Bases de données réparties

Nacer.Boudjlida@loria.fr

http://www.loria.fr/~nacer

Université de Lorraine, FST/MIAE, ESIAL

Support de cours. Des compléments importants seront donnés en cours.

Chapitre I : Introduction

Bases de données distribuées : Définitions

- Intégration des technologies bases de données (BDD) et réseau
- Intégration des données d'une entreprise sans centralisation
- Distribution "naturelle" :
  - "Agence mère" et ses filiales
  - Unité centrale et sous-système d'entrées/sorties, etc.
- *Définition orientée bases de données* :

  Système de traitement distribué = *(1) Unités d'exécution* de programmes, *(2) autonomes*, éventuellement *(3) hétérogènes*, *(4) reliés par un réseau* de communication et *(4) coopérant* à la réalisation de tâches.

Bases de données distribuées : Définitions (fin)

- *Base de données distribuée (DDB)* :

  Collection de BDD logiquement *reliées* et physiquement *distribuées* sur un réseau
- *SGBD Distribué* :

  Système logiciel de gestion rendant les applications insensibles à la distribution des données (TRANSPARENCE)
- *Base de données distribuée N'EST PAS* :
  - {Fichiers indépendants} sur des sites différents,
  - Base centralisée accessible via le réseau

QUOI distribuer ?

- Données
- Traitements
- Fonctions du système
- Contrôle et coordination des tâches

Problèmes liés à la distribution

- *Conception des bases distribuées* :
  - Partitionnement des données (FRAGMENTATION, LOCALISATION)
  - DUPLICATION totale/partielle
  - Administrateur global, local
- *Dictionnaire(s)*
  - Extension avec des informations sur les sites, la fragmentation, la duplication et la localisation des données
  - Centralisé ou distribué ?
  - Copie simple ou multiple ?

Problèmes liés à la distribution (suite)

- *Traitement des requêtes* :
  - *Objectif* : Optimiser en exploitant le parallélisme
  - Problème “NP-complexe” : Approches heuristiques
  - *Facteurs* :
    - Localisation/Duplication des données
    - Coût des communications
    - Disponibilité locale d’informations suffisantes
  - *Stratégies* : Sous-requêtes, jointures parallèles, semi-jointure, etc.

Problèmes liés à la distribution (suite)

- *Accès concurrents*
  - Contrôle réparti des accès concurrents prenant en compte la duplication
  - Techniques : verrouillage, estampillage
  - Traitement du deadlock distribué ?

Problèmes liés à la distribution (suite)

- *Sécurité et reprise en cas d’incident*
  - Nouveaux types d’incidents : perte de messages, panne d’une liaison, etc.
  - Préservation des bases des sites opérationnels en cas de panne d’un site
- *Confidentialité* sur un réseau

Problèmes liés à la distribution (fin)

- *Hétérogénéité*
  - Matériels/Logiciels
  - Modèles de données et Langages de manipulation
  - Introduite aussi dans les systèmes MULTI-BASEs : regroupement de BDD centralisées autonomes
  - Mécanimes de traduction (données, programmes)
- *Systèmes hôtes* :

  Quelles solutions (homogènes) pour supporter les applications distribuées et celle non distribuées (TRANSPARENCE) ?

Bases de données réparties : Contenu

Chapitre II : Typologie des SGBD distribués

Caractéristiques d'un SGBD distribué

1. *Transparence du réseau*
   - Nommage des objets
   - Localisation des objets
   - Services (Exemple : *cp f1,f2; rcp site:f1, site:f2*)

2. *Transparence de la duplication* : Points de vue
   - Utilisateur : *"Duplication ? Connais pas !"*
   - Système : implications en recherche, en mise à jour, en gestion des accès concurrents

Caractéristiques d'un SGBD distribué

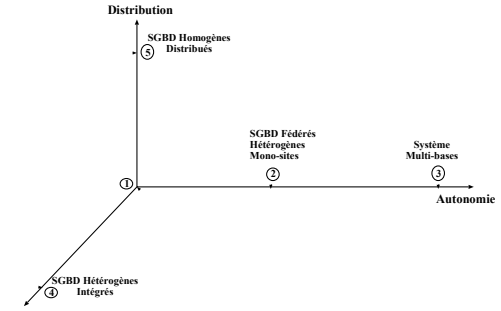
3. *Transparence de la fragmentation* dans le traitement des requêtes
   - Requête sur UNE relation (utilisateur)
   - Décomposée en sous-requêtes (système)
   - Sous-requêtes évaluées sur les fragments (système)
   - Union des réponses (système)

4. *Transparence des langages dans le cas de DDB hétérogènes*

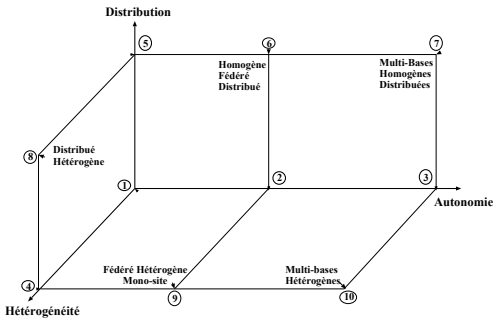
Critères de classification

1. *AUTONOMIE*
   - Concerne la distribution du contrôle
   - Mesure le degré d'indépendance
   - Fonction de :
     - Volume des échanges inter-sites
     - Capacités d'exécution indépendamment des autres sites
2. *DISTRIBUTION* : (ou pas) des données
3. *HETEROGENEITE*
   - Hardware, protocoles réseaux
   - Modèles de données : puissances d'expression
   - Langages : paradigmes, dialecte(s)

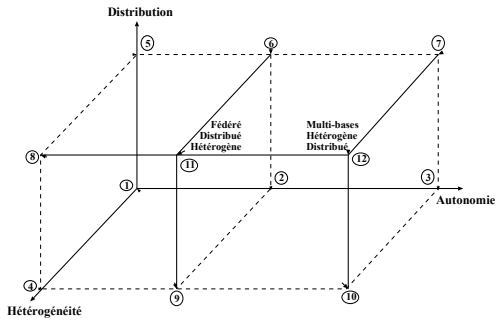
Typologie (1/3)



Typologie (2/3)

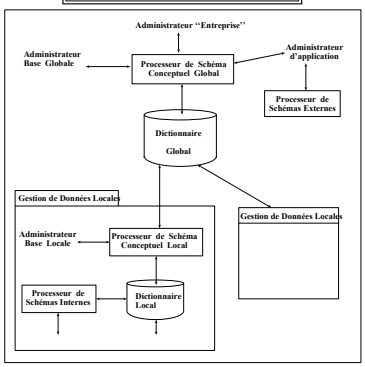


Typologie (3/3)



