Septième partie

Groupes et diffusion

Contenu de cette partie

Abstraction de la communication entre plusieurs sites (en présence de défaillances)

- Ensemble de sites → groupe
- Diffusion : définitions, propriétés
- Prise en compte des défaillances : fiabilité, uniformité, atomicité
- Mise en œuvre
- Exemple : JavaGroups

Source : S. Krakowiak

Situation

Difficultés majeures de la répartition

- asynchronisme
- absence d'état global
- pannes (partielles)

Nécessité de services/abstractions pour réduire/encadrer l'incertitude liée aux pannes :

- en termes de contrôle → décision partagée
  → consensus, détecteur de fautes
- en termes de propagation de l'information
  → protocoles de groupes, diffusion atomique

Plan

Groupe de processus

But

Permettre de transmettre une information de manière homogène à un ensemble (groupe) de sites désigné

Opérations sur les groupes de processus

- *appartenance* (group membership)
  - lister les membres du groupe (*vue* du groupe)
  - évolution du groupe : ajout/retrait de membres
- *diffusion* : communication d'ensemble vers les membres du groupe

Utilisation

- travail coopératif, partage d'information : chat, simulation répartie
- tolérance aux fautes, équilibrage de charge : gestion d'un ensemble de services redondants, ou de données dupliquées

Plan

Diffusion : définitions

Diffusion générale (broadcast)

- destinataires = processus d'un seul ensemble (implicite)
- l'émetteur est aussi destinataire.

Exemples

- les membres d'un groupe unique, vus de l'intérieur du groupe ;
- « tous » les processus joignables

Diffusion de groupe (diffusion sélective, multicast)

- les destinataires sont les membres d'un (ou plusieurs) groupe(s) désignés(s) explicitement.
- L'émetteur peut être extérieur au(x) groupe(s)

Interface

```
broadcast (site émetteur, message)
multicast (site émetteur, message, liste groupes destinataires)
deliver(site récepteur, message)
```

Diffusion : paramètres

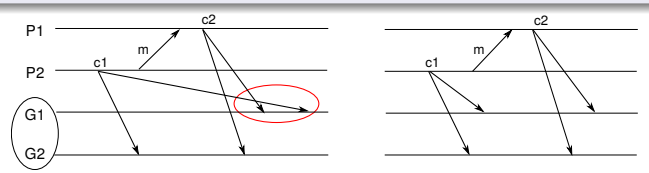
Propriétés relatives à l'ordre d'émission

Diffusion FIFO

Deux messages d'un même émetteur sont délivrés (à chaque membre du groupe) dans l'ordre de leur envoi.

Diffusion causale

Si deux diffusions sont causalement liées, l'ordre de délivrance des messages respecte cet ordre causal.



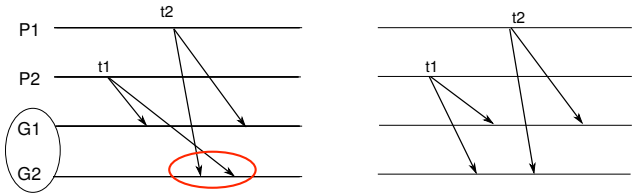
Diffusion : paramètres
Propriétés indépendantes de l'ordre d'émission

Diffusion fiable

Un message est délivré à tous les destinataires corrects ou à aucun.

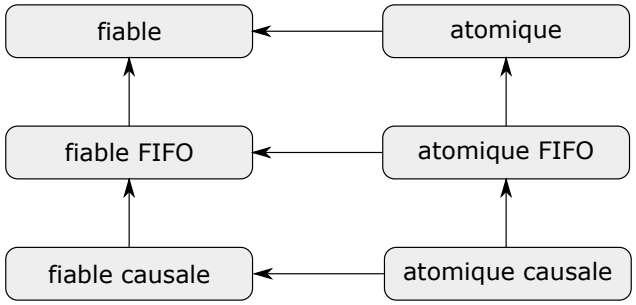
Diffusion totalement ordonnée (*diffusion atomique*)

La diffusion est fiable et les messages sont délivrés dans le même ordre sur tous leurs destinataires.



