CHAPITRE

25 SGBD distribués—Avancé Notions

objectifs du chapitre

Dans ce chapitre, vous apprendrez :

- How data distribution affects the transaction management protocols.
- How centralized concurrency control techniques can be extended to handle data distribution.
- How to detect deadlock when multiple sites are involved.
- How to recover from database failure in a distributed environment using:
  - two-phase commit (2PC)
  - three-phase commit (3PC)
- The difficulties of detecting and maintaining integrity in a distributed environment.
- About the X/Open DTP standard.
- About distributed query optimization.
- The importance of the Semijoin operation in distributed environments.
- How Oracle handles data distribution.

Dans le chapitre précédent, nous avons discuté des concepts de base et des problèmes associés aux systèmes de gestion de bases de données distribuées (DDBMS). Du point de vue des utilisateurs, les fonctionnalités offertes par un DDBMS sont très attractives. Cependant, du point de vue de la mise en œuvre, les protocoles et algorithmes requis pour fournir cette fonctionnalité sont complexes et donnent lieu à plusieurs problèmes qui peuvent l'emporter sur les avantages offerts par cette technologie. Dans ce chapitre, nous poursuivons notre discussion sur la technologie DDBMS et examinons comment les protocoles de contrôle de la concurrence, de gestion des blocages et de récupération que nous avons présentés au chapitre 22 peuvent être étendus pour permettre la distribution et la réplication des données.

Une alternative, et potentiellement une approche plus simplifiée, à la distribution des données est fournie par un serveur de réplication, qui gère la réplication des données vers des sites distants. Chaque grand fournisseur de bases de données a une solution de réplication d'un type ou d'un autre, et de nombreux autres fournisseurs proposent également des méthodes alternatives pour répliquer les données. Dans le chapitre suivant, nous considérons également le serveur de réplication comme une alternative à un DDBMS.

Structure de ce chapitre Dans la section 25.1, nous passons brièvement en revue les objectifs du traitement distribué des transactions. Dans la section 25.2, nous examinons comment la distribution des données affecte la définition de la sérialisabilité donnée dans la section 22.2.2, puis discutons de la manière d'étendre les protocoles de contrôle de concurrence présentés dans les sections 22.2.3 et 22.2.5 pour l'environnement distribué. Dans la section 25.3, nous examinons la complexité accrue de l'identification des interblocages dans un SGBD distribué et discutons des protocoles de détection des interblocages distribués. Dans la section 25.4, nous examinons les défaillances qui peuvent survenir dans un environnement distribué et discutons des protocoles qui peuvent être utilisés pour assurer l'atomicité et la durabilité des transactions distribuées. Dans la section 25.5, nous passons brièvement en revue le modèle de traitement des transactions distribuées X/Open, qui spécifie une interface de programmation pour le traitement des transactions. Dans la section 25.6, nous donnons un aperçu de l'optimisation des requêtes distribuées et dans la section 25.7, nous donnons un aperçu de la façon dont Oracle gère la distribution. Les exemples de ce chapitre sont à nouveau tirés de l'étude de cas DreamHome décrite à la section

25.1 Gestion des transactions distribuées

Dans la section 24.5.2, nous avons noté que les objectifs du traitement distribué des transactions sont les mêmes que ceux des systèmes centralisés, bien que plus complexes, car le DDBMS doit également assurer l'atomicité de la transaction globale et de chaque sous-transaction qui la compose. Dans la section 22.1.2, nous avons identifié quatre modules de base de données de haut niveau qui gèrent les transactions, le contrôle de la concurrence et la récupération dans un SGBD centralisé. Le gestionnaire de transactions coordonne les transactions pour le compte des programmes d'application, communiquant avec l'ordonnanceur, le module chargé de mettre en œuvre une stratégie particulière de contrôle de la concurrence. L'objectif du planificateur est de maximiser la concurrence sans permettre aux transactions exécutées simultanément d'interférer les unes avec les autres et de compromettre ainsi la cohérence de la base de données. En cas de panne survenant au cours de la transaction, le gestionnaire de récupération s'assure que la base de données est restaurée dans l'état où elle se trouvait avant le début de la transaction, et donc dans un état cohérent. Le gestionnaire de récupération est également responsable de la restauration de la base de données dans un état cohérent suite à une panne du système. Le gestionnaire de tampon est responsable du transfert efficace des données entre le stockage sur disque et la mémoire principale.

Dans un SGBD distribué, ces modules existent toujours dans chaque SGBD local. En outre, il existe également un gestionnaire de transactions global ou un coordinateur de transactions sur chaque site pour coordonner l'exécution des transactions globales et locales initiées sur ce site. La communication inter-sites passe toujours par le composant de communication de données (les gestionnaires de transactions des différents sites ne communiquent pas directement entre eux).

La procédure pour exécuter une transaction globale initiée sur le site S1 est la suivanteÿ:

- Le coordinateur de transaction (TC1) du site S1 divise la transaction en un certain nombre de sous-transactions à l'aide des informations contenues dans le catalogue du système global.

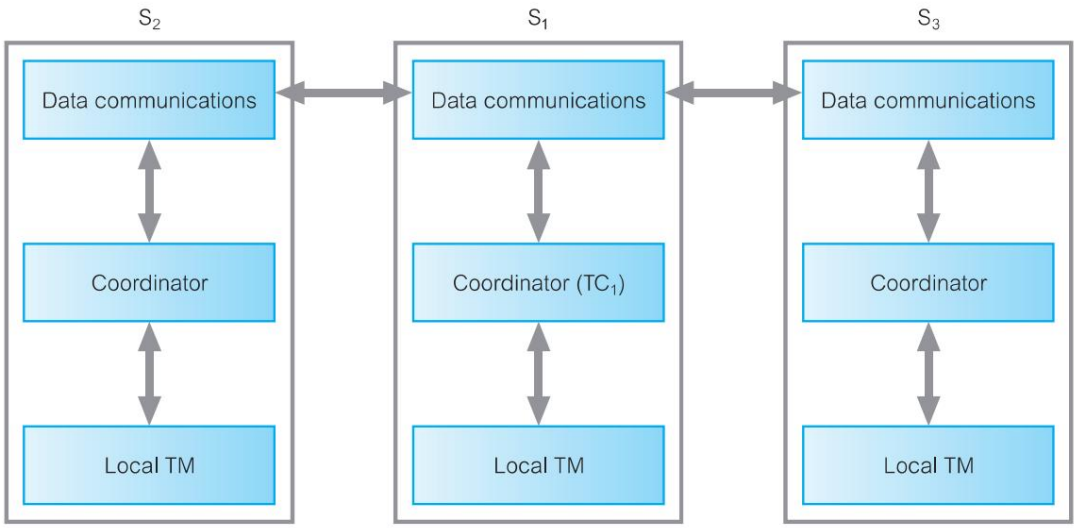


Illustration 25.1 Coordination of distributed transaction.

- Le composant de communication de données du site S1 envoie les sous-transactions aux sites appropriés, disons S2 et S3.
- Les coordinateurs de transactions des sites S2 et S3 gèrent ces sous-transactions. Les résultats des sous-transactions sont renvoyés à TC1 via les composants de communication de données .

Ce processus est illustré à la figure 25.1. Avec cet aperçu de la gestion distribuée des transactions, nous abordons maintenant les protocoles de contrôle de la concurrence, de gestion des interblocages et de récupération.

25.2 Contrôle de concurrence distribué

Dans cette section, nous présentons les protocoles qui peuvent être utilisés pour fournir un contrôle de concurrence dans un SGBD distribué. Nous commençons par examiner les objectifs du contrôle de concurrence distribué.

25.2.1 Objectifs Etant

donné que le système n'a pas échoué, tous les mécanismes de contrôle de concurrence doivent garantir que la cohérence des éléments de données est préservée et que chaque action atomique est réalisée en un temps fini. De plus, un bon mécanisme de contrôle de la concurrence pour les SGBD distribués devraitÿ:

- être résilient aux pannes de site et de communicationÿ;
- permettre le parallélisme pour satisfaire aux exigences de
- performanceÿ; engager des frais de calcul et de stockage modestesÿ;
- fonctionner de manière satisfaisante dans un environnement de réseau qui présente un retard de communication importantÿ; imposent peu de contraintes sur la structure des actions atomiques
- (Kohler, 1981).

Dans la section 22.2.1, nous avons discuté des types de problèmes qui peuvent survenir lorsque plusieurs utilisateurs sont autorisés à accéder simultanément à la base de donnéesÿ: perte de mise à jour, dépendance non validée et analyse incohérente. Ces problèmes existent également dans l'environnement distribué. Cependant, d'autres problèmes peuvent survenir à la suite de la distribution des données. L'un de ces problèmes est le problème de cohérence des copies multiples, qui se produit lorsqu'un élément de données est répliqué à différents emplacements. De toute évidence, pour maintenir la cohérence de la base de données globale, lorsqu'un élément de données répliqué est mis à jour sur un site, toutes les autres copies de l'élément de données doivent également être mises à jour. Si une copie n'est pas mise à jour, la base de données devient incohérente. Nous supposons dans cette section que les mises à jour des éléments répliqués sont effectuées de manière synchrone, dans le cadre de la transaction englobante. Au chapitre 26, nous expliquons comment les mises à jour des éléments répliqués peuvent être effectuées de manière asynchrone, c'est-à-dire à un moment donné après la fin de la transaction qui met à jour la copie d'origine de l'élément de données.

25.2.2 Sérialisabilité distribuée

Le concept de sérialisabilité, dont nous avons parlé à la section 22.2.2, peut être étendu pour l'environnement distribué pour répondre à la distribution des données. Si le calendrier de l'exécution des transactions sur chaque site est sérialisable, alors le planning global (l'union de tous les plannings locaux) est également sérialisable, à condition que les ordres de sérialisation locaux soient identiques. Cela nécessite que toutes les sous-transactions apparaissent dans le même ordre dans le programme de série équivalent sur tous les sites. Ainsi, si la sous-transaction de Ti au site S1 est notée T1 , on doit s'assurer que si T1 , T1 alors : j

, Tx jÿ ÿ pour tout Sx pour lequel Ti et Tj ont des sous-transactions

Les solutions au contrôle de la concurrence dans un environnement distribué sont basées sur les deux approches principales du verrouillage et de l'horodatage, que nous avons considérées pour les systèmes centralisés dans la section 22.2. Ainsi, étant donné un ensemble de transactions à exécuter simultanément, alorsÿ:

- le verrouillage garantit que l'exécution simultanée est équivalente à une exécution en série (imprévisible) de ces transactionsÿ; l'horodatage garantit que l'exécution simultanée est équivalente
- à une exécution en série spécifique de ces transactions, correspondant à l'ordre des horodatages.

Si la base de données est centralisée ou fragmentée, mais non répliquée, de sorte qu'il n'existe qu'une seule copie de chaque élément de données, et que toutes les transactions sont soit locales, soit peuvent être effectuées sur un site distant, alors les protocoles discutés à la section 22.2 peuvent être utilisé. Cependant, ces protocoles doivent être étendus si les données sont répliquées ou si les transactions impliquent des données sur plusieurs sites. De plus, si nous adoptons un protocole basé sur le verrouillage, nous devons fournir un mécanisme pour gérer l'interblocage (voir Section 22.2.4). L'utilisation d'un mécanisme de détection et de récupération d'interblocage implique la vérification de l'interblocage non seulement à chaque niveau local mais également au niveau mondial, ce qui peut impliquer la combinaison de données d'interblocage provenant de plusieurs sites. Nous considérons le blocage distribué dans la section 25

25.2.3 Protocoles de verrouillage

Dans cette section, nous présentons les protocoles suivants basés sur le verrouillage en deux phases (2PL) qui peuvent être utilisés pour assurer la sérialisabilité des SGBD distribués : 2PL centralisé, 2PL de copie primaire, 2PL distribué et verrouillage majoritaire.

2PL centralisé

Avec le protocole 2PL centralisé, il existe un site unique qui conserve toutes les informations de verrouillage (Alsberg et Day, 1976 ; GarciaMolina, 1979). Il n'y a qu'un seul ordonnanceur, ou gestionnaire de verrous, pour l'ensemble du SGBD distribué qui peut accorder et libérer des verrous. Le protocole 2PL centralisé pour une transaction globale initiée sur le site S1 fonctionne comme suitÿ:

(1) Le coordinateur de transaction du site S1 divise la transaction en un certain nombre de sous-transactions, en utilisant les informations contenues dans le catalogue du système global. Le coordinateur a la responsabilité de s'assurer que la cohérence est maintenue. Si la transaction implique une mise à jour d'un élément de données qui est répliqué, le coordinateur doit s'assurer que toutes les copies de l'élément de données sont mises à jour. Ainsi, le coordinateur demande des verrous exclusifs sur toutes les copies avant de mettre à jour chaque copie et de libérer les verrous. Le coordinateur peut choisir d'utiliser n'importe quelle copie de l'élément de données pour les lecturesÿ; généralement la copie sur son site, s'il en existe un.

(2) Les gestionnaires de transactions locaux impliqués dans la transaction globale demandent et libèrent les verrous du gestionnaire de verrous centralisé en utilisant les règles normales pour le verrouillage en deux phases.

(3) Le gestionnaire de verrou centralisé vérifie qu'une demande de verrou sur une donnée est compatible avec les verrous existant actuellement. Si c'est le cas, le gestionnaire de verrous renvoie un message au site d'origine confirmant que le verrou a été accordé. Sinon, il place la demande dans une file d'attente jusqu'à ce que le verrou puisse être accordé.

Une variante de ce schéma consiste pour le coordinateur de transaction à effectuer toutes les demandes de verrouillage au nom des gestionnaires de transaction locaux. Dans ce cas, le gestionnaire de verrouillage n'interagit qu'avec le coordinateur de transaction et non avec les gestionnaires de transaction locaux individuels.

L'avantage du 2PL centralisé est que la mise en œuvre est relativement simple. La détection des interblocages n'est pas plus difficile que celle d'un SGBD centralisé, car un gestionnaire de verrous conserve toutes les informations sur les verrous. Les inconvénients de la centralisation dans un SGBD distribué sont les goulots d'étranglement et une fiabilité moindre. Comme toutes les demandes de verrouillage sont dirigées vers un site central, ce site peut devenir un goulot d'étranglement. Le système peut également être moins fiable, car la défaillance du site central entraînerait des pannes majeures du système. Cependant, les coûts de communication sont relativement faibles. Par exemple, une opération de mise à jour globale qui a des agents (sous-transactions) sur n sites peut nécessiter un minimum de 2n 1 3 messages avec un gestionnaire de verrouillage centraliséÿ:

- 1 demande de
- verrouillageÿ; 1 message d'octroi
- de verrouillageÿ; n messages de
- mise à jourÿ; n remerciementsÿ;
- 1 demande de déverrouillage.

Copie primaire 2PL Ce

protocole tente de pallier les inconvénients du 2PL centralisé en répartissant les gestionnaires de verrous sur plusieurs sites. Chaque gestionnaire de verrous est alors chargé de gérer les verrous pour un ensemble de données. Pour chaque élément de données répliqué, une copie est choisie comme copie principaleÿ; les autres copies sont appelées copies esclaves. Le

le choix du site à choisir comme site principal est flexible et le site choisi pour gérer les verrous d'une copie principale n'a pas besoin de contenir la copie principale de cet élément (Stonebraker et Neuhold, 1977).

Le protocole est une simple extension du 2PL centralisé. La principale différence est que lorsqu'un élément doit être mis à jour, le coordinateur de transaction doit déterminer où se trouve la copie principale afin d'envoyer les demandes de verrouillage au gestionnaire de verrouillage approprié. Il est nécessaire de verrouiller exclusivement uniquement la copie principale de l'élément de données à mettre à jour. Une fois la copie principale mise à jour, la modification peut être propagée aux copies esclaves. Cette propagation doit être effectuée le plus tôt possible pour éviter que d'autres transactions ne lisent des valeurs périmées. Cependant, il n'est pas strictement nécessaire d'effectuer les mises à jour comme une opération atomique. Ce protocole garantit uniquement que la copie primaire est à jour.

Cette approche peut être utilisée lorsque les données sont répliquées de manière sélective, que les mises à jour sont peu fréquentes et que les sites n'ont pas toujours besoin de la toute dernière version des données. Les inconvénients de cette approche sont que la gestion des interblocages est plus complexe en raison de plusieurs gestionnaires de verrous et qu'il existe encore un certain degré de centralisation dans le systèmeÿ: les demandes de verrou pour une copie primaire spécifique peuvent être traitées par un seul site. Ce dernier inconvénient peut être partiellement surmonté en nommant des sites de sauvegarde pour conserver les informations de verrouillage. Cette approche a des coûts de communication inférieurs et de meilleures performances que le 2PL centralisé, car il y a moins de verrouillage à distance.

