Savaş Ali TOKMEN
Master 2 Génie Informatique
MSc Computer Engineering
UFR IMA, Grenoble, FRANCE

23 Octobre 2006

JGroups

Résumé

On appelle "multicast" le fait de communiquer avec un groupe de machines. TCP/IP propose un système de multicast très économe, et JGroups a été initialement crée pour pouvoir faire des multicasts en Java.

En outre, le système multicast de TCP/IP est non-fiable: les messages peuvent se perdre ou se réordonner. Par conséquent, JGroups a été rapidement étendu pour proposer des services additionnels: système en couches (donc un choix de protocoles; UDP, TCP et JMX ayant été implémentés), détection, enlèvement et notification de paires ayant plantés, envois fiables et ordonnés, fragmentation ou encore l'encryption.

Un exemple d'application (un répartiteur de requêtes) a été implémenté pour illustrer ce protocole.

Abstract

A multicast is a communication between a group of machines. TCP/IP has a very economic multicast support, and JGroups has been initially created in order to do multicasts in Java.

Still, TCP/IP multicasts are not reliable: messages can get lost or reordered. To solve this issue, JGroups been extended in order to provide extra services: layered system (therefore a choice of network protocols -to illustrate, UDP, TCP and JMX support has been implemented), crashed peer detection, removal and notification, reliable and ordered transmissions, fragmentation or encryption.

A simple application (a request distributor) has been implemented in order to illustrate this protocol.

Note: The rest of the article is in French.

Motivation initial

On appelle “multicast” le fait de communiquer simultanément avec un groupe de machines identifiés par une adresse spécifique (appelée l'adresse du groupe). TCP/IP propose un système multicast et JGroups a été initialement crée pour pouvoir automatiser le processus d'abonnement, réception et désabonnement d'un multicast IP. Elle supportait donc les fonctionnalités suivantes:

- Création et destruction de groupes multicast. Ces groupes peuvent être sur un réseau local ou un réseau de grande taille.
- Abonnement et désabonnement de ces groupes.
- Transmission de messages (non-fiables) entre groupe.

Extensions

Le multicast IP est utilisé dans de diverses applications:

- Diffusions temps réel de flux audio ou vidéo: dans ce cas, si un paquet est perdu ou arrive dans un mauvais ordre la perte n'est pas très importante, les paquets précédents et suivants sont probablement suffisants pour récupérer.
- Le protocole OSPF, pour la détection automatique de routeurs: dans ce cas, la perte des paquets n'est pas un problème car très peu probable.

Ces applications ne nécessitant pas un contrôle de flux, le multicast IP utilise un protocole non-fiable et n'ayant aucun contrôle d'ordonnancement. JGroups a été rapidement étendu pour pouvoir palier à ces manques et offre plus de fonctionnalités et de flexibilité:

- Un choix parmi des divers protocoles de transport: UDP, TCP ou encore JMS.
- Détection d'abonnement et notification au sujet des membres qui ont rejoint ou quitté le groupe.
- Détection et enlèvement de membres ayant plantés.
- Transmission fiable et ordonné des messages.
- Messages point-à-point.
- Fragmentation des messages trop grands.
- Politiques d'ordonnancement: atomique (tout ou rien), FIFO, ordre total.
- Encryption.

JGroups est donc un système simple, rapide et fiable pour pouvoir mettre en communication un ensemble de machines et proposer services tel que la synchronisation, la répartition de charge ou encore la substitution en cas de problème.

L'API JGroups

JGroups utilise une API simple et qui ressemble grandement aux sockets UDP. L'exemple suivant crée un canal (groupe) et transmet puis réceptionne un message:

```
// On définit les propriétés de la connexion
String props="UDP:PING:FD:STABLE:NAKACK:UNICAST:" +
             "FRAG:FLUSH:GMS:VIEW_ENFORCER:STATE_TRANSFER:QUEUE";

// Création et connexion du canal
Channel channel=new JChannel(props);
channel.connect("TestGroup");

// Envoi du message "Hello world"
channel.send(new Message(null, null, "Hello world"));

// Reception du message -car le message est transmis
// à tout le groupe; inclus l'expéditeur
System.out.println("Received: " + channel.receive(0));

// Fini!
channel.disconnect();
channel.close();
```

Les propriétés d'un canal sont spécifiées à la création. Ces propriétés peuvent inclure:

- UDP: Multicast IP.
- FD: Détection d'erreurs.
- STABLE: “Garbage collection” distribué de messages.
- NAKACK: Messages multicast fiables et FIFO.
- UNICAST: Messages unicast fiables et FIFO.
- FRAG: Fragmentation de messages trop gros.