L'ordre total peut (ou non) être compatible avec la causalité

Diffusion : paramètres
Synoptique

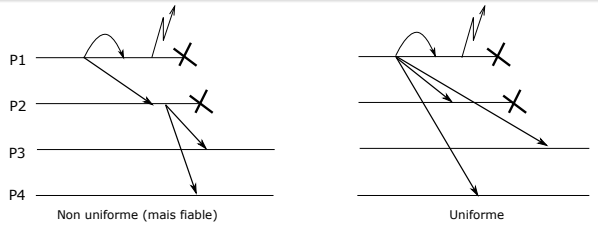


- $\rightarrow \approx$ implication logique/raffinage
- propriétés combinables avec
  - des contraintes de temps (délai maximum pour la délivrance)
    - $\Rightarrow$ hypothèses de synchronisme sur le service de communication
  - une extension à tous les sites (corrects ou fautifs) : uniformité

Diffusion : paramètres
Uniformité

Diffusion fiable uniforme

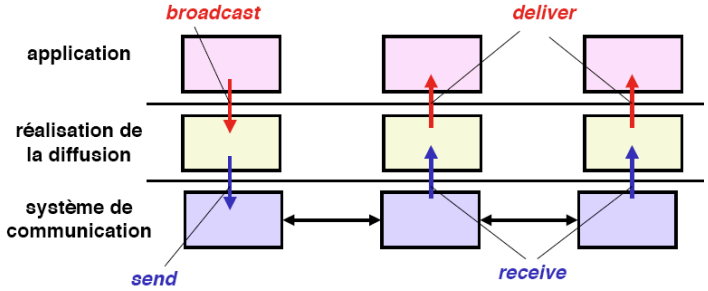
Si un message est délivré à un processus (correct ou défaillant), il sera délivré à tous les processus corrects



Nécessaire si un processus défaillant peut (avant sa défaillance) provoquer des actions irréversibles, notamment vis-à-vis de son environnement (effets de bord).

Plan

Réalisation de la diffusion (propriétés liées à l'ordre d'émission)



- Situation : la diffusion est réalisée au-dessus d'un service de communication permettant l'envoi et la réception de messages (primitives send et receive).
- Principe : distinguer la réception (au niveau du service de communication) et la délivrance (à l'application) : les messages reçus sont mis en attente, jusqu'à ce que les propriétés de diffusion souhaitées soient vérifiées.

Diffusion : résultats (propriétés non liées à l'ordre d'émission)

Hypothèses

- sites/processus sujets à pannes franches
- service de communication
  - fiable : tout message finit par arriver, intact, si l'émetteur et le récepteur restent connectés au service (non défaillants) (⇒ *pas de partition* du réseau)
  - asynchrone (délais de transmission finis non bornés) ⇒ impossible de distinguer un site en panne d'un site lent

Alors

- La diffusion fiable (uniforme) est réalisable simplement.
- La diffusion atomique (totalement ordonnée) n'est pas réalisable
  Remarque : résultat analogue pour les protocoles d'appartenance (impossible de fournir à chaque membre du groupe une même suite de vues totalement ordonnée)

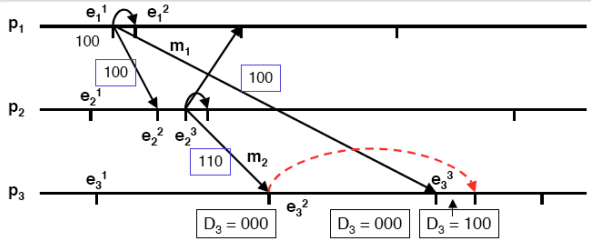
Exemples (rappels)

Diffusion FIFO

- messages estampillés dans l'ordre croissant par chaque émetteur
- un message estampillé <id_émetteur, num> n'est délivré qu'une fois délivré le message estampillé <id_émetteur, num-1>.

Diffusion causale

- diffusions datées sur chaque site par des horloges vectorielles
- messages estampillés par la date de diffusion
- message délivré si son passé causal est inclus dans celui du récepteur



Spécification de la diffusion fiable *[uniforme]*

- Intégrité *(sûreté)* : Quel que soit le message m,
  - il est délivré au plus une fois à tout processus correct *[ou fautif]*,
  - et seulement s'il a été diffusé par un processus
- Accord : si un processus correct *[ou fautif]* délivre 1 message m, tous les processus corrects délivrent m (au bout d'un temps fini)
- Terminaison : si un processus correct diffuse un message m, tous les processus corrects délivrent m (au bout d'un temps fini)

[entre crochets : version uniforme]

Réalisation de la diffusion fiable (rappel)

Réalisation de broadcast(p,m) (p, processus émetteur de m)

pour tous les voisins de p (et p) **faire** send(<m,p>) **fin pour**

Contrôle de deliver(m)

Réalisation de receive(<m, sender(m)>) par le processus q :

si q n'a pas déjà exécuté deliver(m) **alors**
  si sender(m) $\neq$ q **alors**
   pour tous les voisins de q **faire** send(<m,sender(m)>) **fin pour**
  fin si
  deliver(m)
fin si

→ tout site délivrant 1 message l'a auparavant envoyé à ses voisins

→ si un message n'est pas délivré à un site correct, vu que tout site correct est connecté à au moins un voisin correct (pas de partition), il faudrait donc (en remontant de proche en proche jusqu'à atteindre l'émetteur initial) qu'aucun site correct ne l'ait envoyé, donc délivré.

