Examen Client/Serveur, 2002/2003
DESS IIR option Réseau
Durée : 1 heure
Tout document autorisé

Vous devez toujours **argumenter** vos réponses. Une réponse non argumentée ne rapporte pas de points. Recopier le cours ou un livre ne rapporte pas non plus de points car généralement cela ne répond pas exactement à la question qui était posée.

On désire réaliser une architecture client/serveur pour la diffusion et le chargement de gros fichiers (dépassant le giga octet).

Pour toutes les questions suivantes vous donnez :

- une description simplifiée du protocole (ne prenez pas un protocole trop simpliste sinon les questions de cet énoncé n'auront aucun sens),
- la description simplifiée du fonctionnement du client et du serveur,
- une réflexion sur les ressources nécessaires (débit réseau, temps CPU, espace disque, ...)
- les avantages,
- les inconvénients,
- dans quelles situations il est intéressant de l'utiliser,
- dans quelles situations il n'est pas intéressant de l'utiliser,

1. Architecture centralisée : tous les fichiers sont sur la même machine les clients se connectent sur cette machine.
2. Architecture en arbre : les clients demandent à des caches qui demandent à d'autres caches ou à la racine.
3. Architecture en arbre avec réplication : il y a plusieurs racines à l'arbre qui doivent contenir les même fichiers (comment ?).
4. Architecture pair à pair avec index : les clients sont aussi des serveurs, ce sont eux qui stockent les fichiers. Une fois qu'un client a chargé un fichier il le diffuse. Le client démarre en connaissant au moins un autre client raccordé aux réseau. Pour chacun des fichiers le client possède une liste mise à jour dynamiquement (comment ?) des clients le diffusant.
5. Architecture pair à pair sans index : Idem question précédente mais les clients ne mémorisent pas ou se trouvent les fichiers et ne connaissent pas la liste totale des clients.
6. Architecture pair à pair hiérarchisée : Idem question précédente mais les liens qui se tissent entre les clients respectent la topologie du réseau physique.

Remarques

SFTP ne permet pas la reprise des transferts.

FTP offre bien plus que ce qui est nécesaire pour faire ce qui est demandé.

FTP n'implique pas que l'on puisse se connecter en anonyme.

Les problèmes de latence lors des transferts de fichier n'ont aucune importance car les données sont très volumineuses.

Le protocole de transfert de fichier n'a pas besoin de gérer le cryptage et la compression. C'est au client de le faire en lui demandant de lire et écrire des données cryptées/compresées.

Le cache ne met à jour ces données que si on les lui demande sinon il passerait son temps à faire des transferts inutiles. D'autre part il n'a certainement pas assez de mémoire pour tout cacher.

C'est pair à pair (et non paire).

Ce n'est pas par ce que tout le monde peut accéder à une machine ou à un réseau que les données ne peuvent pas être sécurisées, il suffit de ne faire transité que des données cryptées et de les stocker cryptées. Elle ne devront être décryptées que sur une machine sécurisée.

L'espace disque n'est pas un goulot d'étranglement. Cela peut être une limite si vous ne pouvez pas assez brancher de disque sur votre machine. sur un PC normal, on peut facilement mettre plus de 100To (100000 fichiers de 1Go).

Il ne faut pas confondre protocole et medium de communication ce sont deux choses différentes. L'initialisation d'un medium de communication n'est pas réellement l'initialisation d'un protocole. Le protocole est "initialisé" par le premier message envoyé.

Un protocole asynchrone permet d'éliminer le temps de latence du réseau. Il est utile dans le cas de nombreuses questions/réponses.

Le proxy commence à répondre au client *avant* qu'il ai le fichier complet. Sinon la latence serai égale au temps de transfert total du fichier... Quand on utilise un proxy la latence est toujours plus grande, par contre le temps pour faire l'opération est en moyenne plus court.

Le proxy ne met à jours ces fichiers qui si un client les demande. S'il le fait à chaque modification du fichier original alors ce n'est pas un proxy mais de la réplication.

Quand on fait de la réplication, les données répliquées sont identiques partout.

Correction

Architecture centralisée

Protocole : TCP, authentification, demande de lecture, demande d'écriture. On peut bien sur reprendre le protocole FTP ou HTTP.

Fonctionnement : Le client se connecte au serveur, il donne un nom, un mot de passe. Pour une lecture il envoit une commande de lecture avec le nom du fichier à lire, le serveur retourne le nombre d'octets puis le fichier. Pour une écriture il envoit une demande d'écriture avec le nom du fichier à écrire et la taille, puis il envoit le fichier.

Ressources : Le goulet d'étranglement est le débit réseau du serveur. Le débit réseau est largement inférieur au débit des disques. La quantité de RAM du serveur peut être très faible car le cache ne sert à rien car les données sont très grosses et donc la perte de temps pour l'accès à la donnée est négligeable par rapport à l'accès total à la donnée.

