Examen Systèmes Répartis 3A SLE

4 Février 2010

Durée : 3h.
Le barème est à titre indicatif.
Tous documents de cours autorisés (les livres de référence ne font pas partie du cours).
La clarté et la concision des réponses seront pris en compte dans la notation.

1 Fondamentaux *(8p)*

1.1 Gestion du Temps *(3p)*

Dans un système de 3 machines, on considère un ensemble d'échanges de messages (5 messages au total, dont 2 avec plusieur destinataires). La suite d'événements est la suivante :

- Machine 1 :
  e_1^1 : Envoi d'un message M_1 aux machines 2 et 3.
  e_1^2 : Envoi d'un message M_2 à la machine 2.
  e_1^3 : Réception du message M_3.
- Machine 2 :
  e_2^1 : Réception du message M_1.
  e_2^2 : Réception du message M_4.
  e_2^3 : Envoi d'un message M_5 à la machine 3.
  e_2^4 : Réception du message M_2.
  e_2^5 : Réception du message M_3.
- Machine 3 :
  e_3^1 : Événement local.
  e_3^2 : Envoi d'un message M_4 à la machine 2.
  e_3^3 : Réception du message M_5.
  e_3^4 : Réception du message M_1.
  e_3^5 : Envoi d'un message M_3 aux machines 1 et 2.

1. Reconstituez sur un schéma (selon le modèle vu en cours) ces événements et les échanges de messages entre machines. *(1p)*
2. On met en place un système d'horloges logiques (ou horloges de Lamport), chaque horloge étant initialisée à 0. Donnez, après chaque événement et sur chaque message, la valeur de l'horloge logique associée. *(1.5p)*
3. Plusieurs mécanismes de communication peuvent être implémentés à partir de ces horloges logiques ou de variantes. Ces systèmes nécessitent néanmoins un ordre total sur ces horloges (pour pouvoir trier les messages). La description des horloges vue en cours suffit-elle pour obtenir cet ordre total ? Sinon, proposez une extension simple pour pallier au problème. *(0.5p)*

1.2 Cohérence Mémoire *(3p)*

On dispose de 4 systèmes de mémoire distribuée. On observe sur ces systèmes les suites de lectures/écritures suivantes :

$W_1(x)a$		
	$W_2(x)b$	
	$R_3(x)b$	$R_3(x)a$
	$R_4(x)b$	$R_4(x)a$

(1)

$W_1(x)a$			$W_1(x)c$	
	$R_2(x)a$	$W_2(x)b$		
	$R_3(x)a$		$R_3(x)c$	$R_3(x)b$
	$R_4(x)a$		$R_4(x)b$	$R_4(x)c$

(2)

$W_1(x)a$		$W_1(y)c$	
$R_2(x)a$	$W_2(y)b$		
		$R_3(y)c$	$R_3(y)b$
		$R_4(y)b$	$R_4(y)c$

(3)

$W_1(x)a$					
	$R_2(x)a$	$W_2(x)b$	$W_2(x)c$		
			$R_3(x)b$	$R_3(x)a$	$R_3(x)c$
			$R_4(x)a$	$R_4(x)b$	$R_4(x)c$

(4)

1. Justifiez, pour chaque système, s'il respecte la cohérence séquentielle des opérations. De même pour la cohérence causale. *(1.5p)*
2. Identifiez un ensemble minimal d'opérations (une ou deux) pour que, si elles étaient supprimées, le système respecte chaque modèle de cohérence mémoire. *(1p)*
3. Rappelez le principe du Multicast Totalement Ordonné et expliquez son utilisation pour garantir dans un système la cohérence séquentielle des opérations. *(0.5p)*

1.3 Tolérance aux Pannes *(2p)*

1. Donnez la définition d'une panne Fail-Stop et d'une panne byzantine. *(1p)*
2. Quels sont les problèmes que posent ces pannes dans les systèmes distribués ? *(1p)*

2 Construction d'un Gestionnaire de Versions *(12p)*

Vous êtes engagés dans une entreprise afin d'étudier et de proposer des améliorations au système de gestion des codes sources.

2.1 Principes de base d'un VCS

Texte adapté de wikipedia : `http://fr.wikipedia.org/wiki/Gestion_de_versions`

Gestion de versions : la gestion de versions, en anglais *version control* ou *revision control*, est une activité qui consiste à maintenir l'ensemble des versions ou révisions d'un logiciel ou autre document. Essentiellement utilisée dans le domaine de la création de logiciels, elle est surtout concernée par le code source ; mais elle peut être utilisée pour tout type de document informatique.

Modifications du code source : lorsque plusieurs développeurs travaillent sur le même code source, il est nécessaire qu'ils se mettent d'accord sur un façon de transmettre et de discuter des modifications effectuées par chacun d'entre eux. Historiquement, les outils basiques (`diff` et `patch`) existant sous Unix ont servi pour ce travail et restent, de nos jours les plus utilisés. On parle donc d'un **diff** ou d'un **patch** pour un fichier rapportant l'ensemble des modifications effectuées sur un code source (potentiellement plusieurs fichiers). Bien évidemment, pour qu'un

développeur puisse comprendre le patch fourni par un de ses collègues, la version du code à partir de laquelle le patch a été créée est nécessaire.