Réalisation de la diffusion fiable uniforme

Réalisation de broadcast(p,m) (p, processus émetteur de m)

pour tous les sites **faire**
  send(<m,p>)
fin pour

Contrôle de deliver(m)

Réalisation de receive(<m, sender(m)>) par le processus q :

si q n'a pas précédemment exécuté deliver(m) **alors**
  si sender(m) $\neq$ q **alors**
    pour tous les *sites* **faire**
      send(<m,sender(m)>)
    fin pour
  fin si
  deliver(m)
fin si

Impossibilité de la diffusion atomique

Résultat

Dans un système avec une communication asynchrone fiable, et des sites sujets à pannes franches, la diffusion atomique se réduit au consensus.

→ impossibilité de la diffusion atomique ≡ impossibilité du consensus

Réduction

i) *Avec un algo. de diffusion atomique, on peut réaliser le consensus.*
- Chaque processus diffuse atomiquement sa valeur proposée à tous les membres du groupe.
- Tous les processus corrects reçoivent le même ensemble de valeurs dans le même ordre.
- Ils décident la première valeur, la même pour tous.

ii) *Avec un algo. de consensus, on peut réaliser la diffusion atomique*
- les messages sont transmis par diffusion fiable
- algorithme par tours : à chaque tour, chaque site propose l'ensemble des messages reçus non encore délivrés
- le consensus permet de déterminer un ordre commun pour les messages à délivrer dans le tour courant (s'il y en a)

Réalisation de la diffusion atomique

Les algorithmes « pratiques » de réalisation de la diffusion atomique doivent donc relâcher des contraintes
- introduire du synchronisme (utiliser un délai de garde) (supposer un détecteur de fautes)
- interdire les pannes définitives (supposer que certains processus en panne peuvent être rétablis)

Méthodes utilisées pour construire l'ordre global
- utiliser un service produisant une suite croissante (*séquenceur*)
- construire l'*ordre à l'émission*
- construire l'*ordre à la réception*

Utiliser un site séquenceur

- Pour diffuser un message, on l'envoie au séquenceur.
- Celui-ci lui attribue un numéro (estampille), dans une suite croissante (1, 2, 3, etc.), et l'envoie à tous les destinataires.
- Les destinataires délivrent les messages diffusés dans l'ordre croissant des estampilles.

- Le séquenceur peut être fixe ou circulant (jeton)
- Difficulté : résister à la panne du séquenceur
  → redondance (détection (scrutation périodique), élection, reprise)
- Utilisé dans le protocole JavaGroups

Ordre fixé à la réception (ex : protocole ABCAST (Isis))

- Chaque message m est estampillé provisoirement par son heure logique de réception (horloges de Lamport).
- Les différents récepteurs se communiquent leurs estampilles provisoires (l'émetteur peut jouer le rôle de collecteur)
- Quand toutes sont connues, on attribue définitivement à m la plus grande
  → tout message a la même estampille (définitive) sur tous les sites
- Les messages sont délivrés dans l'ordre des estampilles définitives

Quand un message d'estampille définitive *e* est délivré, *e* minore toutes les estampilles définitives (fixées ou à venir) de messages non encore délivrés sur le site :

- l'estampille définitive majore l'estampille provisoire
- avec des canaux FIFO, les estampilles provisoires à venir majorent les estampilles provisoires déjà reçues
  → les estampilles définitives à venir majorent les estampilles définitives déjà fixées

Difficulté : défaillances → envois redondants + délais de garde

Ordre défini à l'émission

Estampilles déterminées au moment de l'émission

- Exclusion mutuelle pour l'émission (ex : jeton)
  → ordre de passage en exclusion mutuelle
- <Horloges logiques (Lamport), id site>
- ou ordre préétabli

- Principe ordre construit
  Lorsqu'un site connaît l'estampille du prochain message proposé par chacun des sites, il peut déterminer le prochain message à délivrer : c'est celui dont l'estampille est la plus petite.
- Difficulté ordre construit : vivacité
  → canaux FIFO
  → envoi périodique de messages vides.

Plan

Etude de cas : JavaGroups

Gestion de groupe, et services de diffusion élaborés : diffusion fiable, ordonnée, causale, sélective...

