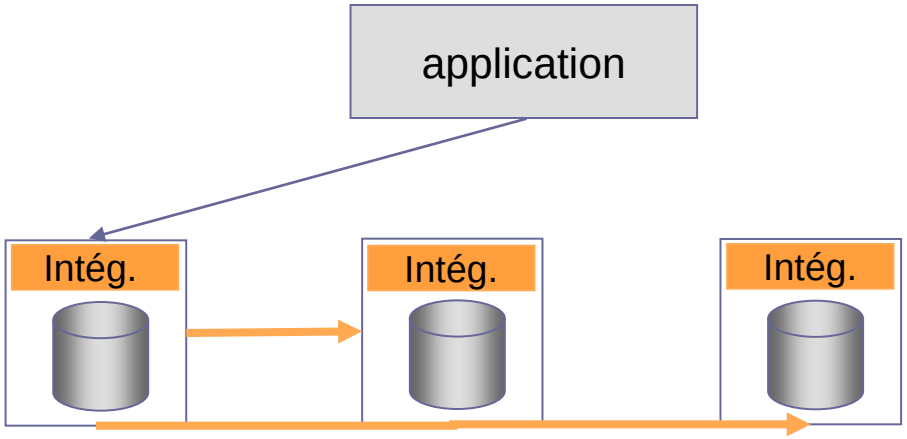
Le processus de *commit* (validation) des transactions distribuées.

Applicabilité au contexte temps réel

Bruno.Sadeg@univ-lehavre.fr

Fonctions d'un SGBD réparti



- Rend la répartition (ou distribution) *transparente*
  - dictionnaire des données réparties
  - traitement des requêtes réparties
  - gestion de transactions réparties
  - gestion de la cohérence et de la confidentialité

Bruno.Sadeg@univ-lehavre.fr

Evaluation de l'approche BDR

- Avantages
  - extensibilité
  - partage des données hétérogènes et réparties
  - performances
  - disponibilité des données
- Inconvénients
  - administration complexe
  - distribution du contrôle (complexe)

Bruno.Sadeg@univ-lehavre.fr

Objectifs de la décomposition

- fragmentation
  - trois types : horizontale, verticale, mixte
  - performances en favorisant les accès locaux
  - équilibrer la charge de travail entre les sites (parallélisme)
- duplication (ou réplication)
  - favoriser les accès locaux
  - augmenter la disponibilité des données
- Conception guidée par des heuristiques

Bruno.Sadeg@univ-lehavre.fr

RAPPEL : Fragmentation horizontale

- Fragments définis par sélection
  - Client1 : Client where ville = "Paris"
  - Client2 : Client where ville ≠ "Paris"

Client

nclient	nom	ville
C 1 C 2 C 3 C 4	Dupont Martin Martin Smith	Paris Lyon Paris Lille

Client1

nclient	nom	ville
C 1 C 3	Dupont Martin	Paris Paris

Client2

nclient	nom	ville
C 2 C 4	Martin Smith	Lyon Lille

Reconstruction

Client = Client1 U Client2

Bruno.Sadeg@univ-lehavre.fr

RAPPEL : Fragmentation horizontale dérivée

Fragments définis par jointure

Cde1 : Cde where
Cde.nclient = Client1.nclient

Cde2 : Cde where
Cde.nclient = Client2.nclient

Reconstruction

Cde = Cde1 U Cde2

Cde

ncde	nclient	produit	qté
D 1	C 1	P 1	10
D 2	C 1	P 2	20
D 3	C 2	P 3	5
D 4	C 4	P 4	10

Cde1

ncde	nclient	produit	qté
D 1	C 1	P 1	10
D 2	C 1	P 2	20

Cde2

ncde	nclient	produit	qté
D 3	C 2	P 3	5
D 4	C 4	P 4	10

Bruno.Sadeg@univ-lehavre.fr

RAPPEL : Fragmentation verticale

- Fragments définis par projection
  - Cde1 : Cde (ncde, nclient)
  - Cde2 : Cde (ncde, produit, qté)
- Reconstruction
  - Cde : [ncde, nclient, produit, qté] where Cde1.ncde = Cde2.ncde
- Utile si forte affinité d'attributs

Cde

ncde	nclient	produit	qté
D 1	C 1	P 1	10
D 2	C 1	P 2	20
D 3	C 2	P 3	5
D 4	C 4	P 4	10

Cde1

ncde	nclient
D 1	C 1
D 2	C 1
D 3	C 2
D 4	C 4

Cde2

ncde	produit	qté
D 1	P 1	10
D 2	P 2	20
D 3	P 3	5
D 4	P 4	10

Bruno.Sadeg@univ-lehavre.fr

RAPPEL : Allocation des fragments aux sites

- Non-dupliquée
  - partitionnée : chaque fragment réside sur un seul site
- Dupliquée
  - chaque fragment sur un ou plusieurs sites
  - maintien de la cohérence des copies multiples
- Règle intuitive:

si le ratio [lectures/màj] est > 1, la duplication est avantageuse (il y beaucoup de lectures)

