Question 1
Un client envoie une requête de 200 octets et reçoit une réponse de 5000 octets. Quel est le temps total requis avec les hypothèses suivantes, pour UDP, TCP, local: latence de réseau 5ms, ouvrir une connexion TCP 5ms, débit 10Mbits/s, unité de transfert maximale 1000 octets, temps de traitement 2ms, réseau peu chargé? En local, la latence est de 2ms et le débit mémoire-mémoire de 400Mo/s.

> *UDP (datagram): 5ms + (200 * 8) / 10000bits/ms + 2ms + 5 (5 + (1000 * 8) / 10000bits/ms) = 36.16ms*
>
> *TCP (avec connexion): 5ms + 5ms + (200 * 8) / 10000bits/ms + 2ms + 5 (5 + (1000 * 8) / 10000bits/ms) = 41.16ms (suivi d'accusé de réception).*
>
> *Accès local: en supposant une latence de 2ms en local dans les deux directions et un transfert mémoire mémoire de 400 mégaoctets/s, 2ms + 200 / 400000 octets/ms + 2ms + 5000 / 400000 octets/ms = 4.013ms.*

Question 2
Les bleus sont divisés en deux camps de part et d'autre des verts et communiquent par messager. S'ils attaquent ensemble, la victoire est à eux. Chacun n'attaquera que s'il a l'assurance d'un accord mutuel confirmé. Quel protocole peuvent-ils utiliser pour s'assurer de manière absolue que l'autre accepte d'attaquer avec eux?

> *Ce n'est pas possible!*

Question 3
Donnez un exemple pour lequel l'ordre d'un message de groupe (multicast) de deux clients n'est pas important? Un ou c'est important?

> *L'ordre n'est pas important pour une requête où un seul ordinateur peut envoyer la réponse (ou lorsque la valeur demandée ne change pas et peut venir de n'importe quel ordinateur). Par contre, pour des mises à jour de serveurs répliqués, il est important que tous les serveurs voient les mises à jour dans le même ordre lorsqu'elles s'intersectent.*

Question 4
Comparez les coûts pour les messages de groupe FIFO, causalement ordonnancé, totalement ordonnancé) en termes de performance et de complexité d'implémentation.

> *L'ordonnancement FIFO requiert que chaque processus maintienne un vecteur de numéros de séquence, et que chaque message transmis contienne un identifiant. Ce dernier est propre à chaque processus.*
> *L'ordonnancement causal requiert également que chaque processus maintenne un vecteur de numéros de séquence, mais requiert aussi que ce vecteur soit transmis avec chaque message, ce qui rajoute du surcoût. De plus, à chaque réception de message, un calcul additionnel doit être effectué pour vérifier la relation de causalité.*
> *L'ordonnancement total requiert un séquenceur qui gère les identifiants puisque ceux-ci sont globaux au système, et non propre à chaque processus. De plus, un plus grand de paquets doivent être envoyés sur le réseau entre le séquenceur et les autres processus ce qui rajoute*

encore plus de surcoût.

Au final: FIFO < Causal < Total

Question 5
Quelle sémantique RPC peut être utilisée pour chacun des cas suivants (peut-être, au moins une fois, au plus une fois):

RAPPEL
- La sémantique “peut-être” désigne une requête sans retransmission:un client envoie une requête et attend une réponse. Si la requête ou la réponse sont perdues, le client ne retransmet pas la requête. L'opération a “peut-être” été exécutée.
- La sémantique “au moins une fois” désigne une requête avec retransmission mais sans filtrage de requêtes du côté du serveur. Le client envoie la requête, s'il n'a pas de réponse, il rentransmet la même requête, qui risque d'être exécutée plusieurs fois sur le serveur. Le client répète la retransmission jusqu'à obtention d'une première réponse, sinon il lève une exception.
- La sémantique “au plus une fois” désigne une requête avec retransmission et avec filtrage de requêtes du côté du serveur. Le client envoie la requête, s'il n'a pas de réponse, il retransmission la même requête. Le serveur garde un historique des requêtes qu'il a exécutées, et ignore une requête si elle se trouve dans cet historique.

a) Forcer un serveur distant à écrire sur disque ce qu'il a en mémoire (flush) sachant que celui-ci retarde ses écritures

Aucune garantie n'est requise, et l'opération est idempotente. Toutes les sémantiques sont acceptables.

b) Enlever une entrée dans une base de données de type <clé-valeur>

Au moins une fois

c) Ajouter une entrée dans une base de données de type <clé-valeur>

Au moins une fois

d) Payer une facture, sachant qu'une facture a un numéro d'identification unique

Au moins une fois

Question 6
Une requête prend 1ms pour être encodée et décodée, le serveur prend 3ms pour le traitement. Chaque envoi ou réception de paquet demande 0.5ms au système d'exploitation et le débit réseau est de 10Mbps. Chaque requête contient deux entiers longs (64 octets chacun) et une chaîne de caractère comme identificateur global de 384 octets. La réponse contient le même identificateur global et une unique réponse en entier long. Quel est le délai d'expiration que cette fonction RPC doit avoir?

