Projet Totally ordered multicast

Hoël Boëdec, Mickaël Fournier

17 Janvier 2016

Contents

1 Spécifications du totally-ordered multicast

- Marche avec n peers.
- Les peers envoient aussi rapidement que possible (pas de temps d'attente).
- Les messages sont bien délivrés dans le même ordre, un vérificateur le confirme.
- Dynamisme du groupe + limitations –TODO–

2 Design du totally ordered multicast

2.1 Types de messages

- **Type 0 : broadcast data**
  Format : [timestamp (4) | id (4) | 0 | data (?)
  Contexte : trame envoyée par le burst thread d'un Peer. La longueur du message dépend du paramètre donné au lancement du peer (64, 256, 512 ou 1024 bytes).
- **Type 1 : ACK**
  Format : [timestamp (4) | id (4) | 1 | timestamp_ack (4) | port (4)]
  Contexte : trame envoyée pour confirmer la réception d'un message.

- **Type 2 : HELLO1**
  Format : [timestamp (4) | id (4) | 2 | id (4)]
  Contexte : trame envoyée pour dire à quelqu'un qu'un peer arrive.
- **Type 3 : HELLO2**
  Format : [timestamp (4) | id (4) | 3 | id (4)] *Contexte* : trame envoyée à tout le groupe pour dire qu'un nouveau peer veut rentrer dans le groupe.
- **Type 4 : EXISTING_MESSAGES** *Format* : [timestamp (4) | id (4) | 4 | data(?)] et data est une suite de [longueur (4), nb_ack (1), data_message (?)] *Contexte* : trame envoyée au nouvel arrivant pour lui faire parvenir les messages non délivrés du groupe.
- **Type 5 : EXISTING_PEERS**
  Format : [timestamp (4) | id (4) | 4 | data(?)] et data est une suite de [id (4)] *Contexte* : trame envoyée au nouvel arrivant pour lui faire parvenir les ids des autres peers du groupe.

2.2 Fonctionnement du vérificateur

Après avoir terminé les éxécutions en appuyant sur la touche "ENTREE" une fonction (lancée par le dernier arrivant seulement) s'occupe de lire les fichiers générés contenant les messages délivrés lors de l'éxécution ligne par ligne. Si il y a une différence alors un message s'affiche exprimant que le total ordonnancement des messages n'a pas correctement fonctionné.

2.3 File d'attente des messages à deliver

Chaque Peer tient à jour une file d'attente dans laquelle il stocke les messages qu'il a reçu (et envoyé). Cette file d'attente prend la forme d'un SortedSet < Message >. En effet les messages sont classés selon leur timestamp.
Ce sont également les messages eux-même qui comptent le nombre de acks reçus qui leur correspondent.

3 The overall class design of your implementation

- Main.java : Point d'entrée du programme. Instancie un Peer et son NioEngine. Instancie également un FileThread qui sera utile pour la vérification de l'ordre. En fin d'execution, lance la vérification de l'ordre des messages delivered.
- BroadcastThread.java : Lancé par chaque Peer une fois qu'ils sont connectés au groupe. Les Peer envoient alors des messages le plus rapidement possible vers l'ensemble des autres peers.
- FileThread.java : Écrit les messages delivered dans un fichier .txt. Ce fichier servira au vérificateur de l'ordre des messages delivered.
- Message.java : Élement de la file d'attente des messages à deliver. Un Message contient les bytes reçus par le peer (le message), le type de ce message, son timestamp et le nombre de ack's reçus pour ce message.
- MonitorMessagesToSend.java : Vérifie en continu si il y a des messages à envoyer.
- NioChannel.java : Dérive de Channel, permet de réaliser des opérations d'écriture/lecture sur un Channel.
- NioEngine.java : Surveille l'ensemble des channels et signale lorsqu'une opération de ACCEPT/CONNECT/WRITE/READ est possible.

- Peer.java : Représente un participant à la “discussion”. Il implémente AcceptCallBack.java, Connect-CallBack.java et DeliverCallBack.java. Un Peer tient à jour une HashMap contenant les connexions actives avec d'autres Peers. Un Peer possède également une file d'attente de messages à deliver et une file d'attente de messages à envoyer. Chaque Peer gère sa propre logical clock.

4 Comment lancer le projet et tester

Pour créer un peer : - Lancer la classe Main.java. - Renseigner le port d'écoute (par exemple ‘2005’) - Comme il s'agit du premier peer, entrer la même valeur de port distant que pour le port d'écoute. - Renseigner la taille des messages souhaitée (64, 256, 512 ou 1024 bytes).

Pour créer d'autres peers : - Lancer la classe Main.java. - Renseigner un port d'écoute différent des autres peers (par ex ‘2006’, ‘2007’, ...) - Renseigner le port d'écoute d'un des peers déjà présent dans le groupe (par ex ‘2005’). - Renseigner la taille des messages souhaitée (64, 256, 512 ou 1024 bytes).

Pour vérifier l'ordre des messages delivered :
Notre implémentation connaît la limite suivante : la vérification automatique ne peut se faire que si les ports utilisés se suivent dans l'ordre chronologique ... Ceci car nous écrivons dans des fichiers.txt pour la vérification et que cela facilite grandement le processus.
Une fois qu'au moins deux peers sont connectés ils commençent à s'envoyer des messages. Pour arrêter l'envoi de messages et vérifier l'ordre, il faut appuyer sur la touche ‘ENTREE’ dans la console de chaque peer. Le vérificateur écrira dans la console si l'ordre a bien été respecté.

Pour observer les statistiques : Après avoir vérifié l'ordre des messages delivered, vous pouvez ouvrir le fichier ‘Stats.txt’ propre à chaque peer qui se trouve à la racine du projet.

5 Problèmes connus

5.1 Vérificateur

Le vérificateur n'est pas très flexible. (choix des ports, ordre de terminaison des peers)

5.2 Statistiques

Nous n'affichons pas le délai de deliver d'un message. Il faudrait que l'emetteur du message observe le temps qui s'écoule entre le moment où il crée le message, et le moment où il le délivre.

5.3 Taille du groupe

On remarque qu'à partir d'une taille de groupe supérieur ou égale à 5 alors le nouvel arrivant est bloqué dans sa délivrance des messages. Cependant, les 4 autres personnes du groupe continuent leur système de délivrance.

<- que des localhost> <System.out.println(“Broadcast en cours. Appuyez sur ENTREE pour arreter et lancer la verification.”);>