Bruno.Sadeg@univ-lehavre.fr

RAPPEL : Exemple d'allocation de fragments

Client1

nclient	nom	ville
C 1 C 3	Dupont Martin	Paris Paris

Client2

nclient	nom	ville
C 2 C 4	Martin Smith	Lyon Lille

Cde1

ncde	client	produit	qté
D 1 D 2	C 1 C 1	P 1 P 2	10 20

Site 1

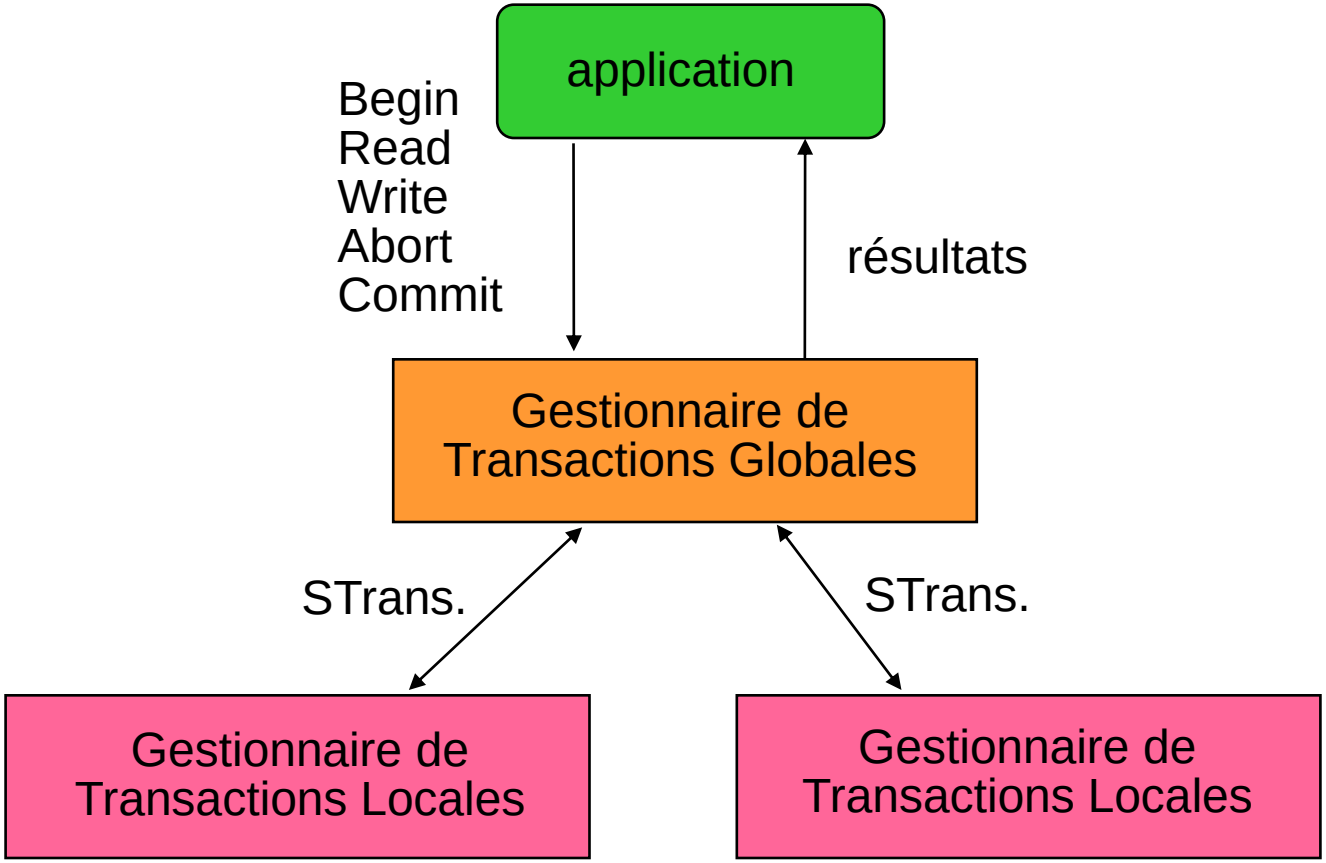
Cde2

ncde	client	produit	qté
D 3 D 4	C 2 C 4	P 3 P 4	5 10

Site 2

Bruno.Sadeg@univ-lehavre.fr

Notion de Transaction Répartie



Bruno.Sadeg@univ-lehavre.fr

Protocole de validation en 2 étapes

- **Objectif** : Exécuter la commande COMMIT pour une transaction répartie
  - Phase 1 : Préparer à écrire les résultats des mises-à-jour dans la BD
  - Phase 2 : Ecrire ces résultats dans la BD

- **Coordinateur** : composant système d'un site qui applique le protocole
- **Participant** : composant système d'un autre site qui participe dans l'exécution de la transaction

Bruno.Sadeg@univ-lehavre.fr

Rappels : Contrôle de concurrence et recouvrement dans les SGBD

Bruno.Sadeg@univ-lehavre.fr

BD : {données nommées}. Chaque donnée a une valeur.

Etat de la BD : valeurs des données à un instant donné.

En pratique, une donnée peut être un mot-mémoire, une page-disque, une enregistrement de fichier ou une rubrique d 'un enregistrement.

SGBD : collection de modules matériels et logiciels qui supportent les commandes d 'accès à la BD.

Une transaction : {actions}

Une transaction T : *début T* ------ *T active* ----------> ------> *Validation (Commit)*

ou ------> *Abandon (Abort)*

T *non commitée* : T abandonnée ou T active

T abandonnée -------> *dû à T elle-même (suite à une condition inhérente à T (ex. fonds insuffisants)*

ou -------> *dû à une raison extérieure (panne système, ...)*

Bruno.Sadeg@univ-lehavre.fr

Actions d 'une transaction : souvent réduites à R et W (supposées atomiques)

Abandon de T (***Abort***) : annule les effets sur la BD des actions déjà effectuées

Validation de T (***Commit***) : garantit que les modifications effectuées par les actions des transactions ont été reportées sur la base (sur disque)

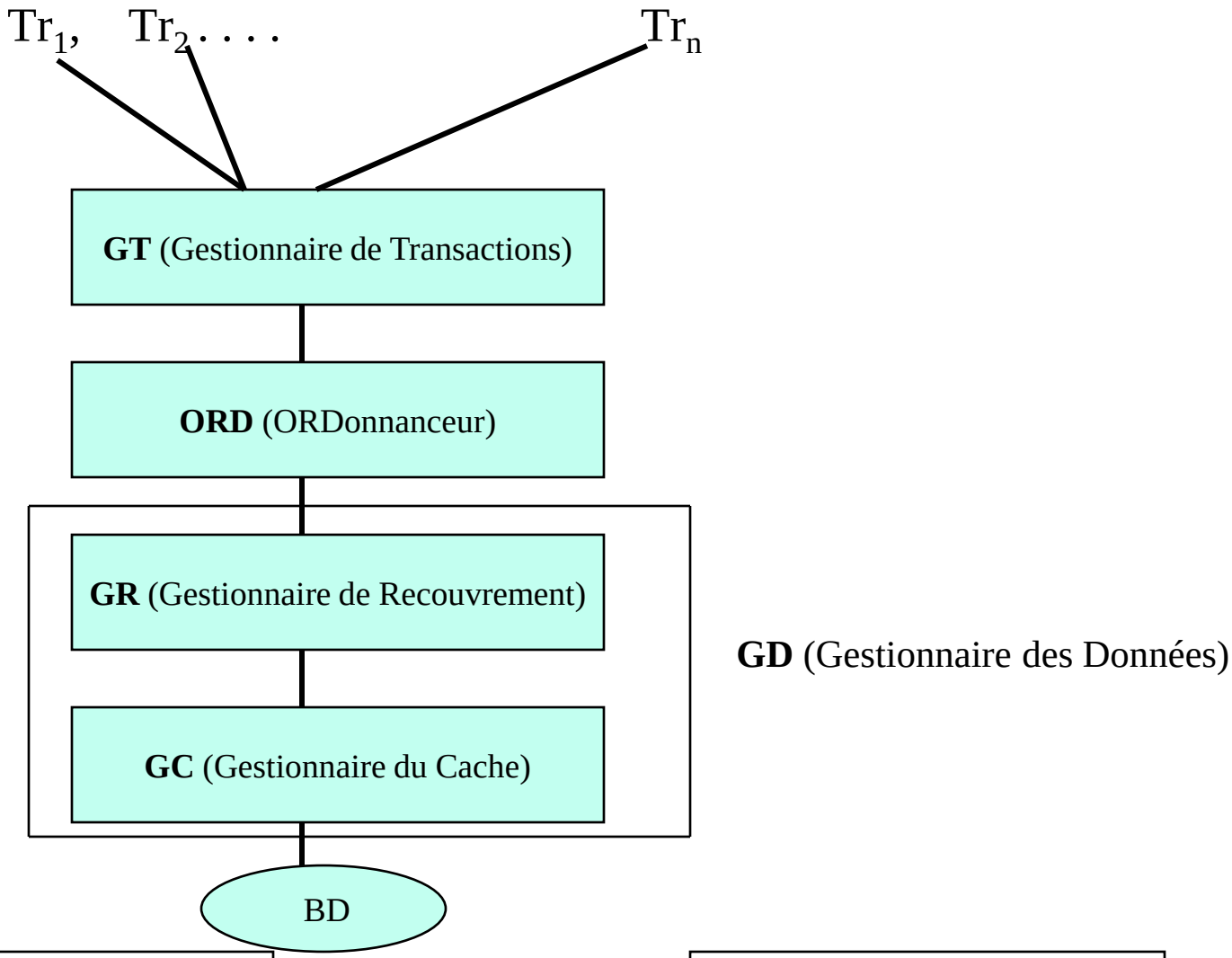
Algorithme de recouvrement (restauration) : fait en sorte que la BD contienne tous les effets des transactions validées et ne contienne aucun effet d 'une transaction abandonnée.

NB : les écritures dans la BD sur disque sont asynchrones. Donc il peut arriver que des modifications ayant été écrites sur le tampon (en MC) ne soient pas encore reportées sur disque quand survient une panne.

