Département de génie logiciel et des TI

Rapport de laboratoire

Étudiants	**Ménard, Pier-Luc** **Lépine, Guillaume**
Codes permanents	**MENP27019200** **LEPG14099201**
Cours	**LOG735**
Session	**Été 2015**
Groupe	**01**
Laboratoire	**02**
Chargé de laboratoire	**Thierry Blais Brossard**
Chargé de cours	**Lévis Thériault**
Date	**04/06/2015**

Introduction:

Afin de bien maitriser les concepts de base des systèmes distribués appris lors des séances de cours théoriques, nous devons mettre en place une application qui permet d'envoyer des messages synchronisés aux divers clients. Dans le cadre de ce laboratoire, il faut mettre en place une application qui permet à trois clients d'établir une connexion et d'envoyer des messages entre les abonnés via l'interface graphique.

La partie de l'envoi de message de type "unicast" a déjà été implémentée mais, le défi de ce laboratoire est de mettre en place l'envoi de message synchronisé de type "Broadcast" à tous les clients connectés. De plus, afin d'assurer que la chaîne de message respecte un ordre bien précis en tenant compte des latences réseaux possibles, il faut tout de même que le message final soit affiché par le client 1, 2 et finalement le 3. Le client 1 doit affiché "***Vous***", le client 2, "***Avez***" puis finalement le client 3 "***Réussi***" dans cet ordre précisément.

Bref, la complexité de ce laboratoire ne se retrouve pas au niveau du traitement des requêtes ou des calculs à faire du côté du serveur mais plutôt la mise en place d'un système qui masque l'hétérogénéité de ses composantes ainsi qu'une transparence au niveau des clients.

Question 1: De façon générale, expliquez les différents défis à relever par rapport à la synchronisation (de messages et/ou d'applications) et parlez brièvement des solutions possibles pour chacun des défis mentionnés.

Les défis rencontrées lors de la réalisation du laboratoire touchent la synchronisation d'événement avec un ordre connu. La difficulté vient du fait que les applications reçoivent les événements avec des délais variables, il faut donc adopté différentes stratégies pour s'assurer que les événements s'affichént dans un ordre bien précis. Un autre défi vient du fait que chaque application ignore l'état des autres applications qui compose le système global, elle ignore si le réseau connait des lenteurs où si une application est inactive. Finalement, nous devions détecter une application qui ne répond pas et s'assurer que le processus continue quand même.

Pour résoudre ces trois défis, les solutions suivantes ont été mises en place:

- Ajout de deux événements qui sont diffusés lorsqu'une application affiche son message à l'écran et lance un avertissement qui est capturé par le prochain noeud dans la liste des applications. Bref, lorsque l'application #1 affiche son message dans l'interface, il lance un message de confirmation sur le réseau (Broadcast) et seulement l'application suivante reçoit cette confirmation. Quant à elle, l'application #2 affiche son texte synchronisé à l'écran et envoie une confirmation qui sera capturée par le noeud suivant et ainsi de suite jusqu'à ce que tous les noeuds ont complétés leur tâche.
- Chaque application a été développée de façon passive de manière à ce qu'elle attend une confirmation du noeud précédent avant d'afficher son message. Ce

mode permet justement de s'assurer que les étapes sont faites dans un ordre précis. Bref, aucune décision hative est prise par le client, l'ordre d'affichage doit être respecter sans quoi, le principe de synchronisation serait brisé.

- Chaque application a un délai d'environ 15 secondes pour répondre. Si après 15 secondes un message de confirmation n'a pas été envoyé l'application suivante va afficher son message et assume que l'application précédente est indisponible.

Question 2: Expliquez l'approche que vous avez prise pour permettre l'envoi synchronisé des messages. Expliquez ensuite les modifications apportées à chaque classe (création de classes, modification des attributs et des méthodes):

Pour répondre à la problématique de ce laboratoire, plusieurs principes et objectifs ont été pris en compte:

1) Toutes les applications sont connectées et sont actives sur le réseau.
2) Le temps de latence est inconnu à l'avance et ne doit pas impacter l'envoi de messages synchronisés.
3) Les événements doivent être appliqués de façon séquentielle et dans un ordre connu à l'avance pour éviter de perdre le contrôle de l'état des applications.

