École Doctorale de Grenoble
Master 2 Recherche “Systèmes et Logiciel”

Validation atomique

Sacha Krakowiak
Université Joseph Fourier
Projet Sardes (INRIA et IMAG-LSR)

http://sardes.inrialpes.fr/people/krakowia

Motivation et objectifs

- **Exécution de transactions réparties**
  - Les transactions sont un outil de base pour la tolérance aux fautes
  - Transaction = suite d’actions A = (A_1 ; A_2 ; … ; A_n) possédant les propriétés ACID
    - Atomicité, Cohérence, Isolation, Durabilité (persistence)
  - Dans une application répartie, les actions peuvent s’exécuter sur des sites différents, d’où nécessité d’une coordination
- **Validation**
  - La validation (opération *commit*) est une opération clé de l’exécution d’une transaction (elle rend les modifications définitives)
  - Nous allons examiner comment cette opération est réalisée dans un système réparti
  - Intérêt
    - pratique (réalisation de transactions réparties)
    - théorique (résultats d’impossibilité ; relations avec autres protocoles)

© 2003-2004, S. Krakowiak

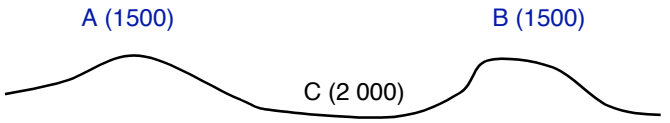
Plan

- **Introduction à la validation**
  - Préambule sur la communication
  - Spécification du problème
- **Solutions “historiques”**
  - Validation à deux phases : le problème du blocage
  - Validation à trois phases
- **Solutions utilisant la diffusion**
  - Spécification du protocole de diffusion (UTRB)
  - Réalisation de la validation
  - Réalisation du protocole de diffusion
- **[à voir plus tard] Relations entre validation et consensus**

© 2003-2004, S. Krakowiak

Préambule : le “paradoxe des généraux” (1)

- **Un premier résultat d’impossibilité**
  - Si le système de communication perd des messages de manière indétectable, il est impossible de réaliser un protocole d’accord en un nombre fini d’étapes
- **Le problème de l’attaque coordonnée**



A → B : message 36 : on attaque demain à 5h00
B → A : bien reçu 36
A → B : bien reçu bien reçu 36
…

J. Gray, Notes on Database Operating Systems, in *Operating Systems, An Advanced Course*, Lecture Notes in Computer Science, nr 60, Springer-Verlag (1978)

© 2003-2004, S. Krakowiak

Le “paradoxe des généraux” (2)

Il n’existe pas de protocole résolvant le problème de l’accord avec un nombre fini de messages si la communication n’est pas fiable

Supposons qu’il existe de tels protocoles, et considérons celui qui réalise l’accord en utilisant le nombre de message minimal, soit n

Supposons que le dernier message soit de B vers A :

…
A → B : $message_{n-1}$
B → A : $message_n$

- Puisque B ne recevra plus de messages, B a pris une décision avant l’envoi de $message_n$
- Puisque le protocole doit fonctionner en cas de perte de messages, la décision de A ne doit pas dépendre du contenu de $message_n$

Mais alors $message_n$ ne sert à rien, et on peut le supprimer.
Ceci est contradictoire avec le fait que le protocole est minimal.

Le “paradoxe des généraux” (3)

Comment vivre avec dans la pratique ?

Rappelons l’hypothèse de défaillance : le système de communication perd des messages **de manière indétectable**

En pratique, on essaie de détecter la perte de messages, soit par signal du système de communication à l’émetteur, soit en général au moyen d’un délai de garde sur l’aller-retour (hypothèse de synchronisme)

Exemple concret : l’accord confirmé

Le protocole d’établissement de liaison de TCP utilise **l’accord confirmé** (3 messages) :

A → B : demande
B → A : accord
A → B : accord confirmé

Chaque partenaire voit donc **un aller-retour**

Le problème de la validation *(commitment)*

- **Motivations**
  - **Pour terminer une transaction, il faut décider de son issue : validation (*commit*) ou annulation (*abort*)**
  - **La validation rend les modifications définitives ; l’annulation ramène les processus à leur état avant transaction (propriété d’atomicité)**
  - **La décision est prise à l’initiative d’un participant quelconque, et doit vérifier différentes propriétés (cf plus loin) pour satisfaire aux spécifications des transactions**
- **Hypothèses**
  - **La communication est fiable (messages délivrés sans altération) et synchrone (une borne est connue sur le temps de transmission des messages, ainsi que sur les vitesses relatives des processus)**

