Techniques de reprise sur sauvegarde en environnement réparti

Master SAR – M2 ARA

Olivier.Marin@lip6.fr
*Transparents originaux de **Pierre Sens***

Définitions - rappels

Points de reprise non coordonnés

Points de reprise de coordonnés

Journalisation

Reprise sur sauvegarde

(rollback- recovery)

Modèles d'exécution

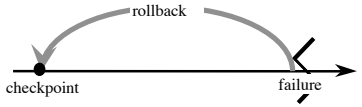
Applications réparties composées de processus communicants

Processus déterministes : définis par un état initial et une séquence d'événements

Processus non déterministes

Principe

Les processus sauvegardent leur état sur support stable pour prévenir les pannes éventuelles



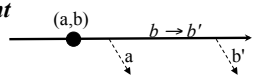
Panne => reprise à partir d'un état antérieur

Point de reprise d'un processus isolé

Copie du contexte sur support fiable (mécanisme lourd)

Pour réduire le coût :

- Méthodes incrémentales
  Sauvegarde limitée aux données mises à jour (pages modifiées)
- Sauvegarde non intrusive
  Continuer l'exécution pendant la réalisation du point de reprise
  Problème : risque de sauvegarder un état incohérent



Copy-on-write : recopie des pages *protégées* au moment de l'écriture
Pre-copying : recopie locale du point de reprise

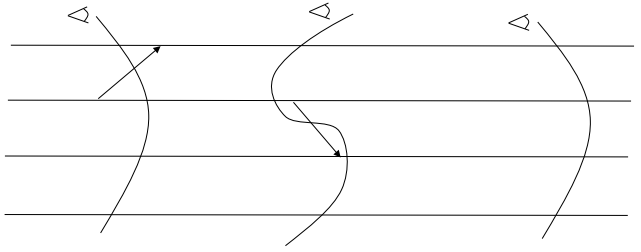
En réparti : Problème de cohérence

Maintenir la cohérence du système

Tout effet doit être précédé de sa cause

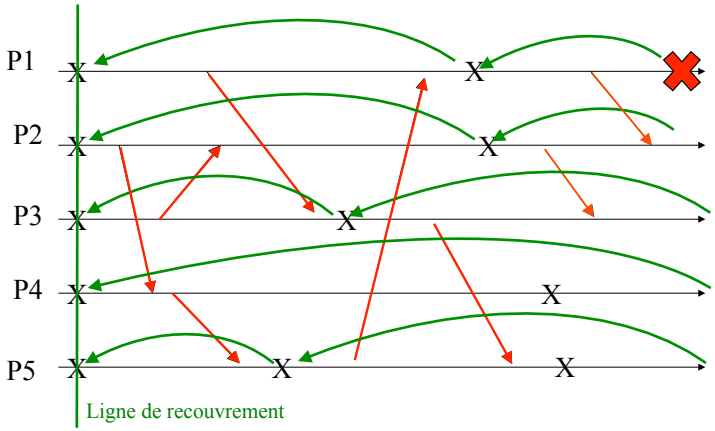
⇒ Dans les systèmes répartis :