Algorithme de contrôle de concurrence : assure que les transactions s 'exécutent de manière atomique (comme si chacune s 'exécute seule dans le système) => il assure que les trans. sont sérialisables (résultat équivalent à leur exécution en série).

Bruno.Sadeg@univ-lehavre.fr

Modèle de mécanisme gérant les transactions



Bruno.Sadeg@univ-lehavre.fr

4 modules :

1. GT : le gestionnaire de transactions traite les demandes d 'exécution venant de la BD et les opérations qu'il reçoit des transactions

2. ORD : l 'ordonnanceur contrôle l 'ordre relatif d 'exécution dans lequel les opérations de la BD et des opérations des transactions sont exécutées

3. GR : le gestionnaire de recouvrement est responsable de l 'abandon (Abort) et de la validation (Commit) des transactions

4. GC : le gestionnaire de cache opère directement sur la BD

Bruno.Sadeg@univ-lehavre.fr

Problème d 'accès concurrent aux données dans les SGBD

Protocole le plus connu dans les SGBD centralisés : **2PL** (2 Phase-Locking ou Protocole de verrouillage à 2 phases).

Principe : à chaque donnée de la base est associé un verrou. Avant qu'une transaction n'accède à une donnée, le gestionnaire de transactions examine d'abord le verrou associé :

=> si verrous compatibles alors la transaction est autorisée à s 'exécuter

=> si verrous incompatibles alors la transaction est mise en attente (voir transparent suivant)

Dans le 2PL, une transaction :

-> commence par une phase d 'acquisition de tous les verrous (données dont elle a besoin) => phase I

-> puis passe par une phase de libération de tous ses verrous (et ne peut plus en acquérir d 'autres) => phase II

Bruno.Sadeg@univ-lehavre.fr

Matrice de compatibilité des verrous

Transaction détentrice

Transaction demandeuse

	V_R	V_W	-
V_R	OUI	NON	OUI
V_W	NON	NON	OUI

Bruno.Sadeg@univ-lehavre.fr

Protocole 2 PL de base

$V_r[x]$: verrou de lecture sur x, $V_w[x]$: verrou d 'écriture sur x

Il est du travail de l 'ordonnanceur 2PL de gérer les verrous en contrôlant quand les transactions obtiennent les verrous et quand elles les libèrent. ==> Règles :

1. Quand l 'ORD reçoit une opération $p_i[x]$ du GT, il teste si le verrou qu 'elle s 'apprête à poser sur x est en conflit avec celui d 'une autre opération $q_j[x]$ déjà active sur x. Si c 'est le cas, il retarde $p_i[x]$ en forçant la transaction T_i à attendre jusqu'à ce qu 'elle puisse poser son verrou. S 'il n 'y a pas conflit, alors l 'ORD pose sur x le verrou de l 'opération p_i .

2. Une fois que l'ORD a posé un verrou pour T_i , il ne peut libérer ce verrou jusqu 'à (au moins) après que le GD lui notifie qu 'il a fini l 'exécution de l 'opération sur la donnée correspondant à ce verrou (c-à-d $p_i[x]$).

3. Une fois que l'ORD a libéré un verrou pour une transaction, il ne peut plus obtenir de nouveaux verrous pour cette transaction (pour aucune donnée)

Bruno.Sadeg@univ-lehavre.fr

Comment exécuter les transactions **en cas de panne ?**

Il y a plusieurs types de pannes :

Celles qui ne concernent pas le mécanisme de recouvrement :

- mauvaise programmation des transactions ou paramètres d 'entrée incorrects
  => du ressort du programmeur (amélioration grâce aux techniques de GL)
- erreurs de l'opérateur qui détruit une partie de la base

Celles qui **concernent** le mécanisme de recouvrement :

- erreurs dans les transactions : abandon de transactions - Abort-
- **pannes systèmes : perte du contenu de la mémoire volatile**
- panne du support : perte d 'une partie du contenu du support stable (secteurs disque défectueux)

Bruno.Sadeg@univ-lehavre.fr

En cas de panne (suite)

On considère seulement les pannes système. On suppose que ces pannes sont détectées.
Fonctionnement :

- Les trans. sont soumises au GT, qui les soumet à l 'ORD. L 'ORD reçoit *Read*, *Write*, *Abort ou Commit*. Il peut passer immédiatement les *Abort* au GD. Pour les autres, il doit décider (éventuellement après un délai) de rejeter ou accepter l 'opération :

a- Si Rejet, l 'ORD envoie un *Non-ACK* au GT qui lui renvoie un *Abort*. L 'ORD envoie *Abort* au GD.

b- Si l 'ORD accepte l 'opération, il l 'envoie au GD qui l 'exécute (selon un algorithme). Quand il a fini, il envoie un *ACK* à l 'ORD qui le passe au GT (s 'il s 'agit de *Read*, la valeur lue est incluse dans l 'ACK).

Bruno.Sadeg@univ-lehavre.fr

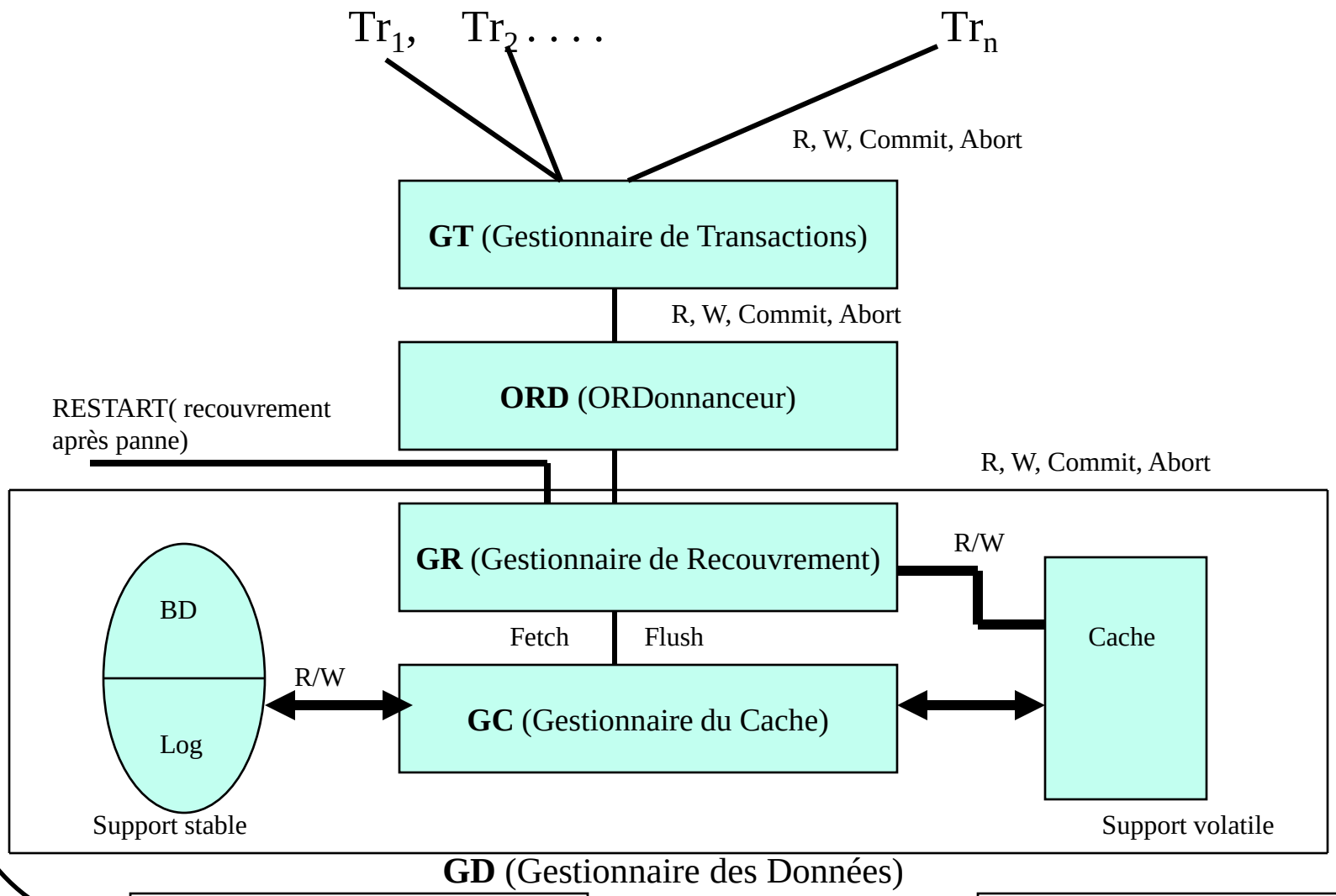
En cas de panne (suite)

c. En plus des opérations précédentes, le GT peut recevoir une opération Réinitialiser (Restart) suite à une panne système