2PL distribué

Ce protocole tente à nouveau de pallier les inconvénients du 2PL centralisé, cette fois en répartissant les gestionnaires de verrous sur chaque site. Chaque gestionnaire de verrous est alors responsable de la gestion des verrous pour les données sur ce site. Si les données ne sont pas répliquées, ce protocole est équivalent à la copie primaire 2PL. Sinon, le 2PL distribué implémente un protocole de contrôle de réplique readonewriteall (ROWA). Cela signifie que toute copie d'un élément répliqué peut être utilisée pour une opération de lecture, mais toutes les copies doivent être exclusivement verrouillées avant qu'un élément puisse être mis à jour. Ce schéma traite les serrures de manière décentralisée, évitant ainsi les inconvénients d'une commande centralisée. Cependant, les inconvénients de cette approche sont que la gestion des interblocages est plus complexe, en raison de plusieurs gestionnaires de verrouillage et que les coûts de communication sont plus élevés que la copie primaire 2PL, car tous les éléments doivent être verrouillés avant la mise à jour. Une opération de mise à jour globale qui a des agents sur n sites peut nécessiter un minimum de 5n messages avec ce protocole : n messages de demande de verrouillage ; n

- verrouiller les messages d'octroiÿ; n messages de mise à jourÿ; n remerciementsÿ; n déverrouiller les
- demandes.
-
-
-

Cela pourrait être réduit à 4n messages si les requêtes de déverrouillage sont omises et traitées par l'opération de validation finale. Le 2PL distribué a été utilisé dans le prototype System R* DDBMS (Mohan et al., 1986).

Verrouillage majoritaire

Ce protocole est une extension du 2PL distribué pour éviter d'avoir à verrouiller toutes les copies d'un élément répliqué avant une mise à jour. Encore une fois, le système maintient un gestionnaire de verrouillage

sur chaque site pour gérer les verrous de toutes les données de ce site. Lorsqu'une transaction souhaite lire ou écrire une donnée qui est répliquée sur n sites, elle doit envoyer une demande de verrouillage à plus de la moitié des n sites où la donnée est stockée. La transaction ne peut pas se poursuivre tant qu'elle n'a pas obtenu de verrous sur la majorité des copies. Si la transaction n'obtient pas de majorité dans un certain délai, elle annule sa demande et informe tous les sites de l'annulation. S'il obtient la majorité, il informe tous les sites qu'il a le verrou. N'importe quel nombre de transactions peut détenir simultanément un verrou partagé sur la majorité des copiesÿ; cependant, une seule transaction peut détenir un verrou exclusif sur la majorité des copies (Thomas, 1979).

Ce schéma évite également les inconvénients d'une commande centralisée. Les inconvénients sont que le protocole est plus compliqué, la détection des interblocages est plus complexe et le verrouillage nécessite au moins [(n 1 1)/2] messages pour les demandes de verrouillage et [(n 1 1)/2] messages pour les demandes de déverrouillage. Cette technique fonctionne mais est trop forte dans le cas des verrous partagésÿ: l'exactitude nécessite qu'une seule copie d'un élément de données soit verrouillée, à savoir l'élément qui est lu, mais cette technique demande des verrous sur la majorité des copies.

25.3 Gestion distribuée des interblocages

Tout algorithme de contrôle de concurrence basé sur le verrouillage (et certains algorithmes basés sur l'horodatage qui nécessitent que les transactions attendent) peut entraîner des blocages, comme discuté à la section 22.2.4. Dans un environnement distribué, la détection des interblocages peut être plus compliquée si la gestion des verrous n'est pas centralisée, comme le montre l'exemple 25.1.

EXEMPLE 25.1 Interblocage distribué

Considérons trois transactions T1, T2 et T3 avecÿ:

- T1 initié sur le site S1 et création d'un agent sur le site S2ÿ;
- T2 initié au site S2 et création d'un agent au site S3ÿ; T3
- initié sur le site S3 et création d'un agent sur le site S1.

Les transactions définissent des verrous partagés (lecture) et exclusifs (écriture) comme illustré dans cet exemple, où read_lock(Ti , xj) désigne un verrou partagé par la transaction Ti sur la donnée xj et write_lock(Ti , xj) désigne un verrou exclusif par transaction Ti sur la donnée xj .

TEMPS	S1	S2	S3
t_1	read_lock(T_1, x_1)	write_lock(T_2, y_2)	read_lock(T_3, z_3)
t_2	write_lock(T_1, y_1)	write_lock(T_2, z_2)	
t_3	write_lock(T_3, x_1)	write_lock(T_1, y_2)	write_lock(T_2, z_3)

Nous pouvons construire les graphes d'attente (WFG) pour chaque site, comme le montre la figure 25.2. Il n'y a pas de cycles dans les WFG individuels, ce qui pourrait nous amener à croire que l'impasse n'existe pas. Cependant, si nous combinons les WFG, comme illustré à la Figure 25.3, nous pouvons voir qu'il existe un blocage dû au cycleÿ:

T1 T2 → T3 → T1

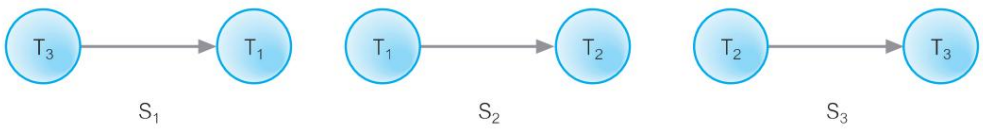


Figure 25.2 Wait-for graphs for sites S., S_2, and S_3.

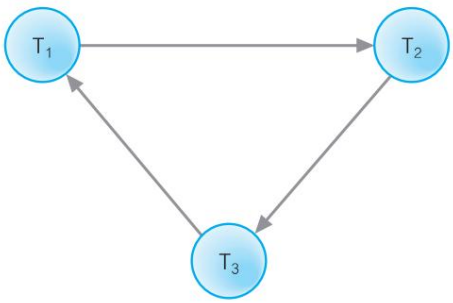


Figure 25.3 Combined wait-for graphs for sites S., S_2, and S_3.

L'exemple 25.1 montre que dans un DDBMS, il ne suffit pas que chaque site construise son propre WFG local pour vérifier l'interblocage. Il est également nécessaire de construire un WFG global qui soit l'union de tous les WFG locaux. Il existe trois méthodes courantes de gestion de la détection des interblocages dans les DDBMSÿ: la détection centralisée, hiérarchique et distribuée.

Détection centralisée des interblocages

Avec la détection centralisée des interblocages, un seul site est désigné comme coordinateur de la détection des interblocages (DDC). Le DDC a la responsabilité de construire et de maintenir le WFG mondial. Périodiquement, chaque gestionnaire d'écluse transmet son WFG local au DDC. Le DDC construit le WFG global et vérifie les cycles qu'il contient. S'il existe un ou plusieurs cycles, le DDC doit interrompre chaque cycle en sélectionnant les transactions à annuler et à redémarrer. Le DDC doit informer tous les sites impliqués dans le traitement de ces transactions qu'elles doivent être annulées et redémarrées.

Pour minimiser la quantité de données envoyées, un gestionnaire de verrouillage n'a besoin d'envoyer que les modifications qui se sont produites dans le WFG local depuis qu'il a envoyé la dernière. Ces modifications représenteraient l'ajout ou la suppression d'arêtes dans le WFG local. L'inconvénient de cette approche centralisée est que le système peut être moins fiable, car la défaillance du site central causerait des problèmes.

Détection hiérarchique des interblocages

Avec la détection de blocage hiérarchique, les sites du réseau sont organisés en hiérarchie. Chaque site envoie son WFG local au site de détection de blocage situé au-dessus de lui dans la hiérarchie (Menasce et Muntz, 1979). La figure 25.4 illustre une hiérarchie possible pour huit sites, S1 à S8. Les feuilles de niveau 1 sont les sites eux-mêmes où la détection locale des interblocages est effectuée. Les nœuds de niveau 2 DDij détectent les blocages impliquant des sites adjacents i et j. Les nœuds de niveau 3 détectent un interblocage entre quatre nœuds adjacents

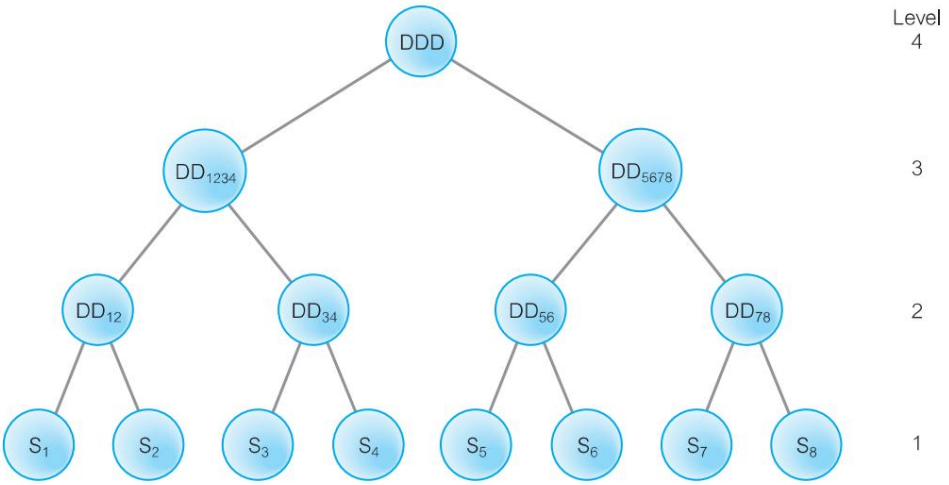


Figure 25.4
Hierarchical deadlock detection.

des sites. La racine de l'arborescence est un détecteur de blocage global qui détecterait un blocage entre, par exemple, les sites S1 et S8.

L'approche hiérarchique réduit la dépendance à un site de détection centralisé, réduisant ainsi les coûts de communication. Cependant, il est beaucoup plus complexe à mettre en œuvre, notamment en présence de pannes de site et de communication.

Détection d'interblocage distribuée

Il y a eu diverses propositions d'algorithmes de détection de blocage distribués, mais nous considérons ici l'une des plus connues qui a été développée par Obermarck (1982). Dans cette approche, un nœud externe Texte est ajouté à un WFG local pour indiquer un agent sur un site distant. Lorsqu'une transaction T1 sur le site S1, par exemple, crée un agent sur un autre site S2, par exemple, un bord est ajouté au WFG local de T1 au nœud Texte. De même, au site S2, une arête est ajoutée au WFG local du nœud Texte à l'agent de T1.

Par exemple, le WFG global illustré à la Figure 25.3 serait représenté par les WFG locaux aux sites S1, S2 et S3 illustrés à la Figure 25.5. Les bords du WFG local liant les agents au texte sont étiquetés avec le site concerné. Par exemple, le bord reliant T1 et Texte au site S1 est étiqueté S2, car ce bord représente un agent créé par la transaction T1 au site S2.

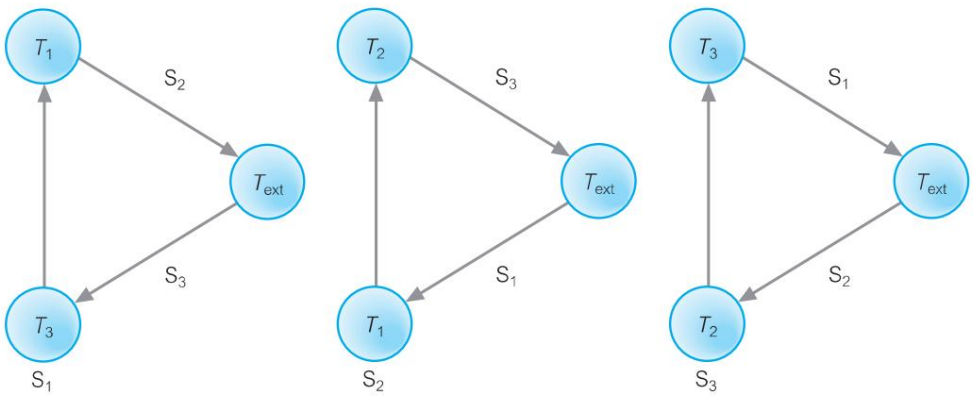


Figure 25.5
Distributed deadlock detection.

Si un WFG local contient un cycle qui n'implique pas le nœud Texte, alors le site et le DDBMS sont dans l'impasse et l'impasse peut être levée par le site local. Un interblocage global existe potentiellement si le WFG local contient un cycle impliquant le Noeud de texte. Cependant, l'existence d'un tel cycle ne signifie pas nécessairement que il y a un blocage global, car les nœuds de texte peuvent représenter différents agents, mais des cycles de ce formulaire doit figurer dans les WFG en cas d'impasse. Pour déterminer si il y a une impasse, les graphiques doivent être fusionnés. Si un site S1, par exemple, a un potentiel interblocage, son WFG local sera de la forme :

Texte → De → Tj → . . . Texte → TK →

Pour éviter que les sites se transmettent leurs WFG entre eux, une stratégie simple alloue un horodatage à chaque transaction et impose la règle selon laquelle le site S1 transmet ses WFG uniquement vers le site pour lequel la transaction Tk est en attente, disons Sk, si ts(Ti) , ts(Tk). Si on suppose que ts(Ti) , ts(Tk) alors, pour vérifier l'interblocage, le site S1 transmettra son WFG local à Sk. Site Sk peut maintenant ajouter ces informations à son WFG local et vérifier cycles n'impliquant pas de texte dans le graphe étendu. S'il n'y a pas un tel cycle, le processus continue jusqu'à ce qu'un cycle apparaisse, auquel cas une ou plusieurs transactions sont annulé et redémarré avec tous leurs agents, ou l'ensemble du WFG mondial est construit et aucun cycle n'a été détecté. Dans ce cas, il n'y a pas de blocage dans le système. Obermarck a prouvé que s'il existe une impasse globale, cette procédure provoque éventuellement l'apparition d'un cycle sur un site.

Les trois WFG locaux de la Figure 25.5 contiennent des cyclesÿ:

S1 : Texte → T3 → T1 → → Texte

S2 : Texte → T1 → T2 → Texte

S3 : Texte → T2 → T3 → → Texte

Dans cet exemple, nous pourrions transmettre le WFG local du site S1 au site pour lequel la transaction T1 est en attente : c'est-à-dire le site S2. Le WFG local à S2 est étendu pour inclure ces informations et devient :

S2 : Texte → T3 → T1 → → T2 → Texte

Cela contient toujours un blocage potentiel, nous transmettrions donc ce WFG au site pour laquelle la transaction T2 est en attente : c'est-à-dire le site S3. Le WFG local à S3 est étendu pour:

S3 : Texte → T3 → T1 → → T2 → T3 → Texte

Ce WFG global contient un cycle qui n'implique pas le nœud Texte (T3 → T1 → → T2 → T3), nous pouvons donc conclure à l'existence d'un blocage et à une reprise appropriée protocole doit être invoqué. Les méthodes distribuées de détection des interblocages sont potentiellement plus robuste que les méthodes hiérarchiques ou centralisées, mais parce qu'aucun site contient toutes les informations nécessaires pour détecter les blocages, intersites considérables communication peut être nécessaire.

25.4 Récupération de base de données distribuée

Dans cette section, nous discutons des protocoles utilisés pour gérer les pannes dans un environnement distribué.

25.4.1 Défaillances dans un environnement distribué

Dans la section 24.5.2, nous avons mentionné quatre types de défaillances propres aux SGBD distribuésÿ:

- la perte d'un message; la
- défaillance d'un lien de communicationÿ; la
- défaillance d'un site ; partitionnement du
- réseau.

La perte de messages ou de messages mal ordonnés relève de la responsabilité du protocole de réseau informatique sous-jacent. En tant que tels, nous supposons qu'ils sont gérés de manière transparente par le composant de communication de données du DDBMS et nous nous concentrons sur les types de défaillance restants.

Un DDBMS dépend fortement de la capacité de tous les sites du réseau à communiquer de manière fiable entre eux. Dans le passé, les communications n'étaient pas toujours fiables. Bien que la technologie réseau se soit considérablement améliorée et que les réseaux actuels soient beaucoup plus fiables, des pannes de communication peuvent toujours se produire. En particulier, les échecs de communication peuvent entraîner la division du réseau en deux partitions ou plus, où les sites d'une même partition peuvent communiquer entre eux mais pas avec les sites d'autres partitions. La figure 25.6 montre un exemple de partitionnement de réseau dans lequel suite à la défaillance du lien reliant les sites S1 ® S2, les sites (S1, S4, S5) sont partitionnés des sites (S2, S3).

Dans certains cas, il est difficile de distinguer si un lien de communication ou un site est défaillant. Par exemple, supposons que le site S1 ne puisse pas communiquer avec le site S2 dans un délai fixe (timeout). Il se peut que :

- le site S2 est tombé en panne ou le réseau est tombé en
- panneÿ; le lien de communication a échouéÿ; le réseau est
- partitionné ; le site S2 est actuellement très occupé et n'a
- pas eu le temps de répondre au message.

Choisir la bonne valeur pour le timeout, qui permettra à S1 de conclure qu'il ne peut pas communiquer avec le site S2, est difficile.