Tout message reçu doit être préalablement émis

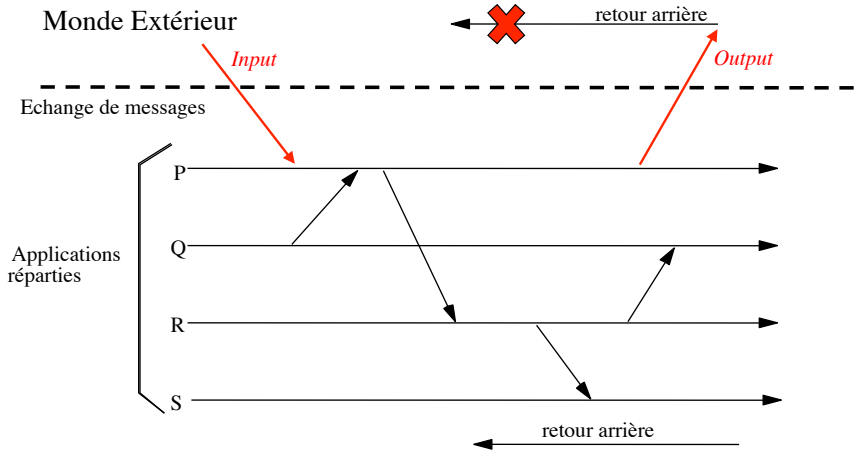


Paradoxe temporel et effet domino

« défaire » une émission => « défaire » une réception

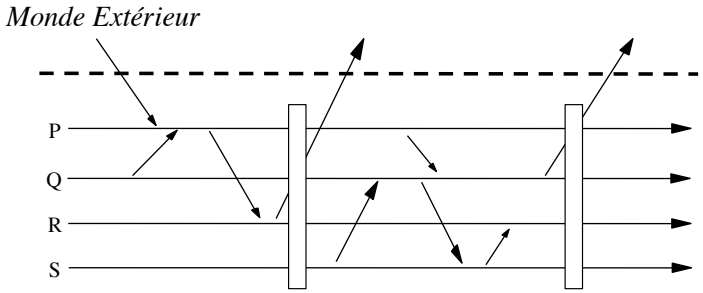


Processus communicants : Contexte



Intéraction avec le monde extérieur

Impossible de « défaire » les actions extérieures
=> « Validation » des envois vers l 'extérieur (*output commit*)



Support stable

Support qui résiste aux fautes

1 Faute : Support stable = un processus distant (en mémoire volatile)
(*Targon-32*)

Fautes transitoires : Support stable = disque local
(*DAWGS*, *Condor*)

Fautes permanentes : Support stable = Système de fichiers répliqué
(*STAR*)

Techniques de reprise sur sauvegarde

Points de reprise non coordonnés
- Trouver une ligne de recouvrement

Points de reprise coordonnés
- Coordination des processus lors des points de reprise
- L'ensemble des derniers points de reprise forme une ligne de recouvrement
- Faute => reprise des processus à partir de leur dernier point de reprise

Journalisation
- Sauvegarde des événements
- Faute => rejouer le même scénario

Point de reprise non coordonnés

Technique naïve
- Sauvegarde locale après chaque émission
- Théoriquement bon
  - Pas de message orphelin => pas d'effet domino
  - Défaillance entre l'envoi et la sauvegarde => retour à la sauvegarde précédente
  - Faible surcoût mémoire (une seule sauvegarde par nœud)
- Pas optimal en pratique
  - Coût prohibitif de la sauvegarde pour les nœuds qui émettent beaucoup
  - Rollback important pour les noeuds émettant peu mais recevant/traitant beaucoup

Sauvegardes indépendantes de l'application
- Pb : trouver une ligne de recouvrement cohérente
  - Construction centralisée d'un graphe de causalité
  - Algorithmes de recouvrement décentralisés

Point de reprise non coordonnés

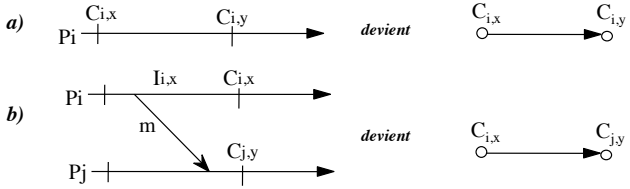
Construction centralisée d'un graphe de causalité
- Chaque processus maintient un graphe localement
- Graphe de dépendance directe entre ses points de reprise et ceux distants

Recouvrement
1. Diffusion d'une requête
2. Collecte des graphes locaux => construction d'un graphe global
3. Déterminer la ligne de recouvrement
4. Reprise des exécutions

Deux approches par graphes
- Rollback dependency graph [Bhargava 88, Wang 95]
- Checkpoint graph [Wang 93]

Déterminer la ligne de recouvrement

Graphe de dépendance des reprises (*rollback dependency graph*)

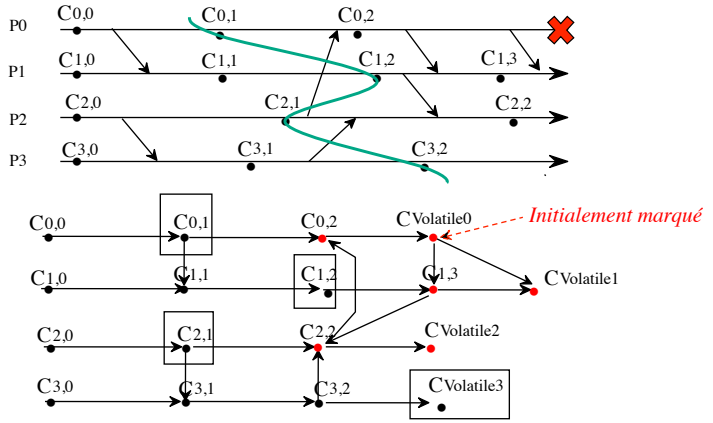


Chaque message estampillé par <Intervalle exp. (i,x)>

Récepteur de mise à jour du graphe en mémoire volatile (checkpoint volatile) : $C_{i,x}$ --> $C_{volatile}$