Pour la requête:
Temps pour encoder: 1ms

Temps pour le système d'exploitation pour envoyer: 0.5ms
*Temps de transmission sur le réseau = (64 + 64 + 384) * 8 / 10,000,000 = 0.409ms*
Temps pour le système d'exploitation pour recevoir: 0.5ms
Temps pour décoder: 1ms

Temps de traitement: *3ms*

Pour la réponse:
Temps pour encoder: 1ms
Temps pour le système d'exploitation pour envoyer: 0.5ms
*Temps de transactions sur le réseau = (64 + 384) * 8 / 10,000,000 = 0.358ms*
Temps pour le système d'exploitation pour recevoir: 0.5ms
Temps pour décoder: 1ms

Le délai d'expiration doit être d'au moins 9.767ms.

Question 7
Napster étant surtout utilisé en amérique du nord, la charge sur le serveur d'indexation varie grandement selon le moment de la journée. Proposez une solution qui soit efficace en consommation d'énergie mais qui garantie une assez bonne disponibilité. Pourquoi Limewire n'a-t-il pas besoin de cette solution?

Un seul serveur d'indexation pourrait servir toutes les requêtes. Lorsque le nombre de transactions par secondes atteint un certain seuil, on augmente le nombre de serveurs pour répartir la charge. Lorsque le nombre de transactions par seconde tombe en dessous du seuil, on réduit le nombre de serveurs.
Dans un système pair-à-pair, il n'y a pas de serveur central, chacun des pairs contribue au système global. La charge de travail est transférée du serveur vers les pairs.

Question 8
Quelle information le module client NFS doit-il conserver pour les processus qui l'utilisent?

Pour chaque fichier ouvert, le client NFS doit conserver le vfs inode, la position courante dans le fichier, et l'identificateur NFS du fichier.
L'exemple donné en classe montrait un serveur de fichiers pour lequel les fonctions d'écriture et d'écriture demandaient une position absolue pour garantir une fonction idempotente et fournir une sémantique au moins une fois. Pour garantir une transparence d'accès chez le client, qui s'attend aux fonctions d'écriture et de lecture POSIX, le module NFS se charge de maintenir la position courante dans le fichier.

Question 9
Quelle est la différence entre NFS et AFS?

Une des différences est qu'avec AFS, l'ouverture d'un fichier requiert que celui-ci soit téléchargé au complet chez le client. Avec NFS, les écritures et lectures chez le client sont en RPC et doivent être envoyés vers le serveur (ne pas oublier les mécanismes de cache, des écritures peuvent être retardées chez le client, et des lectures peuvent se faire en local si

l'information est déjà en mémoire chez le client).

Question 10
Donnez un cas d'utilisation où AFS est plus efficace. Même question pour NFS

Dans un système où les accès à un fichier sont souvent exclusifs et où les écritures sont rares, AFS est intéressant.
Dans un système où les écritures sont plus fréquentes et sont faites par plusieurs clients en même temps, NFS est plus efficace.

Question 11
Pourquoi AFS peut-il supporter plus de clients que NFS?

Avec AFS, les messages de modifications évitent les très nombreuses demandes de revalidation de NFS. De plus, l'ouverture d'un fichier fait en même temps sa lecture du serveur, et la fermeture fait au besoin son écriture sur le serveur. Cela réduit le nombre total d'appels au serveur.

Question 12
Comment les caches peuvent-elles augmenter la disponibilité du service de noms?

L'idéal est d'obtenir la précision et la simplicité d'un système centralisé et la robustesse et performance d'un système réparti. Les caches permettent d'obtenir un excellent compromis. Les caches ne requièrent aucune configuration, elles ne font que mémoriser pour leur période de validité (TTL) les informations qu'elles voient passer; ce faisant elles peuvent servir une grande proportion des requêtes.

Question 13
Quel est le lien entre la mise à l'échelle d'un serveur réparti et la résolution de noms?

Une stratégie de répartition de la charge est de configurer le serveur de noms pour retourner l'adresse IP d'un serveur différent à chaque requête. Ainsi, si les clients C1, C2, C3 et C4 demandent une requête au serveur monserveur.polymtl.ca, le serveur de noms peut retourner une adresse IP différente à chaque requête, garantissant que chacun des serveurs prenne une part de la charge de travail.

