supérieure. En effet, pendant les premières 1000s, seul le serveur original et un seul client sont actifs, alors que ceux qui ont déjà reçu un bloc pourraient transmettre des blocs.

Une simulation naïve de clients bittorrent, selon le niveau de sophistication des heuristiques choisis pour décider quel client demande à quel autre un bloc, donne un temps total de 1150s (chaque client à tour de rôle est le premier à demander un bloc) ou de 1090 (le client avec le moins de blocs reçus demande en premier). Ceci est cohérent avec les bornes trouvées et s'approche passablement de la borne inférieure calculée.

c) Dans le travail pratique 3, plusieurs fichiers ont été créés avec la commande suivante. Donnez le contenu possible de /home/disk1, sur les conteneurs gluster1 et gluster2, pour les deux cas suivants de configuration: i) avec réplication, ii) avec distribution. **(1 point)**

```
for (( i =1; i <= 20; i++ ) ) ; do touch $i; done
```

Avec la réplication, les fichiers 1 à 20 se retrouveront chacun à la fois sur gluster1 et gluster2. Avec la distribution, les fichiers 1 à 20 se répartiront entre gluster1 et gluster2 selon la fonction de hachage utilisée par GlusterFS. Par exemple, on pourrait avoir les fichiers pairs sur gluster1 et impairs sur gluster2.

Question 4 (5 points)

a) Dans une entreprise, chaque client effectue en moyenne 1 requête à un serveur donné à chaque 5 secondes. Le serveur peut répondre à 85% des requêtes en 1ms de CPU (information en mémoire), à 10% des requêtes en 2ms de CPU et 10ms de disque (information sur disque), et à 5% des requêtes en accédant d'autres serveurs en 5ms de CPU, 10ms de disque et 200ms d'attente après les autres serveurs. Ce service utilise plusieurs threads et le serveur possède 16 coeurs de CPU et 10 disques. Quel est le nombre maximal de clients que ce serveur peut desservir efficacement? Quel est le nombre minimal de threads qu'il devrait utiliser? **(2 points)**

Chaque client demande par requête .85 x 1ms + .1 x 2ms + .05 x 5ms = 1.3ms de CPU, .1 x 10ms + .05 x 10ms = 1.5ms de disque et .05 x 200 = 10ms d'attente. Les 16 coeurs peuvent servir: 16 coeurs x 1000 ms/s / 1.3 ms/r = 12307 r/s alors que les 10 disques peuvent servir 10 x 1000 ms/s / 1.5 ms/r = 6666 r/s. Les disques sont le facteur limitant et à 1 requête par 5 seconde par client, le service peut accomoder 5 s/r x 6666 r/s = 33333 clients. Chaque requête reste en moyenne 1.3ms + 1.5ms + 10ms = 12.8ms dans le système, alors qu'une nouvelle requête est servie à chaque 1 / 6666 r/s (ou 1.5ms / 10 disques) = 0.15ms. Le nombre de requêtes simultanées dans le système est donc de 12.8ms / 0.15ms = 85.3, il faut donc au moins 86 threads.

b) Un système client serveur utilise des objets Java qui communiquent avec Java RMI. Au moment d'une nouvelle requête, un objet Java RMI est créé sur le serveur et est accédé par le client. Lorsque la requête est terminée, le proxy sur le client est libéré et un message est envoyé au serveur pour l'informer que l'objet Java RMI correspondant n'est plus utilisé et peut être libéré. Le serveur reçoit environ 100 requêtes par seconde et chaque requête dure environ 5 minutes. Combien d'objets Java RMI pour ce service existent simultanément dans le serveur en moyenne? Si dans 1% des cas, le client termine de manière catastrophique pendant sa requête et le message du proxy libéré n'est pas envoyé au serveur, qu'arrivera-t-il? Est-ce que le serveur pourra continuer sans problème, ou au bout de combien de temps, et selon quels paramètres, deviendra-t-il en difficulté? **(2 points)**

Un serveur qui démarre reçoit une requête. Cette requête sera terminée dans 5m x 60s/m = 300s. Entretemps, il recevra donc 300s x 100r/s = 30000 requêtes qui seront présentes simultanément dans le serveur. Par la suite, à chaque nouvelle requête qui entre, une sort et le nombre de requêtes simultanément dans le serveur reste environ le même. Si 1% des requêtes ne se terminent pas correctement et l'objet RMI n'est pas relâché, cela veut dire que 1 objet RMI par seconde s'accumule. Selon l'espace mémoire pris par l'objet RMI et la taille de mémoire disponible dans le serveur, cela prendra plus ou moins de temps avant que la mémoire ne se remplisse et que le serveur ne devienne très lent et se termine en erreur, faute de mémoire disponible.

c) Une entreprise veut offrir un service en utilisant des serveurs virtuels obtenus d'un fournisseur comme AWS ou Azure. Sur ces serveurs virtuels, elle peut installer Linux et OpenStack afin de déployer et d'orchestrer des machines virtuelles. A la place, elle peut installer Linux et Kubernetes afin de déployer et d'orchestrer des conteneurs. Pour chacune de ces deux solutions, dites si c'est faisable et si c'est efficace. **(1 point)**

Les serveurs "virtuels" obtenus du fournisseur représentent un premier niveau de virtualisation. Il est possible d'installer Linux et OpenStack, et de mettre des machines virtuelles dessus, mais cela veut dire des machines virtuelles imbriquées qui ne pourront bénéficier du support matériel pour la virtualisation, puisque le support matériel ne s'applique qu'au premier niveau. Le surcoût sera assez grand et ce n'est donc pas très efficace. Déployer des conteneurs sur des serveurs virtuels est à la fois faisable et efficace, puisqu'on reste à un seul niveau de virtualisation, qui peut alors être supporté par matériel.

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 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 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 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 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 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