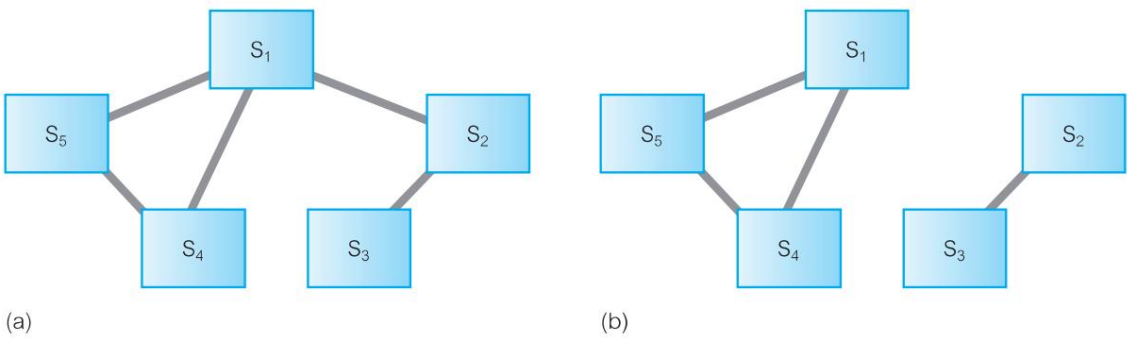


Figure 25.6 Partitioning a network: (a) before failure; (b) after failure.

25.4.2 Comment les défaillances affectent la restauration Comme

pour la restauration locale, la restauration distribuée vise à maintenir l'atomicité et la durabilité des transactions distribuées. Pour garantir l'atomicité de la transaction globale, le DDBMS doit s'assurer que les sous-transactions de la transaction globale sont toutes validées ou toutes abandonnées. Si le DDBMS détecte qu'un site a échoué ou est devenu inaccessible, il doit effectuer les étapes suivantesÿ:

- Abandonnez toutes les transactions affectées par l'échec.
- Marquez le site comme ayant échoué pour empêcher tout autre site d'essayer de l'utiliser.
- Vérifiez périodiquement si le site a récupéré ou, alternativement, attendez le site défaillant à diffuser qu'il a récupéré.
- Au redémarrage, le site défaillant doit lancer une procédure de récupération pour abandonner toutes les transactions partielles qui étaient actives au moment de la panne.
- Après la restauration locale, le site défaillant doit mettre à jour sa copie de la base de données pour la rendre cohérente avec le reste du système.

Si une partition réseau se produit comme dans l'exemple précédent, le DDBMS doit s'assurer que si des agents d'une même transaction globale sont actifs dans différentes partitions, alors il ne doit pas être possible pour le site S1 et les autres sites de la même partition de décider de valider la transaction globale, tandis que le site S2 et les autres sites de sa partition décident de l'abandonner. Cela violerait l'atomicité globale des transactions.

Protocoles de récupération distribuée

Comme mentionné précédemment, la récupération dans un DDBMS est compliquée par le fait que l'atomicité est requise à la fois pour les sous-transactions locales et pour les transactions globales. Les techniques de récupération décrites à la section 22.3 garantissent l'atomicité des sous-transactions, mais le DDBMS doit garantir l'atomicité de la transaction globale. Cela implique de modifier le processus de validation et d'abandon afin qu'une transaction globale ne soit pas validée ou abandonnée tant que toutes ses sous-transactions n'ont pas été validées ou abandonnées avec succès. En outre, le protocole modifié doit tenir compte à la fois des pannes de site et de communication pour garantir que la panne d'un site n'affecte pas le traitement sur un autre site. En d'autres termes, les sites opérationnels ne doivent pas rester bloqués. Les protocoles qui obéissent à cela sont appelés protocoles non bloquants. Dans les deux sections suivantes, nous considérons deux protocoles de validation courants adaptés aux SGBD distribuésÿ: la validation en deux phases (2PC) et la validation en trois phases (3PC), un protocole non bloquant.

Nous supposons que chaque transaction globale a un site qui agit en tant que coordinateur (ou gestionnaire de transaction) pour cette transaction, qui est généralement le site sur lequel la transaction a été initiée. Les sites sur lesquels la transaction globale a des agents sont appelés participants (ou gestionnaires de ressources). Nous supposons que le coordinateur connaît l'identité de tous les participants et que chaque participant connaît l'identité du coordinateur mais pas nécessairement celle des autres participants.

25.4.3 Validation en deux phases (2PC)

Comme son nom l'indique, 2PC fonctionne en deux phases : une phase de vote et une phase de décision. L'idée de base est que le coordinateur demande à tous les participants s'ils sont prêts à valider la transaction. Si un participant vote pour abandonner ou échoue

répondre dans un délai imparti, le coordinateur ordonne à tous les participants d'abandonner la transaction. Si tous votent en faveur de la validation, le coordinateur demande à tous les participants de valider la transaction. La décision globale doit être adoptée par tous les participants. Si un participant vote pour abandonner, il est alors libre d'abandonner la transaction immédiatement ; en fait, n'importe quel site est libre d'interrompre une transaction à tout moment jusqu'à ce qu'il vote pour s'engager. Ce type d'avortement est connu sous le nom d'avortement unilatéral. Si un participant vote pour s'engager, il doit attendre que le coordinateur diffuse le message d'engagement global ou d'abandon global. Ce protocole suppose que chaque site possède son propre journal local et peut donc annuler ou valider la transaction de manière fiable. La validation en deux phases implique des processus attendant des messages provenant d'autres sites. Pour éviter que les processus soient bloqués inutilement, un système de timeouts est utilisé. La procédure pour le coordinateur au commit est la suivanteÿ:

La phase 1

(1) Écrivez un enregistrement begin_commit dans le fichier journal et forcez-le à l'écrire dans un stockage stable. Envoyez un message PREPARE à tous les participants. Attendez que les participants répondent dans un délai d'attente.

Phase 2

(2) Si un participant renvoie un vote ABORT, écrivez un enregistrement d'abandon dans le fichier journal et forcez-le à l'écrire dans un stockage stable. Envoyez un message GLOBAL_ABORT à tous les participants. Attendez que les participants accusent réception dans un délai d'attente.

(3) Si un participant renvoie un vote READY_COMMIT, mettre à jour la liste des participants qui ont répondu. Si tous les participants ont voté COMMIT, écrivez un enregistrement de validation dans le fichier journal et forcez-le à l'écrire dans un stockage stable. Envoyez un message GLOBAL_COMMIT à tous les participants. Attendez que les participants accusent réception dans un délai d'attente.

(4) Une fois tous les accusés de réception reçus, écrivez un message end_transaction dans le fichier journal. Si un site n'accuse pas réception, renvoyez la décision globale jusqu'à ce qu'un accusé de réception soit reçu.

Le coordinateur doit attendre d'avoir reçu les votes de tous les participants. Si un site ne parvient pas à voter, le coordinateur suppose un vote ABORT par défaut et diffuse un message GLOBAL_ABORT à tous les participants. La question de savoir ce qu'il advient du participant défaillant au redémarrage est discutée brièvement. La procédure pour un participant au commit est la suivante :

(1) Lorsque le participant reçoit un message PREPARE, alors soit :

(a) écrire un enregistrement ready_commit dans le fichier journal et forcer l'écriture de tous les enregistrements du journal pour la transaction dans un stockage stable. Envoyez un message READY_COMMIT au coordinateur, ou (b) écrivez un enregistrement d'abandon dans le fichier journal et forcez-le à l'écrire dans un stockage stable. Envoyez un message ABORT au coordinateur. Abandonner unilatéralement la transaction.

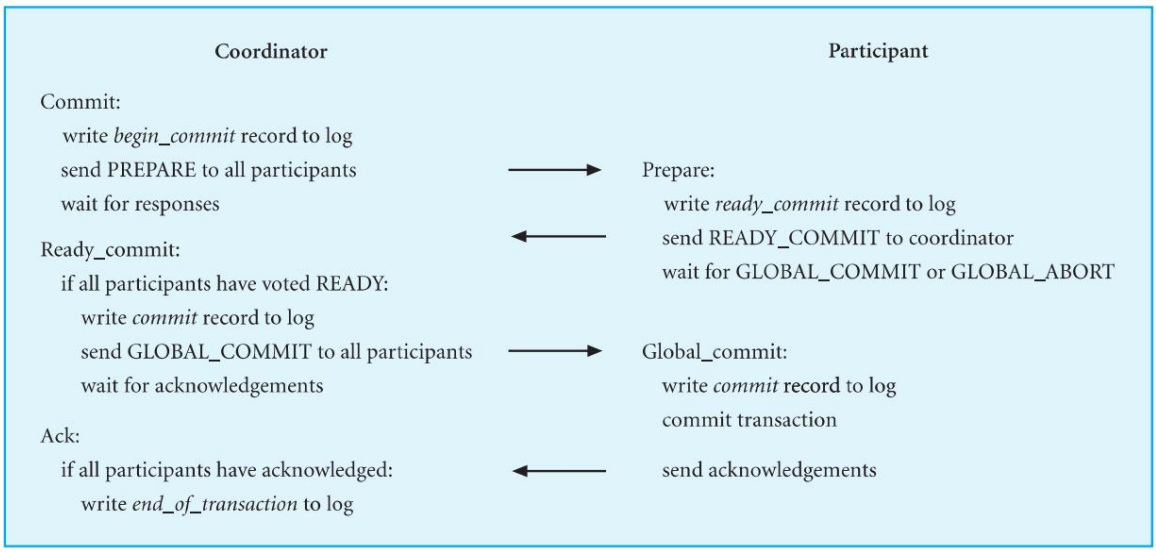
Attendez que le coordinateur réponde dans un délai d'attente.

(2) Si le participant reçoit un message GLOBAL_ABORT, écrivez un enregistrement d'abandon dans le fichier journal et forcez-le à l'écrire dans un stockage stable. Abandonnez la transaction et, une fois terminée, envoyez un accusé de réception au coordinateur.

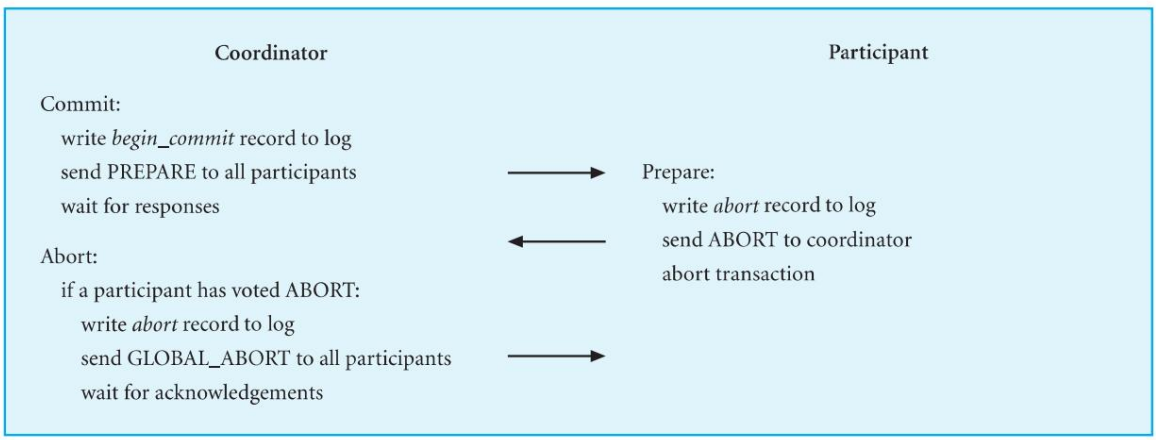
(3) Si le participant reçoit un message GLOBAL_COMMIT, écrivez un enregistrement de validation dans le fichier journal et forcez-le à l'écrire dans un stockage stable. Validez la transaction, libérez tous les verrous qu'elle détient et, une fois terminée, envoyez un accusé de réception au coordinateur.

Si un participant ne reçoit pas d'instruction de vote du coordinateur, il expire simplement et s'interrompt. Par conséquent, un participant peut déjà avoir abandonné et effectué un traitement d'abandon local avant de voter. Le traitement pour le cas où les participants votent COMMIT et ABORT est illustré à la Figure 25.7.

Le participant doit attendre l'instruction GLOBAL_COMMIT ou GLOBAL_ABORT du coordinateur. Si le participant ne reçoit pas l'instruction du coordinateur, ou si le coordinateur ne reçoit pas de réponse d'un participant, il suppose que le site a échoué et un protocole de résiliation doit



(a)



(b)

Figure 25.7 Summary of 2PC: (a) 2PC protocol for participant voting COMMIT; (b) 2PC protocol for participant voting ABORT.

être invoqué. Seuls les sites opérationnels suivent le protocole de résiliation ; les sites qui ont échoué suivent le protocole de récupération au redémarrage.

Protocoles de terminaison pour 2PC Un

protocole de terminaison est invoqué chaque fois qu'un coordinateur ou un participant ne reçoit pas un message attendu et expire. L'action à entreprendre dépend si le coordinateur ou le participant a expiré et le moment où l'expiration s'est produite.

Coordinateur Le coordinateur peut être dans l'un des quatre états pendant le processus de validation : INITIAL, WAITING, DECIDED et COMPLETED, comme indiqué dans le diagramme de transition d'état de la Figure 25.8(a), mais ne peut expirer que dans les deux états du milieu. Les actions à mener sont les suivantes :

- Timeout dans l'état WAITING. Le coordinateur attend que tous les participants reconnaissent s'ils souhaitent valider ou abandonner la transaction. Dans ce cas, le coordinateur ne peut pas valider la transaction, car il n'a pas reçu tous les votes. Cependant, il peut décider d'abandonner globalement la transaction.
- Timeout dans l'état DECIDED. Le coordinateur attend que tous les participants reconnaissent s'ils ont réussi à interrompre ou à valider la transaction. Dans ce cas, le coordinateur renvoie simplement la décision globale aux sites qui n'ont pas acquitté.

Participant Le protocole de terminaison le plus simple consiste à laisser le processus de participant bloqué jusqu'à ce que la communication avec le coordinateur soit rétablie. Le participant peut alors être informé de la décision globale et reprendre le traitement en conséquence. Cependant, d'autres mesures peuvent être prises pour améliorer les performances.

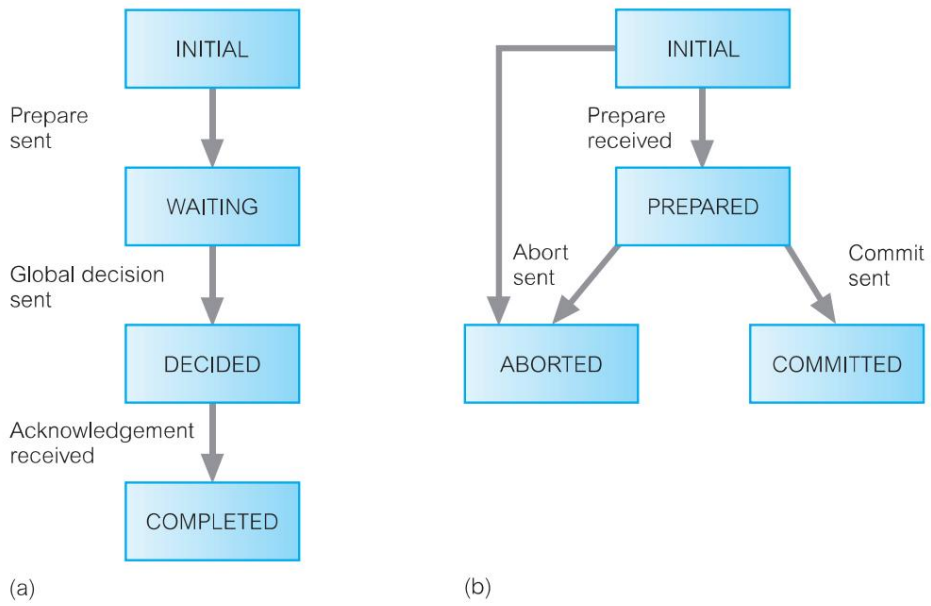


Figure 25.8
State transition diagram for 2PC: (a) coordinator; (b) participant.

Un participant peut être dans l'un des quatre états pendant le processus de validation : INITIAL, PREPARED, ABORTED et COMMITTED, comme le montre la Figure 25.8(b). Cependant, un participant ne peut expirer que dans les deux premiers états, comme suitÿ:

- Timeout dans l'état INITIAL. Le participant attend un message PREPARE du coordinateur, ce qui implique que le coordinateur a dû échouer alors qu'il était dans l'état INITIAL. Dans ce cas, le participant peut interrompre unilatéralement la transaction. S'il reçoit ensuite un message PREPARE, il peut soit l'ignorer, auquel cas le coordinateur expire et abandonne la transaction globale, soit il peut envoyer un message ABORT au coordinateur.

