Les problèmes fondamentaux des systèmes répartis tolérants les pannes

G. Florin – S. Natkin

Rappel des différentes étapes d'un mécanisme de tolérance (1)

Les principaux mécanismes de la tolérance reposent sur l'existence de **services redondants** dont les éléments peuvent tomber en panne.

a) Il faut tout d'abord détecter les pannes d'un ou plusieurs serveurs.

b) Si l'on peut formuler une hypothèse de panne transitoire et si les contraintes de l'application sont compatibles avec les techniques de recouvrement d'erreur (pas trop temps réel fortement contraint) il faut appliquer une technique de rééxécution.

c) Si les redondances temporelles sont insuffisantes il faut mettre en place des redondances spatiales.

=> décider de la suite des situations d'appartenance au groupe de serveurs redondants.

Les situations successives résultent:

- des pannes des membres
- des réinsertions de composants ou des adjonctions de serveurs nouveaux.

Rappel des différentes étapes d'un mécanisme de tolérance (2)

d) Il faut assurer des transmissions fiables vers des groupes de composants redondants (diffusion d'informations de chacun des clients vers le groupe de serveurs redondants implémentant un service).

Ces diffusions doivent être réalisées sous les différentes hypothèses de pannes et satisfaire des propriétés d'ordre.

e) Pour la redondance massive il faut rechercher un consensus sur une valeur calculée n fois (vote réparti).

f) Il faut éventuellement tolérer les pannes temporelles (satisfaction de contraintes temporelles pouvant être sévères) => l'ordonnancement temps réel réparti des tâches.

Point essentiel pour la détection des pannes temporelles et pour la satisfaction des contraintes temps réel réparti:

=> l'existence d'une **synchronisation des horloges** entre les différents sites.

Rappel des différentes étapes d'un mécanisme de tolérance (3)

g) Si la programmation de l'application organise des données réparties partagées sur différents sites il faut assurer le **contrôle de l'accès concurrent** aux données (maintien de la cohérence) pour des **données dupliquées**.

h) Si la programmation de l'application comporte encore des sites centraux (redondances sélectives) il faut prévoir la **défaillance de ces serveurs**.

D'ou une liste de problèmes types à résoudre pour une algorithmique répartie de la tolérance aux pannes

LA DÉTECTION DE COMPORTEMENT FAUTIF

Notion de **composant autotestable**:
un composant qui incorpore son propre logiciel de diagnostic => qui fait passer le plus vite possible un composant de l'état de **faute latente** à l'état de **faute détectée**.

Existence de très nombreuses techniques Quelques exemples

- Utilisant des **programmes de test**
  Détection **hors ligne** (diagnostics).
  Détection **en ligne** (détection continue).
  Chien de garde=>surveillance mutuelle.

- **Détection des erreurs de données**
  Codes détecteurs d'erreur.
  Assertions/Tests d'acceptance.
  Vote entre plusieurs alternants.

- **Détection des erreurs d'enchaînement**
  Protection (anneaux, domaines).
  Observation de points spécifiques de l'exécution => comparaison à un référentiel.
  Signature de séquences
  => comparaison à une tabulation des séquences valides.

REPRISE ARRIERE

Problème classique de l'univers réparti.

Pour une application coopérative faisant intervenir une architecture quelconque de clients et de serveurs (éventuellement redondés):

=> Comment déterminer des points de reprise "cohérents" à une fréquence optimale.

=> Comment assurer si nécessaire la journalisation des messages en transit sur les canaux de communications.

=> Comment effectuer la reprise arrière en cas de panne détectée.

Difficile pour obtenir des performances satisfaisantes

PROTOCOLE D'APPARTENANCE A UN GROUPE

Objectif : Assurer que tous les usagers ayant à connaître la situation d'un groupe de serveurs atteignent un consensus sur la composition du groupe.

- La perception de cette composition est relative à des **communications de groupes** (le protocole d'appartenance à un groupe est en général utilisé conjointement avec un protocole à diffusion).

- Dans un tel protocole on admet souvent que le temps est divisé en **époques** ou la composition du groupe est fixe et identique pour tout le monde.

- A l'intérieur d'une même époque les communications en diffusion atteignent la même liste de processeurs ou échouent ce qui peut advenir en période de changement de liste.

LA DIFFUSION ET LE CONSENSUS

Objectif : Ces deux problèmes précédents sont des variantes peu différentes consistant à faire s'accorder différents composants d'un groupe sur une valeur

- Valeur diffusée par un site.
- Valeur votée après un calcul en redondance massive.

- La valeur est utilisée comme **un signal** pour déclencher un traitement (valeur binaire)

Exemple du problème de validation dans la mise à jour des données ("commit")

- La valeur est utilisée comme **une donnée quelconque**.

Exemple du problème de redondance massive sur des données calculées comme des valeurs à envoyer sur des actionneurs.

LE PROTOCOLE DE SYNCHRONISATION D'HORLOGES

Objectif : Assurer que des horloges situées sur des sites distincts fournissent une datation absolue des événements avec une incertitude définie.

On distingue dans ce contexte deux sous-problèmes :

- Assurer que différents sites arrivent à **démarrer avec la même heure absolue** (au même moment avec une incertitude connue).

- Maintenir aussi longtemps que nécessaire **les différentes horloges** dans une variation relative connue.

. En contrôlant la dérive relative

Solutions par échange de messages.

Solutions par asservissement sur une horloge hertzienne.

LE PROTOCOLE D'ÉLECTION

Objectif : Lorsqu'une solution à un problème est basée sur l'existence d'un site **coordinateur unique** la panne du coordinateur doit être tolérée**.**

Cas des solutions en redondance sélective passive et sélective active.

Le protocole d'élection vise à désigner un et un seul coordinateur remplaçant.

CONCLUSION

Architecture de systèmes répartis en vue de la tolérance aux pannes

Nécessité de fournir deux catégories de services.

1 Les services standards utilisés dans des systèmes répartis: micro-noyaux, systèmes d'objets répartis

- Gestion des ressources (mémoire, processeur /ordonnancement..)
- Désignation, liaison
- Création des objets, migration,
- Réalisation des interactions de base
  (IPC, RPC légers/distants)
- Synchronisation

... etc

La sûreté n'est pas leur objectif principal

La sûreté/tolérance aux pannes de ces algorithmes est fondamentale car ils sont utilisés dans un système dont la sûreté doit être excellente.

2 Algorithmes utilisés dans les systèmes tolérants les pannes pour la tolérance.

Exemples vus:

Détection de panne
Reprises arrière
Élection
Diffusion fiable
Gestion des groupes
Vote réparti
Synchronisation d'horloges
Copies multiples
etc....

- La tolérance aux pannes des solutions présentées dans un système réparti pour la sûreté de fonctionnement est un problème essentiel.

- Certains de ces algorithmes doivent être étudiés pour tolérer tout type de panne.

LA SYNCHRONISATION DES HORLOGES

S. Natkin

NOTATIONS

t : temps absolu

Ei(n) le nème évènements perçu par le processeur i

ti(n) : la date de cet événement

Cj(t) valeur de l'horloge locale du processeur i

Wi(n)=Cj(ti(n)) : la datation de Ei(n) par le processeur j

LES PROBLEMES FONDAMENTAUX LIES AU TEMPS DANS LES SYSTEMES REPARTIS

1) LES HORLOGES LOGIQUES

Soit :

$$W = \bigcup_i Wi(n)$$

L'ensemble des évènements perçus par le système, Trouver un algorithme réparti tel que W soit totalement ordonné et que sur chaque site tous les évènements perçus soient ordonnés selon cet ordre total.

1) Conservation de l'ordre temporel

Trouver un algorithme réparti tel que :

$$\forall i \, \forall j \forall n \, \forall m : \; ti(n) < tj(m) \Rightarrow Wi(n) < Wj(m)$$

1) Conservation des intervalles

Trouver un algorithme réparti tel que :

$$\forall i \, \forall j \, \forall n \, \forall m \quad ti(n) - tj(m) = Wi(n) - Wj(m)$$

1) Synchronisation interne des horloges

Trouver un algorithme réparti tel que :

$$\forall i \quad Ci(t)=t+b$$

1) Synchronisation externe des horloges

Trouver un algorithme réparti tel que :

$$\forall i \quad Ci(t)= t$$

EXEMPLES D'APPLICATIONS

- **Datation en notarisation électronique**

- **Comparaison des horloges pour éviter des rejeux en protocole de sécurité**

- **Analyse de la causalité des incidents d'un processus physique**

- **Construction de make répartis**

- **Décision d'une action synchrone dans un système répartis (changement d'un plan de numérotation, de tables de routage…)**

- **Ordonnancement a contraintes stricte réparties**

- **Construction de systèmes répartis synchrones :**
  - **La synchronisation fournit un temps global**
  - **La diffusion une mémoire partagée**

RELATION ENTRE LES PROBLEMES

5⇒4⇔3⇒2⇒1

Aucun des problèmes 3,4 et 5 n'a de solution exacte Compte tenu des propriétés des mécanismes sur lesquels il est possible de s'appuyer :

- **Horloges matérielles**
- **Réseaux**
- **Primitives système**

LES HORLOGES MATERIELLES

Reposent sur une horloge physique à quartz pilotant un chien de garde sous interruption ou une horloge temps réel.