    N.B. il suffit de faire cette hypothèse sur une durée couvrant celle de la transaction
  - **Les processus peuvent subir des pannes franches. Un processus en panne peut être réparé et réintégrer la transaction. Un processus est dit correct s’il n’est jamais tombé en panne**
  - **Chaque processus peut écrire des données en mémoire stable, ou permanente, qui survit aux défaillances**

Spécifications de la validation

Chaque processus contribue à la décision par un vote (OUI ou NON). Un processus qui vote OUI est prêt à rendre permanentes ses modifications. Le protocole doit avoir les propriétés suivantes :

Validité : La décision est soit valider, soit annuler

Intégrité : Tout processus décide au plus une fois (une décision est définitive)

Accord uniforme : Tous les processus (corrects ou non) qui décident prennent la même décision

Justification$_x$: Si la décision est valider, alors il y a au moins x processus qui ont voté OUI [différentes variantes du protocole selon choix de x ; en pratique, x = n (le nombre total de processus)]

Obligation$_x$: Si x processus votent OUI, et si ces x processus sont corrects, alors la valeur décidée est valider [en pratique x = n]

Terminaison : tout processus correct décide au bout d’un temps fini

Remarques sur les spécifications

Les décisions valider et annuler ne sont pas symétriques. Il n'est pas nécessaire que tous les processus aient voté NON pour décider d'annuler

L'implication de la propriété de justification n'est pas symétrique : la décision d'annuler peut être prise même si tous les processus ont voté OUI

Exemple : tous les processus ont voté OUI ; un processus tombe en panne avant que la décision de valider ne soit prise

La propriété d'obligation est nécessaire pour exclure les solutions triviales (par exemple : tous les processus décident toujours d'annuler)

L'hypothèse de la communication fiable et synchrone exclut le partitionnement du réseau (le réseau est maintenu connexe malgré les pannes de site)

Un peu d'histoire

Solution initiale : Gray (1978), protocole de validation à 2 phases (2-*phase commit*, ou **2PC**)

Le protocole 2PC peut ne pas se terminer (même avec communication synchrone). Recherche d'un protocole de validation "non bloquante" (*Non-Blocking Atomic Commitment*, ou NBAC)

Protocole de validation non bloquant à 3 phases (**3PC**) (Skeen, 1982)

Protocole non bloquant utilisant la diffusion uniforme temporisée (Babaoğlu - Toueg, 1992)

Relations entre validation et consensus (détails plus tard)

La validation est "plus difficile" que le consensus, mais une forme plus faible (pratiquement acceptable) de validation se ramène au consensus

avec le protocole Paxos (Lamport)

avec détecteurs de pannes imparfaits (Chandra - Toueg)

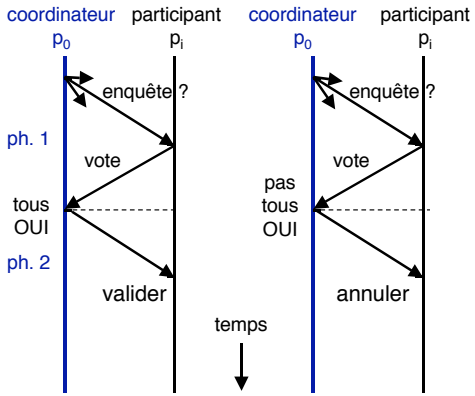
Protocole à 2 phases (2PC)

Un processus coordinateur interroge tous les processus (participants)
Chaque participant envoie son vote au coordinateur
Quand le coordinateur a reçu tous les votes, il prend sa décision :

tous oui : valider

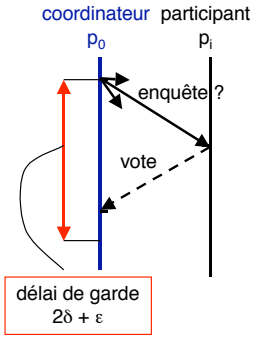
autrement : annuler

Le coordinateur envoie la décision à tous les participants ; chacun décide conformément à ce message



