CSC4509 : Algorithmique répartie et communications entre processus distants

Denis Conan, avec Éric Lallet

Mai 2025

Plan de la présentation

1 Communications au cœur des systèmes répartis

- Communications (réparties)
  $\Longrightarrow$ espace mémoire réparti sans unicité de temps
  - Réseaux transmettant l'information au mieux (en anglais, *best effort*)
  - Machines et réseaux hétérogènes (matériel et logiciel)
- Propriétés physiques des communications
  - Bande passante et latence des lecture et écriture de plusieurs ordres de grandeur plus importants que ceux d'une mémoire locale
  - Entraves multiples : dépendances nombreuses vis-à-vis de services distants potentiellement défaillants, administrations différentes, etc.

1.1 Quid de la sémantique de l'acquittement

- Qu'est-ce qu'un acquittement ?

1. Le traitement demandé par le message a été effectué
2. Le traitement du message est accepté et sera effectué à terme, c.-à-d. *in fine*
3. La temporisation associée au message envoyé peut être arrêtée
4. Les zones mémoires associées au message envoyé peuvent être libérées
5. Un autre message peut être émis
6. Etc.

1.2 Quid de la sémantique des transmissions de messages ?

- « Exactement une fois » : c'est la sémantique idéale car le client peut ignorer le fait que son appel est distant ou non
  - Les mises en œuvre fournissant cette sémantique approchent le cas idéal en ajoutant de la surveillance mutuelle $\Longrightarrow$ par des algorithmes répartis
    - Mais, comme le client et le service sont indépendants, cela est théoriquement impossible à garantir dans tous les cas à cause des défaillances
- « Au plus une fois » : si le client ne reçoit pas de réponse en un temps donné, l'appel revient en erreur, indiquant que le service ne peut pas être rendu
  - C'est moins triviale qu'il n'y paraît, p.ex., si c'est le message de réponse qui est perdu alors que le service a été rendu $\Longrightarrow$ impact sur l'algo. côté « client »
- « Au moins une fois » : si le client ne reçoit pas de réponse en un temps donné, il retransmet jusqu'à réception d'une réponse
  - Le « serveur » doit tolérer les appels redondants $\Longrightarrow$ impact sur l'algo. côté « serveur »

1.2.1 Illustration avec OASIS MQTT pour l'IoT

Les connaissances de ce cours sont à la base des intergiciels (*Distributed Middleware*), ici le standard MQTT.

- MQTT v.5.0: an OASIS standard in Dec. 2019
  - *For Internet of Things (IoT) and for Machine-to-Machine (M2M)*
  - *Use of the publish/subscribe message pattern which provides one-to-many message distribution and decoupling of applications*
  - *Three qualities of service for message delivery :*
    1. *"At most once" : Message loss can occur. This level could be used, for example, with ambient sensor data where it does not matter if an individual reading is lost as the next one will be published soon after*
    2. *"At least once" : Messages are assured to arrive but duplicates can occur*
    3. *"Exactly once" : Messages are assured to arrive exactly once. This level could be used, for example, with billing systems where duplicate or lost messages could lead to incorrect charges being applied*
- Ces propriétés sont garanties dans le cadre de sessions
  - Une session = un ensemble de connexions

2 Algorithmique répartie

- Une application répartie est un ensemble de processus
  - Avec dans ce module
    - Les processus qui sont modélisés comme des machines à états
      = Un état initial
      + une séquence d'actions donnant une séquence d'états
    - Les processus communiquent par échange de messages
- La répartition est à la base de la tolérance aux fautes
  - Lorsqu'un service est défaillant, un autre « prend le relais »
  - La problèmatique de la tolérance aux fautes est étudiée dans le cadre d'études d'articles de recherche (en quadrinôme)

Des exemples d'algorithmes répartis / de primitives réparties ?

— Communication de groupe

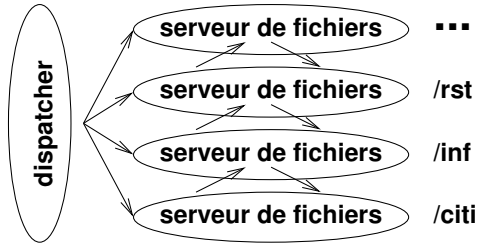
— Terminaison

2.1 P. ex., communication de groupe ?

Il existe des cas où un processus doit envoyer le même message ou la même suite de messages à tous les autres processus : c'est la diffusion.

- Solution de départ : on utilise *IP multicast* ou de multiples envois TCP/UDP
  - Pb 1 : Que se passe-t-il si un processus (l'émetteur ou l'un des récepteurs) défaille pendant la diffusion d'un message ?
  - Pb 2 : Supposons une application répartie de réservation de places d'avion.
    Un client réserve des places (1^{er} msg) puis supprime sa réservation (2^{nd} msg).
    Que se passe-t-il si le premier message est perdu ou non encore arrivé/traité ?
  - Pb 3 : Supposons une application de tchat.
    Un premier utilisateur annonce la réservation d'une table dans un restaurant pour ce soir 20h.
    Un second utilisateur reçoit ce 1^{er} message et répond avec, dans sa réponse (2^{nd} message), la nouvelle heure de réservation sans la date.
    Que se passe-t-il si un troisième utilisateur reçoit le 2^{nd} message sans le 1^{er} ?
  - Etc.

