POLYTECHNIQUE MONTRÉAL

Département de génie informatique et génie logiciel

Cours INF8480: Systèmes répartis et infonuagique (Automne 2022)
3 crédits (3-1.5-4.5)

CORRIGÉ DU CONTRÔLE PÉRIODIQUE

DATE: Lundi le 31 octobre 2022

HEURE: 12h45 à 14h35

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) De nombreux clients font chacun 8 requêtes par seconde vers un serveur. Chaque requête demande au serveur 5ms sur un coeur de CPU et en plus, dans 40% des cas, la lecture d'un disque pendant 20ms. Chaque serveur possède 2 coeurs de CPU et 4 disques. Les requêtes sont bien réparties entre les coeurs de CPU et entre les disques. Le service utilise plusieurs fils d'exécution afin de servir en parallèle les requêtes. Quel est le nombre maximal de clients possible avant que le service ne sature? Combien de fils d'exécution devrait-on rouler sur le serveur? **(2 points)**

Puisque chaque requête prend 5ms sur un coeur de CPU, chaque coeur peut servir 1000ms/s / 5ms/r = 200 requêtes / seconde. Le total est donc de 400 requêtes / seconde pour les deux coeurs de CPU. Chaque requête prend en moyenne 0.4 x 20ms = 8ms de disque. Chaque disque peut servir 1000ms/s / 8ms/r = 125 requêtes / seconde. Le total est donc de 500 requêtes / seconde pour les 4 disques. Le facteur limitant est les coeurs de CPU qui peuvent supporter 400 requêtes / seconde soit 400r/s / 8r/client-s = 50 clients. Si chaque requête reste dans le système 5ms CPU + 8ms disque = 13ms en moyenne, cela demande 400r/s x 1s/1000ms x 13ms-thread/r = 5.2 thread, soit 6 fils d'exécution.

b) Un service s'exécute sur un serveur et reçoit des requêtes par des appels de procédures à distance (RPC). Les requêtes arrivent selon un processus de Poisson et sont mises en file d'attente lorsque le serveur est déjà occupé par une requête. Les requêtes arrivent au rythme moyen de 200 / seconde et le serveur peut traiter chaque requête en 3ms. Calculez N, le nombre moyen de requêtes dans le système, et R le temps de réponse moyen pour ce cas? Pendant une période de pointe, le taux d'arrivée des requêtes passe à 300 / seconde. Que deviennent N et R? **(2 points)**

La capacité de traitement u dans le premier cas est de 1000ms/s / 3ms/r = 333.33r/s, alors que le taux d'arrivée l est 200r/s (un intervalle moyen entre les requêtes de 5ms). Le taux d'utilisation est ainsi de l/u = 200r/s / 333.33r/s = 0.6. En conséquence, N = 0.6 / (1 - 0.6) = 1.5, et le délai total W = N/l = 1.5 / 200r/s = 0.0075s ou 7.5ms. On peut aussi calculer R, le temps relatif au temps de service 1/u, R = 1 / (1 - U) = Wu = 2.5, et W = R/u = 2.5/333.33r/s = .0075s ou 7.5ms. Attention, on peut répondre que le temps de réponse moyen R (relatif) est de 2.5, mais il ne faut pas dire que ce temps est de 2.5 secondes, ce qui est bien sûr incorrect. Il est de 2.5 (temps relatif) x 5ms (intervale entre les requêtes) = 7.5ms. Dans le second cas, u est le même, mais le taux d'arrivée l passe à 300r/s, ce qui donne un taux d'utilisation U = l/u de 300r/s / 333.33r/s = 0.9 et N devient 0.9 / (1 - 0.9) = 9. En conséquence, W = 9 / 300r/s = 0.03 ou 30ms. On peut aussi calculer R = Wu = 1 / (1 - 0.9) = 10 et w = R/u = 10 / 333.33r/s = 0.03 ou 30ms.

c) Le système Android est basé sur le système d'exploitation Linux. Sur Linux, les permissions sur les fichiers sont généralement exprimées sous la forme de permissions données à l'usager, au groupe ou aux autres pour: la lecture, l'écriture, et l'exécution. Toutefois, pour déterminer quelles permissions sont données à chaque application Android, une granularité plus fine est souvent requise. Par exemple, différentes informations, qui demandent des permissions différentes, pourraient se retrouver dans le même fichier. Une permission peut aussi être donnée temporairement, au moment où l'application est exécutée , après validation auprès de l'usager. Comment cela est-il réalisé sur Android pour donner ou non les permissions aux applications, avec une granularité plus fine que celle permise pour les fichiers par Linux? **(1 point)**