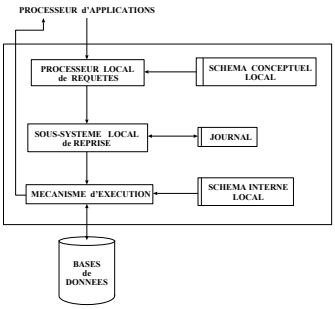
Architecture de référence ANSI/SPARC (1/2)

Architecture de référence (2/2)



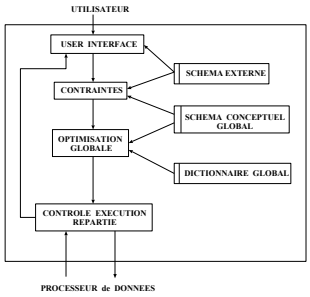
Composants fonctionnels d'un SGBD distribué

1. *Processeur de données*



Composants fonctionnels d'un SGBD distribué

2. *Processeur d'applications*



Chapitre III : Distribution de données

III.1- Fragmentation et duplication de données

- *DUPLICATION TOTALE* :
  - **(+)** Disponibilité des données
  - **(+)** Parallélisme en lecture
  - **(+)** Moindre flux sur le réseau
  - **(-)** Coût des mises à jour

- *FRAGMENTATION* :
  - Découpage d'une relation R en *fragments* $R_1, R_2, \cdots, R_n$
  - *Fragmentation Horizontale* : Reconstruction par *UNION*
  - *Fragmentation Verticale* : Reconstruction par *JOINTURE*

Fragmentation horizontale

- Peut être définie par une sélection
- Fragments disjoints ou non (*Duplication partielle*)

Fragmentation verticale

- Forme la plus simple : Décomposition de R
- "Identifiant" (clé) dans chaque fragment

Fragmentation mixte et duplication

1. *FRAGMENTATION MIXTE* : horizontale et verticale
2. *FRAGMENTATION et DUPLICATION* :
   - Duplication de fragment(s)
   - Fragmentation de duplicata, etc.

- Convention de *nommage des fragments et des copies* :
  - *Désignation_Objet*•F_i : Fragment
  - *Désignation_Objet*•C_i : Copie
  - Exemple : *Site2.Produit.F3.C4*

III.2- Transparence et autonomie

1. *Transparence* :
   - du réseau
   - de la localisation
   - connaissance minimale sur les lieux de rangement des données
2. *Autonomie* :
   - Liée à la transparence
   - Degré d'indépendance vis-à-vis du système distribué

- *Analyse selon les points de vue* :
  (1) Nommage des objets
  (2) Fragmentation et Duplication des données
  (3) Localisation

(1/3) Transparence et autonomie : Nommage des objets

- Assurer *l'unicité* des noms
- *Solution 1* : Serveur de noms centralisé
  (-) Autonomie locale
  (-) Surcharge du serveur ; Blocage en cas de panne

- *Solution 2* : *Nom_du_site*•*Nom_Objet*
  (+) Pas de contrôle centralisé : Meilleure autonomie locale
  (-) Pas de Transparence du réseau

(2/3) Transparence : Fragmentation et duplication

- *DUPLICATION* :
  - Utilisateur : Pas de désignation de copie
  - *Quelle copie accéder ?*
  - *Quelle(s) copie(s) mettre à jour ?*

- *FRAGMENTATION* :
  - Utilisateur : Pas de désignation de fragment
  - *Comment localiser ?*

(3/3) Transparence : Localisation

- Définition d'*alias* (sans nom du site)

  (+) (Légère) transparence à l'utilisation

  (+) Tranparence si changement de localisation

- *Exemple* : Site1 : alias *compte_local* pour le fragment Site1.compte.F1
- *Principe de la localisation*
  1. Remplacer *compte_local* par *Site1.compte.F1*
  2. Si *Site1.compte.F1* dupliqué
     - Accès à la table des copies (dictionnaire)
     - Choisir une copie
  3. Si copie fragmentée $\Longrightarrow$ Consulter la table des fragments

Schéma d'algorithme de localisation

Chapitre IV : Traitement des requêtes

- Processus de traitement des requêtes :

Traitement des requêtes

- *PLAN d'EXECUTION REPARTI* :
  - {*Traitements Locaux* et Opérations de *Communication de données* intermédiaires}
  - Comprend le *nécessaire* pour les *exécutions locales* et la *synchronisation globale*
- *Et la fragmentation* ?
  - *Représentation canonique* de requête algébrique
  - Identification et localisation des fragments
  - *Extension* de la représentation canonique par les sous-arbres de *reconstruction des fragments*

Traitement des requêtes : Exemple

- $Prod1 = \Pi_{prod\#,libelle}(Produit(\underline{prod\#}, libelle, pu))$
- $Prod2 = \Pi_{prod\#,pu}(Produit)$
- $Stock1 = \sigma_{dep\#=4}(Stock(\underline{prod\#, dep\#}, qte))$
- $Stock2 = \sigma_{dep\#<>4}(Stock)$
- *Libellés des produits du depôt 4 ?*

```
select libelle from Produit p, Stock s
where s.prod# = p.prod# and s.dep# = 4
```

$\Pi_{libelle}(\bowtie_{(prod\#=prod\#)} (Produit, \sigma_{(dep\#=4)}(Stock)))$

Traitement des requêtes : Règles de transformation

- *REGLES "CLASSIQUES"* de transformation
- *AUTRES REGLES* :
  1. Condition de sélection et expression de fragmentation *identiques* $\Longrightarrow$ *Sélection INUTILE*
  2. Fragmentation horizontale : *Résultat vide si*
     - Sélection
     - Condition de sélection et expression de fragmentation *CONTRADICTOIRES*
  3. Fragmentation verticale : *Résultat vide si*
     - Fragmentation verticale de liste L'
     - Projection de liste L
     - $L \setminus \{Attribut(s)\ de\ reconstruction\}$ $\not\subset$ $L' \setminus \{Attribut(s)\ de\ reconstruction\}$

Optimisation : Objectifs et facteurs

- Minimiser une fonction coût
- Fonction générale : $\sum_{i=1}^{n} Temps\_Execution_i$ (n : nombre de sites impliqués)
- Autres : Tenir compte
  - du parallélisme
  - des coûts des transferts
  - des *profils* des fragments (Taille, nombre de n-uplets, etc.)
  - de la taille des résultats intermédiaires
  - de l'instant de l'optimisation (compilation/exécution)
  - de la topologie du réseau
  - du coût de l'optimisation, etc.