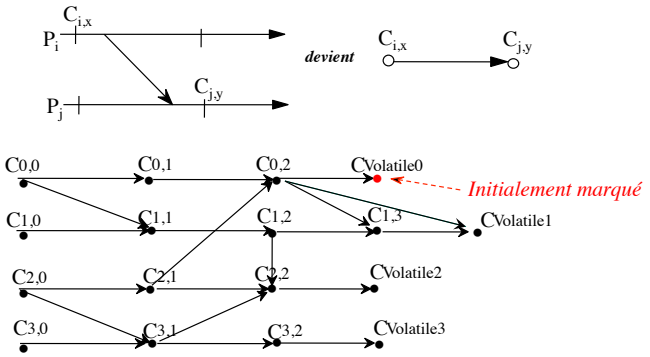
Point de reprise => Ecriture du graphe sur support stable

Graphe de reprise : Exemple



Reprise à partir des premiers points de reprise non marqués

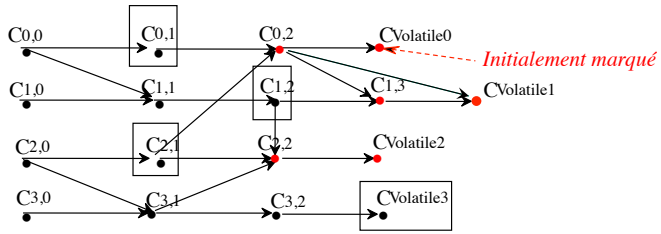
Graphe de points reprise *(checkpoint graph)*



Remarque : Graphe utilisé aussi pour le GC des points de reprise

Graphe de points de reprise : ligne de recouvrement

RootSet = {ensemble des derniers points de reprise (PR)}
Marquer le point de faute & tous les PR accessibles depuis les membres du *RootSet*
TQ (au moins 1 membre du *RootSet* est marqué) {
remplacer dans le *RootSet* chq PR marqué par son prédécesseur sur le même processus
marquer tous les PR accessibles depuis les membres du *RootSet* }
La ligne de recouvrement est alors constituée par les membres du *RootSet*



Remarque : Graphe utilisé aussi pour le GC des points de reprise

Points de reprise non coordonnés

Algorithmes de recouvrement décentralisés

But : déterminer une ligne de recouvrement sans l'aide d'un graphe

Des exercices théoriques généralemment difficiles à mettre en œuvre
- Complexes
- Coûteux
- Associés à des hypothèses fortes

2 exemples
1. Juang-Venkatesan
2. Peterson-Kearns

Point de reprise non coordonnés

Juang-Venkatesan

Méthode
- Comptabilisation des messages sur tous les canaux

Hypothèses
- Canaux fiables et FIFO
- Délais de transmission non connus mais bornés
- Mémoire locale infinie
- Processus à causalité événementielle (*piece-wise deterministic*)

En local, chaque processus P_i conserve
- Un compteur d'état S_i incrémenté à chaque sauvegarde
- Pour chaque voisin P_j :
  - un compteur d'émission $SENT_{ij}$ et un compteur de réception $RCVD_{ji}$
- Chaque sauvegarde inclut Si, ainsi que $SENT_{ij}$ et $RCVD_{ji}$ pour chaque canal

Point de reprise non coordonnés

Juang-Venkatesan

Algorithme

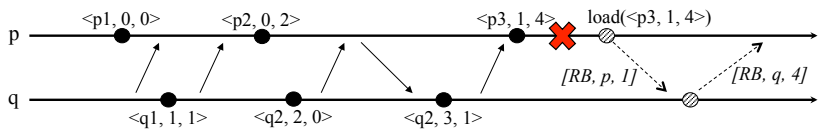
Si (P_i défaillant) revenir au dernier état sauvegardé
Sinon créer nouveau S_i volatile