Android offre de nombreux services qui sont accessibles via des daemon. *Ces* daemon *sont des processus serveur qui roulent en arrière-plan et acceptent des messages inter-processus de requête. Ainsi, la validation des permissions se fait dans ces processus, après avoir validé l'identité du processus ayant envoyé la requête, et peut donc être beaucoup plus flexible et granulaire que ce qui est offert normalement pour les fichiers dans le système d'exploitation Linux.*

Question 2 (5 points)

a) Un groupe de 45 processus, sur autant de noeuds connectés sur le même réseau local, communiquent à l'aide de messages de groupe. Un processus du groupe doit envoyer un message de groupe de manière atomique aux autres processus du groupe. Comment cela peut-il être réalisé? Combien de messages seront envoyés sur le réseau par les différents processus, pour un message de groupe atomique, si la multi-diffusion est disponible? Si la multi-diffusion n'est pas disponible? **(2 points)**

Il faut procéder en deux phases. Lors de la première phase, on envoie le message en question aux autres membres du groupe en demandant un accusé de réception et en spécifiant d'attendre la confirmation avant de livrer le message à l'application. Si tous accusent réception du message, la confirmation est ensuite envoyée. Si la multi-diffusion est disponible, il faut l'envoi initial, 1 message en multi-diffusion, n-1 soit 44 accusés de réception venant des autres membres du groupe, et un message de confirmation en multi-diffusion, soit un total de 46 messages. Sans la multi-diffusion, il faut 44 messages pour le message initial, 44 pour les accusés de réception et 44 pour la confirmation, soit un total de 132 messages.

b) Un appel de procédure à distance doit contenir les arguments suivants: string client, string produit, int quantité, int prix. Si les valeurs sont: "Jean Tremblay", "Bicyclette pliante", "2", "80000", combien d'octets seront requis pour encoder cette information avec CORBA CDR (32 bits)? Avec gRPC protobuf? **(2 points)**

Avec CORBA CDR, nous avons 4 octets pour la longueur de client et 16 octets (multiple de 4) pour les 13 lettres de "Jean Tremblay", 4 octets pour la longueur de produit et 20 octets (multiple de 4) pour les 18 lettres de "Bicyclette pliante". Il faut aussi 4 octets pour la quantité et 4 octets pour le prix, soit un total de 4+16+4+20+4+4 = 52 octets. Pour le format protobuf, nous avons pour le client 1 octet type/champ, 1 octet longueur et 13 octets pour "Jean Tremblay", pour le produit 1 octet type/champ, 1 octet longueur et 18 octets pour "Bicyclette pliante", pour la quantité 1 octet type/champ et 1 octet pour 2, et pour le prix 1 octet type/champ et 3 octets pour 80000. Le total est de 1+1+13+1+1+18+1+1+1+3 = 41 octets.

c) Vous êtes l'expert en infonuagique et des collègues viennent vous consulter pour savoir s'ils devraient utiliser des machines virtuelles (e.g. avec KVM ou VirtualBox) ou des conteneurs (e.g. Kubernetes et Docker qui utilisent les cgroup sur Linux). Le premier collègue veut rouler sur le même système embarqué, qui roule Linux, de nouveaux services sur Linux, ainsi qu'un ancien service fourni par une ancienne application Windows. Le second collègue doit bâtir un serveur qui compile une application pour plusieurs versions différentes de distributions Linux (e.g. Ubuntu 20.04, Ubuntu 22.04, Fedora 35, Fedora 36). Pour chaque version de distribution, il

faut utiliser la bonne version de compilateur et les bonnes versions de librairies, mais la version du noyau du système d'exploitation Linux importe peu. Que suggérez-vous d'utiliser, machine virtuelle ou conteneur, dans chaque cas? Pourquoi? **(1 point)**

Pour le premier cas, pour executer une application Windows, il faudra prendre une machine virtuelle afin de pouvoir exécuter ce système d'exploitation différent. Il serait aussi possible d'envisager un émulateur d'API comme Wine, mais cette solution ne fait pas partie du choix proposé. Pour le second cas, des conteneurs feront très bien l'affaire. Chaque conteneur peut venir avec une distribution différente (librairies, compilateurs...) mais ils sont contraints à utiliser le noyau Linux de l'hôte, ce qui était spécifié comme n'étant pas un problème. Lorsque c'est possible, on préfère utiliser des conteneurs, plutôt que des machines virtuelles, car le surcoût est beaucoup moindre et le temps de démarrage est beaucoup plus court.