Une fois le canal crée, la méthode **`connect`** est appelée pour la connexion. Cette méthode retourne une fois la connexion établie au groupe (si le groupe n'existait pas, il est crée et la machine courante devient le premier membre du groupe).

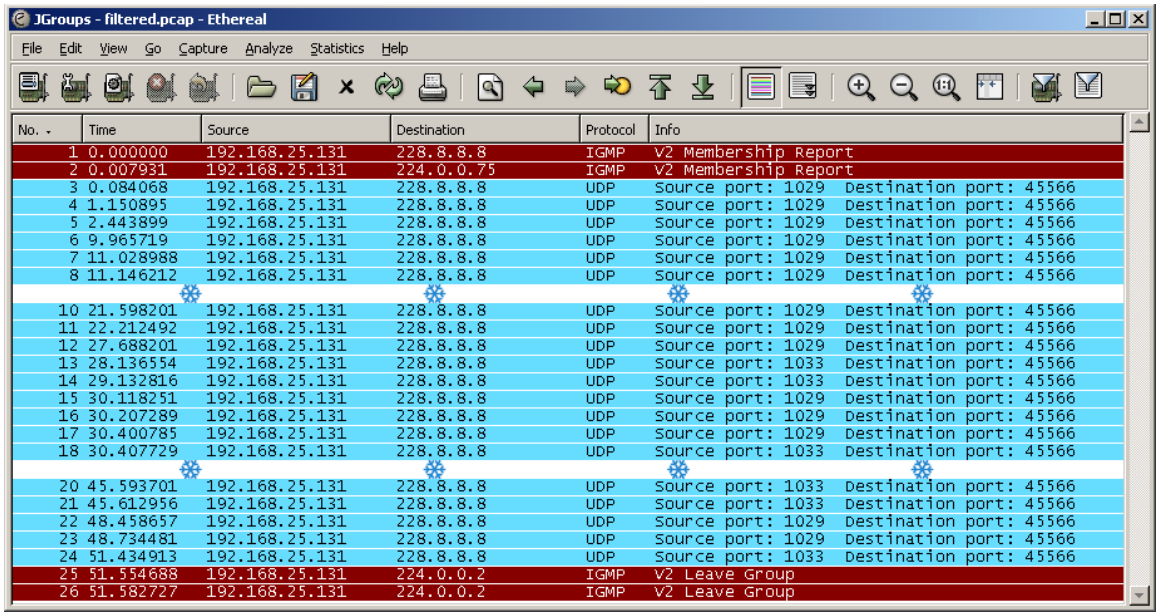
Puis, on envoie un message grâce à la méthode **`send`**. Les arguments sont le destinataire (`null` pour tous les membres), l'expéditeur (**`null`** pour que JGroups le remplisse automatiquement) et un contenu à envoyer (sérialisable).

Comme le message est envoyé à tous les membres, l'expéditeur le reçoit aussi. La méthode `receive` va donc recevoir le “Hello world”. On notera que l'on aurait pu très bien spécifier une architecture publish / subscribe, donc qu'une méthode soit appelée automatiquement à chaque réception de message.

Finalement, on déconnecte et ferme le canal.

Le fonctionnement de JGroups

On peut aussi analyser les messages envoyés par JGroups. Pour ceci, on va utiliser le logiciel Ethereal.