Répéter N-1 fois *// N le nb de sites*
  Pour tout site $j \neq i$
    envoyer (<ROLLBACK, i, $SENT_{ij}(S_i)$>)
  Répéter N-1 fois
    recevoir(<ROLLBACK, j, v>)
    Si ($RCVD_{ji}(S_i) > v$) *// il y a des messages orphelins*
      Trouver (S'_i) le plus tardif tel que $RCVD_{ji}(S'_i) \leq v$
      $S_i = S'_i$
  FinR
FinR

Point de reprise non coordonnés

Juang-Venkatesan

Exemple



Point de reprise non coordonnés

Peterson-Kearns

Méthode
- Horloges vectorielles sur un anneau

Hypothèses
- Canaux fiables avec délais de transmission non connus mais bornés
- Processus organisés en anneau (chacun connaît son successeur)

En local, chaque processus P_i maintient une horloge vectorielle
Un numéro d'*incarnation* est incrémenté à chaque recouvrement

Point de reprise non coordonnés

Peterson-Kearns

Algorithme

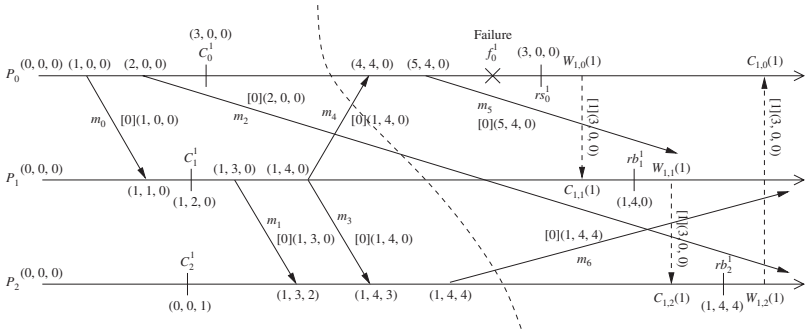
Si (P_∂ défaillant)
- Revenir au dernier état sauvegardé S_∂
- Création d'un jeton <[incarnation++] (horloge de S_∂)>
- Propagation dans l'anneau

Réception de <[incarnation] ($horloge_i$)> sur Pj
- Si ((Pj initiateur) && ($horloge_j[j] \geq horloge_i[j]$))
  - FIN
- Si ($incarnation_{locale} < incarnation$)
  - $incarnation_{locale} = incarnation$
- Si ($horloge_j[j] < horloge_i[j]$)
  - $S_j = S_{j'}$ telle que $horloge_{j'}[j] \geq horloge_i[j]$
  - Propagation de <[incarnation] (horloge de S_j)>
- Sinon
  - Propagation de <[incarnation] (horloge de S_i)>

Point de reprise non coordonnés

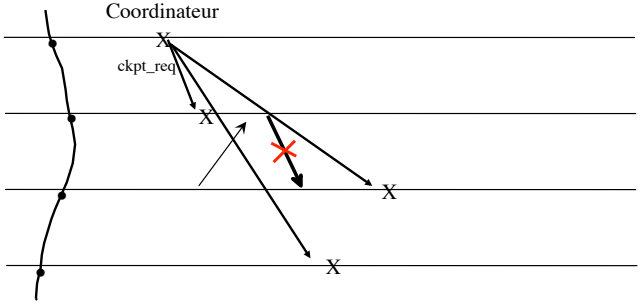
Peterson-Kearns

Exemple



Points de reprise coordonnés : Problème

1 coordinateur synchronise les points de reprise



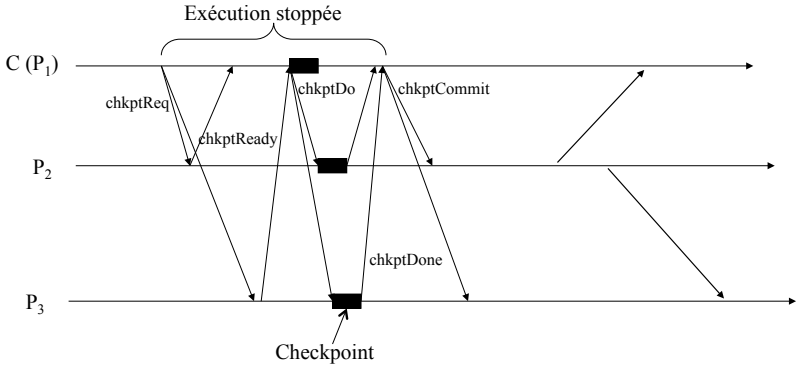
Point de reprise coordonnés : Sync-and-Stop (SaS)