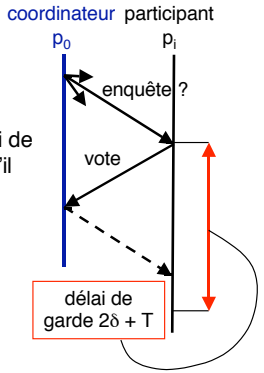
Noter que le coordinateur et chaque participant voient un aller-retour (délai de garde utilisable)

2PC : détection de défaillances



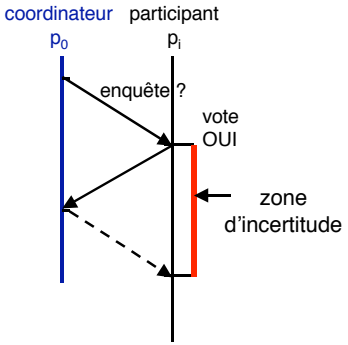
Coordinateur : si des réponses manquent après le délai de garde, décision = annuler
(tous les sites n'ont pas voté OUI)

Participant : si le coordinateur n'a pas répondu après le délai de garde, on suppose qu'il est en panne



Le participant essaie de déterminer si une décision a été prise, en interrogeant les autres participants (un autre participant peut avoir reçu une réponse du coordinateur)

2PC : scénario de blocage



Entre le moment où il a voté OUI et le moment où il reçoit une réponse du coordinateur, un participant est dans une zone d'incertitude (un autre participant peut se trouver soit en phase de validation soit en phase d'annulation, et les deux issues sont possibles)

Cela peut conduire à un scénario de blocage :

- p_i a voté OUI et se trouve dans la zone d'incertitude
- Le coordinateur tombe en panne après avoir envoyé une décision valider ou annuler à certains participants
- ces participants tombent en panne à leur tour

Une décision a été prise mais p_i (correct) n'a aucun moyen de la connaître (au moins jusqu'à un éventuel redémarrage du coordinateur)

2PC : scénario de blocage (2)

La possibilité de blocage provient de l'incertitude sur l'état global (pas de connaissance partagée)

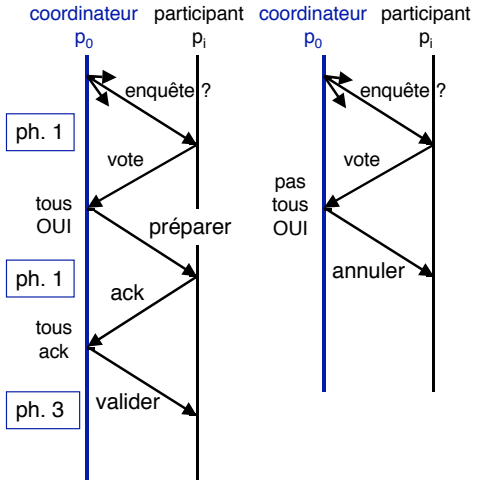
état p_j \ état p_i	voté (OUI)	voté (NON)	décidé valider	décidé annuler
voté (OUI)	possible	possible	possible	possible
voté (NON)	possible	possible		possible
décidé valider	possible		possible	
décidé annuler	possible	possible		possible

À la suite d'un vote OUI, toutes les possibilités restent ouvertes

Protocole à 3 phases (3PC)

- Le coordinateur interroge les participants
- Chaque participant renvoie son vote
- Si la décision est annuler, elle est envoyée aux participants (comme en 2PC)
- Sinon, le coordinateur envoie préparer
- Chaque participant renvoie un acquittement
- Quand il a reçu tous les acquittements, le coordinateur renvoie valider

N.B. s'il n'y a pas de perte de messages, alors les acquittements sont redondants



3PC : détection de défaillances

Réaction à l'écoulement du délai de garde

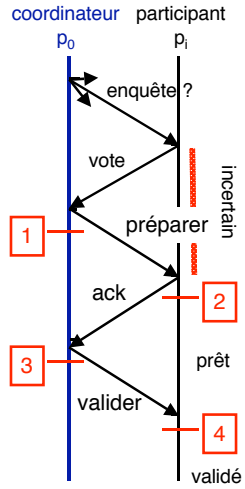
1. Décider annuler
2. Élire un nouveau coordinateur
3. Rien
4. Élire un nouveau coordinateur

Le nouveau coordinateur interroge les participants

- certains annulés : décider annuler
- certains validés : décider valider
- tous incertains : décider annuler
- certains prêts mais aucun validé : reprendre à l'envoi du message préparer

Reprise après défaillance

- si n'avait pas voté : annuler
- si avait décidé : continuer avec décision prise
- si incertain : consulter autres participants



3PC : connaissance partagée

Il n'y a pas d'état où p_i soit incertain (après vote OUI) et p_j ait décidé valider.
Le protocole 3PC introduit une connaissance partagée.