- Timeout dans l'état PREPARED. Le participant attend une instruction pour valider ou abandonner globalement la transaction. Le participant doit avoir voté pour valider la transaction, il ne peut donc pas modifier son vote et abandonner la transaction. De même, il ne peut pas aller de l'avant et valider la transaction, car la décision globale peut être d'abandonner. Sans information supplémentaire, le participant est bloqué. Cependant, le participant pourrait contacter chacun des autres participants en essayant d'en trouver un qui connaît la décision. C'est ce qu'on appelle le protocole de terminaison coopératif. Une façon simple de dire aux participants qui sont les autres participants est que le coordinateur annexe une liste des participants à l'instruction de vote.

Bien que le protocole de terminaison coopératif réduise la probabilité de blocage, le blocage est toujours possible et le processus bloqué devra simplement continuer à essayer de se débloquer au fur et à mesure que les pannes sont réparées. Si seul le coordinateur a échoué et que tous les participants le détectent à la suite de l'exécution du protocole de terminaison, ils peuvent alors élire un nouveau coordinateur et résoudre le blocage de cette manière, comme nous en discuterons brièvement.

Protocoles de récupération pour

2PC Après avoir discuté de l'action à entreprendre par un site opérationnel en cas de panne, nous considérons maintenant l'action à entreprendre par un site défaillant lors de la récupération. L'action au redémarrage dépend à nouveau du stade atteint par le coordinateur ou le participant au moment de l'échec.

Échec du coordinateur Nous considérons trois étapes différentes pour l'échec du coordinateurÿ:

- Échec à l'état INITIAL. Le coordinateur n'a pas encore lancé la procédure de validation. Dans ce cas, la récupération démarre la procédure de validation.
- Échec dans l'état WAITING. Le coordinateur a envoyé le message PREPARE et bien qu'il n'ait pas reçu toutes les réponses, il n'a pas reçu de réponse d'abandon. Dans ce cas, la récupération redémarre la procédure de validation.
- Échec à l'état DECIDED. Le coordinateur a demandé aux participants d'abandonner ou de valider globalement la transaction. Au redémarrage, si le coordinateur a reçu tous les accusés de réception, il peut se terminer avec succès. Sinon, il doit initier le protocole de terminaison décrit précédemment.

Défaillance d'un participant L'objectif du protocole de récupération pour un participant est de s'assurer qu'un processus participant au redémarrage effectue la même action que tous les autres participants et que ce redémarrage peut être effectué indépendamment (c'est-à-dire sans qu'il soit nécessaire de consulter le coordinateur ou le autres participants). Nous considérons trois étapes différentes pour l'échec d'un participantÿ:

- Échec à l'état INITIAL. Le participant n'a pas encore voté sur la transaction. Par conséquent, lors de la récupération, il peut interrompre unilatéralement la transaction, car il aurait été impossible pour le coordinateur d'avoir pris une décision de validation globale sans le vote de ce participant.
- Échec à l'état PREPARED. Le participant a envoyé son vote au coordinateur. Dans ce cas, la récupération se fait via le protocole de terminaison décrit précédemment.
- Échec dans les états ABORTED/COMMITTED. Le participant a terminé la transaction. Par conséquent, au redémarrage, aucune autre action n'est nécessaire.

Protocoles d'élection

Si les participants détectent l'échec du coordinateur (par temporisation), ils peuvent élire un nouveau site pour agir en tant que coordinateur. Un protocole d'élection est que les sites aient un ordre linéaire convenu. Nous supposons que le site Si a l'ordre i dans la séquence, le plus bas étant le coordinateur, et que chaque site connaît l'identification et l'ordre des autres sites du système, dont certains peuvent également avoir échoué. Un protocole d'élection demande à chaque participant opérationnel d'envoyer un message aux sites avec un numéro d'identification supérieur. Ainsi, le site Si enverrait un message aux sites Si11, Si12, . . . , Sn dans cet ordre. Si un site Sk reçoit un message d'un participant de numéro inférieur, alors Sk sait qu'il ne s'agit pas du nouveau coordinateur et arrête d'envoyer messages.

Ce protocole est relativement efficace et la plupart des participants arrêtent d'envoyer des messages assez rapidement. A terme, chaque participant saura s'il existe un participant opérationnel avec un numéro inférieur. S'il n'y en a pas, le site devient le nouveau coordinateur. Si le coordinateur nouvellement élu expire également au cours de ce processus, le protocole d'élection est à nouveau invoqué.

Après la récupération d'un site défaillant, il démarre immédiatement le protocole d'élection. S'il n'y a pas de sites opérationnels avec un numéro inférieur, le site force tous les sites avec un numéro supérieur à le laisser devenir le nouveau coordinateur, qu'il y ait ou non un nouveau coordinateur.

Topologies de communication pour 2PC Il

existe plusieurs topologies de communication différentes (modes d'échange de messages) qui peuvent être utilisées pour mettre en œuvre 2PC. Celui dont il a été question précédemment est appelé 2PC centralisé, car toutes les communications passent par le coordinateur, comme le montre la Figure 25.9(a). Un certain nombre d'améliorations du protocole 2PC centralisé ont été proposées pour tenter d'améliorer ses performances globales, soit en réduisant le nombre de messages à échanger, soit en accélérant le processus de prise de décision. Ces améliorations dépendent de l'adoption de différentes manières d'échanger des messages.

Une alternative est d'utiliser le 2PC linéaire, dans lequel les participants peuvent communiquer entre eux comme le montre la Figure 25.9(b). En 2PC linéaire, les sites sont ordonnés 1, 2, . . . , n, où le site 1 est le coordinateur et les sites restants sont les participants. Le protocole 2PC est implémenté par une chaîne de communication aller du coordinateur au participant n pour la phase de vote et une chaîne de communication retour du participant n au coordinateur pour la phase de décision. Dans la phase de vote, le coordinateur transmet l'instruction de vote au site 2, qui vote puis transmet son

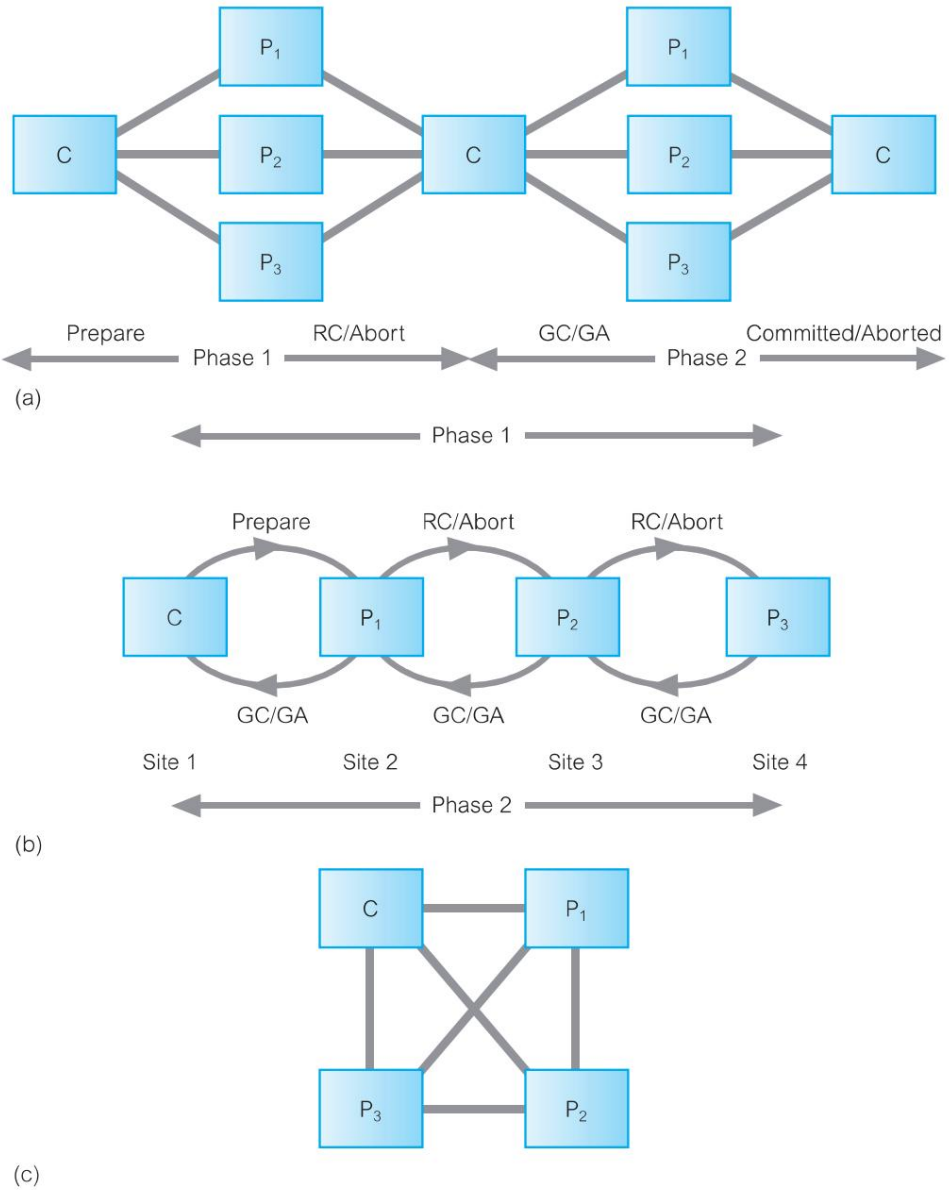


Figure 25.9
2PC topologies: (a) centralized; (b) linear; (c) distributed. C 5 coordinator; PI 5 participant; CR 5 READY_COMMIT; GC 5 GLOBAL_COM-MIT; GA 5 GLOBAL_ABORT.

vote au site 3. Le site 3 combine alors son vote avec celui du site 2 et transmet le vote combiné au site 4, et ainsi de suite. Lorsque le nième participant ajoute son vote, la décision globale est obtenue et transmise aux participants n 2 1, n 2 2, et ainsi de suite et éventuellement au coordinateur. Bien que le 2PC linéaire entraîne moins de messages que le 2PC centralisé, le séquençage linéaire ne permet aucun parallélisme.

Le 2PC linéaire peut être amélioré si le processus de vote adopte le chaînage linéaire des messages alors que le processus de décision adopte la topologie centralisée, de sorte que le site n puisse diffuser la décision globale à tous les participants en parallèle (Bernstein et al., 1987).

Une troisième proposition, connue sous le nom de 2PC distribué, utilise une topologie distribuée comme le montre la Figure 25.9(c). Le coordinateur envoie le message PREPARE à tous les participants, qui à leur tour envoient leur décision à tous les autres sites. Chaque participant attend

pour les messages des autres sites avant de décider de valider ou d'abandonner la transaction. Cela élimine en fait le besoin de la phase de décision du protocole 2PC, car les participants peuvent prendre une décision de manière cohérente, mais indépendante (Skeen, 1981).

25.4.4 Validation triphasée (3PC)

Nous avons vu que 2PC n'est pas un protocole non bloquant, car il est possible que des sites soient bloqués dans certaines circonstances. Par exemple, un processus qui expire après avoir voté la validation mais avant de recevoir l'instruction globale du coordinateur est bloqué s'il ne peut communiquer qu'avec des sites qui ne sont pas non plus au courant de la décision globale. La probabilité qu'un blocage se produise dans la pratique est suffisamment rare pour que la plupart des systèmes existants utilisent 2PC. Cependant, un protocole alternatif non bloquant, appelé protocole de validation en trois phases (3PC), a été proposé (Skeen, 1981). La validation en trois phases est non bloquante pour les défaillances de site, sauf en cas de défaillance de tous les sites. Les défaillances de communication peuvent cependant amener différents sites à prendre des décisions différentes, violant ainsi l'atomicité des transactions mondiales. Le protocole exige que :

- aucun partitionnement du réseau ne doit se produireÿ;
- au moins un site doit toujours être disponibleÿ; au plus K
- sites peuvent échouer simultanément (le système est classé comme Kresilient).

L'idée de base de 3PC est de supprimer la période d'incertitude pour les participants qui ont voté COMMIT et attendent l'abandon global ou le commit global du coordinateur. Le commit en trois phases introduit une troisième phase, appelée precommit, entre le vote et la décision globale. A réception de tous les votes des participants, le coordinateur envoie un message PRECOMMIT global. Un participant qui reçoit le precommit global sait que tous les autres participants ont voté COMMIT et que, avec le temps, le participant lui-même s'engagera définitivement, à moins qu'il n'échoue. Chaque participant accuse réception du message PRECOMMIT et, une fois que le coordinateur a reçu tous les accusés de réception, il émet le commit global. Un vote ABORT d'un participant est traité exactement de la même manière qu'en 2PC.

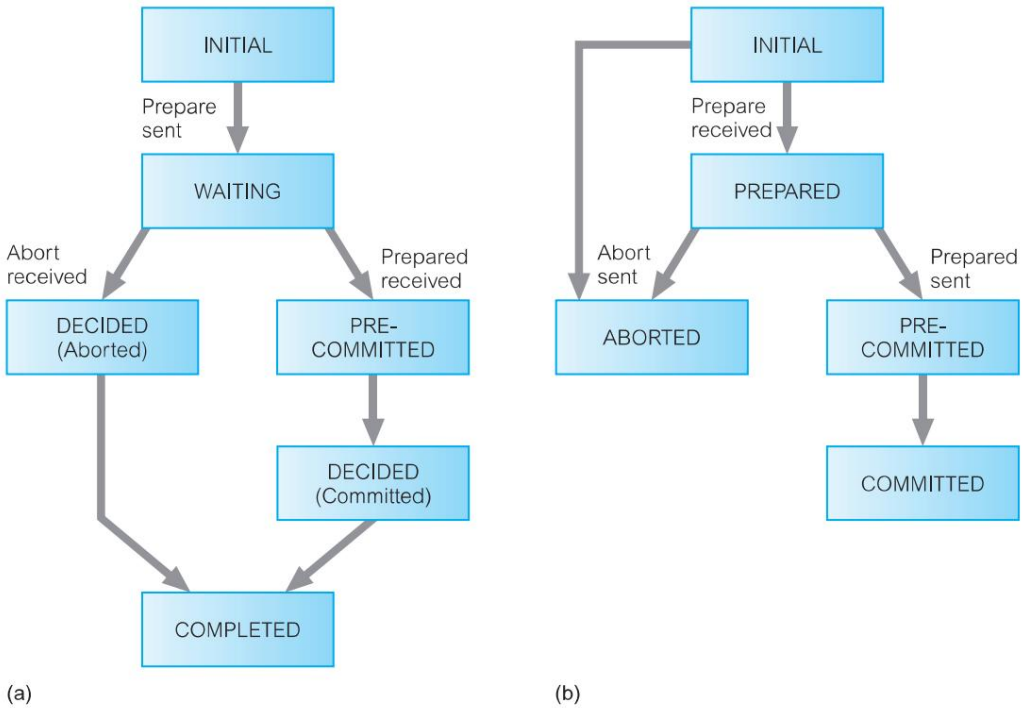
Les nouveaux diagrammes de transition d'état pour le coordinateur et le participant sont illustrés à la Figure 25.10. Le coordinateur et le participant ont encore des périodes d'attente, mais la caractéristique importante est que tous les processus opérationnels ont été informés d'une décision globale d'engagement par le message PRECOMMIT avant l'engagement du premier processus, et peuvent donc agir indépendamment en cas d'échec. . Si le coordinateur échoue, les sites opérationnels peuvent communiquer entre eux et déterminer si la transaction doit être validée ou abandonnée sans attendre que le coordinateur récupère. Si aucun des sites opérationnels n'a reçu de message PRECOMMIT, ils abandonneront la transaction.

Le traitement lorsque tous les participants votent COMMIT est illustré à la Figure 25.11.
Nous discutons maintenant brièvement des protocoles de terminaison et de récupération pour 3PC.

Protocoles de terminaison pour 3PC

Comme pour 2PC, l'action à entreprendre dépend de l'état dans lequel se trouvait le coordinateur ou le participant au moment de la temporisation.

Image 25.10
State transition diagram for 3PC: (a) coordinator; (b) participant.



Coordinateur		Participant
Commettre:		
écrire begin_commit enregistrer dans le journal		
envoyer PREPARE à tous les participants	→	Préparerÿ:
attendre les réponses		écrire l'enregistrement ready_commit dans le
	←	journal envoyer READY_COMMIT au coordinateur
Pre_commitÿ:		attendre PRE_COMMIT ou GLOBAL_ABORT
si tous les participants ont voté PRÊTÿ: écrire		
l'enregistrement pre_commit dans le journal		
envoyer PRE_COMMIT à tous les participants attendre	→	Pre_commitÿ:
les accusés de réception		écrire l'enregistrement pre_commit dans le
	←	journal envoyer un accusé de réception
Ready_commitÿ: une		
fois qu'au moins Kÿparticipants ont acquitté PRE_COMMITÿ:		
écrire un enregistrement de validation	→	Global_commitÿ: écrire
dans le journal envoyer GLOBAL_COMMIT à tous les		l'enregistrement de validation dans le journal
participants attendre les accusés de réception		valider la transaction
	←	envoyer un accusé de réception
Accuséÿ:		
si tous les participants ont reconnu :		
écrire end_of_transaction dans le journal		

Image 25.11 3PC protocol for participant voting COMMIT.