Un coordinateur appelle une barrière de synchronisation pour geler l'application (arrêt des envois de message)

Algorithme :

1. Le coordinateur diffuse un message **chkptReq** à tous les processus
2. Réception **chkptReq** sur p :
   p stoppe l'application (plus d'envoi) et renvoie **chkptReady** au coordinateur
3. Lorsque le coordinateur a reçu chkptReady de tous les processus, il diffuse un message **chkptDo** et sauvegarde son état
4. Réception de chkptDo sur p :
   p sauvegarde son état et renvoie **chkptDone**
5. Lorsque le coordinateur a reçu tous les **chkptDone**, il diffuse **chkptCommit** pour débloquer les processus

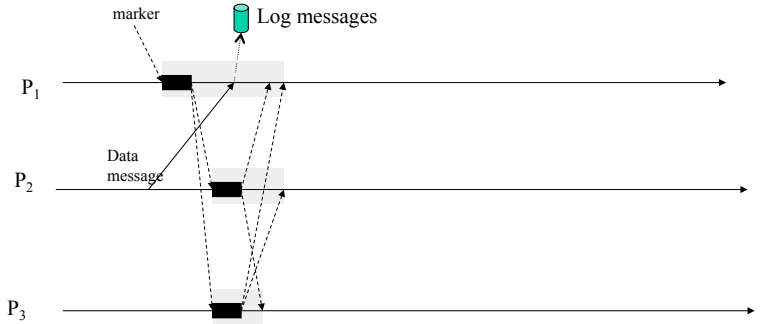
SaS : Exemple



Algorithme non bloquant : Chandy-Lamport 85

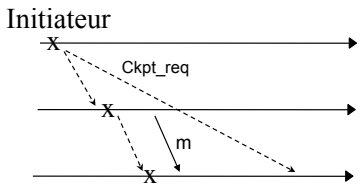
- Algorithme “distributed snapshot” : sauvegarde cohérente de l’état global (état des processus + canaux de communication)
- Hypothèse : canaux FIFO
- Le coordinateur diffuse un *marker* et fait son point de reprise
- Réception du marker sur p sur le canal c:
  - Si p n’a pas encore sauvegardé son état :
    - Rediffusion du marker et sauvegarde d’état
- Tout message reçu sur le canal c entre la sauvegarde et la réception du marker sur c est sauvé (message en transit)

Chandy-Lamport : Exemple

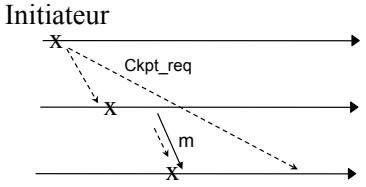


- Problème : Coûteux en message, accès au support stable
- Chandy-Lamport modifié (MCL)
  - [Agbaria06] :réduire le nombre d’accès au support, sauvegarde uniquement après la réception de tous les markers

Points de reprise coordonnés non bloquants



[Chandy-Lamport 85]
Hypothèse : canaux FIFO



[Lai-Yang 87]
Hypothèse : canaux non FIFO

Points de reprise coordonnés à synchro. minimale

Koo & Toueg 87

Messages estampillés – Chaque processus q conserve :

Last_rmsg$_q$(p) - numéro du dernier message reçu de p depuis le dernier checkpoint

First_smsg$_q$(p) - numéro du premier message envoyé à p depuis le dernier checkpoint

Création d'un point de reprise PR

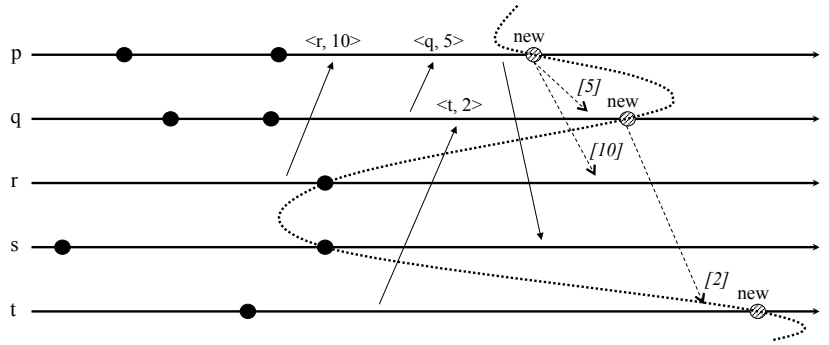
1. Notification aux processus dont on a reçu un message postérieur à PR-1
2. Test d'incohérence de chaque processus émetteur