Notée HCi(t) sur le processeur i

Deux primitives essentielles :
--Initialisation
SET(in T :time, out OK :status)
-- Lecture
GET(out T :time, out OK :status)

PROPRIETES DES HORLOGES MATERIELLES CORRECTES

1. **HCi(t) est monotone croissante non bornée**

1. **Pour s suffisamment petit HCi(t)=HCi(t+s) (granularité), négligée dans la suite**

Donc HC est considérée comme strictement monotone croissante non bornée

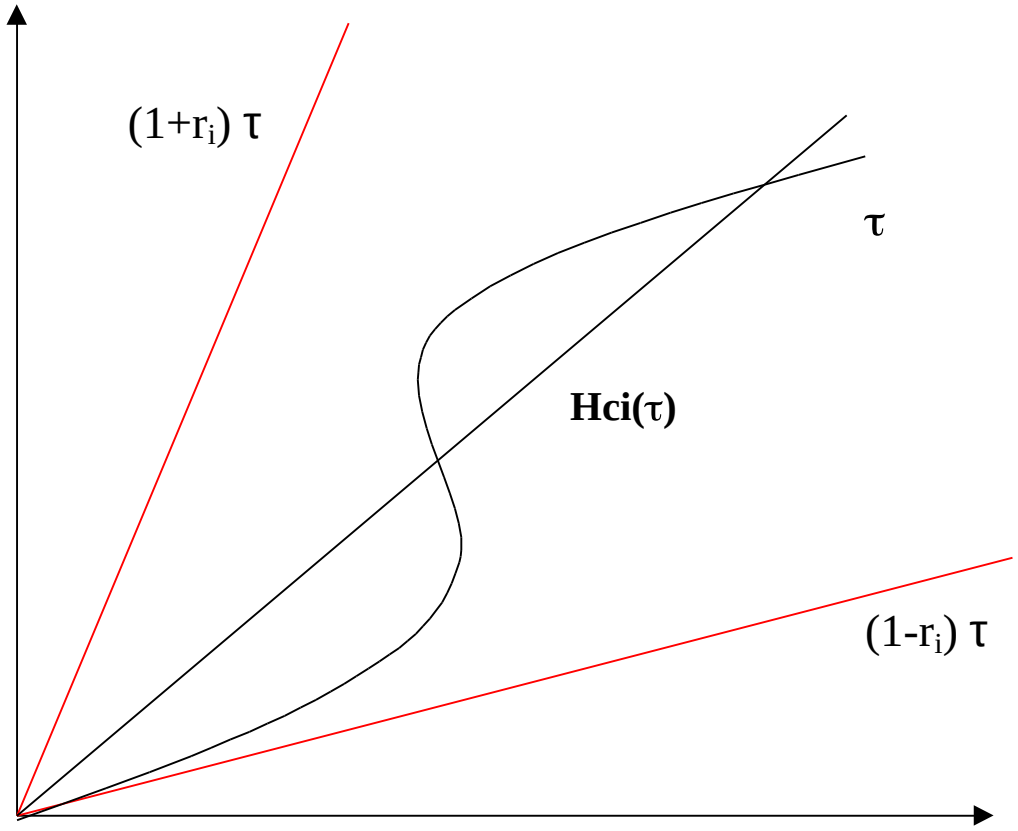
DERIVE DES HORLOGES CORRECTES

1. Il existe une constante réelle r telle que
$\forall i, \forall t \geq 0, \forall \tau \geq 0$

$$(1+r_i)\,\tau \leq HCi(t+\tau)-HCi(t) \leq (1-r_i)\tau$$

avec $r_i \leq r$

En pratique $10^{-4} < r < 10^{-9}$



SERVICE TRANSPORT DES MESSAGES

N processeurs
Trois primitives essentielles :

– Envoi a un
SEND(in destinator : Processor name ,in M : Message, out OK :status)

--Envoi à tous
BROADCAST(in M : Message, out OK :status)

--Réception
Receive (out sender : Processor name, M : Message, out OK :status)

PROPRIETES DU SERVICE DE TRANSPORT DES MESSAGES (1)

1. **Le réseau est logiquement fortement connecté : Si i et j sont deux processus non fautifs et que i envoie périodiquement M à j, au bout d'un temps fini et (presque sûrement) borné j recevra au moins une copie de M**

1. **Les seules pannes observables au niveau du service sont des pannes d'omissions ou temporelles (pas de pannes byzantines)**

Les pannes byzantines peuvent être couvertes par un codage détecteur d'erreur (pouvant aller jusqu'à la signature numérique des messages). Un message erroné est détruit par le service de transport du récepteur.

PROPRIETES DU SERVICE DE TRANSPORT DES MESSAGES (2)

1. **Un message émis par i pour j est correctement reçu si j le reçoit à l'instant t+s avec**

dmin≤s≤dmax (version déterministe)

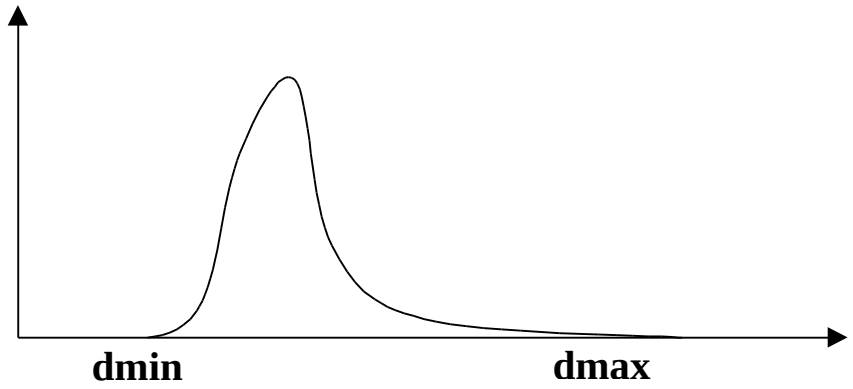
ou

Probabilité(dmin≤s≤dmax)≥1-ε

(version probabiliste)

dmin, dmax, ε sont des constantes données

Distribution de la durée de transmission des messages



PROPRIETES DU MATERIEL ET DU SYSTEME LOCAL

Le matériel et le système local peuvent subir toutes sortes de défaillances, qui vont amener à considérer que chaque processeur peut avoir dans l'exécution des algorithmes un comportement byzantin.

On suppose de plus qu'il existe une spécification définissant (de façon déterministe ou probabiliste) la durée minimale et maximale d'exécution d'une primitive système.

SPECIFICATION DU SERVICE D'HORLOGE SYNCHRONISEE INTERNE

Le service d'horloge fournit deux procédures:

--Lecture de Ci(t)

READ_CLOCK(out C : time, out OK : status)

-- Mesure sur le site i du temps écoulé depuis T, Di(T, D)

DURATION(in T :time, out D :time_interval, OK :status)

On cherche une solution approximative du problème 4 (synchronisation interne)

La solution du problème 5 est examinée plus loin

PROPRIETES DU SERVICE D'HORLOGES SYNCHRONISEES (1)

On considère l'ensemble des sites corrects

Trouver un algorithme qui détermine pour tout site correct Ci(t) tel que:

1) ∀i Ci(t) est une fonction positive monotone croissante non bornée de t

2) Deux horloges ont à peu près la même heure:
∀i, ∀j, ∀t, ∃B (précision de l'algorithme)

$$|Ci(t)-Cj(t)| \leq B$$

PROPRIETES DU SERVICE D'HORLOGES SYNCHRONISEES (2)

3) Toute horloge mesure approximativement le temps (Propriété d'enveloppe linéaire du temps)
$\forall i, \forall t, \forall \tau, \exists ri, \exists bi, \exists ki$

$$(1-ri)\tau \leq Di(t, t+\tau) \leq (1+ri)\tau + bi$$
$$\text{avec } |Di(t, t+\tau) - Ci(t+\tau) + Ci(t)| \leq ki$$

2 et 3 $\Rightarrow$ 4

4) Toutes les horloges mesurent le temps presque de la même façon:

$\forall i, \forall j, \forall t, \forall \tau, \exists R$ (Justesse de l'algorithme)

$$(1-R)\tau \leq Ci(t+\tau) - Cj(t) \leq (1+R)\tau$$

R peut être considéré comme la dérive du service d'horloges réparties

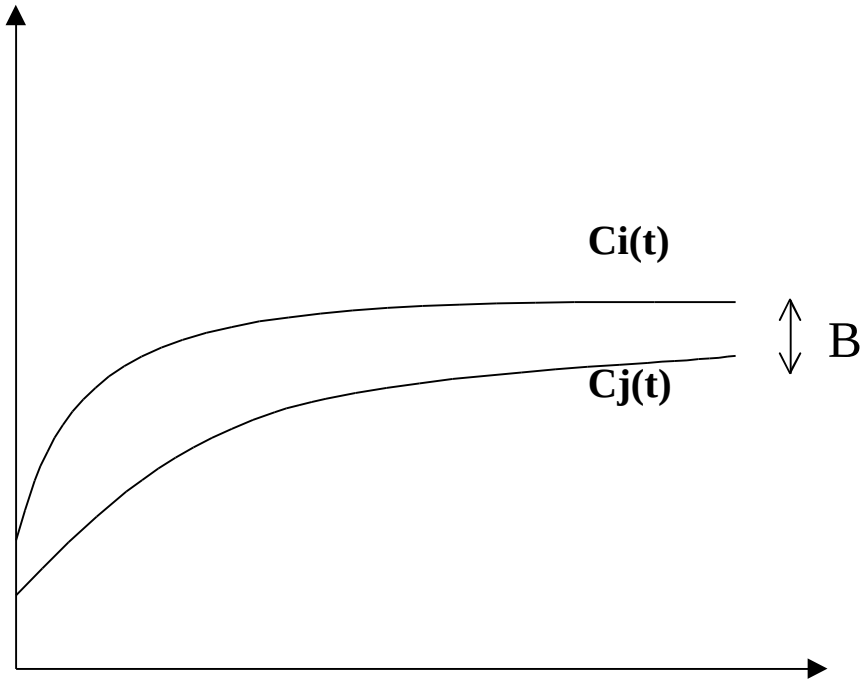
PARADOXE LOGARITHMIQUE

1 et 2 n'impliquent pas 3:

Par exemple soit un algorithme sans message :

Ci(t)= log(HCi(t))

Alors 1 et 2 sont vérifiées:



REFORMULATION DU PROBLEME 4

Soit un système réparti comportant N processeurs et tel que:

- **les processeurs peuvent subir des pannes byzantines**

- **Le service de transport des messages peut subir des pannes de performance mais le réseau reste logiquement connecté**

Trouver un algorithme réparti tel que:

Le service d'horloges synchronisées fournit sur tous les processeurs corrects les primitives READ_CLOCK et DURATION en vérifiant les propriétés 1,2 et 4 avec une précision B et une justesse R données

RESULTATS D'EXISTENCE

Pour des algorithmes déterministe on montre:

Soit f le nombre de processeurs byzantins

Pour f>N/3 et sans signature numérique des messages le problèmes n'a pas de solution

Pour f≤N/3 et sans signature numérique des messages le problèmes a une solution

Si un schéma d'authentification numérique fiable est utilisé le problème a toujours une solution

Dans toute solution la précision et la justesse doivent vérifier:

$$B<(dmax-dmin)/(1-1/N)$$

$$R \geq r$$

Il existe des solutions telle que R=r (optimale)

PRINCIPES GENERAUX DES SOLUTIONS

Les algorithmes reposent sur:

- **Le choix d'une horloge idéale sur laquelle toutes les horloges locales tendent à se recaler**

- **Le choix d'un schéma d'échange de messages: ou et comment les processeurs échangent leur opinion sur le temps**

- **La définition des points de synchronisation: Quand est ce qu'un processeur doit exécuter un traitement pour rester dans les contraintes de justesse et de précision**

- **Le choix d'un algorithme de convergence de l'horloge locale vers l'horloge idéale**

ALGORITHME GENERAL

Sur le processeur i

Initialiser : partir avec un bonne valeur de Ci(0)

Boucle

Sélectionner
Quand c'est le moment accepter
Envoyer(Ci(t))
Quand une heure Cj est reçue de j
Accepter Cj et contrôler sa validité
Quand arrive un point de synchronisation accepter

Calculer une nouvelle estimation de l'horloge idéale C(t)
Faire converger Ci(t) vers C(t)

Fin boucle

L'HORLOGE IDEALE

Pour résoudre le problème on impose à la solution d'être telle que:

L'horloge idéale vérifie la propriété d'enveloppe linéaire:

$$(1-R)\tau \leq C(t+\tau)-C(t) \leq (1+R)\tau$$

et

$\forall i, \forall t$

$$|Ci(t)-C(t)| \leq B/2$$

EXEMPLE D'HORLOGES IDEALES

1. **Le serveur de temps C(t)= Ci(t) pour un i donné (ou un groupe de serveurs eux mêmes synchronisés)**
Si le serveur est lui même synchronisé sur le temps universel, on obtient une solution approximative du problème 5 (Synchronisation externe)

1. **L'horloge la plus rapide**

$$C(t) = Si(Ci(t))$$

1. **La moyenne**

$$C(t) = Moy(Ci(t))$$

1. **La médiane**

$$C(t) = Med(Ci(t))$$

Ou tout autre algorithme (moindre carré, lissage exponentiel…).
S doit être un sous ensemble de processeur tel que la propriété d'enveloppe linéaire est vérifiée (corrects par exemple)

ECHANGE DES MESSAGES

Trois modes d'échange possibles:

Sur interrogation:
Quelle heure avez vous?
Il est midi

Spontanément (en diffusion)
Dormez bien brave gens il est minuit

Portés par d'autres messages (piggybacking)

POINTS DE SYNCHRONISATION

L'algorithme calcule Ci comme une fonction de l'horloge matérielle (et de les valeurs des messages qu'il a reçus jusqu'au dernier point de synchronisation qui ont déterminées son estimation de C(t)):

$$Ci(t)=f_{i,k}(HCi(t))$$

A certains instants $t_1,t_2,\ldots t_k,\ldots$déterminés soit par:

- **La réception d'un message**
- **La fin d'un délai**

le processeur i décide de changer de fonction et de passer à $f_{i,k+1}(HCi(t))$.

Deux possibilités:

- **Points synchrones périodiques**
- **Sur dépassement d'un seuil de la différence entre Ci(t) et l'estimation courante de C(t).**

LE SERVICE DE TEMPS DISCONTINU

A t_k on estime $C(t_k)$

On pose

$$Ci(t)= C(t_k)+HC(t)- HC(t_k)$$

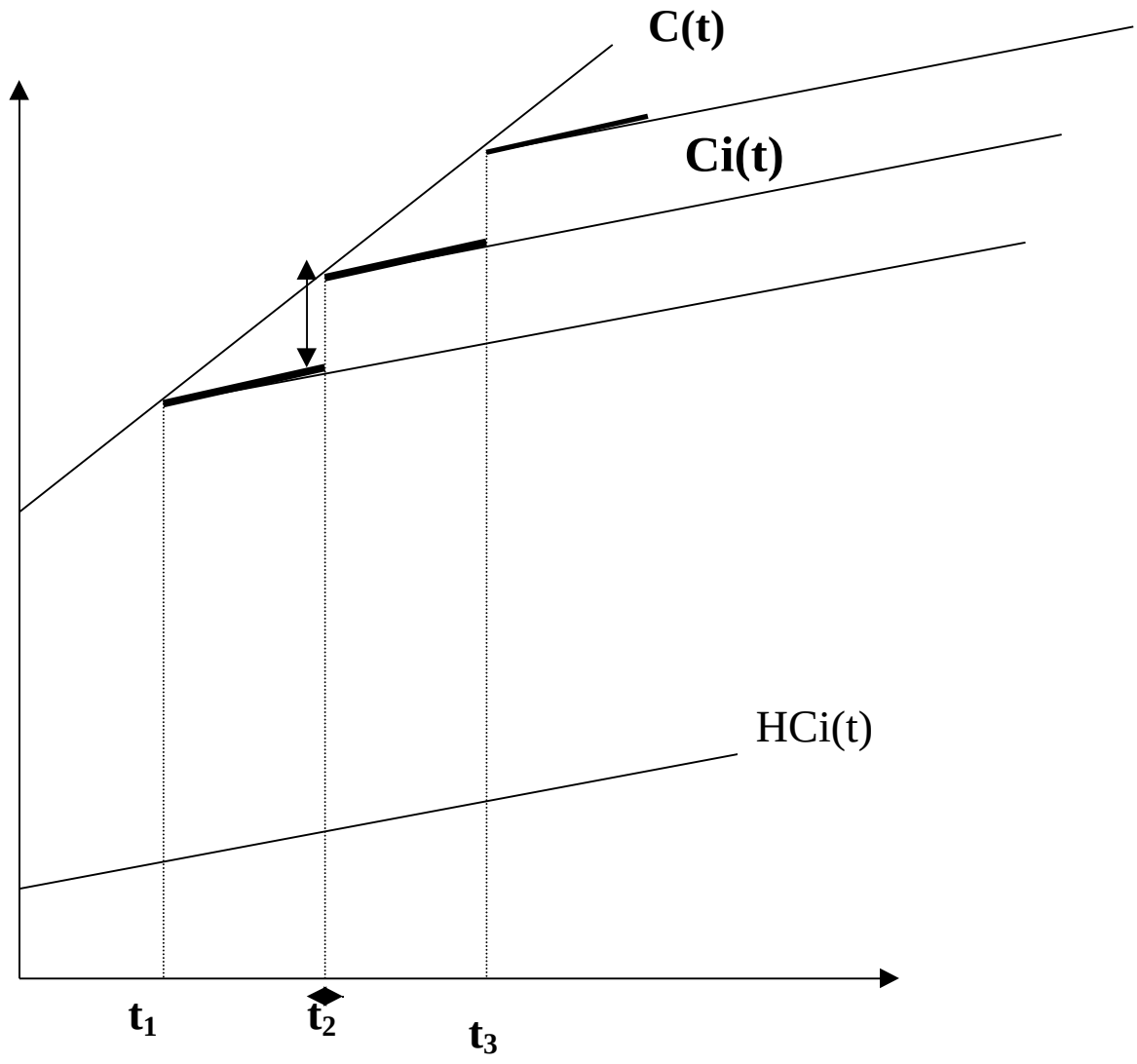
Pour assurer la propriété de croissance on doit avoir