=> remet la BD dans un état cohérent : annule les effets des trans. non commitées (Undo) et applique à la BD les effets des trans. commitées qui n 'y sont pas déjà (Redo).

Bruno.Sadeg@univ-lehavre.fr

Modèle de SGBD centralisé



Bruno.Sadeg@univ-lehavre.fr

UNDO et REDO

La BD (stable) peut contenir des valeurs écrites par des transactions *non commitées* et peut ne pas contenir les valeurs des transactions *commitées*.

En cas de panne, le GR par son opération *Restart* doit transformer la BD stable en BD *commitée*

⇒le GR écrit habituellement sur un support stable les informations (en plus de la BD stable) : ce sont les fichier log. De plus le GR tient à jour sur support stable la liste des id des trans. a*ctives, commitées et abandonnées*

⇒ suite à une panne syst. le GR consulte cette liste pour savoir quelles trans. rejouer et lesquelles abandonner.

∀ stratégie de remplacement, le GR doit parfois forçer les écritures depuis le cache sur la BD stable ou le log.

=> Restart doit toujours trouver l ’information dont elle a besoin dans un support stable.

Bruno.Sadeg@univ-lehavre.fr

UNDO et REDO (suite)

Plusieurs types de GR en fonction des stratégies adoptées :

- le GT a besoin de faire UNDO s'il permet d'écrire dans la BD les valeurs de transactions non commitées. Si panne, l 'opération Restart doit éliminer de la BD les effets de ces transac.

- le GT a besoin de faire REDO s'il permet à une transaction de commiter avant que les valeurs qu 'elle a écrites soient reportées sur la BD stable. Si panne, l 'opération Restart doit «rejouer» ces trans. pour retrouver un état cohérent de la BD

Bruno.Sadeg@univ-lehavre.fr

Cas de panne : systèmes distribués

Système distribué = {sites} + {liens de communication entre sites}

On suppose qu 'il y a toujours un chemin (direct ou non) entre 2 sites

Panne de site : perte du contenu de la mémoire volatile. Quand le site revient après une panne, on dit que le site exécute un processus de recouvrement qui remet le site dans un état cohérent. On suppose un modèle de panne appelé FAIL-STOP (le site est opérationnel ou non. Il n 'exécute pas d 'opérations incorrectes).

Il peut y avoir panne partielle (certains sites en panne) ou totale (tous les sites en panne)

Bruno.Sadeg@univ-lehavre.fr

Cas de panne : systèmes distribués

Panne de communication : messages corrompus, fonctionnement incorrect d ’un lien (temporaire : un message perdu, ou lien rompu : tous les messages sont perdus). Solutions : détection d ’erreurs, retransmission, recherche d ’autres chemins pour transmettre, …

Mais malgré cela, la combinaison des pannes sites et réseaux peut couper la communication entre sites.

=> cas où TOUS les chemins allant de A à B contiennent un site en panne ou un lien coupé : Problème de PARTITION du réseau.

Bruno.Sadeg@univ-lehavre.fr

Pannes (suite)

=> Tenir compte des messages non délivrés.

Un message n'est pas délivré car (i) son récipient est en panne, (ii) ou bien son envoyeur et son récipient sont sur 2 composantes différentes d'une partition du réseau.

=> 2 options :

- le message est persistant, auquel cas le gestionnaire de réseau le stocke et le délivre à son récipient quand cela devient possible

- le message est perdu : le gestionnaire de réseau ne fait aucune tentative pour l 'envoyer de nouveau. C'est l'option adoptée ici.

Détection de pannes par timeouts : *A* envoie un message à *B* et attend une réponse pendant une période de temps δ (période de timeout). Si la réponse arrive, il est clair que *A* et *B* peuvent communiquer. Si la réponse n 'arrive pas, *A* conclut qu 'il ne peut pas communiquer avec *B*.

Bruno.Sadeg@univ-lehavre.fr

Validation atomique (Atomic Commitment)

Soit une transaction T qui s 'exécute sur les sites S1, S2, …, Sn. Le GT sur S1 supervise l 'exécution :

Avant qu'il n'envoie l'opération Commit aux autres sites, il doit s'assurer que l'ORD et le GD sur chaque site sont prêts.

⇒conditions :

- l'ORD sur un site peut accepter d'exécuter Commit(T) aussi longtemps que T satisfait à la condition de **recouvrabilité** sur ce site, c-à-d que chaque donnée lue par T sur ce site est écrite par une transaction ayant validée (effectuée son commit).

.

Bruno.Sadeg@univ-lehavre.fr

Validation atomique (Atomic Commitment) (suite)

- le GD sur un site peut accepter d'exécuter *Commit(T)* aussi longtemps qu'il satisfait à la **règle Redo** (toutes les valeurs écrites par T sur ce site sont stockées sur support stable -BD ou log). Noter que si T ne soumet que des lectures à un site, il n'est pas nécessaire qu'elle demande le consentement du GD de ce site.

Le GT de S_1 peut initier *Commit(T)* aux ORD et GD de S_1, S_2, …, S_n seulement après avoir reçu leurs consentements.

Il existe un processus sur chaque site où la transaction est exécutée. Ces processus réalisent la validation atomique.

Bruno.Sadeg@univ-lehavre.fr

Validation atomique

Hypothèses : Le site initiateur de la transaction est le coordinateur, les autres sont les participants. Le coordinateur connaît les noms de tous les participants, et peut donc leur envoyer des messages. Les participants connaissent le nom du coordinateur mais pas nécessairement les noms des autres participants.

Chaque site contient un log (journal) de transactions distribuées (DT log) sur support stable où des informations sont enregistrées. Les DT log doivent **survivre** aux pannes.

Un protocole de Validation Atomique (ACP) est un algorithme pour le coordinateur et les participants tels que le coordinateur et tous les participants valident la transaction ou bien tous l 'abandonnent. Plus précisément, chaque processus doit effectuer exactement un vote *OUI* ou *NON* et atteindre exactement une décision : Commit ou Abort.

Bruno.Sadeg@univ-lehavre.fr

Validation atomique : conditions

AC1 : tous les process qui atteignent une décision, atteignent la même décision

AC2 : un process ne peut pas inverser sa décision une fois qu 'il l 'a prise

AC3 : le décision de *Commit* peut être prise seulement si tous les process votent *OUI* (si un process vote NON, il peut faire un *Abort* à tout moment. Mais après avoir voté *OUI*, il ne peut plus prendre de décision unilatéralement : il est dans une période d 'incertitude jusqu 'à ce qu 'il reçoive Commit ou Abort du coordinateur)

AC4 : En l 'absence de pannes et si tous les process votent *OUI*, alors la décision est **Commit**

AC5 : Considérons toute exécution contenant seulement des pannes que l 'algorithme peut tolérer. A tout point de cette exécution, si toutes les pannes survenues sont réparées et si aucune nouvelle panne ne survient, alors tous les process pourront atteindre une décision par la suite.