- 2PC : un processus qui décide valider sait que tous ont voté OUI
- 3PC : un processus qui décide valider sait (en outre) que tous savent que tous ont voté OUI

état p_j \ état p_i	voté (OUI)	voté (NON)	prêt	décidé valider	décidé annuler
voté (OUI)	possible	possible	possible		possible
voté (NON)	possible	possible			possible
prêt	possible		possible	possible	
décidé valider			possible	possible	
décidé annuler	possible	possible			possible

Validation utilisant la diffusion

- **Motivations**
  - **Le problème de la "connaissance partagée" identifié dans 3PC peut être abordé en utilisant un protocole de diffusion**
  - **Il existe de nombreux protocoles de diffusion (étudiés plus tard dans le cours). Les spécifications de chaque protocole doivent être adaptées aux hypothèses de défaillance et aux besoins spécifiques d'utilisation**
- **Démarche**
  - **Définir un protocole générique de validation utilisant la diffusion**
    - **générique : acceptant un protocole de diffusion quelconque**
  - **Spécifier un protocole de diffusion ayant les propriétés souhaitées**
  - **Examiner la réalisation de la validation avec ce protocole**
  - **Examiner la réalisation du protocole de diffusion**

Un protocole générique de validation (1)

Schéma d'ensemble d'une transaction

Initialisation (par un processus quelconque)
send [transaction, Δ_c, participants] to all participants

Programme des participants (y compris initiateur)

upon (receipt of [transaction, Δ_c, participants])
C_{know} = local clock
// perform operations requested by transaction
if (willing and able to make updates permanent) **then**
vote = YES
else
vote = NO
// decide commit or abort for the transaction
atomic_commitment(transaction, participants)

Ö. Babaoğlu, S. Toueg, Non-blocking Atomic Commitment, in S. Mullender (ed.), *Distributed Systems* (2nd edition), Addison-Wesley, 1993

Un protocole générique de validation (2)

Coordinator

send [VOTE_REQUEST] **to all**
set-timeout (local_clock + 2δ)
wait-for (receipt of votes from all)
if (all votes YES) **then**
broadcast(commit, participants)
else broadcast(abort, participants)
on-timeout
broadcast(abort, participants)

La structure est la même que celle du protocole 2PC

Ö. Babaoğlu, S. Toueg, Non-blocking Atomic Commitment, in S. Mullender (ed.), *Distributed Systems* (2nd edition), Addison-Wesley, 1993

Participant

set-timeout (C_{know} + Δ_c + δ)
wait-for (receipt of VOTE_REQUEST from coord.)
send (vote) to coordinator
if (vote = NO) **then**
decide (abort)
else
set-timeout (C_{know} + Δ_c + 2δ + Δ_b)
wait-for(delivery of decision messages)
if (decision mess. is abort) **then**
decide (abort)
else decide (commit)
on-timeout // coordinator failed
decide according to termin. protocol
on-timeout
decide (abort)

Validation avec diffusion simple (1)

Les propriétés de l'algorithme de validation dépendent du protocole broadcast inséré dans le protocole générique

Cas 1 : broadcast = diffusion simple (synchrone) :

- si un processus correct diffuse m, tous les processus destinataires
- corrects délivrent m
- pour tout message m, tout destinataire délivre m au plus une fois, et seulement si un processus a diffusé m
- il existe une constante connue Δ_b telle que si la diffusion a été lancée au temps (physique) t, aucum message ne sera délivré après t + Δ_b

Cette diffusion n'a pas la propriété "tout ou rien" (diffusion fiable). En cas de panne de l'émetteur pendant la diffusion, un message peut être délivré à certains destinataires seulement

Validation avec diffusion simple (2)

Le protocole de terminaison (réaction à la panne du coordinateur) tente de déterminer si une décision a été prise en contactant les participants survivants.
Sa terminaison en temps fini n'est pas garantie

Exemple (déjà vu) :

- le coordinateur prend une décision, la diffuse à certains participants, puis tombe en panne
- ces participants tombent en panne à leur tour

En fait le protocole de validation qui utilise la diffusion simple n'est autre que **2PC**