$$C(t_k)\geq Ci(t_k)$$

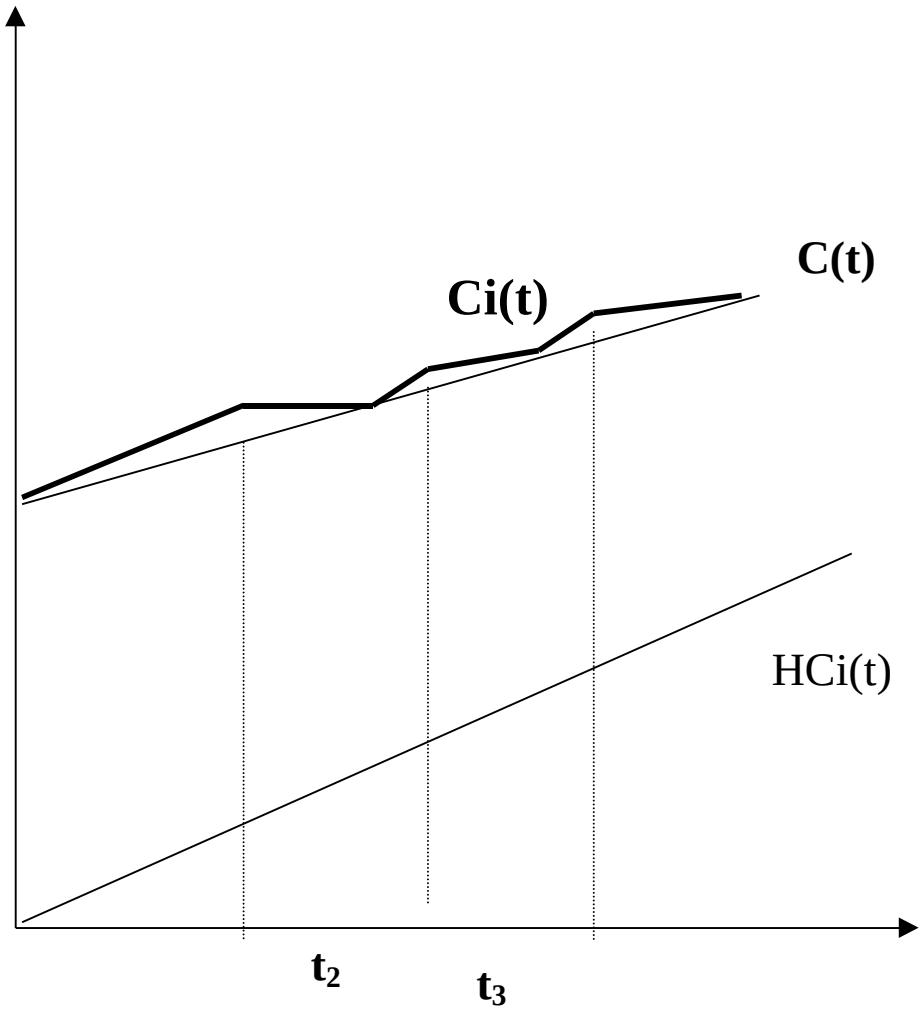
Donc on utilise en général pour l'horloge idéale la plus rapide

ERREUR DE MESURE DES INTERVALLES

Solution: mesurer les intervalles sur l'horloge de début de période (Dolev)



LE SERVICE CONTINU D'ORDRE 0



DETECTION DES FAUTES

Test des tolérances:

Quand i reçoit Cj(s) à t on doit avoir

$t-s \leq dmax$

Donc

$(1-R)\, dmax \leq |Ci(t)-Cj(s)| \leq (1+R)dmax$

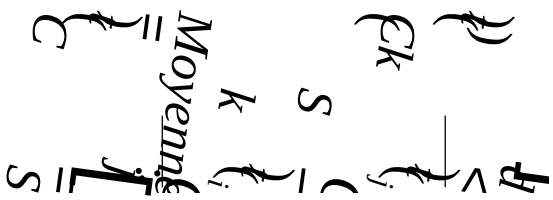
et
$|Ci(t)-Cj(s)| \leq B+dmax$

On peut affiner ces inégalités

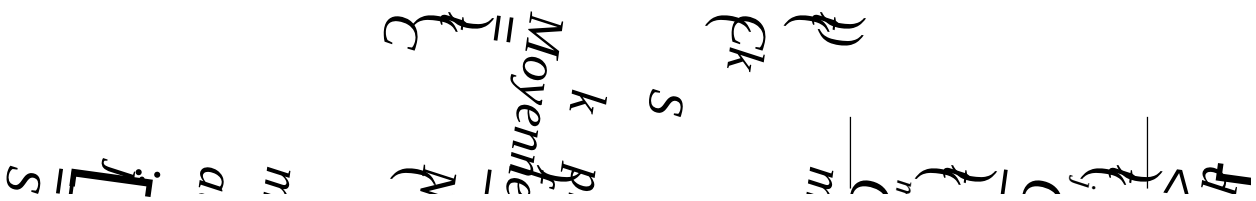
LES HORLOGES TOLERANT LES PANNES BYZANTINES

Horloges permettant de tolérer f processeurs byzantins f<N/3

Moyenne egocentrique



Moyenne à convergence rapide



Moyenne tolérant les pannes

$$C(t) = \mathit{Moyenne}_{k \in S}(C_k(t))$$

S = les f plus petites et plus grandes valeurs sont exclues

Médiane tolérant les pannes

$$C(t) = \mathit{Médiane}_{k \in S}(C_k(t))$$

S = les f plus petites et plus grandes valeurs sont exclues

LES PROBLÈMES

DE DIFFUSION

ET DE CONSENSUS

DANS LES SYSTÈMES

REPARTIS

G. Florin

INTRODUCTION

Nombreux articles consacrés à des classes de problèmes voisins

- Dénominations diverses.

- Hypothèses parfois difficiles à cerner.

  - Diffusion "**fiable**", "**atomique**"
    "Reliable" , "Atomic" Broadcast
  - **Généraux byzantins**
    "Byzantine Generals"
  - **Consensus** (byzantin)
    "Agreement" (Byzantine agreement)
  - **Accord mutuel**
  - **Consensus approché**
  - **Consistance interactive**
    "Interactive consistency"
  - **Vote réparti**

Problèmes connexes

- Protocole de **Validation** atomique
  "ACP Atomic Commitment Protocol"
- Protocole d'appartenance à un groupe
  "GMP Group Memberhip Protocol"

1 LE PROBLÈME DE BASE : LA DIFFUSION

Un message **émis** doit être **reçu par n destinataires**.

- Problème abordé dans les **réseaux généraux** ("IGMP").

=> **En développement significatif**.

- Problème abordé dans les **systèmes répartis (**Chorus, Amoeba, Isis/Horus) (surtout pour des réseaux locaux).

Le protocole à diffusion est l'un des éléments de base du noyau.

Nombreux critères de conception

Critères **fonctionnels**

. gestion des connexions,
. gestion des groupes de diffusion,
. propriétés d'ordres...

Critères de **tolérance aux pannes**

des hypothèses les plus simples (ethernet) jusqu'aux pannes quelconques.

Critères de **performance**

Débit, temps de réponse, variation du délai, ...

LES GÉNÉRAUX BYZANTINS

- Une armée est composée de **corps** d'armée **séparés** et doit livrer bataille.

- Cette armée dispose d'une supériorité lui permettant de vaincre si **tous** les corps d'armée attaquent ensemble.

- Si l'attaque est réalisée en ordre dispersé la défaite est probable: **il vaut mieux ne rien faire** qu'attaquer en ordre dispersé (notion **d'état de sécurité**).

- Le général en chef décide de l'heure de l'attaque en envoyant des messagers qui circulent sur le champ de bataille.

Les généraux de corps d'armée peuvent aussi utiliser des messagers.

- Les messagers peuvent être pris (**messages perdus**).

Le **contenu** des messages peut-être **altéré** pour désorganiser l'attaque.

- Certains généraux peuvent trahir selon des modes quelconques (**comportement byzantin**).

LE PROBLÈME DES GÉNÉRAUX BYZANTINS: ÉNONCÉ

Peut-on **trouver:**

sous les hypothéses précédentes

une solution au problème du commandement

permettant de **vaincre** ou de **rester en sécurité.**

En fait **problème de diffusion sous hypothèse de pannes quelconques**.

Une valeur de décision particulière (**valeur par défaut**) correspond au maintien en état de sécurité.

2 LE PROBLÈME DU CONSENSUS VOTE REPARTI

Un ensemble de processus Pi calculent
la **valeur** d'une variable **Vi**
à partir de données d'entrée **Ei**.
Vi = f (Ei)

Hypothèses de pannes

Les **sites** de calcul tombent en **panne**.
Les données d'entrée sont **erronées**.
Les calculs sont **faux**.
L'**identité** des fautifs **est inconnue**.

Un algorithme de **consensus** est tel que par un **échange des valeurs** calculées Vi **tous les processus corrects décident de la même valeur V**.

Exemple type

Processus redondants => Vote réparti
V = majorité (Vi)
ou V = Vdef valeur par défaut.

Problème de validation ("commit")
Un ensemble de sites doit décider d'une modification d'ensemble ou du maintien d'une situation de cohérence précédente.

Réduction réciproque des problèmes de diffusion, de consensus, de validation

Problème posé (équivalence, réduction):

Trouver un **algorithme simple** (peu complexe) qui fournit une solution à un problème lorsque l'on utilise une solution existante pour un autre problème.