Optimisation : Types de décisions

- *Décisions incluses dans le plan d'exécution*
  - Ordre des jointures
  - Stratégie de jointure (voir plus loin)
  - Sélection des copies (site le plus proche, le moins engorgé)
  - Choix des sites d'exécution (coûts des communications)
  - Choix des algorithmes d'accès répartis
  - Choix du mode de transfert (tuple/tuple, paquets)

Stratégies d'évaluation des requêtes

- En centralisé : coût des accès
- Cadre distribué :
  - Transmission de données
  - Traitement parallèle des (sous-)requêtes
- Attention particulière aux jointures
- *STRATEGIES* :
  1. Jointures simples avec transferts
  2. Jointures parallèles
  3. Semi-Jointures

Stratégie 1/3 : Jointures simples

- Pas de fragmentation ni de duplication
- SITE S_Q : *Produit* $\bowtie$ *(Depot* $\bowtie$ *Stock)*
- *Produit* : Site S_{Pr}
- *Depot* : Site S_{De}
- *Stock* : Site S_{St}
- *Site émetteur de la requête* : S_Q

Stratégie 1/3 : Jointures simples

- Jointures simples : Stratégie 1
  1. Transfert des relations sur S_Q
  2. Optimisation sur S_Q
  3. Exécution locale

Stratégie 1/3 : Jointures simples

- Jointures simples : Stratégie 2
  1. Transfert de *Stock* vers S_{De}
  2. Calcul de *TEMP1 = (Depot* $\bowtie$ *Stock)*
  3. Transfert de *TEMP1* vers S_{Pr}
  4. Calcul de *TEMP2 = (Produit* $\bowtie$ *TEMP1)*
  5. Transfert de *TEMP2* vers S_Q

Jointures simples : Discussion

- *STRATEGIE 1* :
  - Reconstruction éventuelle des index sur S_Q
  - Volume des transferts
- *STRATEGIE 2* : Volume des transferts

Stratégie 2/3 : Jointures parallèles

- $(R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4)$
- R_i sur site S_i
- Jointures parallèles :
  1. Sur S_1 : $(R_1 \bowtie R_2)$
  2. Sur S_3 : $(R_3 \bowtie R_4)$
  3. Envoi des nuplets vers S_Q au fur et à mesure du calcul
  4. S_Q évalue $(R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4)$ *parallèlement* aux évaluations sur S_1 et S_3

Stratégie 3/3 : Semi-Jointure ($\propto$)

- $(R(X) \propto S(Y)) = \Pi_X(R \bowtie S)$
- Stratégie permettant d'éviter le transfert de tuples non utiles pour la jointure
- *EXEMPLE* :
  - $R_1 \bowtie R_2$
  - R_1, R_2 sur sites S_1, S_2, respectivement
  - Stratégie Semi-Jointure, résultat sur S_1
    1. S_1 : $TEMP1 = \Pi_{X \cap Y}(R_1)$
    2. S_1 : Envoi de *TEMP1* à S_2
    3. S_2 : $TEMP2 = R_2 \propto TEMP1$
    4. S_2 : Envoi de *TEMP2* à S_1
    5. S_1 : $RES = R_1 \bowtie TEMP2$

Stratégie 3/3 : Semi-Jointure ($\propto$)

- Exemple :
- Correction de la stratégie : Démontrer que $R_1 \bowtie R_2 = (R_1 \bowtie (R_2 \propto (\Pi_{X \cap Y}(R_1))))$?

Stratégie 3/3 : Discussion

- Hypothèse : Sélectivité faible
- $CARD(R_1 \bowtie R_2) < CARD(R_2)$
- Transfert de $(R_2 \propto R_1)$ moins coûteux que celui de R_2
- Ce gain devrait compenser le coût du transfert de $\Pi_{X \cap Y}(R_1)$
- Augmentation du nombre d'opérations
- MAIS sur de plus petites relations
- Gain proportionnel à la sélectivité de la jointure

Traitement des requêtes : Cas des Vues

- *Rappel* : Vue = relation calculée
- Vue définissable sur fragments répartis
- Traitement de requêtes sur vues
  1. Par *ré-écriture*
  2. En utilisant des *vues concrètes*
  3. En utilisant des *clichés* (*Snapshots*)

1/3. Requêtes sur vues et ré-écriture

1. Requête Q_V sur vue ré-écrite en une requête Q_R sur relations
2. Traiter Q_R comme une requête répartie

2/3. Requêtes sur vues et vues concrètes

- Calcul et stockage des valeurs des vues
- Mises à jour systématiques
- Duplication éventuelle sur les sites utilisateurs fréquents
- *SOLUTION ACCEPTABLE* si vue définie sur beaucoup de fragments (Diminution du coût de son "calcul")

3/3. Requêtes sur vues et clichés

- *Cliché* : Vue mise à jour périodiquement
- Réduction du coût par *mises à jour différentielles* (Vue sur une relation, définie par $\Pi_L(\sigma_E(\cdots))$
  - Estampillage des tuples de la relation
  - Estampillage du cliché
  - Calcul d'une $\triangle$relation contenant les tuples du cliché à modifier
  - Mise à jour du cliché (et de ses copies)

3/3. Requêtes sur vues et clichés

Produit

estampille	#prod	libellé	pu
10:00	11	P11	10
10:15	22	P22	20
10:30	33	P33	30
11:00	44	P44	40

Cliché 10:30

#prod	pu
11	10
22	20
33	30

$\triangle$ **Produit 11:00**

#prod	pu
44	40

Cliché 11:00

#prod	pu
11	10
22	20
33	30
44	40

Chapitre V : Contraintes d'intégrité et Confidentialité

Confidentialité

1. *Identification des utilisateurs distants* :

   Sous-transactions d'une transaction T_{Si} comportent l'identification de l'auteur de T_{Si} (Nom, Mot de passe)

2. *Identification du site émetteur* :

   Un site S_i n'accepte des (sous-)transactions de S_j que si S_i connaît S_j

Contraintes d'intégrité

- CI = Formule du calcul relationnel
- Vérifier CI :

  Produit cartésien des relations impliquées satisfait la formule
- *Solution immédiate* : CI satisfaite si

  $Vide(\sigma_{\neg CI}$(X (Relations impliquées)))
- $\sigma_{\neg CI}$(X (Relations impliquées)) : comme une requête répartie
- *Fragmentation et CI contradictoires* : Pas d'accès aux données
- *En mise à jour* : Si violation alors ROLLBACK

Chapitre VI : Transactions et Accès concurrents

- Validation ou avortement sur chacun des sites impliqués
- Validations indépendantes ? NON car certaines transactions peuvent avoir avorté
- Un gestionnaire local par site
- *Modèle de système* :
  1. *Un gestionnaire des transactions* locales et distantes
  2. *Un coordonnateur de la gestion des transactions*

Gestion des transactions

- Chaque gestionnaire de transactions :
  - *Gère* le Log
  - *Participe à la coordination* des exécutions selon un protocole
- Chaque coordonnateur :
  - *Lance* l'exécution d'une transaction
  - *Décompose* une transaction
  - *Transfère* les transactions vers les "bons" sites
  - *Coordonne* la validation

Gestion des transactions : Plan

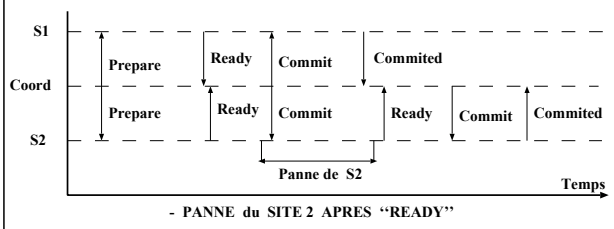
La validation des transactions

- *Protocole de validation à deux phases (Two-Phase Commit Protocol)* :
  - Lampson 1976, Gray 1978
  - Le plus connu et implémenté
  - Exécuté par chaque site
  - Contrôlé par un site maître (coordonnateur)
  - *Assure l'**A**tomicité* (sauf dans le cas de la destruction du log)
  - *Correction prouvée* dans "J.L. Baer and al.: *"The Two-Step Commitment Protocol: Modeling, Specification and Proof Methodology"*, ICSE'81, San Diego.