Question 3 (5 points)

a) Un processus serveur reçoit des requêtes de clients par le biais d'appels de méthode à distance. Le serveur reçoit 25 requêtes par seconde et chaque requête crée un nouvel objet réseau de type *session* qui sera utilisé pendant 350 secondes. On envisage deux stratégies possibles pour déterminer quand les objets réseau peuvent être libérés. Pour la première stratégie, une notification est envoyée par le client lorsque l'objet n'est plus utilisé. Cependant, on estime que pour 1% des requêtes, le message de notification ne parviendra pas au serveur et ainsi l'objet ne sera pas libéré et restera en mémoire dans le serveur. Pour cette raison, le serveur est redémarré au milieu de chaque nuit afin de repartir à 0 et que les objets ne s'accumulent pas d'un jour à l'autre. Pour la seconde stratégie, l'objet est créé pour une durée de *bail* de 500 secondes, durée qui peut être prolongée au besoin en demandant une extension de *bail* de 500 secondes à la fois. Quel est le nombre d'objets réseau de *session* qui se retrouvent simultanément en mémoire dans le serveur dans le pire cas pour la première stratégie? Pour la seconde? **(2 points)**

Avec la première stratégie, nous avons 1% des requêtes qui créeront un objet qui ne sera pas libéré avant la fin de la journée. Ceci crée une accumulation de 0.01 x 25r/s x 60 s/m x 60m/h x 24 h/j = 21600r/j, soit 21600 objets orphelins à la fin de la journée avant le redémarrage. En plus, il y a les objets actifs. Lorsqu'une première requête arrive, elle ne sortira qu'après 350s, le nombre de requêtes présentes simultanément (entrées avant que la première ne sorte) sera donc de 25r/s x 350s = 8750r, soit autant d'objets réseau. Le total avant de redémarrer est donc de 21600+8750 = 30350. Avec la seconde solution, nous aurons 25r/s x 500s = 12500r, soit 12500 objets. Il y a un peu plus d'objets (que ceux actifs dans la première stratégie) avec la seconde stratégie, car ils sont conservés pour 500s, alors qu'ils auraient pu être libérés au bout de 350s. Par contre, on sauve au niveau des messages de notification qui ne sont plus requis, et surtout en évitant les fuites causées par les notifications manquantes.

b) Le travail pratique 3 a mis en contexte un scénario dans lequel vous disposiez de plusieurs serveurs formant une grappe de calcul. Vous avez dû implémenter un gestionnaire permettant de soumettre des tâches à exécuter selon une file d'attente FIFO. Pour cela, vous avez utilisé le système d'appel de procédure à distance gRPC. Deux fichiers principaux devaient être modifiés lors de cet exercice: "Manager.cc" et "operation.proto". Expliquez le rôle que joue chacun de ces

fichiers et détaillez la démarche suivie pour définir les éléments nécessaires pour rendre votre implémentation fonctionnelle. **(2 points)**

Le fichier `Manager.cc` avait pour rôle de prendre les opérations, de les mettre dans une queue FIFO et de les distribuer aux différents serveurs.

Le fichier `Operation.proto` avait pour rôle de définir les messages et les services selon le langage de description d'interface (IDL) de gRPC. ProtoC utilise ces définitions pour générer les classes utilisées par `Manager.cc` (proxy) et `Server.cc` (squelette).

Dans le travail pratique, il n'y avait qu'une seule opération donnée en argument et un seul serveur. Tout d'abord, il a fallu définir les messages et les services selon l'implémentation du service dans `server.cc` et `manager.cc`. Ensuite, il a fallu passer l'argument le programme principal (main) à la méthode `ProcessOperation`. Finalement, il a fallu rajouter l'appel gRPC vers le serveur.

c) Pour les différents services infonuagiques comme Amazon EC2, on parle de stockage d'instance, de stockage de bloc (EBS), et de stockage d'objets (S3). Expliquez les différences entre ces trois types de stockage. **(1 point)**

Le stockage d'instance est propre à chaque activation de l'instance. Son contenu est perdu lorsque l'instance est arrêtée. Le stockage de bloc est comme un disque local. Son contenu persiste après l'arrêt de l'instance et il peut être accédé à nouveau par une nouvelle instance. Un stockage de bloc ne peut toutefois être attaché qu'à une seule instance à la fois, comme un disque local. Le stockage d'objets est comme une page Web qui supporte GET et PUT. On peut lire son contenu complet, ou remplacer son contenu complet, mais il n'est pas possible d'en remplacer seulement une partie, comme le permettrait l'API POSIX.