Le plus simple et populaires des ACP est le 2PC (2 Phase Commit). Puis, il y a PA, PC, 3PC, ..

Bruno.Sadeg@univ-lehavre.fr

Validation atomique : scénarios de pannes

1. Une panne coupe la communication entre un process *p* et les autres, pendant que *p* est incertain (durant sa période d 'incertitude) :

p ne peut atteindre une décision qu 'après que la communication soit rétablie. Le processus est bloqué => Une panne de communication peut engendrer un blocage d 'un processus pour une période indéterminée.

2. Le process *p* tombe en panne pendant sa période d 'incertitude :

Quand *p* est restauré (recover), il ne peut pas atteindre une décision tout seul (independant recovery). Il doit communiquer avec les autres process pour savoir quelle était la décision. L 'impossibilité de recouvrement indépendant en conjonction avec la panne totale (tous les sites) engendre des blocages : Supposons que *p* du scénario 2 soit le premier à recouvrir (revenir de la panne totale). Comme *p* est incertain, il doit communiquer avec les autres process avant de pouvoir prendre une décision. Mais il ne peut pas communiquer car ils sont tous en panne : => blocage.

=> Les pannes durant la période d 'incertitude des process posent de gros problèmes.

Bruno.Sadeg@univ-lehavre.fr

Validation atomique : observations

1. Si des pannes de communication ou des pannes totales sont possibles, alors tout protocole de validation atomique (ACP) peut engendrer un blocage de processus.

(Rmq. si panne de site seulement, il existe des ACP qui n ’engendrent pas des blocages)

2. Aucun ACP ne garantit le recouvrement indépendant de sites en panne.

Bruno.Sadeg@univ-lehavre.fr

Protocoles de CC dans les SGBD distribués

- 2 PC (Two Phase-Commit)
- PA (Presumed Abort)
- PC (Presumed Commit)
- 3 PC (Three Phase-Commit)

Modèle utilisé :

- un processus maître (coordinateur) qui s 'exécute sur le site où la transaction est soumise
- d 'autres processus (participants) qui s 'exécutent sur les sites contenant les données auxquelles accède cette transaction.

Bruno.Sadeg@univ-lehavre.fr

Hypothèses

- Données non répliquées (chaque donnée est localisée sur un seul site)

  => il y a un seul ORD et un seul GD pour contrôler les accès à toute donnée de la base.

- Le GT dans le site maître diffuse les opérations de la transaction aux sites appropriés. Une opération *Read(x)* ou *Write(x)* est diffusée sur le site où x est stockée. Elle est exécutée par l 'ORD et le GD de ce site comme une opération locale.
- **MAIS l 'opération de COMMIT est plus compliquée.**

Bruno.Sadeg@univ-lehavre.fr

Le protocole 2PC :

En supposant l ’absence de panne, il s’énonce ainsi :

- **1.** le coordinateur envoie un message VOT-REQ (requête de vote) à tous les participants.

- **2.** Quand un participant reçoit VOT-REQ, il répond en envoyant au coordinateur un message contenant son vote : OUI ou NON. Si le participant a décidé d ’abandonner la transaction (Abort), il fait Abort et s ’arrête. S ’il a décidé de voter OUI, il envoie OUI et il entre dans sa période d ’incertitude où il ne peut plus faire COMMIT ou ABORT unilatéralement : il doit attendre la décision du maître. Il est **incertain**.

Bruno.Sadeg@univ-lehavre.fr

Le protocole 2PC (suite)

3. Le coordinateur récolte les messages-votes de tous les participants. Si tous ces messages sont *OUI* et le vote du coordinateur est *OUI*, alors le coordinateur décide de Commiter et envoie le message *COMMIT* à tous les participants. Sinon le coordinateur décide d 'abandonner, il fait Abort et envoie le message *ABORT* à tous les participants ayant voté *OUI* (les autres auront déjà décidé de faire un *ABORT* à l 'étape 2).

4. Chaque participant ayant voté *OUI* attend de recevoir un message *COMMIT* ou *ABORT* du coordinateur. A la réception du message, le participant décide en conséquence et s 'arrête.

Bruno.Sadeg@univ-lehavre.fr

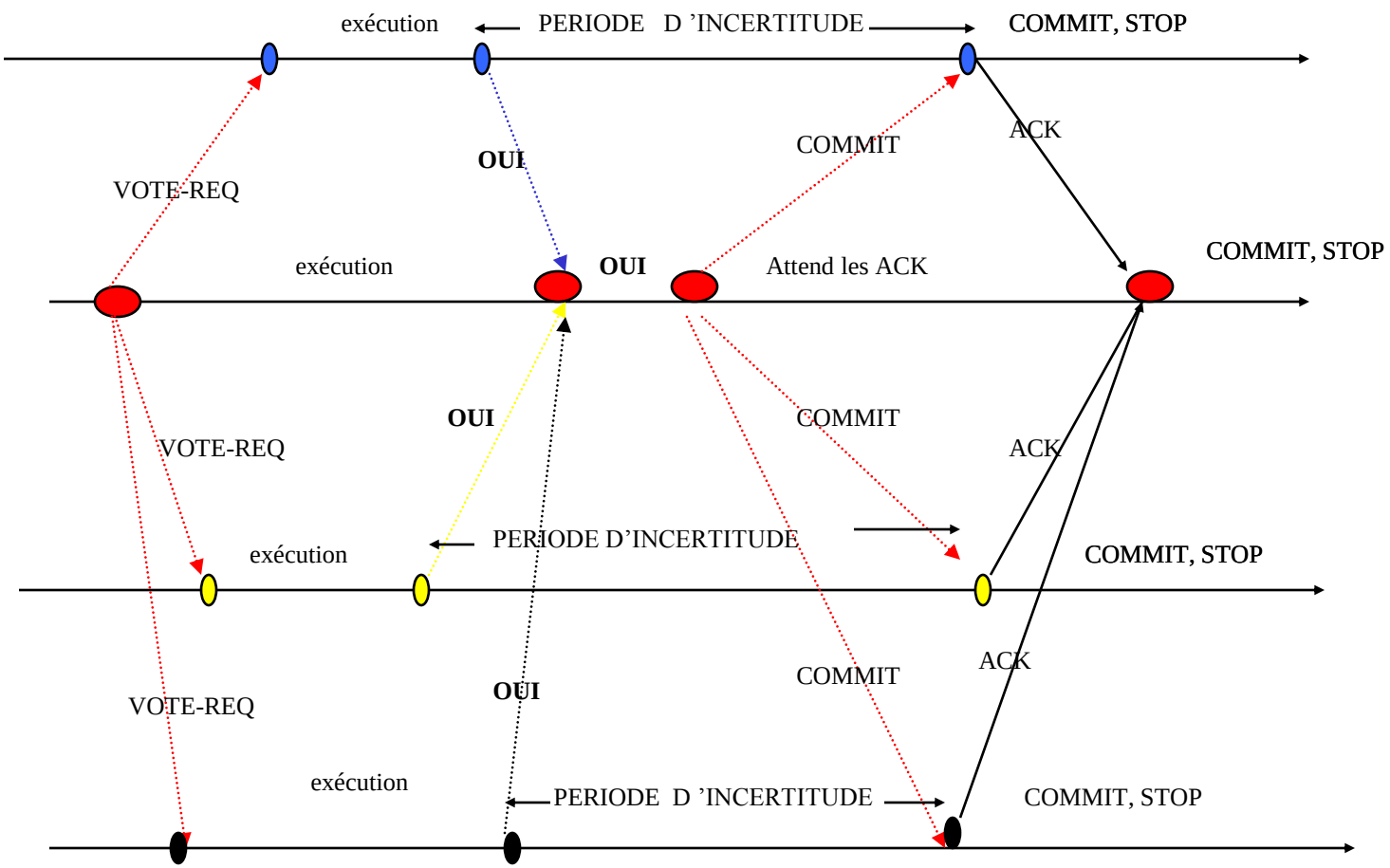
Satisfaction des conditions AC_i

Les 2 phases du 2PC sont la phase de vote (étapes 1 et 2) et la phase de décision (étapes 3 et 4). La période d 'incertitude d 'un participant commence quand il envoie le vote *OUI* au coordinateur (étape 2) et se termine quand il reçoit le message *COMMIT* ou *ABORT* (étape 4).