Coordinateur Le coordinateur peut être dans l'un des cinq états pendant le processus de validation, comme illustré à la Figure 25.10(a), mais peut expirer dans seulement trois états. Les actions à mener sont les suivantes :

- Timeout dans l'état WAITING. C'est la même chose que dans 2PC. Le coordinateur attend que tous les participants reconnaissent s'ils souhaitent valider ou abandonner la transaction, afin qu'il puisse décider d'abandonner globalement la transaction.
- Timeout dans l'état PRECOMMITTED. Les participants ont reçu le message PRECOMMIT, donc les participants seront dans l'état PRECOMMIT ou READY. Dans ce cas, le coordinateur peut terminer la transaction en écrivant l'enregistrement de validation dans le fichier journal et en envoyant le message GLOBALCOMMIT aux participants.
- Timeout dans l'état DECIDED. C'est la même chose que dans 2PC. Le coordinateur attend que tous les participants reconnaissent s'ils ont réussi à abandonner ou à valider la transaction, de sorte qu'il peut simplement envoyer la décision globale à tous les sites qui n'ont pas confirmé.

Participant Le participant peut être dans l'un des cinq états pendant le processus de validation, comme illustré à la Figure 25.10(b), mais peut expirer dans seulement trois états. Les actions à mener sont les suivantes :

- Timeout dans l'état INITIAL. C'est la même chose que dans 2PC. Le participant attend le message PREPARE et peut donc interrompre unilatéralement la transaction.
- Timeout dans l'état PREPARED. Le participant a envoyé son vote au coordinateur et attend le message PRECOMMIT ou ABORT. Dans ce cas, le participant suivra un protocole d'élection pour élire un nouveau coordinateur pour la transaction et terminera comme nous en discuterons ensuite.
- Timeout dans l'état PRECOMMITTED. Le participant a envoyé l'accusé de réception du message PRECOMMIT et attend le message COMMIT. Encore une fois, le participant suivra un protocole d'élection pour élire un nouveau coordinateur pour la transaction et terminera comme nous en discuterons ensuite.

Protocoles de récupération pour

3PC Comme pour 2PC, l'action au redémarrage dépend de l'état atteint par le coordinateur ou le participant au moment de l'échec.

Défaillance du coordinateur Nous considérons quatre états différents pour la défaillance du coordinateur :

- Echec à l'état INITIAL. Le coordinateur n'a pas encore lancé la procédure de validation. Dans ce cas, la récupération démarre la procédure de validation.
- Échec dans l'état WAITING. Les participants peuvent avoir élu un nouveau coordinateur et mis fin à la transaction. Au redémarrage, le coordinateur doit contacter d'autres sites pour déterminer le sort de la transaction.
- Échec dans l'état PRECOMMITTED. Encore une fois, les participants peuvent avoir élu un nouveau coordinateur et mis fin à la transaction. Au redémarrage, le coordinateur doit contacter d'autres sites pour déterminer le sort de la transaction.
- Échec dans l'état DECIDED. Le coordinateur a demandé aux participants d'abandonner ou de valider globalement la transaction. Au redémarrage, si le coordinateur a reçu tous les accusés de réception, il peut se terminer avec succès. Sinon, il doit initier le protocole de terminaison décrit précédemment.

Participant Nous considérons quatre états différents pour l'échec d'un participantÿ:

- Echec à l'état INITIAL. Le participant n'a pas encore voté sur la transaction. Par conséquent, lors de la récupération, il peut interrompre unilatéralement la transaction.
- Échec à l'état PREPARED. Le participant a envoyé son vote au coordinateur. Dans ce cas, le participant devra contacter d'autres sites pour déterminer le sort de la transaction.
- Échec dans l'état PRECOMMITTED. Le participant devra contacter d'autres sites pour déterminer le sort de la transaction.
- Échec dans les états ABORTED/COMMITTED. Le participant a terminé la transaction. Par conséquent, au redémarrage, aucune autre action n'est nécessaire.

Protocole de résiliation suite à l'élection d'un nouveau coordinateur Le protocole d'élection discuté pour 2PC peut être utilisé par les participants pour élire un nouveau coordinateur après un délai d'attente. Le coordinateur nouvellement élu enverra un message STATEREQ à tous les participants impliqués dans l'élection afin de déterminer la meilleure façon de poursuivre la transaction. Le nouveau coordinateur peut utiliser les règles suivantesÿ:

(1) Si un participant a abandonné, alors la décision globale est abandonnée.

(2) Si un participant a engagé la transaction, alors la décision globale est commettre.

(3) Si tous les participants qui répondent sont incertains, alors la décision est abandonnée.

(4) Si un participant peut valider la transaction (est dans l'état PRECOMMIT), alors la décision globale est commit. Pour éviter le blocage, le nouveau coordinateur enverra d'abord le message PRECOMMIT et, une fois que les participants auront accusé réception, enverra le message GLOBALCOMMIT.

25.4.5 Partitionnement du réseau

Lorsqu'un partitionnement du réseau se produit, le maintien de la cohérence de la base de données peut être plus difficile, selon que les données sont répliquées ou non. Si les données ne sont pas répliquées, nous pouvons autoriser une transaction à se poursuivre si elle ne nécessite aucune donnée d'un site en dehors de la partition dans laquelle elle est initiée. Sinon, la transaction doit attendre que les sites auxquels elle a besoin d'accéder soient à nouveau disponibles. Si les données sont répliquées, la procédure est beaucoup plus compliquée. Nous considérons deux exemples d'anomalies pouvant survenir avec des données répliquées dans un réseau partitionné basé sur une simple relation de compte bancaire contenant un solde client.

Identification des mises à jour Les opérations de mise à jour réussies par les utilisateurs dans différentes partitions peuvent être difficiles à observer, comme illustré à la Figure 25.12. Dans la partition P1, une transaction a retiré 10 £ d'un compte (avec solde balx), et dans la partition P2, deux transactions ont chacune retiré 5 £ du même compte. En supposant qu'au début, les deux partitions ont 100 £ en balx, puis à la fin, elles ont toutes les deux 90 £ en balx. Lorsque les partitions se rétablissent, il ne suffit pas de vérifier la valeur dans balx et de supposer que les champs sont cohérents si les valeurs sont identiques. Dans ce cas, la valeur après l'exécution des trois transactions devrait être de 80 £.

Time	P_1	P_2
t_1	begin_transaction	begin_transaction
t_2	$bal_x = bal_x - 10$	$bal_x = bal_x - 5$
t_3	write(bal_x)	write(bal_x)
t_4	commit	commit
t_5		begin_transaction
t_6		$bal_x = bal_x - 5$
t_7		write(bal_x)
t_8		commit

Image 25.12 Identifying updates.

Maintien de l'intégrité Les

opérations de mise à jour réussies par les utilisateurs dans différentes partitions peuvent facilement violer les contraintes d'intégrité, comme illustré à la Figure 25.13. Supposons qu'une banque place une contrainte sur un compte client (avec solde balx) qu'il ne peut pas descendre en dessous de 0 £. Dans la partition P1, une transaction a retiré 60 £ du compte et dans la partition P2, une transaction a retiré 50 £ du même compte. En supposant qu'au début les deux partitions ont 100 £ en balx, puis à la fin, l'une a 40 £ en balx et l'autre a 50 £. Il est important de noter qu'aucun des deux n'a violé la contrainte d'intégrité. Cependant, lorsque les partitions se rétabliront et que les transactions seront toutes deux entièrement mises en œuvre, le solde du compte sera de ÿ10 £ et la contrainte d'intégrité aura été violée.

Le traitement dans un réseau partitionné implique un compromis entre disponibilité et exactitude (Davidson, 1984 ; Davidson et al., 1985). L'exactitude absolue est plus facilement assurée si aucun traitement des données répliquées n'est autorisé pendant le partitionnement. D'autre part, la disponibilité est maximisée si aucune restriction n'est imposée sur le traitement des données répliquées lors du partitionnement.

En général, il n'est pas possible de concevoir un protocole de validation atomique non bloquant pour des réseaux arbitrairement partitionnés (Skeen, 1981). Étant donné que la récupération et le contrôle de la concurrence sont étroitement liés, les techniques de récupération qui seront utilisées après le partitionnement du réseau dépendront de la stratégie particulière de contrôle de la concurrence utilisée. Les méthodes sont classées comme pessimistes ou optimistes.

Protocoles pessimistes Les protocoles pessimistes choisissent la cohérence de la base de données plutôt que la disponibilité et ne permettraient donc pas aux transactions de s'exécuter dans une partition s'il n'y a aucune garantie que la cohérence puisse être maintenue. Le protocole utilise un algorithme de contrôle de concurrence pessimiste tel que la copie primaire 2PL ou le verrouillage majoritaire, comme discuté à la section 25.2. La récupération à l'aide de cette approche est beaucoup plus simple, car les mises à jour auraient été confinées à une seule partition distincte. La récupération du réseau consiste simplement à propager toutes les mises à jour sur tous les autres sites.

Time	P_1	P_2
t_1	begin_transaction	begin_transaction
t_2	$bal_x = bal_x - 60$	$bal_x = bal_x - 50$
t_3	write(bal_x)	write(bal_x)
t_4	commit	commit

Image 25.13 Maintaining integrity.

Protocoles optimistes Les protocoles optimistes, quant à eux, choisissent la disponibilité de la base de données au détriment de la cohérence et utilisent une approche optimiste du contrôle de la concurrence dans laquelle les mises à jour sont autorisées à se dérouler indépendamment dans les différentes partitions. Par conséquent, des incohérences sont probables lorsque les sites se rétablissent.

Pour déterminer s'il existe des incohérences, des graphiques de priorité peuvent être utilisés pour suivre les dépendances entre les données. Les graphes de priorité sont similaires aux graphes d'attente abordés dans la section 22.2.4 et montrent quelles transactions ont lu et écrit quels éléments de données. Pendant que le réseau est partitionné, les mises à jour se déroulent sans restriction et les graphiques de priorité sont conservés par chaque partition. Lorsque le réseau a récupéré, les graphiques de priorité de toutes les partitions sont combinés. Les incohérences sont indiquées s'il y a un cycle dans le graphique. La résolution des incohérences dépend de la sémantique des transactions, et il n'est donc généralement pas possible pour le gestionnaire de reprise de rétablir la cohérence sans l'intervention de l'utilisateur.

25.5 Le modèle de traitement des transactions distribuées X/Open

L'Open Group est un consortium international indépendant des fournisseurs d'utilisateurs, de fournisseurs de logiciels et de fournisseurs de matériel dont la mission est de provoquer la création d'une infrastructure d'information mondiale viable. Elle a été formée en février 1996 par la fusion de X/Open Company Ltd (fondée en 1984) et de l'Open Software Foundation (fondée en 1988). X/Open a créé le groupe de travail sur le traitement des transactions distribuées (DTP) dans le but de spécifier et de promouvoir des interfaces de programmation appropriées pour le traitement des transactions. À cette époque, cependant, les systèmes de traitement des transactions étaient des environnements d'exploitation complets, de la définition de l'écran à la mise en œuvre de la base de données. Plutôt que d'essayer de fournir un ensemble de normes couvrant tous les domaines, le groupe s'est concentré sur les éléments d'un système de traitement des transactions qui fournissaient les propriétés ACID (atomicité, cohérence, isolation et durabilité) dont nous avons parlé à la section 22.1.1. La norme (de jure) X/Open DTP qui a émergé spécifiait trois composants interactifsÿ: une application, un gestionnaire de transactions (TM) et un gestionnaire de ressources (RM).

Tout sous-système qui implémente des données transactionnelles peut être un gestionnaire de ressources, tel qu'un système de base de données, un système de fichiers transactionnel et un gestionnaire de session transactionnel. Le TM est chargé de définir la portée d'une transaction, c'est-à-dire quelles opérations font partie d'une transaction. Il est également chargé d'attribuer une identification unique à la transaction qui peut être partagée avec d'autres composants, et de coordonner les autres composants pour déterminer le résultat de la transaction. Une MT peut également communiquer avec d'autres MT pour coordonner l'exécution de transactions distribuées. L'application appelle le TM pour démarrer une transaction, puis appelle les RM pour manipuler les données, en fonction de la logique de l'application, et appelle enfin le TM pour terminer la transaction. Le TM communique avec les RM pour coordonner la transaction.

De plus, le modèle X/Open définit plusieurs interfaces, comme illustré à la Figure 25.14. Une application peut utiliser l'interface TX pour communiquer avec un TM. L'interface TX fournit des appels qui définissent la portée de la transaction (parfois appelée la démarcation de la transaction) et s'il faut valider/abandonner la transaction. Un TM communique des informations transactionnelles avec des RM via l'interface XA. Enfin, une application peut communiquer directement avec les RM via une interface de programmation native, telle que SQL ou ISAM.

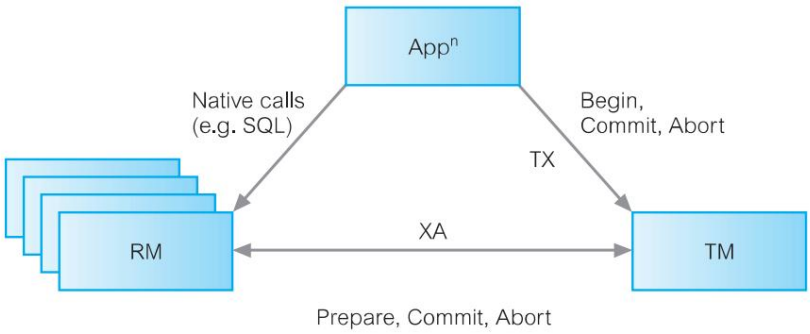


Illustration 25.14 X/Open interfaces.

L'interface TX comprend les procédures suivantesÿ:

- tx_open et tx_close, pour ouvrir et fermer une session avec un TMÿ;
- tx_begin, pour démarrer une nouvelle transactionÿ; tx_commit et tx_abort,
- pour valider et abandonner une transaction.

L'interface XA se compose des procédures suivantesÿ:

- xa_open et xa_close, pour se connecter et se déconnecter d'un RMÿ; xa_start
- et xa_end, pour démarrer une nouvelle transaction avec l'ID de transaction donné et pour la terminerÿ; xa_rollback, pour annuler la transaction avec l'ID de transaction donnéÿ; xa_prepare, pour
- préparer la transaction avec l'ID de transaction donné pour la validation/abandon globalÿ; xa_commit,
- pour valider globalement la transaction avec l'ID de transaction donnéÿ; xa_recover, pour récupérer une liste des transactions préparées, validées de manière heuristique ou abandonnées de manière
- heuristique. Lorsqu'un RM est bloqué, un opérateur peut imposer une décision heuristique
- (généralement l'abandon), permettant de libérer les ressources bloquées. Lorsque la MT récupère, cette liste de transactions peut être utilisée pour indiquer aux transactions dans le doute leur décision réelle (commit ou abort). A partir de son journal, il peut également notifier à l'application toute décision heuristique erronée. xa_forget, pour permettre à un RM d'oublier la transaction heuristique avec le transac donné
- d'identification.

Par exemple, considérez le fragment de code d'application suivantÿ:

```
tx_begin();
  EXEC SQL UPDATE Personnel SET salaire 5 salaire *1.05
  WHERE position 5 'Manager'; EXEC SQL UPDATE Personnel
  SET salaire 5 salaire *1.04 WHERE poste ,. 'Gestionnaire';
  tx_commit();
```

Lorsque l'application appelle la fonction d'interface de niveau d'appel (CLI) tx_begin(), le TM enregistre le début de la transaction et attribue à la transaction un identifiant unique. La MT utilise ensuite XA pour informer le serveur de base de données SQL qu'une transaction est en cours. Une fois qu'un gestionnaire de ressources a reçu ces informations, il suppose que tous les appels qu'il reçoit de l'application font partie de la transaction, dans ce cas les deux instructions de mise à jour SQL. Enfin, lorsque l'application invoque la fonction tx_commit(), le TM interagit avec le RM pour valider la transaction. Si la demande était

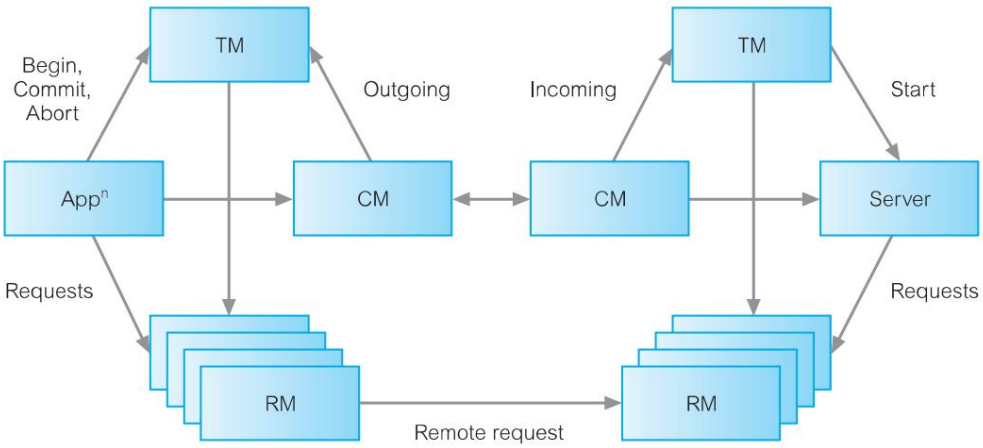


Image 25.15
X/Open interfaces in a distributed environment.

travaillant avec plus d'un RM, à ce stade, le TM utiliserait le protocole de validation en deux phases pour synchroniser la validation avec les RM.