Question 4 (5 points)

a) Un réseau de clients est servi par 3 serveurs CODA répliqués. Chaque client ouvre en moyenne 4 fichiers par seconde et en ferme autant. Lors de l'ouverture, le fichier n'est pas présent localement dans 30% des cas et doit être lu à partir d'un serveur. Lors de la fermeture, le fichier a été modifié dans 15% des cas et doit alors être écrit sur chacun des 3 serveurs. Chaque requête de lecture prend 6ms de coeur de CPU sur un serveur et en plus, dans 35% des cas, une lecture d'un disque de 25ms. Chaque écriture prend sur chaque serveur 8ms de coeur de CPU et 30ms de temps d'un disque. Chaque serveur possède 4 coeurs de CPU et 6 disques. Si les requêtes sont bien réparties entre les serveurs, les coeurs de CPU et les disques, et le service utilise plusieurs fils d'exécution afin de servir en parallèle les requêtes, quel est le nombre maximal de clients possible avant que le service ne sature? **(2 points)**

Chaque client ouvre 4 fichiers par seconde, ce qui demande: 4 x 0.3 = 1.2 lecture d'un des serveur, soit 0.4 lecture par serveur. Chaque client ferme 4 fichiers par seconde, ce qui demande 4 x 0.15 = 0.6 écriture par serveur. Un client demande donc par serveur par seconde 0.4 lectures x 6ms + 0.6 écritures x 8ms = 7.2ms de coeur de CPU, et 0.4 x 0.35 x 25ms + 0.6 x 30ms = 21.5ms de disque. Les 4 coeurs de CPU peuvent soutenir 4 x 1000ms/s / 7.2ms/client-s = 555.55 clients. Les 6 disques peuvent soutenir 6 x 1000ms/s / 21.5ms/client-s = 279.06 clients. Le facteur limitant est les disques et le système peut servir jusqu'à 279 clients avant de saturer.

b) Un volume sur un service GlusterFS est configuré pour être Réparti, Redondant (Distributed Replicate), avec un degré 2 autant pour la distribution que la réplication, pour un total de 4 serveurs impliqués, nommés `S1` à `S4`. Si 10 fichiers, nommés `file0` à `file9`, sont placés sur ce volume et se répartissent aléatoirement, et relativement équitablement, entre les différentes locations possibles, donnez une répartition possible de ces fichiers sur les 4 serveurs, en indiquant où chaque serveur se situe dans l'organisation répartie redondante. Si la bande passante de chaque serveur pour l'écriture ou la lecture de fichiers est de 100MiO/s, combien de fichiers de 1MiO pourrait-on écrire sur ce volume par seconde? Combien de fichiers de 1MiO pourrait-on lire de ce volume par seconde? **(2 points)**

La répartition pourrait se faire entre deux groupes de serveurs, S1 et S2 d'un côté, et S3 et S4 de l'autre. Les fichiers se répartiront aléatoirement et relativement équitablement entre ces deux groupes. A l'intérieur de chaque groupe, la réplication demande que chaque fichier soit écrit sur les deux serveurs. Ainsi, S1 et S2 auront le même contenu (dupliqué) pour ce volume, et il en sera de même pour S3 et S4. Nous pourrions ainsi avoir, par exemple: S1 contient (file0, file2, file4, file6, file8), S2 (file0, file2, file4, file6, file8), S3 (file1, file3, file5, file7, file9), S4 (file1, file3, file5, file7, file9). Avec les 4 serveurs en parallèle, on pourrait écrire 400MiO/s soit 400 fichiers de 1MiO par seconde, mais en raison de la duplication, chaque fichier est écrit deux fois, on se retrouve donc en fait avec 200 fichiers / seconde. Pour la lecture, les 4 serveurs peuvent vraiment fonctionner en parallèle, puisqu'on ne lit chaque fichier que d'un seul des deux réplicat, et on peut donc lire 400 fichiers de 1MiO / seconde.

c) Dans les services de fichiers comme Lustre et Google File System, des serveurs redondants sont utilisés autant pour les métadonnées que pour les fichiers eux-mêmes. Les serveurs de métadonnées redondants sont-ils en configuration Active / Active ou Active / Passive? Les serveurs redondants pour les fichiers eux-mêmes? **(1 point)**

Sur la plupart des systèmes, incluant Lustre et Google File System, les serveurs de métadonnées sont en configuration Active / Passive. Ceci offre moins de performance mais il est plus facile d'assurer la cohérence, qui est très importante pour les métadonnées. Pour les serveurs de fichiers, une configuration Active / Active est utilisée, ce qui assure un maximum de performance pour cette partie qui peut facilement avoir à soutenir un débit de données très intense.

Le professeur: Michel Dagenais