   Incohérence ⇒ création d'un nouveau point de reprise par l'émetteur

Reprise

Annulation des envois postérieurs aux derniers points de reprise

Koo & Toueg 87 par la pratique

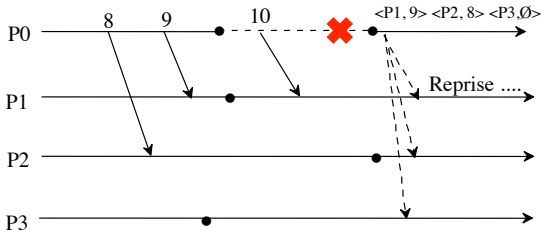


Koo & Toueg 87 : Reprise minimale

Reprise : propager aux processus dont on "défait" un envoi de message

Reprise de q
1. Diffusion des numéros de dernier message envoyé *avant* le dernier point de reprise (*last_smsg$_q$(p)*)

2°) Reception sur p :
Si last_smsg$_q$(p) < last_rmsg$_P$(q)
Reprendre ==> Lancer l'algorithme localement



Point de reprise coordonnés à synchronisations implicites

Coordination implicite : utilisation des messages d'applications

Algorithme "naïf"

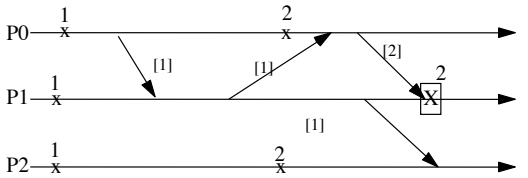
Sauvegarde (atomique) après chaque émission [Wu & Fuchs 90]

Algorithme de base : [Briatico 84]

Un compteur date les points de reprise (+1 à chaque checkpoint)

Chaque message est estampillé par le compteur de l'émetteur

Réception d'un message m :
ckpt si m.compteur > compteur_local



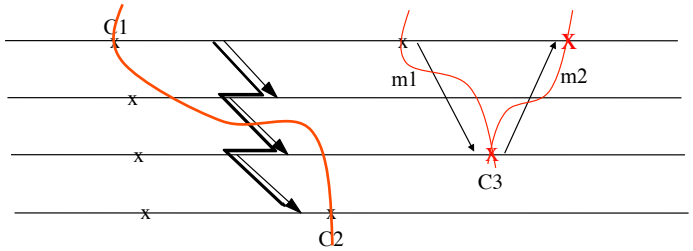
Zigzag-Path (Xu & Netzer 93)

- Extension des chemins de causalité de Lamport
- Un *Zigzag chemin* (*Z-path*) de $C_{p,i}$ à $C_{q,j}$ est une séquence de messages $(m_1, m_2, \ldots, m_l)$; $l \geq 1$, telle que:
  - m_1 est envoyé par p *après* $C_{p,i}$
  - Si m_k $(1 \leq k < l)$ est reçu par r, alors m_{k+1} est envoyé par r dans le même intervalle (ou plus tard)

    m_{k+1} peut être envoyé après la réception de m_k
  - m_l est reçu par q avant $C_{q,j}$

Points de reprise coordonnés : synchro. implicite (2)

Netzer et Xu 95

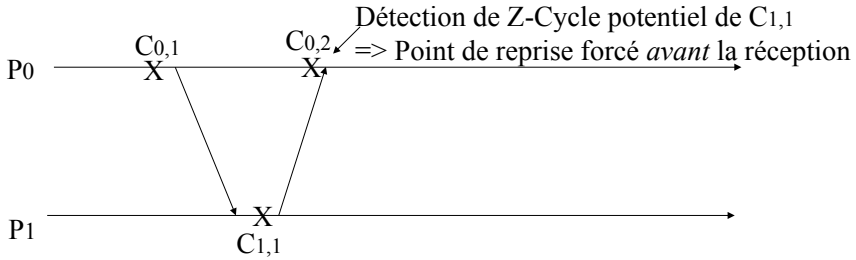


Propriété fondamentale :
S'il existe un Z-path de C1 à C2 alors C1 et C2 ne peuvent faire partie du même ensemble cohérent de points de reprise

Z-cycle : Il existe un Z-Path de C vers lui-même.
Si C appartient à un Z-cycle alors il ne peut faire partie d'une ligne de recouvrement
C est inutile
Ex : [m2, m1] forme un Z-chemin de C3 vers lui-même