Dans l'environnement distribué, nous devons modifier le modèle décrit précédemment pour permettre une transaction composée de sous-transactions, chacune s'exécutant sur un site distant par rapport à une base de données distante. Le modèle X/Open DTP pour un environnement distribué est illustré à la Figure 25.15. Le modèle X/Open communique avec les applications via un type spécial de gestionnaire de ressources appelé Communications Manager (CM). Comme tous les gestionnaires de ressources, le CM est informé des transactions par les TM et les applications appellent un CM en utilisant son interface native. Deux mécanismes sont nécessaires dans ce cas : un mécanisme d'invocation à distance et un mécanisme de transaction distribuée. L'invocation à distance est assurée par le ROSE (Remote Operations Service) de l'ISO et par les mécanismes d'appel de procédure à distance (RFC). X/Open spécifie le protocole de communication OSITP (Open Systems Interconnection Transaction Processing) pour la coordination des transactions distribuées (l'interface TM–TM).

X/Open DTP prend en charge non seulement les transactions plates, mais également les transactions chaînées et imbriquées (voir Section 22.4). Avec les transactions imbriquées, une transaction s'arrêtera si une sous-transaction échoue.

Le modèle de référence X/Open est bien établi dans l'industrie. Un certain nombre de moniteurs tiers de traitement des transactions (TP) prennent en charge l'interface TX, et de nombreux fournisseurs de bases de données commerciales proposent une implémentation de l'interface XA. Les principaux exemples incluent CICS et Encina d'IBM (qui sont principalement utilisés sur IBM AIX ou Windows NT et désormais intégrés dans IBM TXSeries), Tuxedo de BEA Systems, Oracle, Informix et SQL Server.

25.6 Optimisation des requêtes distribuées

Au chapitre 23, nous avons discuté du traitement et de l'optimisation des requêtes pour les SGBDR centralisés. Nous avons discuté de deux techniques d'optimisation des requêtesÿ:

- le premier qui utilisait des règles heuristiques pour ordonner les opérations dans
- une requêteÿ; la seconde comparait différentes stratégies en fonction de leurs coûts relatifs et sélectionnait celle qui minimisait l'utilisation des ressources.

Dans les deux cas, nous avons représenté la requête sous la forme d'un arbre d'algèbre relationnelle pour faciliter le traitement ultérieur. L'optimisation des requêtes distribuées est plus complexe, en raison de la distribution des données. La figure 25.16 montre comment la requête distribuée est traitée et optimisée en plusieurs couches distinctes composées deÿ:

- Décomposition des requêtes. Cette couche prend une requête exprimée sur les relations globales et effectue une optimisation partielle en utilisant les techniques décrites au chapitre 23. Le résultat est une forme d'arbre d'algèbre relationnelle basé sur des relations globales.
- Localisation des données. Cette couche tient compte de la manière dont les données ont été distribuées. Une autre itération d'optimisation est effectuée en remplaçant les relations globales aux feuilles de l'arbre d'algèbre relationnelle par leurs algorithmes de reconstruction (parfois appelés programmes de localisation de données) ; c'est-à-dire les opérations d'algèbre relationnelle qui reconstruisent les relations globales à partir des fragments constitutifs.
- Optimisation globale. Cette couche prend en compte des informations statistiques pour trouver un plan d'exécution quasi optimal. La sortie de cette couche est une stratégie d'exécution basée sur des fragments avec des primitives de communication ajoutées pour envoyer des parties de la requête aux SGBD locaux pour y être exécutées et recevoir les résultats.
- Optimisation locale. Alors que les trois premières couches sont exécutées sur le site de contrôle (généralement le site qui a lancé la requête), cette couche particulière est exécutée sur chacun des sites locaux impliqués dans la requête. Chaque SGBD local effectuera sa propre optimisation locale en utilisant les techniques décrites au chapitre 23.

Nous abordons maintenant les deux couches médianes de cette architecture.

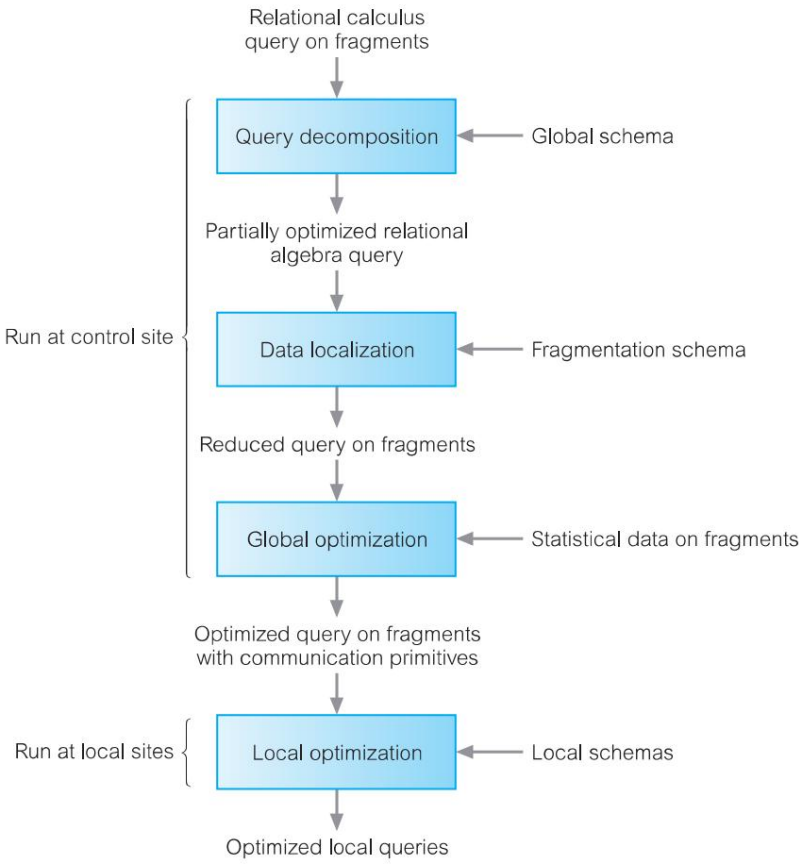


Figure 25.16
Distributed query processing.

25.6.1 Localisation des données

Comme discuté précédemment, l'objectif de cette couche est de prendre une requête exprimée sous la forme d'une forme d'arbre d'algèbre relationnelle et de prendre en compte la distribution des données pour effectuer une optimisation supplémentaire à l'aide de règles heuristiques. Pour ce faire, nous remplaçons les relations globales aux feuilles de l'arbre par leurs algorithmes de reconstruction, c'est-à-dire les opérations d'algèbre relationnelle qui reconstruisent les relations globales à partir des fragments constitutifs. Pour la fragmentation horizontale, l'algorithme de reconstruction est l'opération Union ; pour la fragmentation verticale, il s'agit de l'opération Join. L'arbre d'algèbre relationnelle formé en appliquant les algorithmes de reconstruction est parfois appelé arbre d'algèbre relationnelle générique. Par la suite, nous utilisons des techniques de réduction pour générer une requête plus simple et optimisée. La technique de réduction particulière que nous employons dépend du type de fragmentation impliqué. Nous considérons des techniques de réduction pour les types de fragmentation suivantsÿ:

- fragmentation horizontale primaire;
- fragmentation verticale; fragmentation
- horizontale dérivée.

Réduction pour la fragmentation horizontale primaire

Pour la fragmentation horizontale primaire, nous considérons deux cas : réduction avec l'opération de sélection et réduction pour l'opération de jointure. Dans le premier cas, si le prédicat de sélection contredit la définition du fragment, alors il en résulte une relation intermédiaire vide et les opérations peuvent être éliminées. Dans le second cas, on utilise d'abord la règle de transformation qui permet de commuter l'opération Join avec l'opération Union :

$$(R \bowtie R_2) \bowtie R_3 \quad (R \bowtie R_3) \quad (R_2 \bowtie R_3)$$

Nous examinons ensuite chacune des opérations de jointure individuelles pour déterminer s'il existe des jointures redondantes qui peuvent être éliminées du résultat. Une jointure redondante existe si les prédicats de fragment ne se chevauchent pas. Cette règle de transformation est importante dans les DDBMS, permettant d'implémenter une jointure de deux relations comme une union de jointures partielles, où chaque partie de l'union peut être effectuée en parallèle. Nous illustrons l'utilisation de ces deux règles de réduction dans l'exemple 25.2.

EXEMPLE 25.2 Réduction pour la fragmentation horizontale primaire

Dressez la liste des appartements à louer avec les coordonnées de l'agence correspondante.

Nous pouvons exprimer cette requête en SQL commeÿ:

```
SELECT *
FROM Branche b, PropertyForRent
p WHERE b.branchNo 5 p.branchNo AND p.type 5 'Flat';
```

Supposons maintenant que PropertyForRent et Branch sont fragmentés horizontalement comme suitÿ:

P1: sbranchNo 5'B003' Ù type 5'House' (PropertyForRent)
P2: sbranchNo 5'B003' Ù type 5'Flat' (PropertyForRent)
P3: sbranchNo 5'B003' (PropertyForRent)

B1: sbranchNo 5'B003 ' (Branche)
B2: sbranchNo 5BB3 ' (Branche)

L'arbre d'algèbre relationnelle générique pour cette requête est montré dans la Figure 25.17(a). Si nous commutons les opérations de sélection et d'union, nous obtenons l'arbre d'algèbre relationnelle illustré dans la Figure 25.17(b). Cet arbre est obtenu en observant que la branche suivante du tree est redondant (il ne produit aucun tuple contribuant au résultat) et peut être supprimé:

sp.type5'Flat'(P1)5stype5'Flat' (sbranchNo5'B003' Ù type5'House'(PropertyForRent)) 5 Ø

De plus, parce que le prédicat de sélection (sp.type 5'Flat') est un sous-ensemble de la définition du fragmentation pour P , la sélection n'est pas nécessaire. Si nous commutons maintenant les Join et opérations d'union, nous obtenons l'arbre représenté sur la Figure 25.17(c). Parce que la seconde et les troisièmes jointures ne contribuent pas au résultat, elles peuvent être éliminées, ce qui réduit requête illustrée à la Figure 25.17(d).

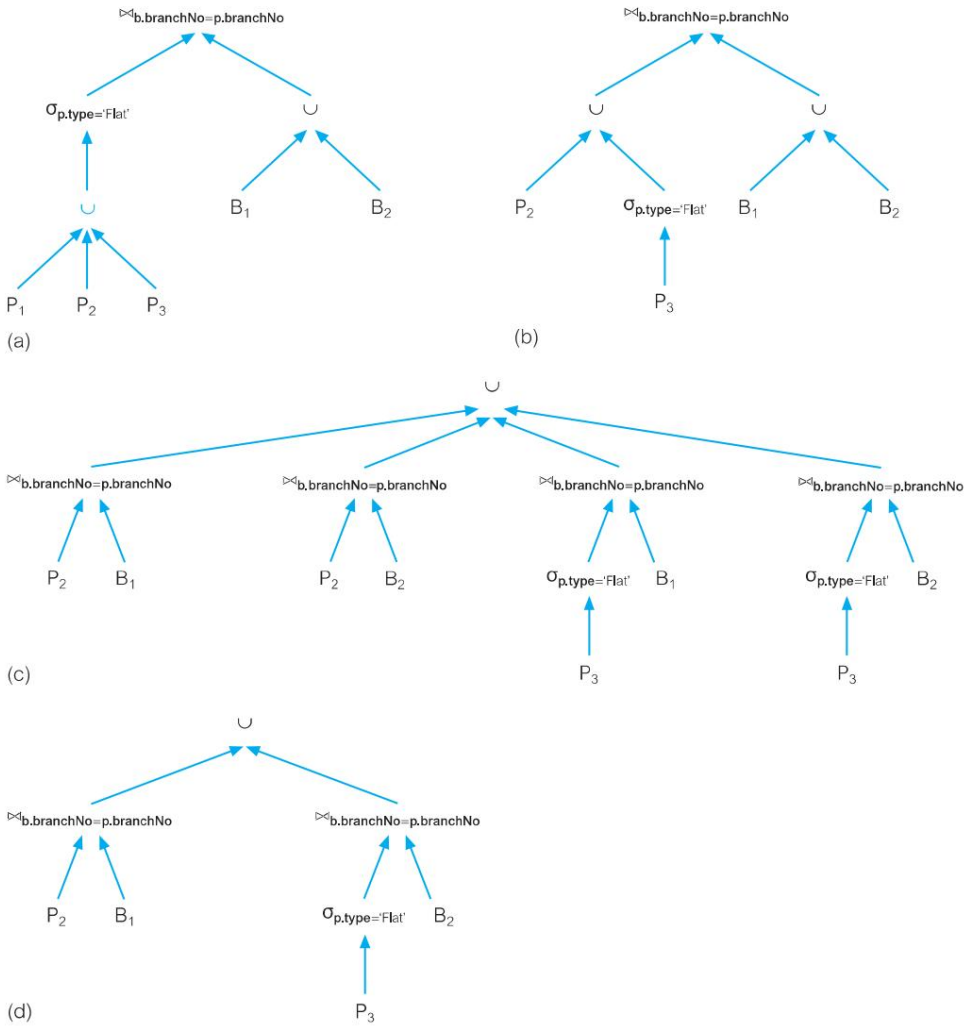


Illustration 25.17 Relational algebra trees for Example 25.2: (a) generic tree; (b) tree resulting from reduction by selection; (c) tree resulting from commuting join and union; (d) reduced tree.

Réduction pour fragmentation verticale La

réduction pour fragmentation verticale consiste à supprimer les fragments verticaux qui n'ont aucun attribut en commun avec les attributs de projection, à l'exception de la clé de la relation.

EXEMPLE 25.3 Réduction pour fragmentation verticale

Dressez la liste des noms de chaque membre du personnel.

Nous pouvons exprimer cette requête en SQL commeÿ:

```
SELECT fName, IName
DU Personnelÿ;
```

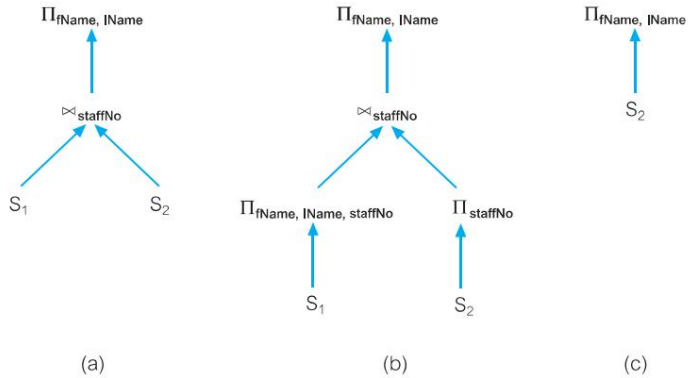


Illustration 25.18 Relational algebra trees for Example 25.3: (a) generic tree; (b) commuting Projection and Join; (c) reduced tree.

Nous utiliserons le schéma de fragmentation pour Staff que nous avons utilisé dans l'exemple 24.3ÿ:

S : staffNo, poste, sexe, date de naissance, salaire(Staff)

S_2: staffNo, fName, lName, branchNo(personnel)

L'arbre d'algèbre relationnelle générique pour cette requête est montré dans la Figure 25.18(a). En commutant les opérations Projection et Jointure, l'opération Projection sur 1 est redondante car les attributs de projection fName et lName ne font pas partie de 1. L'arbre réduit est représenté sur la Figure 25.18(b).

Réduction pour la fragmentation horizontale dérivée La

réduction pour la fragmentation horizontale dérivée utilise à nouveau la règle de transformation qui permet de commuter les opérations Join et Union. Dans ce cas, nous utilisons la connaissance que la fragmentation d'une relation est basée sur l'autre relation et, en commutant, certaines des jointures partielles devraient être redondantes.

EXEMPLE 25.4 Réduction pour la fragmentation horizontale dérivée

Dressez la liste des clients enregistrés à la succursale B003 avec les détails de la succursale.