Le protocole de validation à deux phases

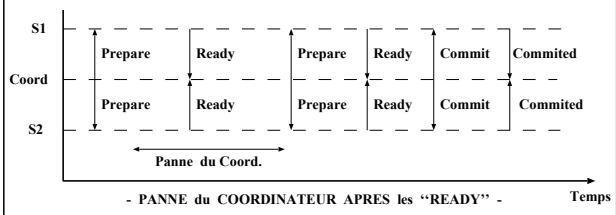
1. *Préparation* : Coordonnateur demande aux sites de se préparer (Ecriture Logs coordonnateur et sites, ...)
2. *Validation/Annulation* : Coordonnateur ordonne la validation ou l'avortement en fonction des résultats de la première phase

Validation à deux phases et échecs



- PANNE du SITE 2 APRES "READY"

Validation à deux phases et échecs



Validation à deux phases

- Transaction T initiée sur site S_i
- C_i coordonnateur de S_i
- Chaque site S_j participant à T informe C_i de la fin de sa "contribution"

1. *Phase 1* :
   - C_i :
     - Journalise < PREPARE T >
     - Envoie < PREPARE_TO_COMMIT > aux sites S_j
   - Chaque S_j :
     - Si validation possible
       Alors Journalise et envoie < PRET T > à C_i
     - SINON Journalise < NO T > et envoie < ABORT T > à C_i

Validation à deux phases

2. *Phase 2* : CAS REUSSITE (tous les < PRET T > sont parvenus)
   - Journalisation de < COMMIT T > sur S_i et les S_j
   - Les S_j envoient un "ACK" à C_i
   - C_i journalise < COMPLETE T> après réception des ACK.

3. *Phase 2* : CAS ECHEC
   - Un des S_j répond < ABORT T >
   - ou tous les < PRET T > non parvenus à C_i dans un délai d
   - C_i :
     - Journalise < ABORT T >
     - Envoie < ABORT T > aux S_j

Validation à deux phases

Exercice :
Décrire le protocole de validation à deux phases sous la forme d'un diagramme d'interaction, sachant que tous les sites utilisent le *Write Ahead Log Protocol*.

Validation à deux phases et pannes

1. Panne d'un site participant
2. Panne du site du coordinateur
3. Rupture de liaison
4. Partition du réseau

1/4. Validation à deux phases : Panne d'un site S_P

- *Après REPRISE, examen du Log de S_P* : Cas
  1. Pas d'information sur T dans le Log : S_P n'a pas répondu au $< PREPARE\ T >$ $\Longrightarrow$ UNDO(T)
  2. $< COMMIT\ T > \in Log$: REDO(T)
  3. $< ABORT\ T > \in Log$: UNDO(T)
  4. $< PRET\ T > \in Log$:
     - Consulter C_i puis UNDO(T) ou REDO(T)
     - *Cas S_i en Panne : Comment obtenir des informations sur T ?*
       * Envoi de $< QUERY\ STATUS >$ aux autres S_j
       * Si réception d'une réponse : UNDO ou REDO
       * Si pas de réponse ou si aucun S_j n'a l'information, poursuite des envois de $< QUERY\ STATUS >$ par S_P (avec état $< PRET\ T >$ et verrous éventuels "chez" S_P)

2/4. Validation à deux phases et Panne du coordinateur

- T est validée ssi $\exists\ S_j$ actif $\wedge\ <\ COMMIT\ T\ >\ \in\ Log\ de\ S_j$
- T est annulée ssi $\exists\ S_j$ actif $\wedge\ <\ NO\ T\ >\ \in\ Log\ de\ S_j$
- Sinon attendre la reprise de C_i

3/4. Validation à deux phases et Rupture d'une liaison

- Messages ne parviennent plus
- Chaque "partie" pense que l'autre est en panne
- Application du schéma "Panne d'un site"

4/4. Validation à deux phases et Partition du réseau

- Si S_i et tous les S_j dans la même partition : Pas de problème
- Sinon cas similaire à la rupture d'une liaison

Validation à deux phases : Implantations

1. Centralisé (cf. ce qui précède)
2. Linéaire
3. Distribué

Validation à deux phases : Mise en œuvre pratique

- Une transaction "maîtresse" (coordonateur) connaissant la fragmentation
- Une transaction par sous-requête
- Transaction "maîtresse" :
  - lance les sous-requêtes ($\simeq$ appel de procédure distante)
  - Coordonne la validation à deux phases
- Communication par appel de procédures distantes (*Remote Procedure Call*)

Validation à deux phases : conclusion

- Bloquant si panne du coordonnateur : Maintien des ressources jusqu'à sa reprise
- Blocage dû à la transition directe PRET, VALIDER
- *Protocole à trois phases* (Implantation ?)
  - Introduit par D. Skeen
    "Non-Blocking Protocols", ACM-SIGMOD Conference, An Arbor, Michigan, May 1981
  - Etat intermédiaire < PRET à VALIDER >
  - Si atteint alors VALIDER TOUJOURS
  - C_i en panne et aucun S_j n'a reçu < PRET à VALIDER > alors UNDO

Gestion de transactions : Plan

I. Les techniques d'estampillage

- Ordre de sérialisation obtenu par association d'une estampille unique à chaque transaction
- Gestion centralisée ou distribuée des estampilles
  1. *Gestion centralisée*
     - UN site
     - Compteur logique ou horloge locale
  2. *Gestion décentralisée*
     - < Estampille, Identifiant_du_Site >
     - Estampille : compteur logique ou fonction de l'horloge locale
     - *SYNCHRONISATION des HORLOGES ou des COMPTEURS*

Estampillage

- *SYNCHRONISATION des HORLOGES ou des COMPTEURS*
  - Problème : Un site génère plus rapidement que les autres
  - Mécanisme de contrôle de la génération "saine" des estampilles
    - Site S_i : Horloge H_i
    - S_i reçoit un message estampillé < t, s> avec $t > Valeur(H_i)$
    - S_i "avance" H_i à t + 1
  - Mécanisme similaire si utilisation d'horloges physiques