Quand on l'utilise en mode UDP, JGroups agit comme un agent multicast IP classique:

- Premièrement, un paquet “PING” est envoyé au groupe multicast (adresse IP 228.8.8.8, port 45566). Comme le routeur sait que ce multicast n'existe pas, des paquets “IGMP V2 Membership Report” (Rapport d'Abonnement IGMP version 2) sont générés et le groupe est crée.
- Puis, le membre initial envoie, de façon périodique, des paquets de “PING” et “NAKACK”.
- Au paquet 13, notre deuxième membre (avec port de départ 1033) rejoint le groupe. Pour ceci, il suffit d'envoyer un paquet “PING” au groupe multicast.
- Comme on a deux membres, ces derniers envoient périodiquement des paquets “PING” pour montrer qu'ils sont bien vivants.
- Quand le groupe est détruit, le routeur génère les messages “IGMP V2 Leave Group”. Le groupe est donc détruit.

Exemple d'application: répartiteur de requêtes

Pour cette application, nous supposons que l'architecture est une architecture disposant d'un point d'entrée et d'un ensemble de serveurs pour servir des clients. JGroups sera utilisé d'un part pour que le point d'entrée soit au courant de la liste des serveurs disponibles, d'autre part pour les requêtes provenant des clients puissent être distribués aux serveurs.

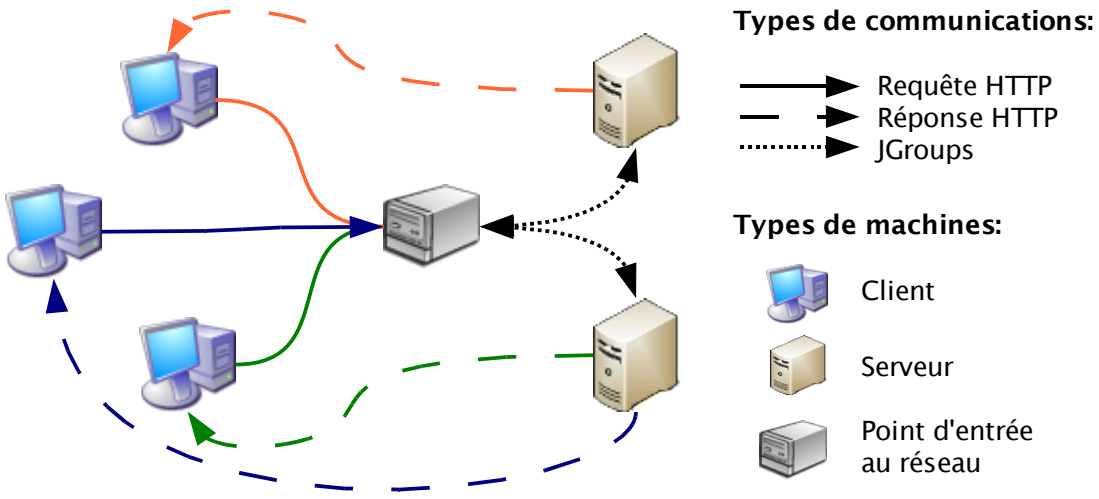


Schéma: Architecture du service proposé

La méthode `OnRequest` du point d'entrée au réseau, appelé quand un client essaye d'accéder à une ressource (on suppose qu'il essaye d'accéder à du contenu HTTP) du réseau, est programmée comme suit:

```
public void OnRequest( HTTPRequest request ) {
    // Récupérer la liste des serveurs disponibles
    Vector<Address> servers = channel.getView().getMembers();
    if( servers.size() <= 1 ){
        // Dans ce cas là, il n'y a aucun serveur!
        OnError( 501 );
    } else {
        int gatewayPosition = servers.indexOf(channel.getLocalAddress());
        int targetServer = gatewayPosition;

        // Sélectionner le serveur qui doit executer la requête en
        // vérifiant bien que c'est un serveur et pas le point d'entrée
        while ( targetServer  == gatewayPosition ) {
            targetServer = random.nextInt(servers.size());
        }

        // Rediriger la requête au serveur, qui lui répondra directement au client
        channel.send(new Message( servers.get(targetServer), null, request ));
    }
}
```

Chaque serveur aura un code d'écoute comme suit:

```
while( true ) {
    // Attendre pour une requête
    Object o = channel.receive(0);

    // Ne traiter que les messages normaux
    if( o instanceof Message ) {
        o = ((Message) o).getObject();
        // Vérifier que c'est une requête HTTP
        if( o instanceof HTTPRequest ) {
            // Appeler le serveur HTTP classique avec la requête
            ProcessRequest( (HTTPRequest) o );
        }
    }
}
```

L'initialisation des canaux peut se faire comme dans l'exemple, ou encore en utilisant des modes de transports ou encore des options de fiabilité et d'ordonnancement différents.