Idée de base pour la diffusion et le consensus

- Si l'on dispose d'un consensus
  tolérant un type de pannes donné

On effectue une **diffusion** de base et on applique un algorithme de **consensus** entre les destinataires.

- Si l'on dispose d'une diffusion
  tolérant un type de pannes donné
  possédant des propriétés d'ordre

Tous les processus **diffusent** leur valeur à tous les autres qui appliquent un processus de décision identique sur toutes les valeurs reçues => **consensus**.

Propriétés de réduction établies

Réduction démontrée dans le cas de la **diffusion atomique** (diffusion ordonnée) sous hypothèse de pannes franches et du **consensus** sous hypothèse de pannes franches (Chandra et Toueg)

Réduction de la **validation atomique** au problème de **consensus** pour des pannes franches (Raynal)

A construire pour les autres cas (au cas par cas).

Pas forcément utilisable car les solutions obtenues ne seront pas **optimales**.

GÉNÉRALITÉS

PROPRIÉTÉS DES PROTOCOLES A DIFFUSION

1 INTRODUCTION A LA GESTION DES GROUPES

Notion de groupe

Rassemblement d'entités ayant une finalité commune (type ensemble):

- groupes de **processus,**
- groupes d'**objets**,
- groupes **de portes**

Exemple d'utilisation des groupes

=> L'ensemble des destinataires d'une diffusion.
L'ensemble des votants dans un protocole de consensus.

Exemple: Emission vers les files d'attentes d'un groupe de portes (Chorus)

=> L'ensemble des processus impliqués dans un comportement de groupe.
Election, point de reprise, etc ...

Protocole d'appartenance à un groupe ou protocole de gestion de groupe ("Group membership")

Un aspect important des comportements de groupe limité aux problèmes de composition, de désignation et de liaison

- La composition du groupe

Qui est présent et qui est absent?
Suivi des arrivées et des départs.
Très relié à la détection de pannes.

- La désignation

La stucture, l'attribution, le stockage et la restitution des identifiants uniques de groupe.

- La liaison

La localisation des membres en fonction de l'identifiant et de l'appartenance..

Situation de la gestion de groupe

- En association avec **le protocole de détection de comportement fautif** si la prise en compte des pannes est l'élément essentiel de la modification du groupe.

- En association avec **le protocole à diffusion** qui est le principal utilisateur de la notion de groupe et souvent le principal informateur sur les comportements fautifs.

- **Service et protocole de gestion des groupes**: une entité en soi qui demande des services et fournit des services.

Exemple de hiérarchie selon F Christian pour un noyau de système sûr

Gestion de groupe

Diffusion de base

Synchronisation d'horloge.

Niveaux de connaissance des groupes

Groupe **indéfini**

. Il existe un identifiant de groupe

. Il n'existe pas de liste d'appartenance complète instanciée sur un site.

. Les messages visitent les différents sites qui déterminent localement s'il y a ou non des récepteurs.

Exemple: ethernet

Groupe **défini**

. Le protocole de gestion de groupe gère une liste complète des membres du groupe instanciée sur un ou plusieurs sites.

Modes de composition des groupes

Groupe **statique**

. **Entre deux événements** d'ouverture et de fermeture d'une connexion **la composition du groupe est figée**.

Groupe **dynamique**

. Les processus peuvent **rejoindre ou quitter le groupe** à tout moment (sur pannes, réparations ou selon les besoins).

Groupe **invoqué**

. **L'ensemble prédéfini** des processus qui peuvent **rejoindre un groupe dynamique** (mais fermé).

Groupe **actif**

. **Les processus opérationnels** (pouvant participer aux diffusions) à un instant donné.

Problème de cohérence entre les opérations de diffusion et les modifications de composition du groupe.

Groupes d'émetteurs et de récepteurs

MODE DIFFUSION "CENTRALISÉE"

Un seul émetteur (toujours le même) et un groupe destinataire (toujours le même).

MODE CENTRALISÉ A CENTRE MOBILE

L'émetteur est **unique par période**.

MODE MULTI-CENTRE

p émetteurs peuvent à tout instant diffuser **vers un groupe de récepteurs.**

MODE DÉCENTRALISÉ
OU MODE CONVERSATION ("Broadcast")

Les membres d'un groupe peuvent être à tout instant **émetteurs** et sont **tous destinataires** des messages.

MODE DIFFUSION SUR GROUPES ("Multicast")

. Les membres d'un **ensemble de groupes** peuvent être à tout instant **émetteurs vers l'un des groupes** existants.

Diffusions sur groupes: cas particuliers

MODE DIFFUSION "A SOI MÊME"

L'émetteur est par définition **membre de tous les groupes** de diffusion.

Version "ordres"

Toutes les diffusions d'un même émetteur sont réordonnées comme si l'émetteur était destinataire.

MODE DIFFUSION "ARCHIVÉE"

L'architecture des groupes est telle qu'**un site particulier appartient à tous les groupes** de diffusion.

Version "ordres"

Toutes les diffusions sont réordonnées sur un même site.

2 PROPRIÉTÉS TEMPORELLES

Protocole à délai borné (synchrone)

Les messages sont délivrés avant un certain délai d_{max}.

Protocole à délai non borné (asynchrone)

Quelquesoit d_{max} la probabilité pour un message diffusé d'arriver dans un délai supérieur à d_{max} est non nulle.

Simulation d'un protocole à délai borné

Il existe une synchronisation d'horloge à δ près entre sites émetteurs et récepteurs.

- Les messages émis à t_e et arrivés à t_r tels que $t_r - t_e \geq d_{max} + \delta$ sont ignorés et ne sont pas acquittés.
- Les messages tels que $t_r - t_e < d_{max} + \delta$ sont délivrés et acquittés.

Z On crée des pertes de messages.
Problème des messages non délivrés
Problèmes d'un taux de rejet trop élevé

3 PROPRIÉTÉS D'ORDRE

Problème à résoudre: réaliser des délivrances ordonnées

Deux messages M et M' sont en relation de délivrance ordonnée s'ils sont délivrés dans le même ordre à tous les destinataires communs.

$$M \rightarrow_{do} M' \Leftrightarrow \quad i \quad G(M) \cap G(M')$$
$$i.recevoir(M) \rightarrow i.recevoir(M')$$

G(M), G(M') groupes destinataires

Diffusion locale

- Pour deux diffusions successives du même processus, les messages sont délivrés dans le même ordre sur tous les sites distants.

$$M \rightarrow_{l} M' \Leftrightarrow i.envoyer(M) \rightarrow i.envoyer(M')$$
$$M, M' \quad M \rightarrow_{l} M' \qquad M \rightarrow_{do} M'$$

Diffusion causale

- Toute diffusion de messages en relation de causalité (émission(M) précède causalement émission(M')) implique la délivrance des messages sur tous les sites destinataires dans la même relation (recevoir(M) précède recevoir(M')).

Relation causale directe entre messages

Un message M est délivré sur le site émetteur de M' avant que M' ne soit émis.

$$M \rightarrow_{cd} M' \Leftrightarrow i.recevoir(M) \rightarrow i.envoyer(M')$$

Relation causale entre messages

Fermeture transitive des relations locales et causales directes.

$$M \rightarrow_{c} M' \Leftrightarrow M_0 = M, \ldots M_q, \ldots M_{m+1} = M'$$

Diffusion causale

$$M, M' \quad M \rightarrow_{c} M' \qquad M \rightarrow_{do} M'$$

Diffusion totale

Pour tous les messages il existe un ordre de délivrance aux applications réceptrices.

$$M, M' \quad M \rightarrow_{do} M' \qquad M' \rightarrow_{do} M$$

Diffusion totale numérotable

Pour tous les messages il existe un ordre de délivrance qui correspond à un numéro unique des messages.

$$M, M' \quad n(M), n(M')$$

$$n(M) < n(M') \Leftrightarrow M \rightarrow_{do} M'$$

Diffusion totale et causale

Pour tous les messages qui sont en relation causale l'ordre causal celui-ci est respecté lors des délivrances. Pour tous les messages indépendants causalement un ordre de délivrance identique sur tous les destinataires est appliqué.

$$M, M' \quad M \rightarrow_c M' \qquad M \rightarrow_{do} M'$$

$$M, M' \quad \neg(M \rightarrow_c M') \qquad \neg(M' \rightarrow_c M)$$

$$M \rightarrow_{do} M' \qquad M' \rightarrow_{do} M$$

Quelques propriétés

Les deux propriétés de délivrance causale et totale sont différentes.

On a

¬ (**ordre total => ordre causal)**

Un ordre arbitraire strict n'a pas de raison de respecter la causalité.

¬ (**ordre causal => ordre total)**

L'ordre causal n'est que partiel et ne permet pas de relier tous les événements.

On peut montrer par contre que pour une diffusion en conversation ("broadcast")

ordre total causal <=>

ordre total + ordre local

(Ordre total qui respecte l'ordre local)

4 PROPRIÉTÉS DE TOLÉRANCE AUX PANNES

Analyse des catégories de diffusion précédentes en relation avec les différentes pannes des processeurs ou du réseau:

- panne franche
- panne d'omission
- panne temporelle
- panne quelconque (ou byzantine)

Une combinatoire très élevée.

Le problèmes de diffusion ou de consensus: le mieux étudié du point de vue des modes de pannes.