- On voit aisément que le 2PC satisfait les conditions $AC_1 \ldots AC_4$ mais ne satisfait pas AC_5 => Deux raisons : (1) en différents points du protocole, les process doivent attendre des messages pour s 'exécuter. Ces messages peuvent ne pas arriver à cause des pannes. Donc les process peuvent attendre indéfiniment. Une solution : les *timeouts*. (2) quand un process revient d 'une panne, la condition AC_5 stiplule que le process doit atteindre une décision consistante avec les décisions que les autres process pourraient avoir pris (il peut arriver que cette décision ne peut être prise qu 'après que d 'autres éventuelles pannes soient réparées). Donc, pour satisfaire AC_5, le process doit stocker certaines informations (voir plus loin) en mémoire stable (dans le DT log).

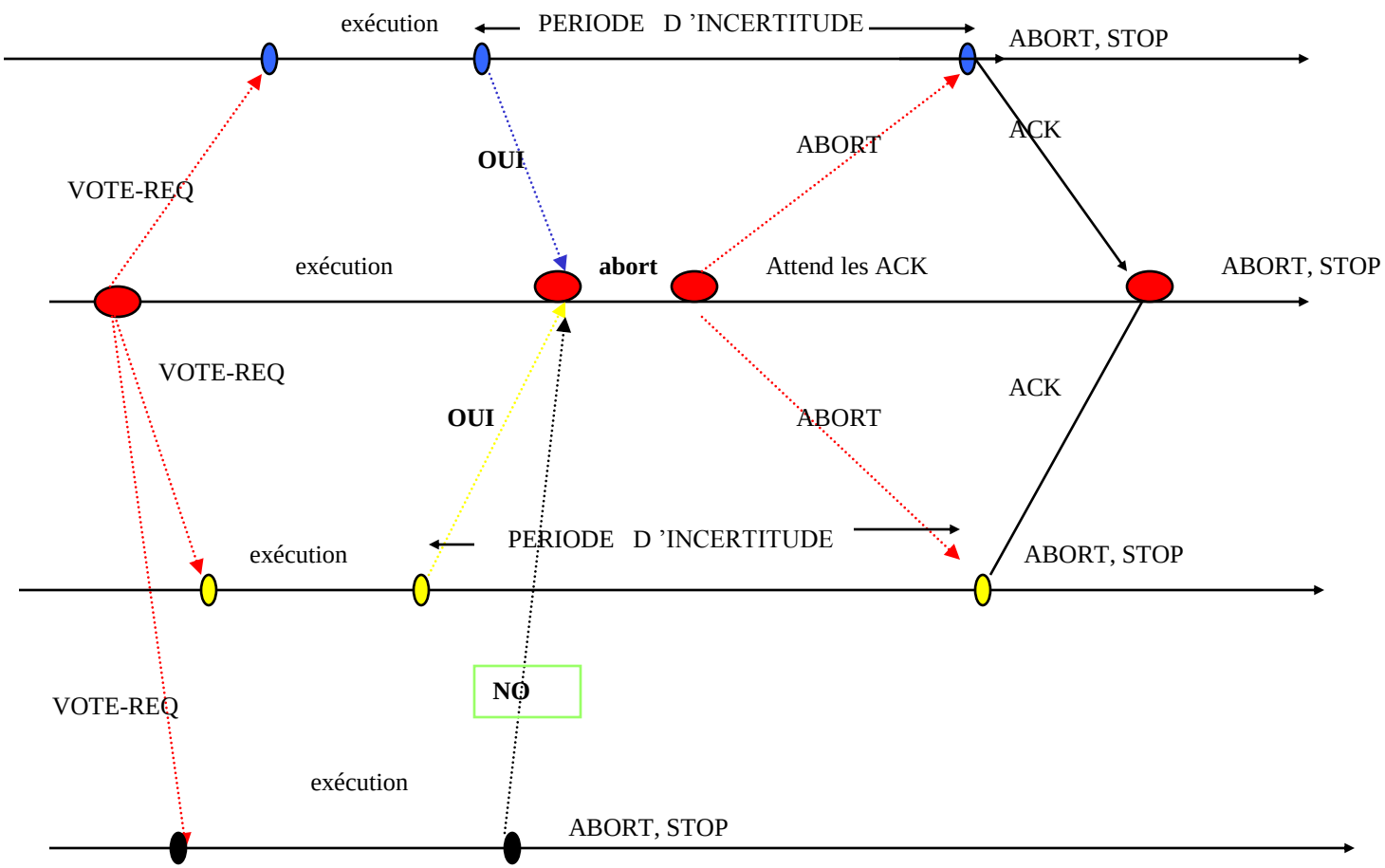
Bruno.Sadeg@univ-lehavre.fr

COMMIT



Bruno.Sadeg@univ-lehavre.fr

ABORT



Bruno.Sadeg@univ-lehavre.fr

Actions en cas de timeout : le protocole de terminaison

Il existe 3 places où le 2PC est en attente d ’un message :

- **1.** Au début de l ’étape 2 : le participant attend la demande de vote du coordinateur. Si timeout, il fait Abort et s’arrête

- **2.** au début de 3 : le coordinateur attend les votes (OUI ou NON) des participants. Si timeout, le coordinateur décide de faire ABORT et doit envoyer ABORT à tous les participants desquels il a reçu OUI.

Bruno.Sadeg@univ-lehavre.fr

Actions en cas de timeout : le protocole de terminaison (suite)

• **3.** au début de 4 : le participant *p* ayant voté OUI attend un message *Commit* ou *Abort* du coordinateur. A ce point, *p* est *incertain* et ne peut donc pas prendre une décision unilatéralement.
=> Il doit consulter les autres sites pour décider.
⇒cela se fait au cours du mécanisme de *protocole de terminaison* (pour 2PC) : le + simple est que p attendre le rétablissement de la liaison avec le coordinateur. Mais inconvénient : p peut attendre (bloqué) inutilement.

Ex. 2 participants p et q attendent la décision du coordinateur. Si q la reçoit mais que p ne la reçoit pas (panne juste avant de lui envoyer la décision). Si p pouvait communiquer avec q, il aurait reçu la décision de la part de q sans attendre.

Bruno.Sadeg@univ-lehavre.fr

Actions en cas de timeout : le protocole de terminaison coopératif

- => **suggestion** : que les participants puissent communiquer entre eux. Cela est possible car le coordinateur peut envoyer à chacun la liste des participants (avec le message VOT-REQ).

  => Le protocole de terminaison coopératif : un participant qui fait un timeout envoie à tous les participants q le message DECISION-REQ pour voir si l'un d'eux connaît la décision : p est l'initiateur, q est le répondeur

- => 3 cas possibles :

Bruno.Sadeg@univ-lehavre.fr

1- q à déjà décidé Commit (ou Abort). Il envoie sa décision à p qui décide en conséquence.