Nous avons créé deux événements ***EventTriggerForPartTwo*** et ***EventTriggerForPartThree*** qui sont respectivement envoyé par l'application un et écouté par l'application deux et envoyé par l'application deux et écouté par l'application trois. Ces deux événements implémentent l'interface ***IAckEvent*** et héritent de la classe ***EventBase***. Une interface ***IAckEvent*** a été créé pour regrouper les classes de type ***EventTrigger***, cette interface est une extension de l'interface ***IEvent***. Ensuite, nous avons modifié la classe ***EventBusConnector***, une file d'événements a été ajoutée, lorsqu'un événement de type ***IEventSynchronized*** est reçu, il est ajouté dans la pile. Si l'application est la première dans la chaîne alors elle affiche le message et envoie un ***EventTrigger***. Nous avons modifiés la section qui écoute sur le socket, si un événement de type ***EventTrigger*** est reçu et que l'application est sensible à cet événement alors l'application va retirer de la file l'événement synchronisé, va l'afficher puis diffuser un ***EventTrigger*** pour la prochaine application de la chaîne. La dernière modification a été d'ajouter un paramètre de type date à l'événement ***EventThatShouldBeSynchronized***. Ce paramètre sert à savoir quand l'événement a été mis dans la file. Si un délai trop élevé a été dépassé l'application va l'afficher même si les événements ***EventTrigger*** précédent n'ont pas été recus.

Question 3: Discutez des améliorations possibles à effectuer à votre implémentation en fonction des notions d'extensibilité et de performance : que devriez-vous corriger pour que l'envoi de messages synchronisés fonctionne avec un nombre variable d'applications:

L'avantage de notre méthode vient du fait qu'elle est facilement extensible. Si on veut ajouter une nouvelle application il faut seulement ajouter un événement de type EventTrigger, modifier la dernière application de la chaîne pour qu'elle envoie le nouvel

événement, cette modification implique seulement d'appeler un constructeur différent, puis de s'assurer que la nouvelle application écoute pour le nouvel événement.

Question 4: Discutez d'au moins une solution alternative qui aurait pu répondre à la tâche demandée ; pensez à comment les responsabilités auraient pu être différemment assignées entre les applications et le bus d'événements:

Dans le cas de notre application, le délai de latence est inconnu à l'avance et peut varier dans le temps donc, il faudrait trouver une façon de réduire l'impact de celui-ci sur l'ensemble de l'application. Connaissant ce contexte, une alternative intéressante pourrait être mise en place si un client a une connexion réseau non fiable ou s'il est physiquement éloigné, de façon à réduire l'utilisation de la bande passante sur le réseau.

Une solution alternative serait d'ouvrir une connexion de type TCP entre deux noeuds successifs afin de confirmer l'affichage du message au noeud suivant au lieu de diffuser ce message sur l'ensemble du réseau. Dans le cas où le nombre de clients augmente, la bande passante nécessaire augmente elle aussi mais, de façon linéraire contrairement à une augmentation de manière exponentielle avec le modèle courant. La connexion resterait ouverte tant que le prochain noeud n'aurait pas exécuté son opération.

Conclusion

Ce laboratoire nous a permis de comprendre la problématique de l'envoi d'événements qui doivent être synchronisés ainsi que l'importance des délais de propagations des événements sur le réseau lorsqu'une synchronisation est nécessaire. Il faut premièrement, identifier les latences puis il faut s'assurer que tous les clients soient connectés simultanément pour garantir le fonctionnement. Différentes stratégies de synchronisation sont possible mais, dans ce laboratoire nous avons utilisé le principe de diffusion de message "Broadcast", le prochain noeud impliqué traite l'événement de façon indépendante. De manière à garantir l'affichage synchronisé, cette méthode permet d'assurer un ordre bien précis et assure que la latence ne cause pas de problème de fonctionnement de l'application globale.

Par contre, cette méthode n'est pas optimale, car elle consomme beaucoup de bande passante rapidement lors des diffusions mais, d'autre part, l'ajout ou le retrait des clients est simple à mettre en place et à configurer. Une autre méthode aurait été de créer des connexions TCP entre chaque noeud successif pour valider que la confirmation de réception a bien été faite et aussi, d'assurer que les événements ne soient pas capturés par un tier non impliqué. Bref, en tenant compte des contextes généraux et des besoins de l'application, nous serons en mesure de proposer une solution qui répond aux exigences et au contexte général. Dans notre cas, la notion de diffusions de messages est efficace puisqu'elle est simple à mettre en place et génère beaucoup de trafic certes, mais, permet de réduire le coût de traitement de connexion de type TCP par exemple.