Algo. adaptatif à base de détection de Z-Cycle

- [Hélary 97] – sur les principes de [Briatico 84]



Recouvrement par journalisation

Point de reprise non coordonné

Principe : Reconstruire l'état précédant la faute en rejouant le même scénario

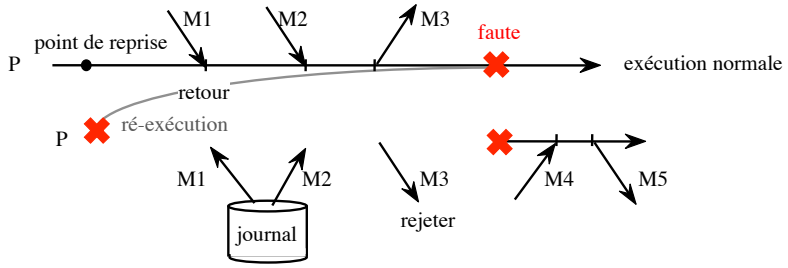
=> Tracer tous les événements non déterministes (logging)

Hypothèse d'application "*piecewise deterministic*" (PWD) :

- Exécution ensemble d'intervalles déterministes
- Chaque intervalle commence par un événement non déterministe (réception d'un message).

Journalisation pessimiste

Journalisation synchrone de tous les messages reçus
⇒ un processus repris consomme les messages du journal
Détection des réémissions (estampillage)
⇒ rejet des messages retransmis par les processus repris



Journalisation Pessimiste : Optimisations

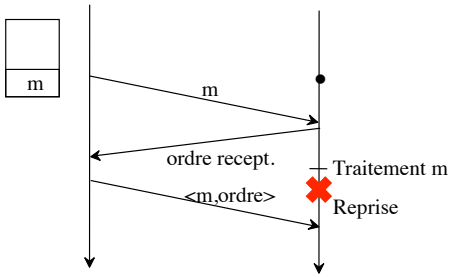
Implantation matérielle du support stable :

Mémoire non volatile (FTM)

Bus spécifique (TARGON-32)

Sans matériel spécifique :

Utiliser la mémoire vive de l'émetteur (Sender Based Message Logging [Johnson 90])



Journalisation Optimiste

Sauvegarde *asynchrone* en RAM

Le journal est sauvegardé périodiquement sur support stable

Faute => risque de perdre une partie du journal

Reprise en cascade

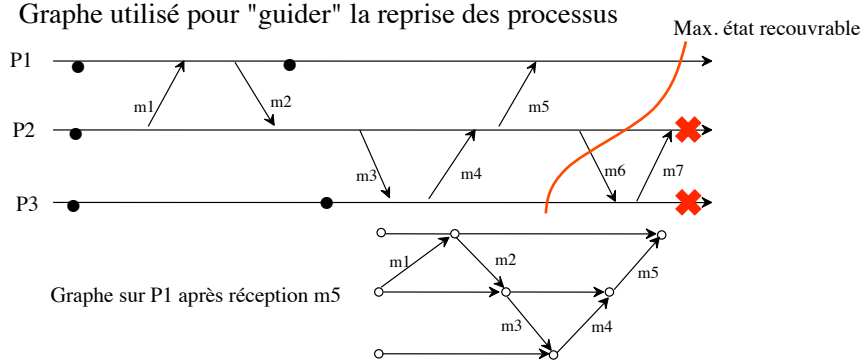
+ Asynchronisme (meilleures performances)
- Possibilité d'effet domino

Journalisation Causale (Manetho, [Elnozahy 94])

Chaque processus maintient un *graphe de précédences*
(historique des événements non déterministes qui précèdent causalement l'état courant)

Graphe contenu dans les messages (piggybacking)

Graphe utilisé pour "guider" la reprise des processus



Comparaison de stratégies (extrait de [Elnozahy 97])