Références

- site du projet JavaGroups (en anglais)
  https ://sourceforge.net/projects/javagroups
- Fournit la documentation (API, manuels, tutoriels, exemples), et les liens de téléchargement
- le blog de l'auteur principal du projet
  http ://belaban.blogspot.com
- page wiki JGroups : http ://community.jboss.org/wiki/JGroups

Principe

- La communication se fait via un canal
  (un canal est associé à un groupe et un seul)
- Les propriétés requises pour la communication sont réalisées par des protocoles
  - chaque protocole assure une propriété particulière : fiabilité, ordonnancement, groupes
  - l'ensemble des propriétés souhaitées pour la communication définit l'ensemble de protocoles associé au canal
    - chaque protocole est implémenté par une classe Java
    - les protocoles utilisés pour un canal sont structurés en une pile de protocoles
    - un fichier XML permet de spécifier et paramétrer chacun des protocoles constituant la pile associée à un canal

Opérations sur un canal

- création/destruction : ch = new JChannel(fprotocole.xml)/ch.close()
- rejoindre un groupe utilisant le canal : ch.connect("nomGroupe")
  le 1^{er} processus qui rejoint le groupe crée le groupe, s'il n'existe pas
- quitter un groupe : ch.disconnect()
- diffuser un message : ch.send(message)
- recevoir (délivrer) un message
  → synchrone : méthode ch.receive(message) associée au canal
  → ou via une interface de rappel :
  - le récepteur doit hériter de la classe Receiver ou ReceiverAdapter
  - ce qui implique qu'il implante une méthode receive(message), appelée à la délivrance
  - le récepteur s'abonne auprès du canal par ch.setReceiver(this)

Autres méthodes de rappel

- un récepteur doit aussi implanter la méthode viewAccepted(groupe_courant), appelée par le service de diffusion chaque fois que le groupe évolue
- deux accesseurs permettent de communiquer l'état d'un objet associé au groupe : membre.getState()/setState(état).
  Pour être mis à jour, un processus doit appeler canal.getState(), ce qui provoquera le transfert de l'état (obtenu d'un autre membre par mb.getState()), par le rappel de mb.setState()
  - l'appel de ch.getState est utile en particulier lorsqu'un membre rejoint un groupe
  - l'appel de m.setState()permet d'ordonner la réception de l'état par rapport aux diffusions
  - l'état du groupe est dupliqué sur les différents membres

Exemple : réalisation d'un chat (source : tutoriel Jgroups)

```
import org.jgroups.JChannel;
import org.jgroups.Message;
import org.jgroups.ReceiverAdapter;
import org.jgroups.View;
import org.jgroups.util.Util;

import java.util.List;
import java.util.LinkedList;
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class SimpleChat extends ReceiverAdapter {
    JChannel channel;
    String user_name=System.getProperty("user.name", "n/a");
    final List<String> state=new LinkedList<String>();

    public void viewAccepted(View new_view) {
        System.out.println("** view: " + new_view); }

    public void receive(Message msg) {
        String line=msg.getSrc() + ": " + msg.getObject();
        System.out.println(line);
        synchronized(state) { state.add(line); }
    }

    public byte[] getState() {
        synchronized(state) {
            try { return Util.objectToByteBuffer(state);}
            catch(Exception e) {e.printStackTrace(); return null;
            }
        }
    }
```


```
    public void setState(byte[] new_state) {
        try {
            List<String> list=(List<String>)Util.objectFromByteBuffer(new_state);
            synchronized(state) { state.clear(); state.addAll(list);}
            System.out.println("rcvd state ("+list.size()+" msgs in chat history):");
            for(String str: list) { System.out.println(str);}
        } catch(Exception e) {e.printStackTrace();}
    }

    private void start() throws Exception {
        channel=new JChannel();
        channel.setReceiver(this);
        channel.connect("ChatCluster");
        channel.getState(null, 10000);
        eventLoop();
        channel.close();
    }

  private void eventLoop() {
        BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
        while(true) {
            try {
                System.out.print("> "); System.out.flush();
                String line=in.readLine().toLowerCase();
                if(line.startsWith("quit") || line.startsWith("exit")) { break;}
                line="[" + user_name + "] " + line;
                Message msg=new Message(null, null, line);
                channel.send(msg);
            } catch(Exception e) {}
        }
    }

  public static void main(String[] args) throws Exception {
        new SimpleChat().start();
    }
}
```


Bilan (JavaGroups)

- Interface simple
- Composition de protocoles simple, modulaire
- Variété de protocoles élémentaires disponibles
  - transport (UDP, TCP)
  - découverte (Ping...)
  - une forme de fiabilité, FIFO
  - sécurité
  - détection de pannes
  - fragmentation
  - transfert d'état
  - trace
  - diffusion probabiliste
  - appartenance
  - ...
  - → outils/protocoles de base pour réaliser les protocoles de plus haut niveau (diffusion ordonnée, causale)
- Logiciel libre
  - communauté active, mais assez réduite