Il faut renforcer les spécifications de l'algorithme de diffusion pour qu'il ait la propriété tout ou rien

Validation avec diffusion fiable uniforme (1)

diffusion simple

Cas 2 : broadcast = diffusion fiable uniforme synchrone :

- si un processus correct diffuse m, tous les processus destinataires
- corrects délivrent m
- pour tout message m, tout destinataire délivre m au plus une fois, et seulement si un processus a diffusé m
- il existe une constante connue Δ_b telle que si la diffusion a été lancée au temps (physique) t, aucun message ne sera délivré après t + Δ_b

+ accord uniforme

- si un processus (correct **ou non**) délivre m, tous les processus corrects délivrent m

uniformité

Cette diffusion a la propriété "tout ou rien" (diffusion fiable). En cas de panne de l'émetteur pendant la diffusion, le message est délivré à tous les participants ou à aucun. En outre l'uniformité garantit l'accord entre tous les processus : si un processus a décidé avant de tomber en panne, tous les processus corrects destinataires sont au courant de cette décision. Donc le scénario de blocage est impossible

Validation avec diffusion fiable uniforme (2)

Le protocole de diffusion fiable uniforme synchrone est appelé UTRB (*Uniform Timed Reliable Broadcast*)

Le protocole du coordinateur est inchangé (broadcast est maintenant la diffusion UTRB)

Le protocole des participants est inchangé, sauf pour le protocole de terminaison (panne du coordinateur) devenu inutile : on décide d'annuler car on sait qu'aucune décision contradictoire ne peut avoir été prise

Ö. Babaoğlu, S. Toueg, Non-blocking Atomic Commitment, in S. Mullender (ed.), *Distributed Systems* (2nd edition), Addison-Wesley, 1993

Participant

set-timeout (C_{know} + Δ_c + δ)
wait-for (receipt of VOTE_REQUEST from coord.)
  send (vote) to coordinator
  if (vote = NO) **then**
    decide (abort)
  else
    set-timeout (C_{know} + Δ_c + 2δ + Δ_b)
    wait-for(delivery of decision messages)
      if (decision mess. is abort) **then**
        decide (abort)
      else decide (commit)
    on-timeout
      decide (abort)
on-timeout
  decide (abort)

Validation avec diffusion fiable uniforme (3)

On peut démontrer (cf Babaoglu & Toueg) que le protocole de validation utilisant la diffusion fiable uniforme synchrone (UTRB) satisfait aux spécifications de la validation, y compris l'absence de blocage :

Tout processus correct décide

On peut en outre prévoir le cas de restauration d'un processus défaillant (protocole de reprise). On peut ainsi assurer que :

Tout participant au courant de la transaction décide, à condition de rester vivant "suffisamment longtemps" (borne connue) après sa reprise sur défaillance

Diffusion simple

Pour conclure l'étude de la validation, il reste à fournir une réalisation des protocoles de diffusion

- **diffusion simple**
- **diffusion fiable uniforme synchrone (UTRB)**

Diffusion simple dans un groupe G de processus (composition fixe)

émetteur :
send (m) to all processes in G
deliver (m)

destinataire (≠ émetteur) :
upon (receipt of m)
deliver (m)

Rappel : un message envoyé via send par un processus correct parvient à son destinataire (correct) en un temps $t < \delta$

Diffusion fiable uniforme synchrone (UTRB)

Un premier algorithme, simple mais peu efficace (nombre de messages ≅ n^2)

Diffusion fiable uniforme synchrone dans un groupe G de processus (composition fixe)

émetteur :
send (m) to all processes in G
deliver (m)

destinataire (≠ émetteur) :
upon (first receipt of m)
send (m) to all processes in G
deliver (m)

Rappel : un message envoyé via send par un processus correct parvient à son destinataire (correct) en un temps $t < \delta$

Un algorithme plus efficace sera présenté plus tard

Conclusion (provisoire) sur la validation

- **Importance**
  - **Pratique : algorithme de base pour la réalisation de transactions réparties**
  - **Théorique : un des deux problèmes fondamentaux de prise de décision concertée (l'autre est le consensus, à voir plus tard)**
- **Solutions**
  - **Nécessitent en pratique des hypothèses fortes sur la communication et le synchronisme**
  - **Protocoles de base : 2PC, 3PC**
- **Reste à voir**
  - **Relations entre validation et consensus**
    - **conséquences théoriques**
    - **conséquences pratiques**