Estampilles et accès concurrents

- Chaque granule garde trace de la dernière transaction qui l'a utilisé
- Ordonnancement :

  T_i veut accéder à un granule estampillé j
  Si j $\leq$ i
  Alors T_i peut s'exécuter
  Sinon ABORT(T_i) et reprise ultérieure
  avec estampille $e > j$
  Finsi
- *ESTAMPILLAGE et ROLLBACK*
  - Conflits gérés par des ROLLBACK et pas par des attentes
  - Donc, risque de cascades de rollback

Estampillage et accès concurrents : Conclusion

- Beaucoup de ROLLBACK et de UNDO
- Mélange de 2-Phase Commit et estampillage : Protocole assurant la sérialisabilité sans cascade de ROLLBACK
- Technique délaissée au profit du verrouillage (+ gestion de deadlock)

II. Accès concurrents : Techniques de verrouillage

1. Données non dupliquées
2. Protocole de la majorité
3. Coordonnateur unique
4. Protocole "partial"
5. Protocole de la copie primaire

1/5. Verrouillage et données non dupliquées

- Un Gestionnaire de verrous GV_i par site S_i
- Demande de verrou d'un granule D, sur un site S_i par une transaction T :
  - T envoie sa demande à GV_i
  - D verrouillé dans un mode incompartible avec celui demandé $\Longrightarrow$ Demande mise en attente
  - Verrou apposé $\Longrightarrow$ GV_i informe T
- Mécanisme *SIMPLE* : Deux envois de messages
  - Demandes de verrouillage et de déverrouillage
- Gestion *COMPLEXE* du deadlock

2/5. Verrouillage et protocole de la majorité

- Version modifiée du protocole sans duplication
- D est détenu par une transaction T si T obtient le verrou sur une majorité de copies
- Un Gestionnaire de verrous GV_i par site S_i
- Granule D localisé sur N sites
- Demande concernant D adressée à M sites, M $>$ N/2
- Verrouillage ou attente sur les M sites
  - **(+)** Pas de contrôle centralisé avec duplication
  - **(-)** Gestion complexe du deadlock
  - **(-)** 2(n/2 + 1) messages (Verrouillage)
  - **(-)** + (n/2 + 1) messages (Déverrouillage)

3/5. Verrouillage et Coordonnateur unique

- Sur un site S_C
- Reçoit les demandes de (dé)verrouillage
- Lecture sur tout site détenant une copie
- Ecriture : concerne tout site détenant une copie
- Si verrou apposable : informer le demandeur
- Sinon : informer et mettre en attente
- *APPROCHE SIMPLE* :
  - Trois transferts de messages : 2 (Verrouillage) + 1 (Déverrouillage)
  - Gestion du deadlock : cf. BDD centralisée

3/5. Verrouillage et Coordonnateur unique

- *INCONVENIENTS* :
  - Engorgement de S_C
  - Blocage en cas de panne
- *ALTERNATIVE* :
  - Coordonnateurs sur plusieurs sites
  - Distribution de la gestion des verrous
  - Chaque gestionnaire se charge d'un sous-ensemble de données
  - **(+)** Réduction engorgement
  - **(-)** Gestion plus complexe du deadlock

4/5. Verrouillage et Protocole "partial"

- Similaire au protocole de la majorité
- Favoriser les verrous partagés au détriment des verrous exclusifs
- Un gestionnaire de verrous GV_i par site S_i
- GV_i chargé des verrous sur les données locales
- *Verrou partagé* : demande adressée à un des sites détenant la donnée
- *Verrou exclusif* : demande adressée à tous les sites
- Mise en attente ou apposition
  - **(+)** "Overhead" inférieur à celui des autres protocoles
  - **(-)** Overhead néanmoins, en écriture
  - **(-)** Gestion complexe du deadlock

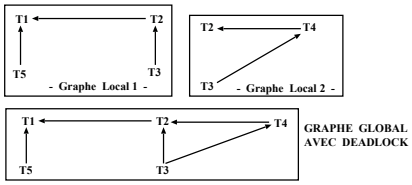
5/5. Verrouillage et Protocole de la copie primaire

- *SITE PRIMAIRE* : site privilégié d'une donnée dupliquée D
- Site primaire reçoit les demandes relatives à D
  - **(+)** Gestion de la concurrence similaire au cas non dupliqué
  - **(+)** Simplicité de l'implémentation
  - **(-)** Blocage en cas de panne sur le site primaire

III. Verrouillage et Interblocage

- *Cadre centralisé* :
  - Graphe d'attente
  - Scrutation périodique
  - Si $\exists$ cycle, tuer une transaction
- *Cadre distribué* : Gestion du graphe d'attente ?
  - Un graphe par site
  - $\neg\exists\ cycle\ local \not\Longleftrightarrow \neg\exists\ deadlock$

Verrouillage et Interblocage



- *Approches pour la gestion de l'interblocage* :
  1. Centralisée
  2. Répartie
  3. Hiérarchique

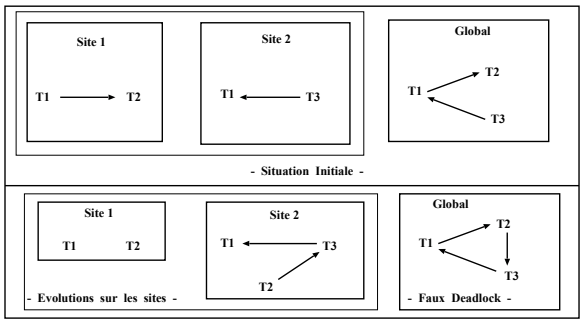
1/3. Gestion centralisée de l'interblocage

- COORDONNATEUR de détection de deadlock sur site S_C
- Graphe global géré sur S_C
- Communication des modifications des graphes locaux
- Mise à jour périodique du graphe global
  - Après chaque modification d'un graphe local
  - Après un certain nombre de modifications
  - Avant la scrutation

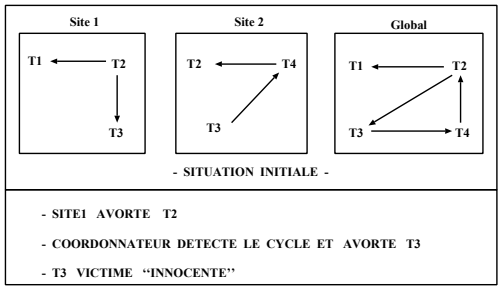
1/3. Gestion centralisée de l'interblocage

- *Graphe global construit* : Approximation du *graphe réel*
- Cycle sur le graphe construit
- Résolution (Tuer transaction T_D et ROLLBACK)
- Information aux sites impliqués dans T_D
- *!!! FAUX DEADLOCK et ROLLBACK INUTILE*