Lorsque des développeurs envisagent de modifier en profondeur un code source il devient difficile de n'utiliser qu'un seul diff pour toutes les modifications. Dans ce cas, un **patchset** est créé : un groupe de patchs, chacun regroupant des modifications plus petites, mais formant tout de même un ensemble cohérent.

Un gestionnaire de versions permet de conserver toutes ces modifications et permet à un développeur de retrouver un état antérieur de son code source.

Révision : Chaque fois qu'un développeur finalise un patch, il l'enregistre dans le système comme une nouvelle version, une nouvelle **révision** du code. On appelle cette opération d'enregistrement un **commit**.

Dépôt et copies locales : Dans la pratique, le développeur travaille toujours sur sa propre version du code aussi appelée **copie locale**. Le gestionnaire de versions peut, par contre, limiter les modifications effectuées par certains développeurs : limiter les patchs à certains fichiers (en vérifiant au moment du commit), ou encore permettre à des développeurs de partager leurs copies.

On appelle **dépôt** les versions centralisées des gestionnaires de versions où tous les commits se font sur une même version du code. Il existe ainsi un point de synchronisation, où les développeurs peuvent trouver la dernière version du code mais aussi les modifications effectuées par les collègues.

Branches : Il est courant que des modifications complexes soient menées en parallèle par plusieurs développeurs. Dans ce cas, et pour éviter un entrecroisement des commits de plusieurs groupes, il est possible de créer des **branches**. Une branche désigne ainsi un ensemble de commits ayant le même objectif, partant d'une révision précise du code.

Plus généralement, les branches sont utilisées pour permettre :

- la maintenance d'anciennes versions du logiciel (sur les branches) tout en continuant le développement des futures versions (sur le **tronc**) ;
- le développement parallèle de plusieurs fonctionnalités volumineuses sans bloquer le travail quotidien sur les autres fonctionnalités.

On peut considérer une branche comme un dépôt différent du dépôt central, à la différence qu'il est possible de fusionner une branche avec le tronc. L'opération de fusion est d'ailleurs la partie la plus délicate de l'implémentation d'un gestionnaire de versions et souvent la raison pour laquelle de nouveaux gestionnaires de versions sont encore créés.

Conflits : Dans le cas d'un développement en équipe, particulièrement lorsque les développeurs sont répartis dans le monde entier, chacun travaille de façon indépendante . C'est tout l'intérêt du système de gestion de version que de rendre cela possible. Toutefois, des fusions régulières sont nécessaires à un avancement global du logiciel.

Il n'est pas rare que certaines modifications soient contradictoires (par exemple lorsque deux personnes ont apporté des modifications différentes à la même partie d'un fichier). On parle alors de **conflit** (de modifications) puisque le logiciel de gestion de versions n'est pas en mesure de savoir laquelle des deux modifications appliquer. Parfois ce n'est même ni l'une ni l'autre et il est nécessaire de reprendre une troisième fois le logiciel pour que ces modifications puissent finalement coexister harmonieusement.

2.2 Description du Système en Activité *(3p)*

L'entreprise dans laquelle vous avez été embauché n'utilise en réalité aucun système de gestion de versions : les développeurs ont juste mis en place un ensemble de règles de communication pour permettre leur collaboration.

Hiérarchie au sein d'un projet : Chaque projet logiciel au sein de l'entreprise est dirigé par un *coordinateur*. Ce dernier est responsable de la maintenance du dépôt du projet. Le dépôt dans ce contexte est défini comme un système de fichiers accessible en lecture seule à tous les membres du projet, et en écriture au coordinateur. Ainsi le coordinateur est chargé d'appliquer les patchs des autres membres du projet, de vérifier que les patchs ne violent pas les règles mises en place par l'entreprise (justification de la correction d'un bug, validation par une tierce personne des changements de fonctionnalités, ...) et ainsi de suite. Les autres développeurs du projet ont donc accès de manière aisée au code source dans sa version actuelle et soumettent les patchs au coordinateur.

Schéma de communication : Pour permettre la communication entre les développeurs, des mailing lists sont en place :

- **coordinateur.projet@entreprise.com :** le coordinateur envoie un mail contenant un patch sur cette liste chaque fois qu'il le valide. Dans cette opération, le coordinateur assigne aussi un numéro unique au patch et applique le patch sur le dépôt.
- **rfc.projet@entreprise.com :** chaque développeur peut par cette mailing list demander à ses collègues ce qu'ils pensent d'un de ces patchs.

Bien entendu, toute autre communication est possible entre les développeurs par le biais de mails professionnels. C'est notamment le cas de la demande de commit : chaque fois qu'un développeur souhaite faire appliquer un patch sur le dépôt, il envoie le patch et les justificatifs par mail au coordinateur.