Avantages : Simple à gérer et à mettre en place, coûts minimums.

Inconvénients : Limitation par le débit réseau du serveur. Plus aucun accès si le serveur plante. Le protocole ne prévoit pas les reprises.

Il faut l'utiliser quand les inconvénients ne sont pas gênants : peu de chargements, fiabilité d'accès non vitale, réseau des clients lent.

Il ne faut pas l'utiliser dans le cas inverse.

Architecture en arbre

Protocole : le même + la demande de date de modification sur un fichier + la demande pour savoir si on a le droit de le lire.

Fonctionnement : Le client ou proxy se connecte à un autre proxy. Le proxy fait suivre les demandes d'authentification. Pour les lectures, il vérifie d'abord le droit d'accès et la date de modification avant d'envoyer la copie locale. Pour l'écriture, il redirige bêtement en gardant une copie locale.

Ressources : Moins de débit réseau pour la racine, mais il devient important d'avoir de la mémoire pour cacher les informations sur tous les fichiers car il va y avoir de nombreuses demandes pour ces petites informations. Il faut des proxy avec de grandes capacités disque pour diminuer le traffic réseau.

Avantages : On supporte beaucoup plus de clients. On peut continuer à diffuser des fichiers quand la racine (ou un proxy père) est inateignable.

Inconvénients : Il faut configurer les client/proxy pour utiliser des proxy. Il faut configurer des proxy.

Il faut l'utiliser dans tous les cas si c'est possible.

Il ne faut pas l'utiliser si l'on n'a pas les ressources humaines et matériels pour les proxy.

Architecture en arbre avec réplication

Protocole : le même.

Fonctionnement : Lorsqu'un serveur racine reçoit un fichier il le stocke sur les autres serveurs racines immédiatement ou en différé si le serveur est innacessible.

Ressources : Les serveurs racines doivent contenir tous les fichiers, il doivent donc avoir beaucoup de disques. Un réseau rapide entre les serveurs racines.

Avantages : Plus de pannes possibles.

Inconvénients : Gros traffic entre les serveurs racines si les fichiers sont souvent modifiés. Possibilité d'incohérences, donc programmes compliqués.

Il faut l'utiliser quand on a les ressources nécessaires.

Il ne faut pas l'utiliser si les fichiers sont souvent modifiés. (Dans ce cas on reste avec une architecture en arbre mais ou les proxy de niveau supérieur on suffisemment de disque pour tout cacher).

Pair à Pair avec index

Protocole : Une seule action possible : récupérer un fichier. Une session Multicast diffuse les *modifications* à faire au fichier index.

Fonctionnement : A la connexion, on récupère l'index, après on le tient à jour en écoutant la session Multicast. Quand on récupère un fichier, on diffuse le fait que l'on diffuse ce fichier. La récupération du fichier ce fait à partir de la machine la plus proche en distane réseau. On demande à enlever de l'index les fichiers que l'on ne diffuse plus.

Ressources : de l'espace disque pour garder les fichiers que l'on a récupéré et une session multicast.

Avantages : non-centralisé, pas de gestion, pas de ressources supplémentaires. Les données souvent chargée sont les plus fiable et les plus facilement accessible. Les données restent là ou elles sont utilisées et ne vont pas ailleurs.

Inconvénients : L'index n'est pas fiable mais tend constamment à ce rapprocher de la réalité. Les fichiers jamais chargés peuvent disparaître du réseau. On utilise du temps CPU quand on ne fait rien...

Il faut l'utiliser si l'on a pas de ressources. Quand les clients restent toujours connectés (pas de transfert d'index).

Il ne faut pas l'utiliser pour un accès modem si le fichier index dépasse le méga octet. (par exemple plus de 10000 fichiers *différent* diffusés et 1000000 de clients).

Il ne faut pas l'utiliser si l'on ne veut pas être censuré car on peut facilement filtrer la session multicast.

Pair à Pair sans index

Protocole : Actions possibles : récupérer un fichier ou chercher un fichier.

Fonctionnement : Avant de récupérer un fichier on demande à ses voisins ou il se trouve. Si on nous pose la question, on recherche dans les fichiers locaux et on fait suivre la requête si elle n'est pas trop vieille. On fait évoluer sa liste des voisins en fonction du temps.

Ressource : Ce que l'on gagne en ne gérant pas l'index on le perd en faisant des recherches.

Avantages : Complètement non-centralisé

Inconvénients : Très nombreuses demandes de recherche. La recherche ne parcour pas *toutes* les machines, on peu donc rater un fichier existant si il est peu répandu.

Il faut l'utiliser quand on ne veut rien de centralisé.

Pair à Pair respectant le réseau

Fonctionnement : On fait suivre les recherches à ses voisins proches avec une grande probabilité et d'autant plus rarement qu'ils sont éloignés.

Avantages : beaucoup moins de débit sur les liaisons longues.

Inconvénients : on est pas certain de trouver le fichier.