Exemple de faux deadlock



Exemple de Rollback inutile

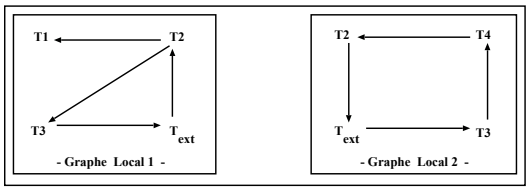


2/3. Gestion répartie de l'interblocage

- T_{ext} : Nouveau type de nœud (Transaction NON locale)
- Arc $T_i \longrightarrow T_{ext}$: T_i en attente d'un autre site
- Arc $T_{ext} \longrightarrow T_i$: T_{ext} en attente d'une ressource locale
- *Principe de la détection* :

  SI $\exists$ *cycle*
  ALORS SI $T_{ext} \notin$ *cycle*
  ALORS DEADLOCK LOCAL
  SINON - RISQUE de DEADLOCK
  - DETECTION [et RESOLUTION]
  FINSI
  FINSI

2/3. Gestion répartie de l'interblocage



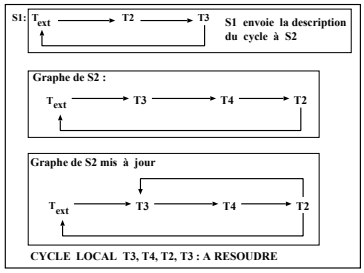
- Cycle nécessairement de la forme :

$$T_{ext} \longrightarrow T_{i1} \cdots \longrightarrow T_{ik} \longrightarrow T_{ext}$$

2/3. Gestion répartie de l'interblocage

- T_{ik} : Transaction k du site S_i
- S_i détecte le cycle : T_{ik} en attente de T_{ext} et *ext* = *j*
- S_i envoie un message à S_j :
  - Description du cycle
  - Demande de détection de deadlock
- S_j :
  - Mise à jour du graphe local
  - Recherche de cycle dans : graphe $\setminus \{T_{ext}\}$
  - $\exists$ cycle local : Résolution
  - $\exists$ cycle AVEC T_{ext} : Faire comme S_i
- *Nombre fini d'essais* : détection échoue ou réussit

2/3. Gestion répartie de l'interblocage : exemple



3/3. Gestion hiérarchisée de l'inter-blocage

- Hiérarchisation des sites
- Graphe d'attente d'un nœud : graphe global aux sites descendants
- Détection de deadlock sur les sous-arbres
- (+) Réduit l'activité à la racine (cf. approche centralisée)
- Performances fonction de l'adéquation *Hiérarchie-Localisation* des transactions

Chapitre VII : Sécurité de fonctionnement

Sécurité de fonctionnement et reprise

- Nouveaux types de pannes :
  - Panne d'un site
  - Rupture d'une liaison
  - Perte de message
  - Partition du réseau
- *Procédure* :
  1. *DETECTION*
  2. *RECONFIGURATION du SYSTEME*
  3. *REPRISE*

1/3. Sécurité de fonctionnement : Détection

- Souvent possible
- Identification de panne difficile
  - S_1 ne communique plus avec S_2
  - S_2 en panne ou rupture de liaison ?

2/3. Sécurité de fonctionnement : Reconfiguration

- Avortement des transactions actives (Libération des ressources)
- Inhiber l'accès aux données dupliquées du site en panne (MAJ catalogue)
- Si site en panne = site central (coordonnateur, etc.) : *Remplacement*
- Si partition du réseau :
  - Tolérer l'exécution de transactions dans les partitions
  - Eviter l'élection de plusieurs serveurs centraux dans une partition
  - Eviter la MAJ de données dupliquées par plusieurs partitions

3/3. Sécurité de fonctionnement : Reprise

- Appliquer les MAJ qui ont eu lieu sur les données dupliquées
- MAIS, ces données peuvent ENCORE être en cours de MAJ
- *Solution 1* :
  - Arrêt du système et remise en état
  - Eventuellement, négociation humaine pour la cohérence des copies
- *Solution 2* : Reprise $\simeq$ Série de transactions
- *Reprise après partition du réseau* :
  - Informer les sites de la réparation de la liaison (message broadcast)

Remplacement d'un site coordonnateur

- Coordonnateur UNIQUE $\Longrightarrow$ RISQUE de BLOCAGE
- Désignation d'un "remplaçant" :
  1. "Doublure" pré-définie (*BACK-UP*)
  2. Élection
  3. Préemption

1/3. Coordonnateur "back-up"

- Reçoit les mêmes messages que le "vrai"
- Exécute les mêmes algorithmes
- Gère les mêmes informations
- Prend le relai en cas de panne

(+) Reprise rapide
(-) Overhead

2/3. Élection de coordonnateur

- Identifiants UNIQUES des sites
- Panne du site de coordination :
  - Choix d'un site
  - Informer les autres sites
  - Prévoir d'informer les sites en panne
- Panne (présumée) du coordinateur : Pas d'écho pendant $\triangle$t $\Longrightarrow$ Élection

2/3. Élection de coordonnateur : Algorithme "bulldozer"

- *Algorithme* : S_i = initiateur de l'élection

  $S_i \longrightarrow$ Message d'élection vers les sites d'identifiants supérieurs
  SI pas de réponse APRES $\triangle$t
  ALORS • $\simeq$ "TOUS en PANNE"
  • S_i : "JE SUIS LE COORDINATEUR"
  • S_i informe tous les sites d'identifiants inférieurs
  SINON Attente élection d'un site d'identifiant supérieur
  SI PAS d'information à l'issue de l'attente
  ALORS • $\simeq$ PANNE du site ayant répondu
  • REFAIRE une ELECTION
  FINSI
  FINSI

3/3. Préemption

- Reprise d'un site S_i : Exécution du même algorithme

  SI $\neg\, \exists k, k = ID(Site_l) \wedge k > ID(S_i)$
  ALORS S_i DEVIENT COORDONNATEUR
  FINSI

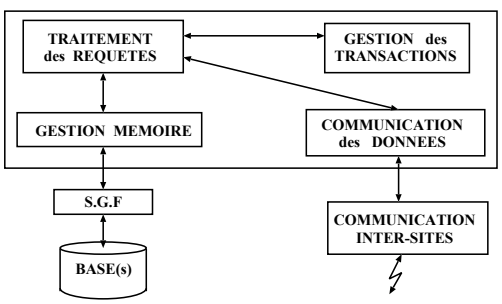
Chapitre VIII : Architecture des SGBD distribués

1. Homogènes
2. Hétérogènes
3. Systémes multi-bases
4. Bases de données parallèles
5. Architectures client/serveur

VIII.1- Bases de données distribuées homogènes

- R^*, successeur de SYSTEM-R, IBM, San José, 1979-1984
- Sites autonomes dotés du même SGBD
- Indépendance vis-à-vis de la localisation
- Transparence vis-à-vis de la duplication et de la fragmentation (ignorée pour simplifier l'implémentation)
- Extensions de SYSTEM-R
  - Définition de données et Dictionnaire
  - Traitement des requêtes
- Dictionnaire réparti
- Unicité des noms et immutabilité : Nom généré par le site qui crée un objet