	Non coordonnés	Coordonnés	Coordonnés implicites	Journalisation pessimiste	Journalisation optimiste	Journalisation causale
Hyp. PWD	Non	Non	Non	Oui	Oui	Oui
Surcoût Comm.	Faible	Aucun	Faible	Le plus élevé	Elevé	Elevé
Surcoût Sauvegarde	Faible	Le + élevé	Faible	Faible	Faible	Faible
Nb de pts de reprise	Plusieurs	1	1	1	Plusieurs	1
GC	Complexe	Simple	Simple	Simple	Complexe	Complexe
Reprise	Complexe	Simple	Simple	Simple	Complexe	Complexe
Effet domino	Possible	Impossible	Impossible	Impossible	Impossible	Impossible
Extension reprise	Illimitée	Dernier checkpoint	Dernier checkpoint	Minimum	Checkpoints précédents	Dernier checkpoint
"Output commit"	Impossible	Très lent	Très lent	Le plus rapide	Lent	Rapide

Tolérance aux fautes et large échelle

La plupart des plates-formes sont peu adaptées au large échelle

- Eloignement => Forte latence des protocoles
- Nombre de sites => Coût en ressources (réseau)
- Dynamicité => Approche statique (stratégie figée ou guidée par l'utilisateur)
- Topologie => Partitionnement

Modèle de faute restreint (crash, recovery)

- Tendance à élargir vers fautes byzantines (dans P2P)
- Outils : librairie BFT, pb très coûteux !

Point de reprise pour les Grilles

Approche hiérarchique utilisée dans ProActive (Thèse de C. Delbé)

- Utilisation de groupe de recouvrement [Sistla et Welch] :
  - 1 groupe par cluster
  - Groupe = unité de reprise
- Point de reprise coordonné au sein d'un groupe
- Journalisation des messages inter-groupes

Bibliographie

[Alvisi 93] L. Alvisi, B. Hoppe, K. Marzullo. Nonblocking and Orphan-Free Message Logging Protocols. Proc. of the 23rd International Symposium on Fault-Tolerant Computing (FTCS 23), pp. 145-154, Toulouse, France 1993.

[Agbaria06] A. Agbaria. Improvements and Reconsideration of Distributed Snapshot Protocols. In *Proceedings of the 25th Symposium on Reliable Distributed Systems (SRDS'06)*. pp. , October 2006, Leeds, UK

[Borg 89] A. Borg, W. Blau, W. Graetsch, F. Herrmann, W. Oberle. Fault Tolerance Under UNIX. ACM Transaction on Computer Systems, 7(1):1-24, février 1989.

[Fischer 85] M.J. Fischer, N.A. Lynch, M.S. Paterson. Impossibility of Distributed Consensus with One Faulty Process. Journal of the ACM 32(2):374-383, 1985.

[Johnson 90] D.B. Johnson, W. Zwaenepoel, Recovery in Distributed Systems using Optimistic Message Logging and Checkpointing. Journal of Algorithms, 11(3):462-491, septembre 1990.

[Elnozahy 96] E.N. Elnozahy, D.B. Johnson, Y.M. Wang. A survey of Rollback-Recovey Protocols in Message-Passing Systems. Technical Report. CMU-CS-96-181.

[Koo 87] Richard Koo, Sam Toueg. Checkpoint ing and Rollback-recovery for Distributed Systems. IEEE Transactions on Software Engineering, SE-13(1):23-31, janvier 1987.

[Sens 98] P. Sens, B. Folliot. The STAR Fault Tolerant Manager. Software Practice and Experience, Aout 1998.

[Wang 92] Y.M. Wang, W.F. Fuchs. Optimistic Message Logging for Independent Checkpointing in Message-Passing Systems. Proc. of the IEEE 11th Symposium on Reliable Distributed Systems, pp. 147-154, octobre 1992.

[Wang 93] Y.M. Wang, W.K. Fuchs. Lazy Coordination for Bounding Rollback Propagation. Proc. of the 12th Symposium on Reliable Distributed Systems, pp.78-85, octobre 1993.

[Netzer 95] R.H.B. Netzer, J. Xu. Necessary and Sufficient Conditions for Consistent Global Snapshots. IEEE Trans. on Parallel and Distributd Systems

[Hélary 98] Hélary, A. Mostefaoui, M. Raynal. Points de Contrôle cohérents dans les systèmes répartis : concepts et protocoles. TSI 17,10, 1998

Quelques "pointeurs"

Manetho :
http://www.cs.cmu.edu/People/mootaz/manetho.html

Libckpt :
http://www.cs.utk.edu/~plank/plank/www/libckpt.html

Algorithmes :
http://www.cs.utexas.edu/users/lorenzo

http://www.irisa.fr/adept/index.html

http://www.dis.uniroma1.it/~baldoni/publications.shtml

MPICH-V :
http://mpich-v.lri.fr