=> Deux grandes catégories de solutions.

- Solutions avec **tolérance aux pannes** et/ou **compte-rendu** des erreurs détectées (à l'émetteur, aux membres du groupe).

Aucune tentative visant à annuler l'effet rémanent des échanges réussis n'est entrepris si certains sites n'ont pas bien reçu les informations.

- Solutions "**atomiques**" **"tout ou rien"**

Tous les membres (ou tous les membres corrects) reçoivent ou personne ne reçoit.

Diffusion de base
(sans contrôle d'erreur)

Un message est envoyé sans garantie de livraison par un processus aux membres d'un groupe de diffusion.

Exemple: Réseau local Ethernet

Mode diffusion sur groupes "multicast":

- Aucune garantie n'est fournie sur la qualité de la transmission.

Le message peut-être non délivré à tous ou à un sous-ensemble des destinataires.

- Pratiquement pas de borne utilisable pour le temps de transmission.

Exemple: Le protocole à diffusion du système réparti Chorus.

- Une diffusion simple sur ethernet est jugée suffisante pour les applications envisagées (localisation de portes ...).
- Le taux d'erreur d'ethernet est jugé suffisamment faible.

Diffusion avec tolérance aux pannes d'omission de messages

Redondance temporelle

- Pour tolérer les pertes de message on effectue en général n tentatives en attente d'un acquittement positif .

Compte-rendu d'erreur

- En cas d'échec persistant on peut retourner un compte-rendu à l'émetteur.

Relation avec le détecteur de pannes

A priori deux protocoles différents.

Hypothèse souvent considérée: un site que l'on ne réussit pas à atteindre est déclaré en panne.

=> **Possibilité de décision erronée**

Remarques:

- Certains sites reçoivent d'autres non.

- On devrait plutôt parler dans ce cas de diffusion "**fiable**"

Exemple: protocole à diffusion du réseau local industriel ARLIC (Sema)

Objectif : Résoudre le problème de diffusion pour des applications en réseau local industriel.

- Protocole **unidirectionnel centralisé**.
- **Détection d'erreurs** (et contrôle de flux par acquittement positif immédiat) de chaque processeur membre du groupe des destinataires.
- **Compte-rendu d'erreurs** selon les besoins des utilisateurs.

Récepteurs type 1 :

Attachement indispensable
=> La connexion est rompue si le récepteur ne peut-être atteint.

Récepteurs type 2 :

Attachement non indispensable
=> Signalisation d'erreur

Récepteurs type 3 :

Aucune signalisation tant qu'au moins un récepteur membre du groupe est actif.

Diffusion fiable
Hypothèses de pannes franches de sites

Notion d'atomicité sur groupe statique (tout ou rien).

Tout message envoyé par un processus aux **membres d'un groupe statique** de diffusion est délivré **à tous les processus ou à aucun d'entre eux**.

Remarques

- Tous les destinataires membres du groupe doivent être atteints.
- Si des sites sont en panne il faut mémoriser les messages de façon suffisamment stable (fichiers journaux) pour que la diffusion aboutisse plus tard.

Utilisation

Un ensemble de processus redondants effectuent tous les mêmes travaux en présence de pannes.

Diffusion fiable
(diffusion atomique sur le sous-groupe actif d'un groupe)

On appelle le plus souvent diffusion fiable **une diffusion sous hypothèse de pannes franches** telle que **tout message est reçu par l'ensemble des processus actifs**.

Diffusion atomique
(diffusion ordonnée totalement)

Dans de très nombreuses publications on appelle diffusion atomique **une diffusion ordonnée totalement** (avec éventuellement un critère de fiabilIté)

Les messages sont reçus dans le même ordre par l'ensemble des processus actifs ou par aucun.

Application: Cohérence séquentielle de données dupliquées sur n sites au moyen d'une diffusion => toutes les écritures sont exécutées dans le même ordre.

Notion d'atomicité transactionnelle

Une transaction est un ensemble d'opérations (de lecture et d'écriture) adressée à un ensemble de données partitionnées, partiellement ou totalement dupliquées

En cas de duplication les diffusions doivent posséder les propriétés suivantes:

- Tout les messages de la suite **sont reçus par tous les sites serveurs** de données ou par aucun.
- Les messages sont **reçus dans le même ordre**.

Diffusion tolérant les pannes temporelles

Diffusion atomique sous hypothèse de pannes temporelles

Un message émis à la date t doit être délivré à tous les destinataires ou a aucun à la date $t+d_{max}$.

Utilisation : systèmes temps réels pour lesquels les informations pertinentes ne peuvent pas "vieillir". Il est préférable de refaire la mesure.

Peu études s'adressaient spécifiquement ce problème.

F Christian propose une solution dans un réseau maillé.

Regain d'intérêt avec les réseaux haut débit pour le multimédia.

Diffusion tolérant les pannes quelconques (byzantines)

Définition de l'atomicité sous hypothèse de faute byzantine.

a) Unanimité (accord)

Tout message diffusé par **un processus fautif ou non fautif** à tous les processus d'une liste de diffusion est **délivré à tous les processus non fautifs** ou **à aucuns d'entr'eux** (v_{def} est une valeur par défaut prise en cas d'échec).

b) Solution non triviale (validité)

Pour éviter des solutions ou l'on décide toujours la même valeur on rajoute des hypothèses sur la valeur de décision.

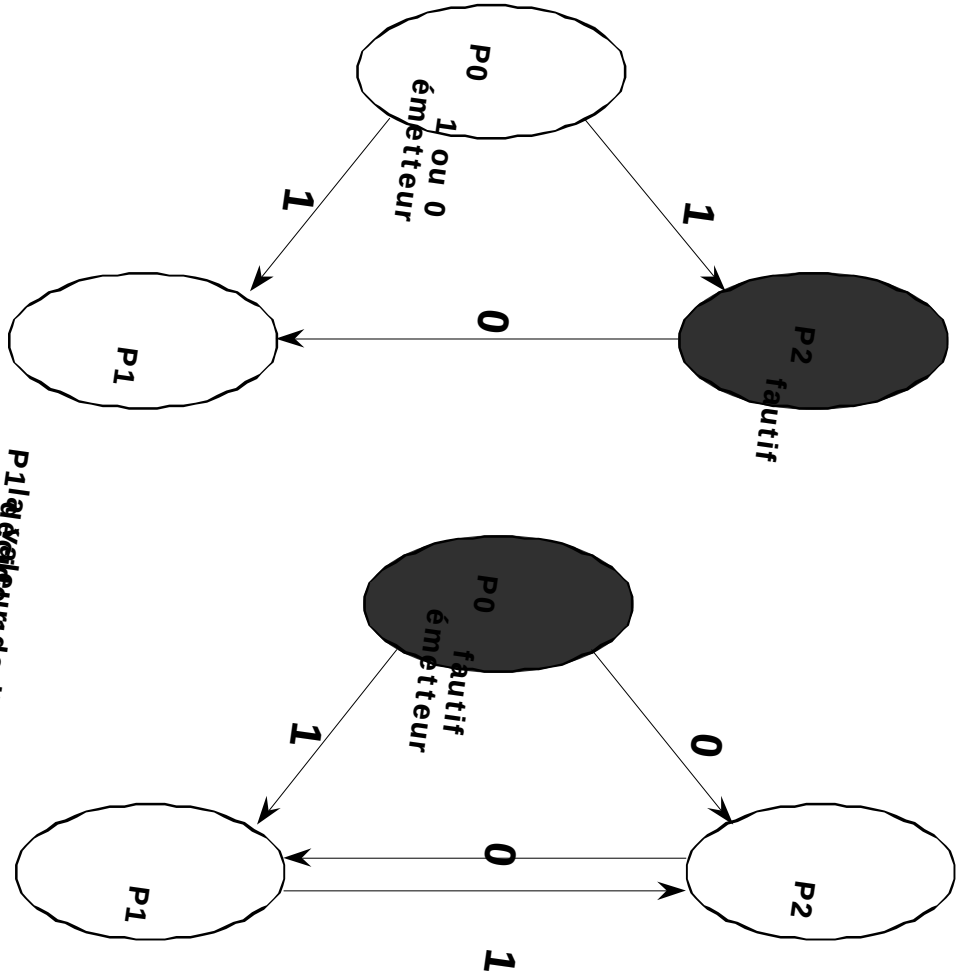
Si l'**émetteur est non fautif** la valeur du message faisant l'unanimité est **celle transmise par l'émetteur**.

Résultat d'impossibilité
(sous hypothèse de panne byzantine)

- n processus - t fautifs
existence d'une solution si n > 3t

Contre exemple

Impossibilité avec 3 sites dont un fautif.



IC1: Tous les sites corrects adoptent la même valeur
IC2: Si l'émetteur est correct la valeur adoptée est celle de l'émetteur.

LE PROBLÈME DU CONSENSUS

N processus Pi disposent chacun d'une valeur Vi. Un algorithme de consensus possède les propriétés suivantes:

Terminaison ("Termination")

Si tous les processus corrects initialisent un consensus (chacun proposant sa propre valeur Vi) alors chaque processus correct inévitablement décide une valeur.

Intégrité ("Integrity")

Un processus correct décide une seule fois.

Consistance ("Agreement")

Tous les processus corrects décident de la même valeur commune.

Solution non triviale ("Validity")

La valeur d'accord doit être corrélée aux valeurs Vi d'où différentes hypothèses complémentaires nécessaires.