Architecture de R^*



R^* : Traitement des requêtes

1. *Compilation de requêtes* :
   - *Site "client"* prend des décisions globales :
     - Choix des sites d'exécution (cf. jointures)
     - Choix de la méthode de transfert
     - Génération de plan
   - *Sites serveurs :*
     - Ordre de jointures
     - Plans d'exécution locaux

R^* : Traitement des requêtes

2. *Exécution* :
   - Un processus R^* par site inter-agissant avec un programme d'aplication
   - Processus vivant jusqu'à la fin du programme
   - Communication inter-sites :
     - Entre processus R^*
     - Totalement contrôlée par R^*

R^* : Gestion des transactions

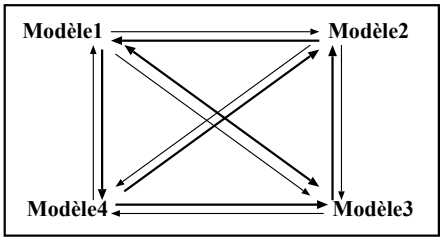
- 2-Phase Commit (Modifié)
- Détection répartie de deadlock
- Résolution de deadlock locaux et globaux

VIII.2- Bases de données distribuées hétérogènes

- Objectif : Indépendance, autonomie, transparence des SGBD locaux
- Intégration en une base globale de bases existantes sous des SGBD différents
- Problème difficile : certains systèmes n'admettent que des SGBD relationnels ou ne supportent que des requêtes d'interrogation
- Nouvelles fonctions : Correspondance/Traduction
  - Modèles (et représentation) de données
  - Langages de manipulation

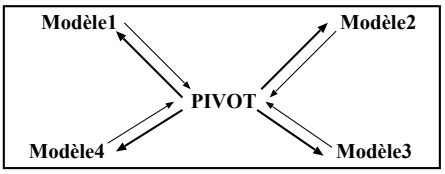
Intégration : approche 1 (approche "naïve")

- n*(n-1) traducteurs
- Difficulté de développement : (in)compatibilité des modèles



Intégration : approche 2 (modèle pivot ou commun)

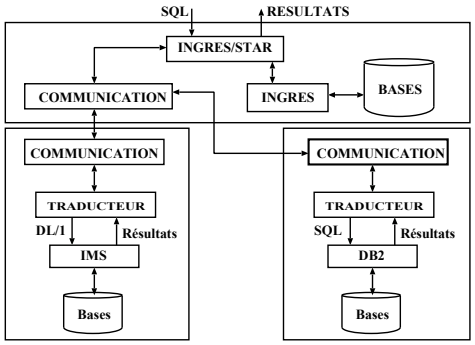
- Modèle pivot (Relationnel, Objet, (XML ?))
- Schéma global et LMD dans le modèle pivot
- Autant de traducteurs (bi-directionnels) que de modèles (2*n)
- Facilite la conception et la manipulation
- Dificulté d'intégration (conflits sémantiques, de nommage, etc.)



Exemples de systèmes

- ORACLE/STAR : SGBD réparti "relationnellement" hétérogène
- Sybase
- INGRES/STAR : SGBD relationnels et non relationnels (théoriquement)

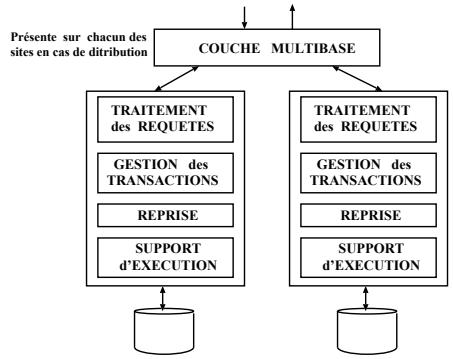
Architecture de Ingres/Star



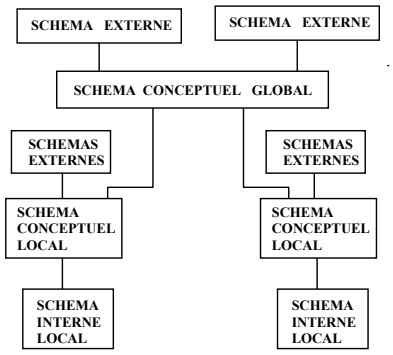
VIII.3- Multibases distribuées

- *SGBD logiquement intégrés distribués* :
  - Vision de toute la base
  - Base globale = ∪ Bases locales
- *SGBD multi-bases distribués* :
  - Fédération de bases de données
  - Vision de parties de bases locales que chaque SGBD veut partager
  - Base globale ⊆ ∪ Bases locales
  - Avec ou sans schéma conceptuel global

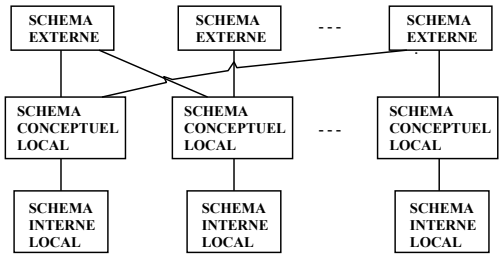
Composants fonctionnels des systèmes multi-bases



1/2. Multi-bases avec schéma global



2/2. Multi-bases sans schéma global



2/2. Multi-bases sans schéma global

- *Couche SGBD locaux + Couche système multibase*
- Accès aux bases délégué au mécanisme de correspondance entre schéma externe et schéma conceptuel local
- *Schéma des dépendances* inter-bases : expression de liens sémantiques
- *Extensions du LDD*
  - Manipulation de schémas (dépendances, externes, multibase)
    - CREATE, ALTER, DROP relation ou base
    - CREATE TABLE à partir de données d'une base privée
    - CREATE DATABASE : introduire une base dans la fédération
  - Nommage : *Nom_base•Nom_relation*
  - CREATE VIEW "multibase"
- *LMD* : SQL + Ouverture/Fermeture de plusieurs bases

Systèmes Multi-bases : Conclusion

- Technologie pas (encore) au point
- Oracle et Sybase offrent des possibilités de requêtes multi-bases centralisées
- Solutions techniques dans les SGBD répartis inadaptées aux multi-bases
- Problématique similaire dans la construction d'entrepôts de données (*Data Warehouse*)

VIII.4- Bases de données parallèles

- Bénéficier des architectures parallèles et multi-processeurs
- *Exemples* :
  - Machine transactionnelle *NonStop SQL*
  - Machine base de données DBC/1012 [Tera Data Corp.]
- Architectures multi-processeurs : Unités de traitement indépendantes coopérant via un réseau