2- q n ’a pas encore voté. Il peut décider de faire Abort et envoie le message Abort à p qui décide à son tour de faire Abort.

3- q a voté oui mais n ’as pas encore arrêté sa décision : q est incertain. Donc, il ne peut pas aider p.

si cas 1 et 2 : p n’est pas bloqué,

si cas 3 (pour tout process avec lequel p communique) : p reste bloqué jusqu’à ce que assez de sites soient réparés pour lesquels le cas 1 ou 2 s ’applique.

=> au moins un tel site existe : le coordinateur

Bruno.Sadeg@univ-lehavre.fr

Le recouvrement (après panne)

- Supposons que p revient d'une panne. Pour satisfaire la condition AC5, il doit prendre une décision consistante avec celle des autres:

  – Si p tombe en panne avant d'avoir envoyé son vote OUI au coordinateur, alors p peut faire Abort unilatéralement.

  – Si p tombe en panne après avoir reçu le message *Commit* ou *Abort* du coordinateur ou après avoir décidé de faire *Abort*, alors *p* aura pris une décision.

  => dans ces 2 cas, *p* **peut recouvrir** de façon indépendante (sans communication)

  – Si p tombe en panne pendant sa période d 'incertitude, il ne peut pas décider seul. Dans ce cas, *p* se trouve exactement dans le cas où il a fait un Timout (attente de la décision du coordinateur et dépassement du temps d'attente) => *p* atteint sa décision en appliquant un *protocole de terminaison*.

  (Noter que dans ce cas aussi, *p* peut être bloqué car il peut arriver qu'il ne puisse communiquer qu'avec des participants *incertains*).

Bruno.Sadeg@univ-lehavre.fr

Gestion des fichiers DT log (journaux)

Pour se rappeler de son état avant une panne, chaque process gère un DT log local. En supposant que le protocole de terminaison coopératif est en vigueur, le DT log est géré de la façon suivante :

1. Avant ou après que le coordinateur ait envoyé VOTE-REQ aux participants, il écrit start-2PC (identités des participants) dans son DT log.

2. Si un participant vote OUI, il écrit d ’abord un enreg. OUI dans son DT log, puis envoie sa décision OUI au coordinateur. S ’il vote NON, il écrit un enreg. Abort dans son DT log avant ou après avoir envoyé sa décision NON au coordinateur.

Bruno.Sadeg@univ-lehavre.fr

Gestion des fichiers DT log (suite)

3. Avant que le coordinateur n'envoie un message COMMIT aux participants, il écrit un enreg COMMIT dans son DT log.

4. Quand le coordinateur envoie le message ABORT aux participants, il écrit (avant ou après l 'envoi) un enreg ABORT dans son DT log.

5. Après avoir reçu COMMIT (ou ABORT), un participant écrit un enreg COMMIT (ou ABORT) dans son DT log.

DONC : l'écriture d'un enreg COMMIT ou ABORT dans le DT log matérialise le COMMIT ou l'ABORT d'un processus.

Bruno.Sadeg@univ-lehavre.fr

- Processus de Commit et exécution de la transaction

- Une fois qu'un enreg Commit (ou Abort) est écrit dans le DT log, le GD du site peut exécuter l'opération de Commit (ou de Abort).
- Quand un site S revient d'une panne, le sort de la transaction distribuée s 'exécutant sur S est déterminé en examinant le DT log :

  > Si le DT log contient un enreg start-2PC, alors S est le site du coordinateur. S 'il contient aussi un enreg Commit ou Abort alors S a déjà pris une décision avant la panne. S'il ne contient pas Commit ou Abort, alors le coordinaeur peut décider de faire Abort unilatéralement (en insérant un enreg Abort dans son DT log en envoyant la décision aux participants ayant voté OUI)

  > Si le DT log ne contient pas d'enreg start-2PC, alors S est un site participant

  ➔ dans ce cas, 3 cas sont possibles :

Bruno.Sadeg@univ-lehavre.fr

- **1.** Le DT log contient un enreg *Commit* ou *Abort*, alors S a déjà pris sa décision avant la panne

- **2.** Le DT log ne contient pas un enreg *OUI* alors le participant est tombé en panne avant d 'avoir voté *OUI* ou a déjà voté *NON* (mais n 'a pas eu le temps d 'écrire un enreg *Abort*). S peut donc faire *Abort* unilatéralement en insérant un enreg *Abort* dans son DT log.

- **3.** Le DT log contient un enreg *OUI* mais pas d 'enreg *Commit* ou *Abort*. Donc S est tombé en panne pendant qu'il est dans sa période d 'incertitude. Il peut essayer d 'aboutir à une décision en utilisant un protocole de terminaison.

Bruno.Sadeg@univ-lehavre.fr

Le protocole PA (Presumed Abort)

- PA est une variante du 2PC qui essaie de réduire le nombre de messages échangés et l ’*overhead* dû à l ’écriture des *log*. PA requiert que tous les sites participants de retour d ’un incident suivent la règle du « en cas de doute, abandonner ». C-à-d qu ’après une panne, le site défaillant interroge le maître sur le devenir de la transaction : s ’il ne reçoit aucune information, il considère que la transaction est (correctement) abandonnée. Avec cette hypothèse (a) les participants n ’ont pas à envoyer des ACK au maître pour une transaction abandonnée et (b) n ’ont pas à écrire sur disque les *abort log record*. De plus, le maître n ’a pas à écrire un *abort log record* ni un *end log record*.
- En résumé, PA se comporte comme le 2PC pour les transactions qui font leur *COMMIT*, mais réduit les *overhead* et le nombre de messages échangés pour les transactions abandonnées.

Bruno.Sadeg@univ-lehavre.fr

Le protocole PC (Presumed Commit)

- PC est une variante de PA venu du constat qu ’en général, le nombre de transactions validées (commit) est très supérieur au nombre de transactions abandonnées (Abort).

  Dans le protocole PC, les *overhead* sont réduits pour les transactions validées en requérant que tous les sites participants suivent la règle du «en cas de doute, valider» après le retour d ’une panne. Dans ce schéma, les participants n’envoient pas d ’ACK après la décision du *commit* global et n ’écrivent pas sur disque le *commit log record*. Et le maître n ’écrit pas le *end log record*. Cependant, le maître doit écrire sur disque les *commit log record* collectés avant d’initier le protocole à 2 phases. Ce *log record* contient les noms de tous les participants intervenant dans l ’exécution de cette transaction.
- Ces optimisations du 2PC sont implémentés sur de nombreux produits commerciaux et PA fait partie des standards d ’exécutions de transactions distribuées ISO-OSI et X/OPEN.

Bruno.Sadeg@univ-lehavre.fr

Le protocole 3PC (Three Phase-Commit)

- L'inconvénient des protocoles précédents est que les participants peuvent être bloqués si une panne d 'un site survient et demeurent bloqués jusqu'à ce que le site défaillant soit réparé.

Par exemple, si le site maître tombe en panne, après avoir initié le protocole mais avant d'envoyer la décision aux participants, alors les participants deviennent bloqués et le restent jusqu'à ce que le maître soit réparé et les informent de la décision finale.

Durant le blocage, les participants continuent à accaparer les ressources systèmes (verrous sur les données, …) les rendant indisponibles pour d'autres transactions, qui, à leur tour restent bloquées en attente de ces ressources ➔ blocages en cascade. Il est aisé de voir que si la période de blocage est grande, il peut résulter une perturbation de l'activité des transactions. Le protocole 3PC est proposé pour résoudre ce problème.