Différentes hypothèses de validité

Consensus d'unanimité

Si tous les processus corrects avaient au départ la même valeur alors l'algorithme de consensus doit décider de cette valeur.

Difficulté: Si tous les processus corrects obtiennent le même résultat la définition est consistante.

Z Pour un problème ou les processus corrects peuvent produire des valeurs différentes s'il y a une seule divergence on peut décider n'importe quoi.

Consensus majoritaire

La valeur décidée est la majorité (selon un règle majoritaire à préciser) des valeurs détenues au départ par les processus corrects.

Z Difficile à réaliser lorsque de nombreux processus sont en pannes byzantines

=> ils peuvent faire échouer une prise de décision majoritaire par les corrects.

Consensus non nécessairement majoritaire

La valeur décidée est:

- La majorité (des valeurs détenues au départ par les processus corrects) pour un nombre de pannes inférieur à un seuil.

- Une valeur détenue au départ éventuellement non majoritaire sinon. (Bracha et Toueg).

Consensus de base

La valeur décidée est l'une des valeurs proposée par l'un des processus corrects.

Consensus uniforme

Tous les processus qui décident une valeur décident la même valeur (qu'ils soient corrects ou incorrects).

Hypothèse plutôt associée aux modes de pannes franches.

LE PROBLÈME DU CONSENSUS APPROCHÉ

Les processus Pi disposent d'une valeur Vi résultant d'un calcul non nécessairement exact mais l'incertitude sur le calcul est bornée par ε.

Exemple: calculs en temps réel sur des données échantillonnées à des instants légèrement différents ou calculs numériques itératifs arrêtés sur horloge à des points de convergence différents.

Unanimité approchée

Tous les processus corrects s'accordent sur des valeurs qui ne différent pas de plus de ε.

Validité approchée

- Multivalence : toutes les valeurs peuvent être atteintes.

- Décision consensuelle approchée : si toutes les valeurs de départ sont égales à ε près deux à deux la valeur d'accord correspond au résultat d'un calcul correct sur une valeur de départ.

Quelques résultats de possibilité sous hypothèse de panne franche

Fisher Lynch Paterson

On ne peut pas trouver de solution au problème du consensus sous hypothèse de panne franche pour un modèle de système asynchrone avec un seul fautif.

Chandra Toueg

1 Si une majorité de processus sont corrects alors on peut résoudre le problème du consensus avec un détecteur de pannes $\diamond W$
(il existe une date au delà de laquelle l'un des processus correct n'est jamais suspecté avant de tomber en panne franche, inévitablement tout processus en panne franche est suspecté de manière permanente par un processus correct).

2 Si un seul processus est correct alors on peut résoudre le problème du consensus avec un détecteur de pannes W
(l'un des processus correct n'est jamais suspecté avant de tomber en panne franche, inévitablement tout processus en panne franche est suspecté de manière permanente par un processus correct).

TRANSACTIONNEL

RÉPARTI

G. Florin

INTRODUCTION

Système d'objets partagés répartis

Ensemble d'objets à accéder à distance
(ou localement) par plusieurs applications.

Enregistrements de fichiers répartis.
Données d'une base de données répartie.
Variables d'instance d'objets répartis.

Actions élémentaires

Opérations sur les objets considérées ici

lire (x , val)
écrire (x , val)

Autres opérations possibles

création , destruction des objets,
opérations complexes.

Contraintes d'intégrité

L'ensemble des relations que doivent vérifier les objets.

État cohérent (du système d'objets)

Un état du système ou les contraintes d'intégrité sont satisfaites.

Transaction

Application qui exécute des actions (notées tlire, técrire) sur les objets partagés

Exemple : Transaction T

```
t_début
    tlire(x);
    técrire (x, val1);
    técrire (y, val2);
t_fin;
```

Propriétés ACID (Atomicité, Consistance, Isolation, Durabilité)

Deux présentations

- Une formulation très générale définie pour un ensemble quelconque d'opérations.

- Une formulation spécifique dans le cas des transactions.

Atomicité ("atomicity")

Forme générale

T**outes les opérations** associées à une **action "atomique"** sont réalisées **complètement** ou **aucune d'entr'elles** n'est exécutée.

Cas du transactionnel

. Ou bien l'exécution d'une transaction est totale.

Toutes les actions lire et écrire d'une transaction sont effectuées et amènent la base jusqu'à un nouvel état.

. Ou bien l'exécution n'a aucun effet sur les objets.

Toutes les modifications partielles engagées sont annulées et l'on reste sur l'ancienne version.

Z Origine des difficultés:

- les accès concurrents
- les pannes.

Consistance ("consistency")

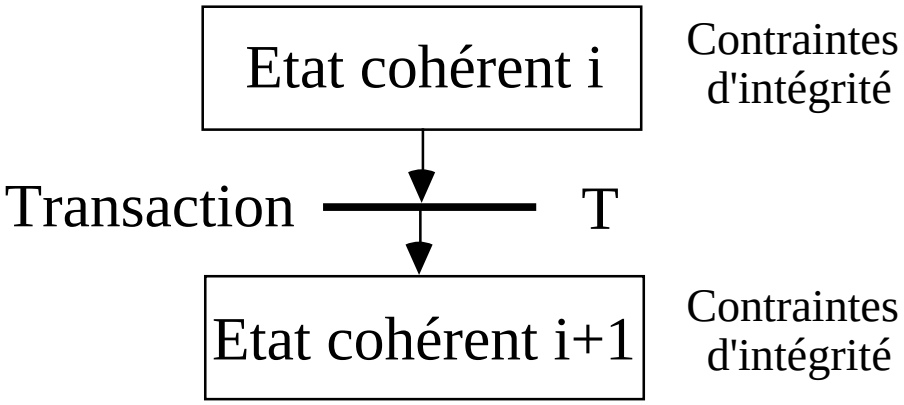
Forme générale

Un ensemble d'opérations d'accès vérifient certaines propriétés (exemple consistance séquentielle d'une mémoire).

Cas du transactionnel

Une transaction est consistante lorsque exécutée seule elle amène un ensemble d'objets d'un état cohérent à un autre état cohérent (consistante avec la sémantique de cohérence d'un ensemble d'objets).

L'objectif primordial est de maintenir la cohérence de la base d'objets.



Isolation ("isolation")

Forme générale

Un utilisateur ne peut avoir accès à aucun résultat intermédiaire d'une action atomique avant la terminaison de la totalité de la transaction.

Cas du transactionnel

L'isolation est la propriété qui permet à un ensemble de transactions s'exécutant simultanément (en concurrence d'accès aux données partagées) d'apparaître comme s'exécutant de façon indépendante les unes des autres (sérialisable).

On ne peut accéder à des valeurs intermédiaires d'une transaction.

C'est le travail de contrôle de concurrence proprement dit.

Durabilité ("durability")

Forme générale

C'est la propriété d'un système de garantir que les effets d'un ensemble d'opérations ne sont altérés par aucune sorte de pannes.

Cas du transactionnel

Les effets des opérations décidées définitivement doivent être durables même en cas de défaillances (de processeurs ou de communications).

Autre terminologie très fréquente:

La persistance

(les objets accédés de façon concurrente sont persistants).

Les trois fonctions principales du transactionnel distribué

Contrôle de concurrence en réparti ("concurrency control")

L'ensemble des **techniques** qui permettent de **synchroniser** les activités parallèles de plusieurs **transactions**,

qui accèdent à **des objets partagés sur des sites différents** et par suite interfèrent les unes avec les autres.

Ils assurent la propriété **d'isolation**.

Reprises arrières en réparti ("backward recovery")

Les mécanismes systèmes de sauvegarde et de reprise sur pannes qui assurent que les pannes ne peuvent corrompre les données

Ils assurent l'**atomicité** et la **durabilité** (**la persistance)** des données.

La gestion des performances

Fournir les mécanismes systèmes qui permettent d'accepter le plus grand nombre de transactions possible par unité de temps sans que le temps de réponse de chaque requête soit trop dégradé:

- ordonnancer les transactions
  (priorités, ...)

- créer, administrer les processus serveurs de transactions (équilibrage de charge, partage de charge)

En résumé

Le moniteur transactionnel réparti est la partie **du système d'exploitation réseau** pour le traitement d'unités de travail de type transactionnel.

Il réalise:
- des fonctions de synchronisation
- des fonction de gestion de ressources
- des fonctions de tolérance aux pannes.

Le problème d'atomicité

Le protocole de validation atomique (ACP, "Atomic Commitment Protocol")

Les participants

Un coordinateur: C ("coordinator")

. Supporte un gérant de transaction TM ("transaction manager").
. Dirige de façon centralisée la validation.

Les exécutants: S1, ... , Sn supportent

. Un gestionnaire de données DM ("data manager") pour réaliser des accès.
. Un journal de transactions distribuées DTlog ("Distributed Transaction Log") ou chaque exécutant enregistre en mémoire stable les informations les plus importantes de façon à survivre aux pannes.

Remarque:
Les exécutants peuvent aussi réaliser une validation atomique "coopérative" sans l'aide d'un coordinateur.

Les principes généraux du protocole de validation atomique transactionnel

a) Le gestionnaire de données travaille sur des copies des données (en espace de travail) lors de la phase de préparation.

- Tout objet est recopié dans l'espace de travail de la transaction lors de sa première lecture.