Question 14
Soit deux transactions T et U, lesquelles de ces sérialisations sont valides avec un contrôle de la conucrrence optimiste? Lesquels de ces entralcements ne peuvent pas se produire avec un contrôle de la concurrence pessimiste avec verrouillage à deux phases?

T: x = read(i); write(j,44);
U: write(i,55); write(j,66);

a) T x=Read(i); U write(i,55); T write(j,44); U write(j,66);

T commence avant U: T à t1 et U à t2
T x=Read(i); dernière écriture en t0, OK
U write(i,55); dernière lecture en t1, dernière écriture en t0, OK
T write(j,44); dernière lecture en t0, dernière écriture en t0, OK
U write(j,66); dernière lecture en t0, dernière écirutre en t1, OK
L'équivalence sérielle est respectée

b) U write(i,55); U write(j,66); T x=Read(i); T write(j,44);

U commence avant T: U à t1 et T à t2
U write(i,55); dernière lecture en t0, dernière écriture en t0, OK
U write(j,66); dernière lecture en t0, dernière écriture en t0, OK
T x=Read(i); dernière écriture en t1, OK
T write(j,44); dernière lecture en t0, dernière écriture en t0, OK
L'équivalence sérielle est respectée

c) T x=Read(i); T write(j,44); U write(i,55); U write(j,66);

T commence avant U: T à t1 et U à t2
T x=Read(i); dernière écriture en t0, OK
T write(j,44); dernière lecture en t0, dernière écriture en t0, OK
U write(i,55); dernière lecture en t1, dernière écriture en t0, OK
U write(j,66); dernière lecture en t0, dernière écriture en t1, OK
L'équivalence sérielle est respectée

d) U write(i,55); T x=Read(i); T write(j,44); U write(j,66);

U commence avant T: U à t1 et T à t2
U write(i,55); dernière lecture en t0, dernière écriture en t0, OK
T x=Read(i); dernière écriture en t1, OK
T write(j,44); dernière lecture en t0, dernière écriture en t0, OK
U write(j,66); dernière lecture en t0, dernière écriture en t2, PAS OK
L'équivalence sérielle n'est pas respectée

Pour le contrôle pessimiste de la concurrence, les scénarios a et d ne pourraient pas se produire. En a, lorsque U tente de lire i, T est encore dans sa phase d'expansion et détient le verrou sur i. Même chose pour le scénario d, mais c'est U qui détient le verrou.

Question 15
Trois transactions, T, U et V, s'exécutent concurremment. Le séquencement des opérations est le suivant:

T: Début
U: Début
V: Début
T: Read(x)
U: Read(z)
T: Write(x,1)
U: Read(x)
V: Read(w)
U: Read(y)
U: Write(y,2)
V: Read(x)
V: Read(y)
V: Write(z,3)

Cet ordonnancement respecte-t-il l'équivalence sérielle? Utilisez la méthode par estampille de temps

T: Début
U: Début
V: Début
T: Read(x): dernière écriture en t0, OK
U: Read(z) dernière écriture en t0, OK
T: Write(x,1) dernie écriture en t0, dernière lecture en t1, OK
U: Read(x) dernière écriture en t1, dernière lecture en t1, OK
V: Read(w) dernière écriture en t0, OK
U: Read(y) dernière écriture en t0, OK
U: Write(y,2) dernière écriture en t0, dernière lecture en t2, OK
V: Read(x) dernière écriture en t1, OK
V: Read(y) dernière écriture en t2, OK
V: Write(z,3) dernière écriture en t0, dernière lecture en t0, OK
L'équivalence sérielle est respectée

Question 16
Donnez deux méthodes pour éviter d'avoir un interblocage.

1. Imposer un ordre dans lequel les verrous doivent être acquis, que tous les processus doivent respecter.
2. Tout ou rien. Un processus tente de prendre un verrou, s'il n'y arrive pas, au lieu de bloquer, il relâche les verrous qu'il détenait, puis recommence.

Question 17
Dans un système de calcul avec une queue de requêtes partagées par plusieurs ordinateurs, quel algorithme allez-vous utiliser pour assurer l'accès exclusif à la queue de requêtes sachant que le temps de calcul de chaque requête est très court et le nombre de noeuds est élevé?

Un algorithme avec verrou centralisé est plus efficace. Comme le nombre de noeuds est élevé, le pire case pour l'algorithme en anneau est très long. De plus, comme le temps de calcul est court, les processus tenteront d'accéder à la ressource partagée très fréquemment.

Question 18

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

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

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

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