Bruno.Sadeg@univ-lehavre.fr

Le protocole 3PC (suite)

- Le 3PC évite les blocages en insérant une autre phase (la phase de précommit) entre les 2 phases du protocole 2PC.

Dans la phase de *précommit*, une décision préliminaire est prise concernant le sort de la transaction. L'information rendue disponible aux sites participants comme résultat de cette décision préliminaire permet de prendre la décision globale malgré la survenue ultérieure d'une panne du site maître.

Noter cependant que le prix à payer pour éviter les blocages est le surcroît d'*overhead* de communication puisqu'il y a des échanges de messages supplémentaires entre le maître et les participants durant la phase de *précommit*. De plus, les participants et le maître ont à écrire sur disque des *log record* additionnels durant la phase de *précommit*.

Bruno.Sadeg@univ-lehavre.fr

Autres protocoles

- Il existe d'autres protocoles sont proposés pour les systèmes de bases de données distribuées connectées par des réseaux très rapides (Linear 2PC, Distributed 2PC, Unsolicited Vote (UV), Early Prepare (EP), Coordinator Log (CL), Implicit Yes Vote (IYV), Two-Phase Abort (2PA)

Bruno.Sadeg@univ-lehavre.fr

Processus de *commit* optimistes

- Dans les protocoles précédents, un participant qui atteint sa phase de préparation libère tous les verrous en lecture qu'il détient, mais doit garder les verrous de mise à jour jusqu'à réception de la décision du site maître pour assurer l'atomicité de la transaction distribuée.

L'intervalle de rétention des verrous n'est pas borné puisque la durée pendant laquelle le participant est dans la phase de préparation peut être arbitrairement longue (à cause des retards du réseau par exemple) => Provoque des effets négatifs sur les performances, puisque d'autres transactions désirant accéder à ces données (préparées) sont forcées à être bloquées jusqu'à la fin du processus de *commit*.

Noter que ce blocage à cause des données est *orthogonal* au blocage en attente de décision, c-à-d que dans tous les protocoles de commit précédents (3PC inclus) les transactions peuvent être affectées par le blocage à cause des données. De plus ce blocage survient durant le processus normal alors que le blocage en attente de décision survient seulement dans les situations de pannes d'un site.

Bruno.Sadeg@univ-lehavre.fr

Processus de commit optimistes (suite)

- Pour palier le problème du blocage dû aux données préparées : le protocole OPT (Gupta, Haritsa, Ramamritham).
- Est une nouvelle version du 2PC où des transactions désirant accéder à des données détenues par d 'autres transactions en phase de préparation peuvent le faire, c-à-d que les participants préparés prêtent des données *non commitées* aux transactions concurrentes. => 2 situations :

- **1.** Le prêteur reçoit la décision en premier : le prêteur reçoit la décision globale avant que l 'emprunteur ne finisse son exécution locale. Si la décision est de *commiter* alors le prêteur complète son processus de façon normale. Si la décision est d 'abandonner alors le prêteur est abandonné de façon normale; de plus l 'emprunteur est également abandonné car il aura utilisé des données inconsistantes.

Bruno.Sadeg@univ-lehavre.fr

Processus de *commit* optimistes (suite)

- **2.** L ’emprunteur finit son exécution en premier : l ’emprunteur finit son exécution avant que le prêteur ne reçoive la décision globale. L ’emprunteur est alors en état d ’attente et n ’est pas autorisé à envoyer un message WORKDONE à son site maître. C-à-d que l ’emprunteur n ’est pas autorisé à initier le processus qui le mènerait éventuellement à atteindre son état de préparation. L ’emprunteur doit attendre que le prêteur reçoive la décision globale. Si la décision est de *commiter*, l ’emprunteur est autorisé à envoyer un message WORKDONE à son maître. Si la décision est d ’abandonner, le prêteur abandonne et l ’emprunteur abandonne également car il aura utilisé des données inconsistantes.
- **En résumé :** le protocole OPT autorise les transactions à accéder à des données non *commitées* détenues par des transactions en phase de préparation dans l’espoir (optimisme) que ces données seront *commitées (c-à-d que les transactions qui les ont fourni valideront).*

Bruno.Sadeg@univ-lehavre.fr

Protocole de commit distribué temps réel

- Les protocoles optimistes commencent à être adaptés au contexte temps réel, plus que les protocoles bloquants, car les transactions ont plus de chances de respecter leurs échéances.

- **Exemple** : le protocole PROMPT = le protocole OPT précédent auquel sont ajoutées 3 caractéristiques. Il est l'un des premiers protocoles conçus spécialement pour le temps réel. Ces caractéristiques sont :
- l 'Abandon actif (active abort), l 'assassinat silencieuse (silent kill), le prêt sein (healthy lending).

Bruno.Sadeg@univ-lehavre.fr

- Abandon actif : le participant qui fait *Abort* prévient immédiatement le maître, qui transmet ensuite cette décision aux sites participants

  => ce qui améliore le respect des CT

- Assassinat silencieux : pour une transaction « tuée » (dû au dépassement d ’échéance), il n ’est pas nécessaire au maître d ’invoquer le protocole d ’Abort car les participants peuvent être au courant du dépassement (présence d’une horloge globale). Donc, la transaction est « tuée » sans besoin de communication entre le maître et les participants.

Bruno.Sadeg@univ-lehavre.fr

Processus de commit temps réel

- prêt sein : seules les transactions « préparées » loin de leur échéance peuvent prêter des données non *commitées* aux autres transactions

  => ce qui évite les abandons en cascade car sinon (si on permet aux transactions proches de leur échéance de prêter des données), elles ont plus de risque d'être abandonnées (dépassement d'échéance) et entraîner ainsi des abandons en cascade.

Des simulations ont été menées pour comparer les performances du protocole PROMPT avec, d'une part, le protocole 2PC et d'autre part, avec un protocole d'héritage de priorité. Dans les deux cas, PROMPT s'était avéré meilleur (il fournit de meilleures performances), c-à-d que davantage de transactions respectent leurs échéances.

Bruno.Sadeg@univ-lehavre.fr

Conclusion

- Le processus de COMMIT distribué dans un contexte temps réel : au stade de la recherche. Peu de travaux actuellement.
- Objectif des ces protocoles :

- --> Tenir compte de la cohérence de la base et des échéances des transactions
- --> le maximum de transactions respectent leurs échéances

- dans un contexte distribué, en présence de pannes de sites et/ou de réseaux éventuelles.

Bruno.Sadeg@univ-lehavre.fr

Some references

- **[BES 87]** Bernstein P.A, Hadzilacos V., Goodman N., “Concurrency Control and Recovery in Database Systems”, *Addison-Wesley Publishers*, 1987.
- **[ESW 76]** Eswaran K.P., Gray J.N., Lorie R.A. and Traiger I.L., “ The notion of consistency and predicate locks in a database system ”, *CACM*, vol. 19, n° 11, pp. 624-633, 1976.
- **[HAR 99]** Haritsa J., Ramamritham K. and Gupta R., “The PROMPT Real-Time Commit Protocol”, IEEE Transactions on Parallel and Distributed Systems, vol. 11, n° 2, pp. 160-181, 1999.
- **[PUR 94]** Purimetla B., Sivasankaran R.M., Ramamritham K. and Stankovic J.A., “ Real-time databases: issues and applications ”, *Principles of RTS*, ed. Printice Hill, 1994.
- **[SON 92]** Son S.H. and Liu J.W.S., “ How well can data temporal consistency be maintained ? ”, *IEEE Symp. on Computer-Aided Control System Design*, 1992.

Bruno.Sadeg@univ-lehavre.fr