- Tout objet est écrit dans l'espace de travail.

b) Lorsque l'on est prêt à valider la transaction, chaque objet est recopié dans le journal en mémoire stable (non affecté par les pannes).

c) Il existe un point de validation ("**commit point**") à partir duquel la transaction sera obligatoirement terminée Soit les valeurs en mémoire stable sont intégrées de façon définitive (**"commit"**)

Soit la transaction abandonnée: tout le travail temporaire est défait (**"rollback"**).

Modèle de transaction avec validation atomique

/* Préparation d'un exécutant */
t_début
tlire(x);
préécrire (x, val1);
préécrire (y, val2);
t_fin;

/* Pas de problèmes pour ce site*/
/* Début de phase de validation */

ecrire_stable (x , val1) ;
écrire_stable (y , val2) ;

/* ----- Point de validation ----- */
/* Pas de problème pour les exécutants*/
/* Écriture mémoire stable dans la base*/

écrire (x , val1) ;
écrire (y , val2) ;

/* Destruction des informations temporaires */
Remarque : modification possible préécrire
- annonce de l'intention de modifier
- écriture directe en mémoire stable.

Propriétés d'un protocole de validation atomique

C'est un protocole de **consensus** sur l'exécution d'une transaction entre des participants qui émettent un vote:

. oui, validation ("commit")

. non, abandon ("abort").

Il faut atteindre un consensus unanime selon les règles suivantes.

AC1 : **Tout participant qui atteint une décision atteint la même décision que les autres (atomicité)**.
Cette hypothèse implique que tous terminent de façon cohérente.

AC2 : **Un participant ne peut plus revenir sur sa décision s'il l'a émise.**
La décision d'un site est irrévocable.

AC3 : **La décision générale de validation est prise si tous les participants ont voté oui (validation).**
La validation n'est atteinte que si tous ont répondu oui.

Propriétés d'un protocole de validation atomique (suite)

AC4 Si la décision de tous est validation et s'il n'y a pas de pannes la validation est réalisée.

Pour éviter de toujours décider abandon.

AC5 Si la décision générale est validation et qu'il n'y a que des pannes que l'algorithme peut tolérer => lorsque les pannes sont réparées et s'il n'y a pas de nouvelles pannes la validation est réalisée.

Pour éviter qu'une indécision perpétuelle ne soit possible.

La validation à deux phases (2PC "Two Phase Commit")

Principes d'une validation atomique (syntaxe de l'API XA de l'X/open)

a) Le coordinateur fait réaliser par les exécutants des actions sur des copies d'objets.

b) Le coordinateur initialise la phase terminale d'une transaction en demandant un vote aux exécutants (message **XA_PREPARE)**.

c) Les exécutants **OK** votent oui (non si **KO)**.

d) Le coordinateur ayant collecté toutes les réponses décide de valider ou d'abandonner par **XA_COMMIT** ou **XA_ROLLBACK**.

e) S'il a reçu **XA_COMMIT**, un exécutant recopie ses données dans la base de données et peut envoyer **XA_END** au coordinateur.

Différentes pannes possibles

. Panne du coordinateur
. Panne des exécutants
. Coupure de réseau

Problèmes liés aux pannes

Pour détecter les situations de pannes on arme différents délais de garde.

Terminaison de la validation en cas de pannes

Délai de garde dépassé chez le coordinateur

Cas 1: Coordinateur en attente de validation exécutant => décision d'abandon

Cas 2 : Coordinateur en attente d'acquittement final
=> répétition (sur demande) du message de validation ou d'abandon

Délai de garde dépassé chez un exécutant

Cas 1 : En phase de préparation non réponse du coordinateur
=> décision d'abandon de l'exécutant.

Cas 2 : Après avoir voté :
=> situation d'incertitude chez un exécutant qui peut durer très longtemps.

Notion de phase d'incertitude d'un exécutant

Le site Sk ayant notifié sa réponse de vote oui **ne reçoit pas de réponse** du coordinateur en raison de pannes.

=> Il ne sait pas s'il doit valider ou abandonner.

Exemples

. Coupure de Sk du reste du réseau
. Relation de Sk avec des sites en état d'incertitude
. Panne de Sk en état incertain et réparation
. Panne du coordinateur entre réponse oui et le message XA_COMMIT.

=> **Blocage d'une transaction** en cours pour une durée arbitrairement grande (en raison des pannes).

Solutions pour lever l'incertitude d'un exécutant

1 Basée uniquement sur la reconnexion avec le coordinateur opérationnel (après réparation).

2 Basée sur un dialogue entre les exécutants : certains sites peuvent savoir quel est le statut de la transaction.

. L'un des exécutants est encore dans la phase préparatoire => il va abandonner.

. L'un des exécutants a déjà abandonné.

. L'un des exécutants connaît le statut validé de la transaction.

. Tous les exécutants que peut contacter le demandeur du statut sont dans le même état d'incertitude.

Traitements de reprise après panne

Reprise après panne du coordinateur

Reprise dans **la phase préparatoire** avant diffusion du statut de la transaction:

=>Abandonner la transaction, (même s'il serait possible de continuer).

Reprise **après décision prise**:

=> Ne rien faire.

Reprise après panne chez un exécutant

Reprise **dans la phase préparatoire**

=> Abandon de la transaction.

Reprise **après abandon** (message envoyé ou non) => Ne rien faire.

Reprise **après validation** (message envoyé ou non) => traitement identique à celui concernant la phase d'incertitude.

Reprise **en état de décision connue**

(transaction validée ou abandonnée)

=> Ne rien faire.

Programmation de la validation à deux phases

Algorithme du coordinateur

début

/* Préparer la transaction */

envoyer message XA_PREPARE à tous ;
écrire_stable début_valid (id_transaction);
armer_délai_réponse;
attendre (événement) ;

si (retombée_garde ou réponse non) **alors**
/* Un/plusieurs sites ne sont pas OK */
écrire_stable abandon(id_transaction);
envoyer XA_ROLLBACK à tous;
finsi;

si (tous les exécutants votent oui) **alors**
écrire_stable validation(id_transaction);
envoyer C_COMMIT à tous les exécutants;
finsi;

fin coordinateur;

Algorithme d'un exécutant

début
/* Réaliser la préparation de la transaction */

armer (délai_message_XA_prepare) ;
attendre (événement) ;

si (retombée_garde) **alors**

écrire_stable (abandon (id_transaction)) ;
fin exécutant;
finsi ;

si (XA_PREPARE et vote oui) **alors**
écrire_stable (oui (id_transaction)) ;
envoyer C_READY au coordinateur ;
armer (délai_message_valid_abandon) ;
attendre(événement) ;
si (retombée_garde) **alors**
/* Cas d'incertitude trop longue */
exécuter programme de terminaison ;
finsi ;

si (message XA_COMMIT) **alors**
écrire_stable(validation(id_transaction));
écrire définitivement les modifications ;
finsi ;

si (message XA_ROLLBACK) **alors**
écrire_stable (abandon(id_transaction)) ;
détruire les informations mémoire stable;
finsi ;

finsi ;

si (XA_PREPARE et vote non) **alors**
écrire_stable (abandon (id_transaction)) ;
envoyer NON au coordinateur ;
détruire les informations mémoire stable ;
fin exécutant ;

finsi ;

fin exécutant **;**

Phase de terminaison par dialogue entre les participants

Algorithme du demandeur de statut de la transaction

début

envoyer DEM_STATUT(transaction) à tous;
armer (délai_réponse) ;
attendre (événement) ;

si (message réponse XA_COMMIT) **alors**
  écrire_stable (validation(id_transaction));
  écrire définitivement les modifications;
finsi ;

si (message réponse XA_ROLLBACK) **alors**
  écrire_stable (abandon(ident_transaction));
 détruire les modifications en mémoire stable;
finsi ;

si (retombée_garde) **alors**
 état d'incertitude non levé **recommencer**;
finsi ;

fin terminaison_demandeur**;**

Algorithme du répondeur
(aux demandes de statut de la transaction)

```
début

attendre (événement ) ;

si (message DEM_STATUT) alors

 si (le site a voté non ou s'il a noté
          abandon(ident_transaction) alors
  envoyer (XA_ROLLBACK);
  sinon
      si (le site a noté
          validation(ident_transaction) alors
          envoyer (XA_COMMIT);
      sinon

      /* Le répondeur est aussi incertain */
      /* problème non résolu              */
      finsi ;
  finsi ;
finsi ;

fin terminaison_répondeur ;
```

Bibliographie

1 R. Strong "Problems in fault-tolerant distributed systems", Publication IEEE
ISBN 135-/85/0000/0300s01.00

2 F. Christian, "Issues in the design of highly available computing systems",
IBM Research Report RJ 5856 (58907) 7/10/1987

3 F. Christian, "Questions to ask when designing or attempting to understand a fault-tolerant distributed system",
IBM Research Report RJ 6980 (66517) 4/8/1989

4 J.C. Laprie, B. Courtois, M.C. Gaudel , D. Powell "Sûreté de fonctionnement des systèmes informatiques matériels et logiciels", AFCET Dunod Informatique 1989

5 Georges Gardarin, Olivier Gardarin
"Le client-serveur" Eyrolles

6 Robert Orfali, Dan Harkey, Jeri Edwards,
"Client-serveur, guide de survie" Wiley.