2.2 P. ex., terminaison d'une application répartie



Comment arrêter une application répartie dont l'algorithme de chaque processus est structuré en une boucle infinie d'attente de messages en provenance du réseau ?

- Solution de départ : on exécute des commandes `ssh` sur les différentes machines pour arrêter les processus
  - Pb 1 : C'est trop violent si chaque processus doit laisser le système de fichiers de la machine hôte dans un certain état
  - Pb 2 : Quand bien même cette première solution serait acceptable, comment sait-on que c'est terminé ?
  - Pb 3 : Quand bien même un processus serait capable de dire qu'il ne fait plus rien, que fait-on des messages en transit dans les canaux de communication ?
  - Pb 4 : Et quid des processus qui créent d'autres processus, c'est-à-dire d'une application répartie dont la « configuration » évolue constamment ?
  - Etc.

3 Objectifs du module

- À l'issue du module, vous serez capables, dans le cadre du développement d'une application répartie de petite taille, mais réaliste, et dont le cahier des charges est fourni, de :

  1. développer entièrement une application répartie (par exemple un tchat multiclient multiserveur) avec TCP/IP en utilisant la bibliothèque JAVA NIO avec des communications en modes connecté et déconnecté, et en modes synchrone et asynchrone,
  2. lire, donner l'exécution, et expliquer les preuves d'algorithmes répartis de base (élection, diffusion, exclusion mutuelle, interblocage, et détection de terminaison),
  3. mettre en œuvre les algorithmes répartis étudiés par insertion dans l'application répartie développée dans le module et avec écriture de scénarios de tests simples (c.-à-d. sans utilisation de canevas logiciel d'émulation ou de simulation) pour vérifier les propriétés de correction et de progression.

- Par ailleurs, vous serez aussi capables de :

  4. étudier en groupe des articles de recherche sur la thématique de la tolérance aux fautes pour en faire une restitution à la classe.

4 Organisation du module I

1. Concevoir et mettre en œuvre une application répartie
   - Mise en contexte + abstraction en concepts + devoir maison
   - Diapositives présentées en cours intégrés
   - Application des concepts avec `JAVA NIO` en TP
   - Travail personnel : finir les TPs, lire des tutoriels, **faire le devoir maison**
2. Comprendre un algorithme réparti
   - Mise en contexte + abstraction en propriétés + exercices et QCM
   - **Classe inversée** : cours en auto-apprentissage et exercices en classe
   - Travail personnel : auto-apprentissage puis **QCM** (après chaque séance d'exos)
3. Mettre en œuvre et tester un algorithme réparti dans une appli. répartie
   - Réalisation d'une étude de cas en binôme
   - Fonctionnement en mode projet : **code + rapport (compte rendu des TPs)**

4 Organisation du module II

4. Présenter un article de recherche en tolérance aux fautes
   `http://www-inf.telecom-sudparis.eu/COURS/CSC4509/?page=articles_recherche`

   - Étude d'un état de l'art par groupe de 4
   - Étude et préparation avec un tuteur
   - **En fin de module, soit présentation devant la classe, soit QCM à la classe**
     - Être capable de répondre à des questions pendant la séance des questions/réponses

Évaluation individuelle : préparation, compréhension, présentation ou QCM, et réponse aux questions

5 Organisation des études d'articles de recherche

1. Sondage pour émettre vos préférences (sondage à venir)
2. Constitution des quadrinômes (suite au sondage)
3. Travail en équipe avec tutorat, 3 réunions avec le tuteur
   (des créneaux sont programmés dans OpenPortal ; vous les utilisez ou pas)
4. En fin de module,
   - soit présentation devant la classe de l'étude bibliographique,
     - 30mn de présentation et 15mn de questions/réponses avec discussion
   - soit présentation de l'étude par le tuteur (ses diapos) et QCM par le quadrinôme
     - 20mn de présentation puis 20mn de QCM et 5mn de questions/réponses
   - Présence obligatoire de tous

6 Évaluation du module

- Note finale =
  - Partie programmation réseau JAVA NIO : 9 points
    - Devoir en binôme sur la programmation réseau avec JAVA NIO : 2 points
    - TP noté prog. réseau JAVA NIO en monôme : 7 points
  - Projet en binôme sur la mise en œuvre d'algorithmes répartis : 8 points
    - Code + rapport de réalisation en binôme
  - Étude d'un article de recherche en tutorat : 3 points
    - Le barème est construit sur la préparation, la présentation ou le QCM, ainsi que la compréhension de l'article
- **VAP = la présence en cours et la participation sont nécessaires**