L'entreprise étant jeune et dynamique, la totalité des développeurs sont administrateurs de leur machine : il peuvent y installer n'importe quelle application et faire les modifications qu'ils veulent.

Questions

1. En terme de gestionnaire de versions, à quelle opération correspond l'envoi par le coordinateur d'un mail sur coordinateur.projet@entreprise.com ? *(1p)*
2. Selon vous, comment un groupe de développeurs peut-il mettre en place une branche pour un projet ? *(1p)*
3. Identifiez les avantages et les inconvénients de ce mode de développement dans les situations suivantes : *(1p)*
   - pour un développeur,
   - pour un coordinateur,
   - lors de l'identification d'un bug dans le projet,
   - lors de conflits entre les patchs de plusieurs développeurs.

2.3 Passage des Emails à une Application Centralisée *(7p)*

Infrastructure de l'entreprise : La totalité des machines de l'entreprise est reliée par un réseau filaire haut débit fiable à 100%. Chacun des serveurs sur lesquels vous seriez amené à déployer une application fonctionne en Fail-Stop : quand il tombe en panne, tout le réseau de l'entreprise tombe aussi (autrement dit : soit tout marche soit rien ne marche).

Tout le personnel de l'entreprise est identifié de manière unique et chaque développeur est joignable par son identifiant. Le système d'identification est accessible sur tout le réseau de l'entreprise.

Construction d'une application à base de RPC : Au vu de vos commentaires et dans un souci de simplification du développement, on vous demande de réaliser une véritable application de gestion de versions. Vous optez pour un système de type client-serveur à base de RPC :

- Chaque coordinateur possède son propre serveur.
- Le coordinateur, une fois le serveur installé, dispose d'un client spécial pour administrer chaque projet (création, commit).
- Les développeurs possèdent un client pour publier des patchs et récupérer le code source d'un projet.

Questions Pour simplifier, on considère dans la suite qu'un dépôt ou une copie locale n'est jamais qu'un ensemble de fichiers disposés quelque part sur la machine d'un utilisateur du système. De même, les opérations de diff ou d'application d'un patch sont déjà disponibles dans le système et fonctionnent sur un ensemble de fichiers.

1. *(1p)* Rappelez le principe des RPC et justifiez en quoi l'infrastructure de l'entreprise est adaptée à ce genre de middleware.
2. *(1p)* Le coordinateur dispose d'une API de création d'un projet :
   (a) Donnez une description de cette API côté client.
   (b) Quelles opérations doivent être réalisées par le serveur lors de cette requête ?
3. *(1p)* Quelles opérations doit réaliser le serveur lorsqu'un coordinateur demande un commit ? Quels problèmes posent le fait de recevoir plusieurs patchs avec la même révision de référence ?
4. *(1.5p)* Les développeurs disposent d'opérations permettant de récupérer n'importe quelle révision du code (get), de réaliser un patch à partir des modifications apportées à une copie locale (diff) et d'envoyer le patch au coordinateur (push).
   - Donnez l'interface (API) de ces opérations.
   - Permettre la demande de n'importe quelle révision sur un projet demande une organisation particulière du serveur, pourquoi ?
5. *(1.5p)* Les développeurs souhaitent pouvoir créer des branches sur un projet et publier des
   modifications dessus sans l'accord du coordinateur (le push et le commit sont confondus). Donnez les modifications à réaliser sur les interfaces précédentes (pour les développeurs, pour le coordinateur).
6. *(1p)* L'entreprise trouve que ce schéma de développement est trop lourd pour le coordinateur. Proposez des changements au système précédent pour qu'il n'y ait plus qu'un seul serveur central de gestion des projets. En quoi ces changements modifient le rôle du coordinateur ?

2.4 Une Entreprise Orientée Open Source *(2p)*

Un changement de business model : L'entreprise décide de s'ouvrir au monde de l'Open Source. Ainsi le code de leurs applications devient public et des développeurs du monde entier souhaitent participer à son écriture. Pour répondre à toutes ces attentes, vous êtes donc chargé de transformer le système existant pour gérer les nouveaux utilisateurs. Bien entendu, pour permettre à n'importe qui d'accéder au code source des applications, le nouveau gestionnaire de versions doit être accessible par internet.

Questions

1. *(1p)* En quoi l'ouverture du gestionnaire de versions sur internet présente des difficultés ? En terme d'identification des utilisateurs ? En terme de tolérance aux pannes ?
2. *(1p)* Le système complètement centralisé ne plaît pas du tout à la communauté du logiciel libre. On vous demande de créer un gestionnaire de versions où il n'existe plus de dépôt central : chaque développeur dispose de son propre dépôt. Quels changements au niveau de l'interface sont nécessaires sur le gestionnaire actuel ?
3. (Bonus) La décentralisation du gestionnaire pose de gros problèmes pour identifier une révision de manière unique. Lesquels ? Quelles solutions pouvez-vous proposer pour les résoudre ?