Nous pouvons exprimer cette requête en SQL commeÿ:

```
SÉLECTIONNER *
FROM Succursale b, Client c
WHERE b.branchNo 5 c.branchNo AND b.branchNo 5 'B003';
```

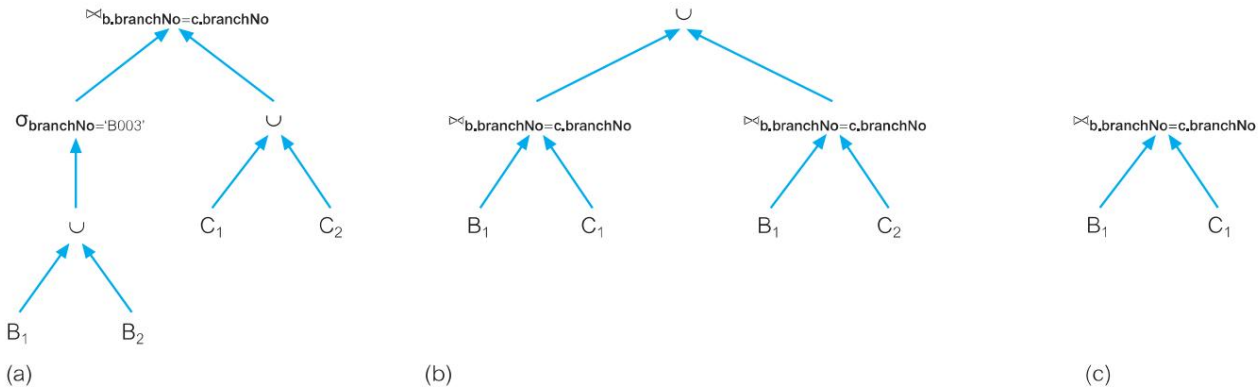


Illustration 25.19 Relational algebra trees for Example 25.4: (a) generic tree; (b) tree resulting from commuting join and union; (c) reduced tree.

Nous supposons que Branch est fragmenté horizontalement comme dans l'exemple 25.2 et que la fragmentation pour Client est dérivée de Branch :

B. 5 sbranchNo5'B003'(Branche) B2 5 sbranchNoÿ! 5'B003 '(Branche)

Vous 5 Client2brancheNonB. je 5 1, 2

L'arbre générique de l'algèbre relationnelle est représenté sur la Figure 25.19(a). Si on commute le opérations de sélection et d'union, la sélection sur le fragment est redondante eB2cette branche de l'arbre peut être éliminée. Toute l'opération de sélection peut être éliminée car le fragment 1Best lui-même défini sur la branche B003. Si nous commutons maintenant les Join et Opérations d'union, nous obtenons l'arbre représenté sur la Figure 25.19(b). La deuxième opération de jointure entre 1 et CBproduit une2 relation nulle et peut être éliminée, donnant la valeur réduite arbre de la Figure 25.19(c).

25.6.2 Jointures distribuées

Comme nous l'avons noté précédemment, la jointure est l'une des opérations d'algèbre relationnelle les plus coûteuses. Une approche utilisée dans l'optimisation des requêtes distribuées consiste à remplacer Jointures par combinaisons de semi-jointures (voir Section 5.1.3). L'opération de semi-jointure a la propriété importante de réduire la taille de la relation d'opérande. Quand le principale composante de coût est le temps de communication, l'opération Semijoin est particulièrement utile pour améliorer le traitement des jointures distribuées en réduisant la quantité de données transférées entre les sites.

Par exemple, supposons que nous souhaitions évaluer l'expression de jointure R1 R sur le site2 S2, où R1 et R sont des2 fragments stockés aux sites S1 et S2, respectivement. R1 et R sont définis sur les attributs A 5 (, a1, a , . . . , ax) et B 5 (, b1, b , . . . , bm), respectivement. Nous pouvons changer cela pour utiliser l'opération Semijoin à la place. Tout d'abord, notez

que nous pouvons réécrire une jointure comme suitÿ:

R1 R5 (R1 avec R) R

On peut donc évaluer l'opération Join comme suit :

(1) Évaluer R 5 (R) à S2 (nécessite uniquement les attributs de jointure à S1).

(2) Transférez R vers le site S1.

(3) Évaluer R 5 (R1 R à S1.

(4) Transférez R vers le site S2.

(5) Évaluer RR à S2.

L'utilisation de Semijoins est bénéfique s'il n'y a que quelques tuples de R1 qui participent à la jointure de R1 et R , alors que l'approche de jointure est meilleure si la plupart des tuples de R1 participent à la jointure, car l'approche Semijoin nécessite un transfert supplémentaire de une projection sur l'attribut join. Pour une étude plus complète des semi-jointures, le lecteur intéressé est renvoyé à l'article de Bernstein et Chiu (1981). Il convient de noter que l'opération Semijoin n'est utilisée dans aucun des principaux DDBMS commerciaux.

25.6.3 Optimisation globale Comme discuté

précédemment, l'objectif de cette couche est de prendre le plan de requête réduit pour la couche de localisation des données et de trouver une stratégie d'exécution quasi optimale. Comme pour l'optimisation centralisée des requêtes abordée à la section 23.5, cela implique d'évaluer le coût de différentes stratégies d'exécution et de choisir la stratégie optimale dans cet espace de recherche.

Frais

Dans un SGBD centralisé, le coût d'exécution est une combinaison des coûts d'E/S et de CPU. Étant donné que l'accès au disque est lent par rapport à l'accès à la mémoire, l'accès au disque a tendance à être le coût dominant dans le traitement des requêtes pour un SGBD centralisé, et c'est celui sur lequel nous nous sommes concentrés exclusivement lors de la fourniture des estimations de coûts au chapitre 23. Cependant, dans l'environnement distribué , la vitesse du réseau sous-jacent doit être prise en considération lors de la comparaison de différentes stratégies. Comme nous l'avons mentionné dans la section 24.5.3, un WAN peut avoir une bande passante de seulement quelques kilo-octets par seconde et dans ce cas, nous pourrions ignorer les coûts de traitement locaux. D'un autre côté, un LAN est généralement beaucoup plus rapide qu'un WAN, bien que toujours plus lent que l'accès au disque, mais dans ce cas, aucun coût ne domine et tous doivent être pris en compte.

De plus, pour un SGBD centralisé, nous avons considéré un modèle de coût basé sur le coût total (temps) de toutes les opérations de la requête. Un modèle de coût alternatif est basé sur le temps de réponse, c'est-à-dire le temps écoulé entre le début et la fin de la requête. Ce dernier modèle tient compte du parallélisme inhérent à un système distribué. Ces deux modèles de coûts peuvent produire des résultats différents. Par exemple, considérez le transfert de données illustré à la Figure 25.20, où x bits de données sont transférés du site 1 au site 2 et y bits du site 2 au site 3. En utilisant une formule de coût total, le coût de ces opérations estÿ:

Temps total 5 2*C0 1 (x 1 y)/transmission_rate

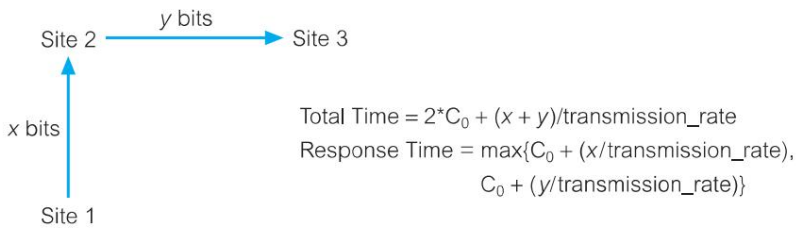


Image 25.20
Example of effect of different cost models when transferring data between sites.

En utilisant une formule de temps de réponse, le coût de ces opérations est deÿ:

Temps de réponse 5 max{C0 1 (x/transmission_rate), C0 1 (y/transmission_rate)}

Dans le reste de cette section, nous discutons de deux algorithmes d'optimisation de requêtes distribuéesÿ:

- Algorithme R*ÿ;
- Algorithme SDD1.

Algorithme R*

R* était un SGBD distribué expérimental construit par IBM Research au début des années 1980 qui incorporait de nombreux mécanismes du projet System R antérieur, adapté à un environnement distribué (Williams et al., 1982). Les principaux objectifs de R * étaient la transparence de l'emplacement, l'autonomie du site et des performances minimales. Les principales extensions de System R pour prendre en charge la distribution des données concernent la définition des données, la gestion des transactions, le contrôle des autorisations et la compilation, l'optimisation et l'exécution des requêtes.

Pour l'optimisation des requêtes distribuées, R* utilise un modèle de coût basé sur le coût total et l'optimisation des requêtes statiques (Selinger et Abida, 1980 ; Lohman et al., 1985). Comme l'optimiseur System R centralisé, l'algorithme d'optimisation est basé sur une recherche exhaustive de tous les ordres de jointure, des méthodes de jointure (boucle imbriquée ou jointure tri-fusion) et des chemins d'accès pour chaque relation, comme discuté dans la section 23.5. Lorsqu'une jointure est requise impliquant des relations sur différents sites, R* sélectionne les sites pour effectuer la jointure et la méthode de transfert de données entre les sites.

Pour une jointure (R A S) avec la relation R au site 1 et la relation au site 2, il y a trois sites candidats :

- le site 1, où se situe la relation R ; le site
- 2, où se situe la relationÿ; un autre site
- (par exemple, le site d'une relation T, qui doit être jointe par la jointure de R et).

Dans R*, il existe deux méthodes pour transférer des données entre sitesÿ:

(1) Expédier toute la relation. Dans ce cas, la relation entière est transférée vers le site de jointure, où elle est soit stockée temporairement avant l'exécution de la jointure, soit jointe tuple par tuple à l'arrivée.

(2) Récupérer les tuples selon les besoins. Dans ce cas, le site de la relation externe coordonne le transfert des tuples et les utilise directement sans stockage temporaire. Le site de coordination analyse séquentiellement la relation externe et, pour chaque valeur, demande les tuples correspondants au site de la relation interne (en effet, en effectuant une semi-jointure tupleatatime, bien qu'encourant plus de messages que ce dernier).

La première méthode implique un transfert de données plus important mais moins de messages que la seconde méthode. Bien que chaque méthode de jointure puisse être utilisée avec chaque méthode de transmission, R* considère que seules les suivantes sont valablesÿ:

(1) Boucle imbriquée, expédier toute la relation extérieure au site de la relation intérieure. Dans ce cas, aucun stockage temporaire n'est nécessaire et les tuples peuvent être joints au fur et à mesure qu'ils arrivent sur le site de la relation interne. Le coût est:

Coût total 5 coût (boucle imbriquée)
1 [C0 1 (nTuples(R)*nBitsInTuple(R)/taux_de_transmission)]

(2) Trier-fusionner, expédier toute la relation intérieure au site de la relation extérieure. Dans ce cas, les tuples ne peuvent pas être joints au fur et à mesure qu'ils arrivent et doivent être stockés dans une relation temporaire. Le coût est:

Coût total 5 coûts (stockage sur le site 1) 1 coût (tri–fusion)
1 [C0 1 (nTuples( )*nBitsInTuple( )/taux_de_transmission)]

(3) Boucle imbriquée, récupère les tuples de la relation interne selon les besoins pour chaque tuple de la relation externe. Encore une fois, les tuples peuvent être joints au fur et à mesure qu'ils arrivent. Le coût est:

Coût total 5 coût(boucle imbriquée) 1
nTuplets(R)*[C0 1 (nBitsInAttribute(A)/transmission_rate)] 1 nTuples(R)*[C0 1 (AVG(R,)*nBitsInTuple( )/transmission_rate)]

où AVG(R,) désigne le nombre de tuples qui correspondent (en moyenne) à un tuple de R, ainsiÿ:

MOY(R,) 5 nuplets(2A R)/nuplets(R)

(4) Trier-fusionner, récupérer les tuples de la relation interne selon les besoins pour chaque tuple de la relation externe. Encore une fois, les tuples peuvent être joints au fur et à mesure qu'ils arrivent. Le coût est similaire au coût précédent et est laissé en exercice au lecteur.

(5) Expédier les deux relations vers un site tiers. La relation interne est déplacée vers le troisième site et stockée dans une relation temporaire. La relation externe est ensuite déplacée vers le troisième site et ses tuples sont joints à la relation temporaire à mesure qu'ils arrivent. La boucle imbriquée ou la jointure tri-fusion peut être utilisée dans ce cas. Le coût peut être obtenu à partir des coûts antérieurs et est laissé en exercice au lecteur.

Bien que de nombreuses stratégies soient évaluées par R* en utilisant cette approche, cela peut être intéressant si la requête est fréquemment exécutée. Bien que l'algorithme décrit par Selinger et Abida traite de la fragmentation, la version de l'algorithme implémentée dans R* ne traite que des relations entières.

Algorithme SDD1 SDD1

était un autre SGBD distribué expérimental construit par la division de recherche de Computer Corporation of America à la fin des années 1970 et au début des années 1980 qui fonctionnait sur un réseau de DEC PDP11 connectés via Arpanet (Rothnie et al., 1980). Il offrait une indépendance totale en matière d'emplacement, de fragmentation et de réplication. L'optimiseur SDD1 était basé sur une méthode antérieure connue sous le nom d'algorithme « hill climbing », un algorithme glouton qui part d'une solution réalisable initiale qui est ensuite améliorée de manière itérative (Wong, 1977). Il a été modifié pour utiliser l'opérateur Semijoin afin de réduire la cardinalité des opérandes de jointure. Comme l'algorithme R*, l'objectif de l'optimiseur SDD1 est de minimiser le coût total, bien que contrairement à R*, il ignore les coûts de traitement locaux et se concentre sur la taille des messages de communication. Encore une fois, comme R*, le délai de traitement des requêtes utilisé est statique.

L'algorithme est basé sur le concept de "semi-jointures bénéfiques". Le coût de communication d'une semi-jointure est simplement le coût de transfert de l'attribut join du premier opérande vers le site du second opérande, ainsi :

Coût de communication(R 2A)
5 C0 1 [taille(A( ))/taux_de_transmission]
5 C0 1 [nTuples( )*nBitsInAttribute(A)/transmission_rate] (A est la clé de)

Le «ÿbénéficeÿ» de la semi-jointure est considéré comme le coût du transfert de tuples non pertinents de R, ce que la semi-jointure éviteÿ:

Avantage(R 2A) 5 (1 2 SFA( )) * [nTuples(R)*nBitsInTuple(R)/ taux_de_transmission]

où SFA( ) est le facteur de sélectivité de jointure (la fraction de tuples de R qui se rejoignent avec des tuples de), qui peut être estimé comme suitÿ:

SFA( ) 5 nTuples(A( ))/nDistinct(A)

où nDistinct(A) est le nombre de valeurs distinctes dans le domaine de l'attribut A. L'algorithme procède comme suitÿ:

(1) Phase 1 : Initialisation. Effectuez toutes les réductions locales à l'aide de la sélection et de la projection. Exécutez des semi-jointures au sein du même site pour réduire la taille des relations. Générer l'ensemble de toutes les semi-jointures bénéfiques sur les sites (la semi-jointure est bénéfique si son coût est inférieur à son bénéfice).

(2) Phase 2 : Sélection des semi-jointures bénéfiques. Sélectionnez itérativement la semi-jointure la plus avantageuse dans l'ensemble généré lors de la phase précédente et ajoutez-la à la stratégie d'exécution. Après chaque itération, mettez à jour les statistiques de la base de données pour refléter l'incorporation de la semi-jointure et mettez à jour l'ensemble avec de nouvelles semi-jointures utiles.

(3) Phase 3 : Sélection du site d'assemblage. Sur l'ensemble des sites sélectionner le site vers lequel la transmission de toutes les relations référencées par la requête engendre un coût minimum. Choisissez le site contenant la plus grande quantité de données après la phase de réduction afin que la somme de la quantité de données transférées depuis d'autres sites soit minimale.

(4) Phase 4 : Post-optimisation. Ignorez les semi-jointures inutiles. Par exemple, si la relation R réside dans le site d'assemblage et R doit être réduite par une semi-jointure, mais n'est pas utilisée pour réduire d'autres relations après l'exécution de la semi-jointure, alors parce que R n'a pas besoin d'être déplacé vers un autre site pendant l'assemblage phase, la semi-jointure sur R est inutile et peut être ignorée.

L'exemple suivant illustre la discussion précédente.

EXEMPLE 25.5 Algorithme SDD1

Répertoriez les détails de la succursale ainsi que les propriétés gérées et les détails du personnel qui les gère.

Nous pouvons exprimer cette requête en SQL commeÿ:

```
SÉLECTIONNER *
FROM Succursale b, PropertyForRent p, Personnel
s, WHERE b.branchNo 5 p.branchNo AND p.staffNo 5 s.staffNoÿ;
```

Supposons que la relation Branch se trouve sur le site 1, la relation PropertyForRent sur le site 2 et la relation Staff sur le site 3. En outre, supposons que le coût d'initiation d'un message, C0, est égal à 0 et que le débit de transmission, transmission_rate, est 1. La figure 25.21 fournit l'ensemble initial de statistiques de base de données pour ces relations. L'ensemble initial de semi-jointures estÿ:

relation	site	cardinality	tuple size	relation size
Branch	1	50	200	10,000
PropertyForRent	2	200	600	120,000
Staff	3	100	500	50,000

attribute	size($\Pi_{attribute}$)	$SF_{attribute}$
b.branchNo	1600	1
p.branchNo	640	0.1
s.staffNo	2880	0.9
p.staffNo	1280	0.2

Illustration 25.21 Initial set of database statistics for Branch, PropertyForRent, and Staff.