Caractéristiques d'un SGBD parallèle

- Ressemble à un SGBD réparti homogène fortement intégré
  - Chaque nœud correspond à un site
- Diffère d'un SGBD réparti :
  - Spécialisation d'un nœud (gestion de données) $\Longrightarrow$ implantation plus simple et plus efficace
  - Nombre de nœuds peut être très élevé (Jusqu'à 1024 processeurs à MIPS dans DBC/1012)

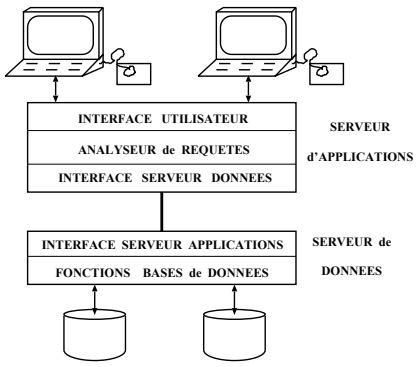
Caractéristiques d'un SGBD parallèle

- *Exemple* : Sybase MPP (Sybase11, 1995)
  - 128 processurs ou plus
  - Sun-Sparc Center 2000E, 16 processurs :
    - 4544 tpmc; 26 % > Informix
  - HP 9000 T500, 12 processurs :
    - 5621 tpmc; 300 % > Oracle

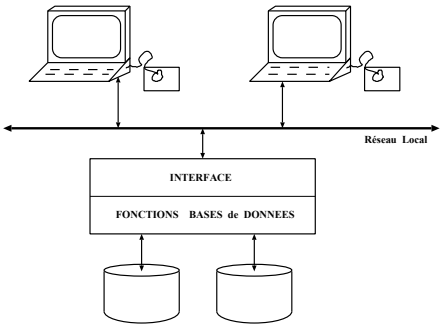
VIII.5- Architectures client-serveur

- Stations de travail et machines parallèles
- *Serveur d'applications* : station exécutant des programmes (anciennement *machine hôte*)
- *Serveur de données* : machine dédiée (anciennement *machine bases de données, back-end*)
- *Architectures* :
  1. Serveur de données centralisé
  2. Serveurs distribués

Architecture client-serveur



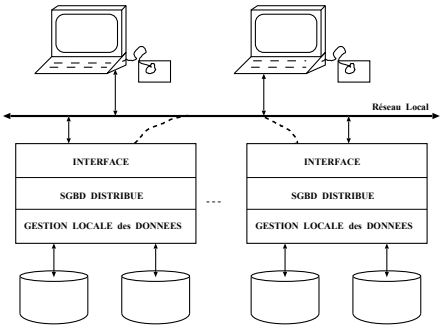
1/2. Architecture centralisée



Architecture centralisée

- Alternative moins coûteuse que SGBD distribué
- Accès efficace au serveur de données
- Mêmes inconvénients qu'une architecture "classique"
- Informations du dictionnaire en lecture seule sur les stations
  - Tâches locales de contrôle (autorisations, vues, etc.)
  - Réduction du flux de communication avec le serveur de données
- Serveur de données mono/multi-processeur(s)

2.2/ Architecture Distribuée



2.2/ Architecture Distribuée

- Serveurs d'applications peuvent être similaires à ceux de l'architecture centralisée
  - Requête addressée au serveur local,
  - Serveur local se charge de l'exécution répartie
- Alternative plus complexe : Dictionnaire et traitements de requêtes distribués sur chaque serveur d'applications

Chapitre IX : Conclusion

Conclusion

- Système distribué : {Sites} gérant des bases locales
- AUTONOMIE, TRANSPARENCE et COOPERATION
- ROBUSTESSE :
  - Détection panne
  - Reconfiguration (Tolérance au fonctionnement en mode dégradé)
  - Reprise après panne
  - Remplacement du coordonnateur
- ATOMICITE des transactions PLUS COMPLEXE (2PC)
- ADAPTATION des mécanismes "classiques" à la gestion des accès concurrents (Verrous, estampilles)

Conclusion

- Gestion centralisée, distribuée, hiérarchisée du deadlock
- Graphes d'attente locaux, globaux

- *Technologie trop complexe ?*
- % architecture client/serveur et outils de connectivité (ODBC, JDBC) ?
- Le Web comme une base de données distribuée ?

References

[1] Abiteboul (S.), Buneman (P.) et Suciu (D.). – *Data on the Web; From Relations to Semi-Structured Data and XML*. – San Francisco, CA, Morgan Kaufmann Publishers, 2000.

[2] Agrawal (R.) and al.. – *The Claremon Report on Database Research*. – CACM, 52(6), June 2009. http://doi.acm.org/10.1145/1516046.1516062.

[3] ANSI/X3/SPARC. – Study Group on Data Base Management Systems. – Interim Report - ACM, 1975.

[4] Besancenot (J.) et al. – *Les systèmes transactionnels: concepts, normes et prouits*. – Paris, Editions Hermes, 1997, *Collection informatique*.

[5] Boudjlida (N.). – Tutoriel "Objets Distribués, Intéropérabilité, CORBA" (*Distributed Objects, Interoperability, CORBA : a tutorial*). *In : Journées Bases de Données Avancées, BDA'98*. – Hammamet, Tunisie, Octobre 1998. (in French, http://www.loria.fr/~ nacer).

[6] Boudjlida (N.). – *Bases de données et systèmes d'informations. Le modèle relationnel: langages, systèmes et méthodes*. – Dunod, Paris, 1999. Cours et exercices corrigés. Collection Sciences Sup.

[7] Boudjlida (N.). – De la technologie bases de données aux technologies web. – Conférence invitée, 7th Maghrebian Conference on Software Engineering and Artificial Intelligence, MCSEAI'2002, Annaba, DZ, May 2002. http://www.loria.fr/~ nacer/PUBLI/Tut-MCSEAI02.ZIP.

[8] Boudjlida (N.). – *Gestion et Administration des Bases de Données: Application à Sybase et Oracle*. – Dunod, Paris, 2003.

[9] Boudjlida (N.) et Belhamissi (Y.). – Traitement de requêtes réparties : des modèles à leur implémentation. *In : Actes des Journées Internationales des Sciences Informatiques, JISI'94*. – Tunis, Tunisie, Mai 1994.

[10] Bouneffa (M.A.) et Boudjlida (N.). – Managing Schema Changes in Object-Relationship Databases. *In : Proceedings of the 14th Object-Oriented and Entity-Relationship International Conference, OO-ER'95*, éd. par Papazoglou (M.P.). pp. 113–122. – Gold Coast, Australia, December 1995.

[11] Elmasri (R.) et Navathe (S.B.). – *Fundamentals of database systems*. – The Benjamin/Cummings Publishing Company, Inc., 1989.

[12] Gardarin (G.) et Gardarin (O.). – *Le Client-Serveur*. – Paris, Eyrolles, 1996.

[13] Oszu (M. Tamer) et Valduriez (P.). – *Principles of Distributed Database Systems*. – Prentice Hall International, December 1998. 2nd edition.