On observe que l'implémentation qui est très simple permet tout de même d'avoir un système où des serveurs peuvent être ajoutés et retirés à tout moment. Bien sûr, dans tous les cas, le client aura l'impression d'être desservi par le point d'entrée au réseau donc le processus est totalement transparent côté client.

Tout de même, il est évident de voir que le système n'est pas un système de répartition de charge, au sens où le répartiteur n'est pas au courant de la charge sur les serveurs et que la répartition des requêtes est assurée par fonction `random`.

Conclusions sur JGroups

JGroups est donc un système simple tout de même fiable pour des applications ayant besoin d'avoir des groupes dynamiques. C'est un projet mature, dont la distribution "source" vient avec 19 exemples et 117 tests dont des tests de fonctionnalité, de performance et de montée en charge. Quelques ouvertures proposées sur le site web http://www.jgroups.org/javagroupsnew/docs/open_projects.html sont:

- Plugin Ethereal: nous avons vu dans la partie "Le fonctionnement de JGroups" qu'Ethereal ne reconnait pas les divers paquets JGroups. Or, ceci pourrait être très utile pour tracer et déboguer.
- Support Bluetooth: un des avantages de JGroups est que le protocole de communication est spécifié à la création de l'objet canal et donc que les méthodes de réception et d'émission sont totalement indépendants des protocoles sous-jacentes. Donc, en plus au support UDP, TCP et JMS on pourrait penser à l'ajout du protocole Bluetooth. Ceci pourrait être utile pour l'utilisation de JGroups sur supports mobiles (jeux sur téléphones portables notamment).
- Compatibilité avec standards Java: JavaSpaces, Jini, ...
- Un projet de répartisseur de charge HTTP "tout fait".

Comparé aux technologies rivales, JGroups a aussi d'autres éléments absents:

- JGroups propose un système pour diviser les messages trop gros, mais pas pour mettre ensemble les messages trop petits (et donc économiser en bande passante).
- La fiabilité est spécifiée à la connexion. D'autres systèmes permettent de spécifier les options tel que la fiabilité ou l'ordonnancement par message.
- JGroups n'a pas de priorité de messages.
- JGroups ne propose pas l'annulation de messages en attente.
- JGroups ne permet pas la création de sous-groupes dans un groupe.
- Avec JGroups, on ne peut pas faire d'émissions ou de réceptions en "flux" (on ne peut seulement le faire par bloc de données).
- JGroups permet l'échange d'informations entre machines mais pas l'échange d'états. Donc, quand on appelle `getMembers`, on n'a que des adresses. Or, si on pourrait avoir la possibilité de mettre des champs (comme la charge actuelle ou l'état du matériel) ceci simplifierait le travail à faire pour des projets tel que le répartisseur de charge.

De plus, JGroups ne possède pas de logo, ce qui influence de façon négative son image. Aussi, le guide d'utilisateur en ligne est en partie vide. Le site web de JGroups donne donc plutôt l'impression d'un “petit projet” que celui d'un projet de production de référence.

Références

- Site web JGroups et les liens de ce site (surtout les présentations JBoss)
- Divers sites web comme Wikipédia ou le site de Cisco pour les protocoles de groupe et de répartissement de charge TCP/IP
- Ethereal
- Cours de Didier Donsez sur Jini
- Librairie MSDN de Microsoft

Article, présentation et code source disponibles sur
http://scholar.alishomepage.com/Master/JGroups/

Article, slideshow and source codes available on
http://scholar.alishomepage.com/Master/JGroups/