SJ1 :. PropertyForRent 2branchNo Branch Benefit is (1 2 1)*120,000 5 0; le coût est de 1 600 Le bénéfice Branch 2branchNo PropertyForRent SJ3 : Propert[illegible] le coût est de 2880 SJ4 : Personnel 2personnelPas de bénéfice à [illegible] 40 000, le gain est de (1 2 0,2)*50 000 5

Dans ce cas, les semi-jointures bénéfiques sont SJ2, SJ3 et SJ4 et nous ajoutons donc SJ4 (celui avec la plus grande différence) à la stratégie d'exécution. Nous mettons maintenant à jour les statistiques basées sur cette semi-jointure, de sorte que la cardinalité de Staff devient 100*0,2 5 20, la taille devient 50 000*0,2 5 10 000 et le facteur de sélectivité est estimé à 0,9*0,2 5 0,18. À la prochaine itération, nous obtenons SJ3ÿ: PropertyForRent 2staffNo Staff comme étant bénéfique avec un coût de 3720 et l'ajoutons à la stratégie d'exécution. Encore une fois, nous mettons à jour les statistiques et ainsi la cardinalité de PropertyForRent devient 200*0,9 5 180 et la taille devient 120 000*0,9 5 108 000. Une autre itération trouve Semijoin SJ2: Branch 2branchNo PropertyForRent comme étant bénéfique et nous l'ajoutons à la stratégie d'exécution et mettons à jour les statistiques de Branch, de sorte que la cardinalité devient 40*0,1 5 4 et la taille devient 10 000*0,1 5 1000.

Après réduction, la quantité de données stockées est de 1 000 sur le site 1, 108 000 sur le site 2 et 10 000 sur le site 3. Le site 2 est choisi comme site d'assemblage. A la post-optimisation, on supprime la stratégie SJ3. La stratégie choisie consiste à envoyer Staff 2staffNo PropertyForRent et Branch wbranchNo PropertyForRent au site 3.

D'autres algorithmes d'optimisation de requête distribués bien connus sont AHY (Apers et al., 1983) et Distributed Ingres (Epstein et al., 1978). Le lecteur intéressé est également renvoyé à un certain nombre de publications dans ce domaine, par exemple, Yu et Chang (1984), Steinbrunn et al. (1997) et Kossmann (2000).

25.7 Diffusion dans Oracle

Pour terminer ce chapitre, nous examinons la fonctionnalité de SGBD distribué d'Oracle11g (Oracle Corporation, 2008d). Dans cette section, nous utilisons la terminologie du SGBD — Oracle désigne une relation comme une table avec des colonnes et des lignes. Nous fournissons une introduction à Oracle dans l'annexe H.2.

25.7.1 Fonctionnalité DDBMS d'Oracle

Comme de nombreux DDBMS commerciaux, Oracle ne prend pas en charge le type de mécanisme de fragmentation dont nous avons parlé au chapitre 24, bien que le DBA puisse distribuer manuellement les données pour obtenir un effet similaire. Cependant, cela place la responsabilité sur l'utilisateur final de savoir comment une table a été fragmentée et de construire

ces connaissances dans l'application. En d'autres termes, Oracle DDBMS ne prend pas en charge la transparence de la fragmentation, bien qu'il prenne en charge la transparence de l'emplacement, comme nous le verrons bientôt. Dans cette section, nous fournissons un aperçu de la fonctionnalité DDBMS d'Oracle, couvrantÿ:

- connectivitéÿ;
- noms de bases de données
- globalesÿ; liens de bases de
- donnéesÿ; transactions;
- intégrité référentielleÿ; bases
- de données distribuées hétérogènesÿ; optimisation
- distribuée des requêtes.

Dans le chapitre suivant, nous discutons du mécanisme de réplication d'Oracle.

Connectivité Oracle

Net Services est l'application d'accès aux données fournie par Oracle pour prendre en charge la communication entre les clients et les serveurs (les versions antérieures d'Oracle utilisaient SQL*Net ou Net8). Oracle Net Services permet à la fois les communications client-serveur et serveur-serveur sur n'importe quel réseau, prenant en charge à la fois le traitement distribué et la capacité de SGBD distribué. Même si un processus s'exécute sur la même machine que l'instance de base de données, Net Services doit toujours établir sa connexion à la base de données. Net Services est également responsable de la traduction de toutes les différences dans les jeux de caractères ou les représentations de données qui peuvent exister au niveau du système d'exploitation. Net Services établit une connexion en transmettant la demande de connexion au substrat de réseau transparent (TNS), qui détermine quel serveur doit traiter la demande et envoie la demande à l'aide du protocole réseau approprié (par exemple, TCP/IP). Net Services peut également gérer la communication entre des machines exécutant différents protocoles réseau via le gestionnaire de connexion, qui était auparavant géré par MultiProtocol Interchange dans Oracle 7.

Dans les versions antérieures d'Oracle, le produit Oracle Names stocke les informations sur les bases de données dans un environnement distribué dans un emplacement unique. Lorsqu'une application émet une demande de connexion, le référentiel Oracle Names est consulté pour déterminer l'emplacement du serveur de base de données. Une alternative à l'utilisation d'Oracle Names consiste à stocker ces informations dans un fichier tnsnames.ora local sur chaque ordinateur client. Dans Oracle 11g, lorsqu'un réseau Oracle utilise un serveur d'annuaire compatible LDAP, le serveur d'annuaire crée et gère automatiquement les liens de base de données globaux (en tant que noms de services réseau) pour chaque base de données Oracle du réseau. Les utilisateurs et les sous-programmes PL/SQL de n'importe quelle base de données peuvent utiliser un lien global pour accéder aux objets de la base de données distante correspondante. Nous discutons des liens de base de données sous peu.

Noms de bases de données globales

Chaque base de données distribuée reçoit un nom, appelé nom de base de données global, qui est distinct de toutes les bases de données du système. Oracle forme un nom de base de données global en préfixant le nom de domaine réseau de la base de données avec le nom de la base de données locale. Le nom de domaine doit suivre les conventions Internet standard, où les niveaux doivent être séparés par des points ordonnés de la feuille à la racine, de gauche à droite. Par exemple, Figure 25.22

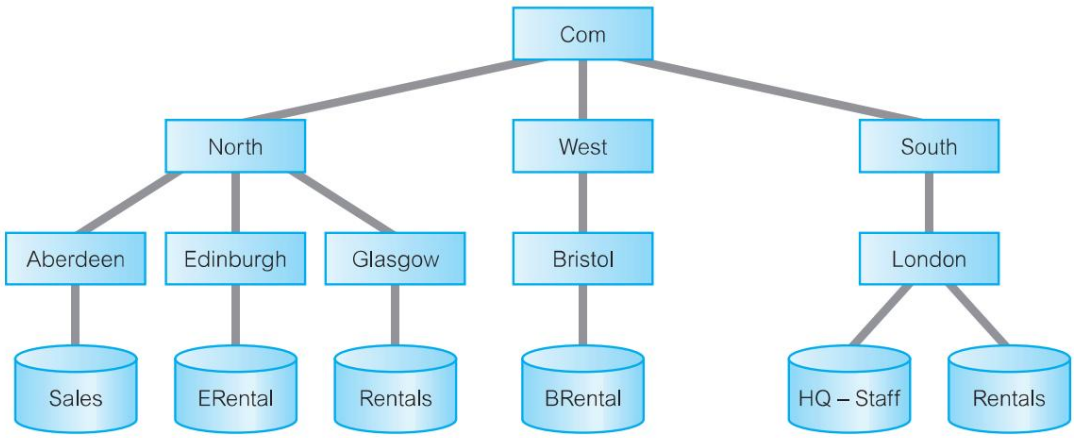


Figure 25.22 Maison de Rêve network structure.

illustre une disposition hiérarchique possible des bases de données pour DreamHome.
Bien qu'il y ait deux bases de données locales appelées Rentals dans cette figure, nous pouvons utiliser le nom de domaine réseau LONDON.SOUTH.COM pour différencier la base de données de Londres de celle de Glasgow.

Dans ce cas, les noms de bases de données globales sontÿ:

RENTALS.LONDON.SOUTH.COM
LOCATIONS.GLASGOW.NORTH.COM

Liens de base de données

Les bases de données distribuées dans Oracle sont construites sur des liens de base de données, qui définissent un chemin de communication d'une base de données Oracle à une autre base de données (éventuellement non Oracle). Le but des liens de base de données est de rendre les données distantes disponibles pour les requêtes et les mises à jour, agissant essentiellement comme un type de connexion stockée à la base de données distante. Un lien de base de données doit porter le même nom que le nom de base de données global de la base de données distante à laquelle il fait référence, auquel cas les liens de base de données sont par essence transparents pour les utilisateurs d'une base de données distribuée. Par exemple, l'instruction suivante crée un lien de base de données dans la base de données locale vers la base de données distante à Glasgowÿ:

CRÉER UN LIEN DE BASE DE DONNÉES PUBLIQUE RENTALS.GLASGOW.NORTH.COMÿ;

Une fois qu'un lien de base de données a été créé, il peut être utilisé pour faire référence à des tables et des vues sur la base de données distante en ajoutant @databaselink au nom de table ou de vue utilisé dans une instruction SQL. Une table ou une vue distante peut être interrogée avec l'instruction SELECT. Avec l'option distribuée Oracle, les tables et vues distantes sont également accessibles à l'aide des instructions INSERT, UPDATE et DELETE. Par exemple, nous pouvons utiliser les instructions SQL suivantes pour interroger et mettre à jour la table Staff au site distant :

SÉLECTIONNEZ * À PARTIR DE Staff@RENTALS.GLASGOW.NORTH.COMÿ; MISE
À JOUR Staff@RENTALS.GLASGOW.NORTH.COM SET salaire 5 salaire*1,05ÿ;

Un utilisateur peut également accéder aux tables appartenant à d'autres utilisateurs dans la même base de données en faisant précéder le nom de la base de données du nom du schéma. Par exemple, si nous supposons que le

l'utilisateur actuel a accès à la table de visualisation dans le schéma du superviseur, nous pouvons utiliser l' instruction SQL suivanteÿ:

```
SÉLECTIONNER * À PARTIR DE Supervisor.Viewing@RENTALS.GLASGOW.NORTH.COMÿ;
```

Cette instruction se connecte en tant qu'utilisateur actuel à la base de données distante, puis interroge la table de visualisation dans le schéma du superviseur. Un synonyme peut être créé pour masquer le fait que la table de visualisation du superviseur se trouve sur une base de données distante. L'instruction suivante force toutes les futures références à Viewing à accéder à une table Viewing distante appartenant à Supervisorÿ:

```
CRÉER UN SYNONYME Affichage POUR
    Supervisor.Viewing@RENTALS.GLASGOW.NORTH.COMÿ;
SELECT * FROM Affichageÿ;
```

De cette manière, l'utilisation de synonymes assure à la fois l'indépendance des données et la transparence de l'emplacement.

Transactions

Oracle prend en charge les transactions sur les données distantes, notammentÿ:

- Instructions SQL distantes. Une requête distante est une requête qui sélectionne des informations dans une ou plusieurs tables distantes, qui résident toutes sur le même nœud distant. Une instruction de mise à jour à distance est une mise à jour qui modifie les données dans une ou plusieurs tables, toutes situées sur le même nœud distant.

- Instructions SQL distribuées. Une requête distribuée récupère des informations à partir de deux nœuds ou plus. Une instruction de mise à jour distribuée modifie les données sur deux nœuds ou plus. Une mise à jour distribuée est possible à l'aide d'une unité de sous-programme PL/SQL telle qu'une procédure ou un déclencheur qui inclut deux ou plusieurs mises à jour à distance qui accèdent aux données sur différents nœuds. Oracle envoie des instructions dans le programme aux nœuds distants et leur exécution réussit ou échoue en tant qu'unité.

- Opérations à distance. Une transaction distante contient une ou plusieurs instructions distantes, qui font toutes référence à un seul nœud distant.
- Opérations distribuées. Une transaction distribuée est une transaction qui comprend une ou plusieurs instructions qui, individuellement ou en groupe, mettent à jour des données sur deux nœuds distincts ou plus d'une base de données distribuée. Dans de tels cas, Oracle garantit l'intégrité des transactions distribuées à l'aide du protocole 2PC décrit à la section 25.4.3.

Intégrité référentielle

Oracle n'autorise pas la définition de contraintes d'intégrité référentielle déclarative dans les bases de données d'un système distribué (c'est-à-dire qu'une contrainte d'intégrité référentielle déclarative sur une table ne peut pas spécifier une clé étrangère faisant référence à une clé primaire ou unique d'une table distante). Cependant, les relations entre les tables parent-enfant entre les bases de données peuvent être maintenues à l'aide de déclencheurs.

Bases de données distribuées hétérogènes Dans

un DDBMS hétérogène Oracle, au moins un des SGBD est un système non Oracle. À l'aide de services hétérogènes et d'un agent de services hétérogènes non spécifique au système Oracle, Oracle peut masquer la distribution et l'hétérogénéité à l'utilisateur. L'agent des services hétérogènes communique avec le

système non Oracle et avec le composant Services hétérogènes dans le serveur Oracle. Au nom du serveur Oracle, l'agent exécute les requêtes SQL, de procédure et transactionnelles sur le système non Oracle.

Les Services Hétérogènes sont accessibles via des outils tels que :

- Les passerelles de base de données Oracle, qui fournissent un accès SQL aux SGBD non Oracle, notamment DB2/400, DB2 pour OS/390, Informix, Sybase, SQL Server, Teradata, IMS, Adabas et VSAM. Ces passerelles s'exécutent généralement sur la machine avec le SGBD non Oracle, par opposition à l'endroit où réside le serveur Oracle. Toutefois, la passerelle transparente pour DRDA (voir Section 24.5.2), qui fournit un accès SQL aux bases de données compatibles DRDA telles que DB2, SQL/DS et SQL/400, ne nécessite aucun logiciel Oracle sur le système cible. La Figure 25.23(a) illustre l'architecture d'Oracle Database Gateway.

- Oracle Database Gateway pour ODBC, une passerelle liée aux pilotes fournis par le client à l'aide d'ODBC. La fonctionnalité de cette passerelle est plus limitée que celle des passerelles de base de données Oracle. La Figure 25.23(b) illustre l'architecture Oracle Database Gateway pour ODBC.

Les caractéristiques des Services hétérogènes comprennentÿ:

- Opérations distribuées. Une transaction peut s'étendre à la fois sur des systèmes Oracle et non Oracle en utilisant une validation en deux phases (voir Section 25.4.3).
- Accès SQL transparent. Les instructions SQL émises par l'application sont transformées de manière transparente en instructions SQL reconnues par le système non Oracle.
- Accès procédural. Les systèmes procéduraux, tels que les systèmes de messagerie et de file d'attente, sont accessibles à partir d'un serveur Oracle11g à l'aide d'appels de procédure à distance PL/SQL.
- Traductions du dictionnaire de données. Pour faire apparaître le système non Oracle comme un autre serveur Oracle, les instructions SQL contenant des références aux tables du dictionnaire de données d'Oracle sont transformées en instructions SQL contenant des références aux tables du dictionnaire de données d'un système non Oracle.
- Passthrough SQL et procédures stockées. Une application peut accéder directement à un système non Oracle en utilisant le dialecte SQL de ce système. Les procédures stockées dans un système non Oracle basé sur SQL sont traitées comme s'il s'agissait de procédures distantes PL/SQL.
- Prise en charge de la langue nationale. Les services hétérogènes prennent en charge les jeux de caractères multi-octets et traduisent les jeux de caractères entre un système non Oracle et Oracle.
- Réplication. Les données peuvent être répliquées entre un système non Oracle et un serveur Oracle à l'aide de vues matérialisées (voir Section 26.8.1).
- Optimisation. Les services hétérogènes peuvent collecter certaines statistiques de table et d'index sur le système non Oracle et les transmettre à l'optimiseur basé sur les coûts Oracle.

Optimisation des requêtes distribuées

Une requête distribuée est décomposée par le SGBD Oracle local en un nombre correspondant de requêtes distantes, qui sont envoyées aux SGBD distants pour exécution. Les SGBD distants exécutent les requêtes et renvoient les résultats au nœud local. Le nœud local effectue ensuite tout post-traitement nécessaire et renvoie les résultats à l'utilisateur ou à l'application. Seules les données nécessaires des tables distantes sont extraites, réduisant ainsi la quantité de données devant être transférées. L'optimisation des requêtes distribuées utilise l'optimiseur basé sur les coûts d'Oracle, dont nous avons parlé à la section 23.6.

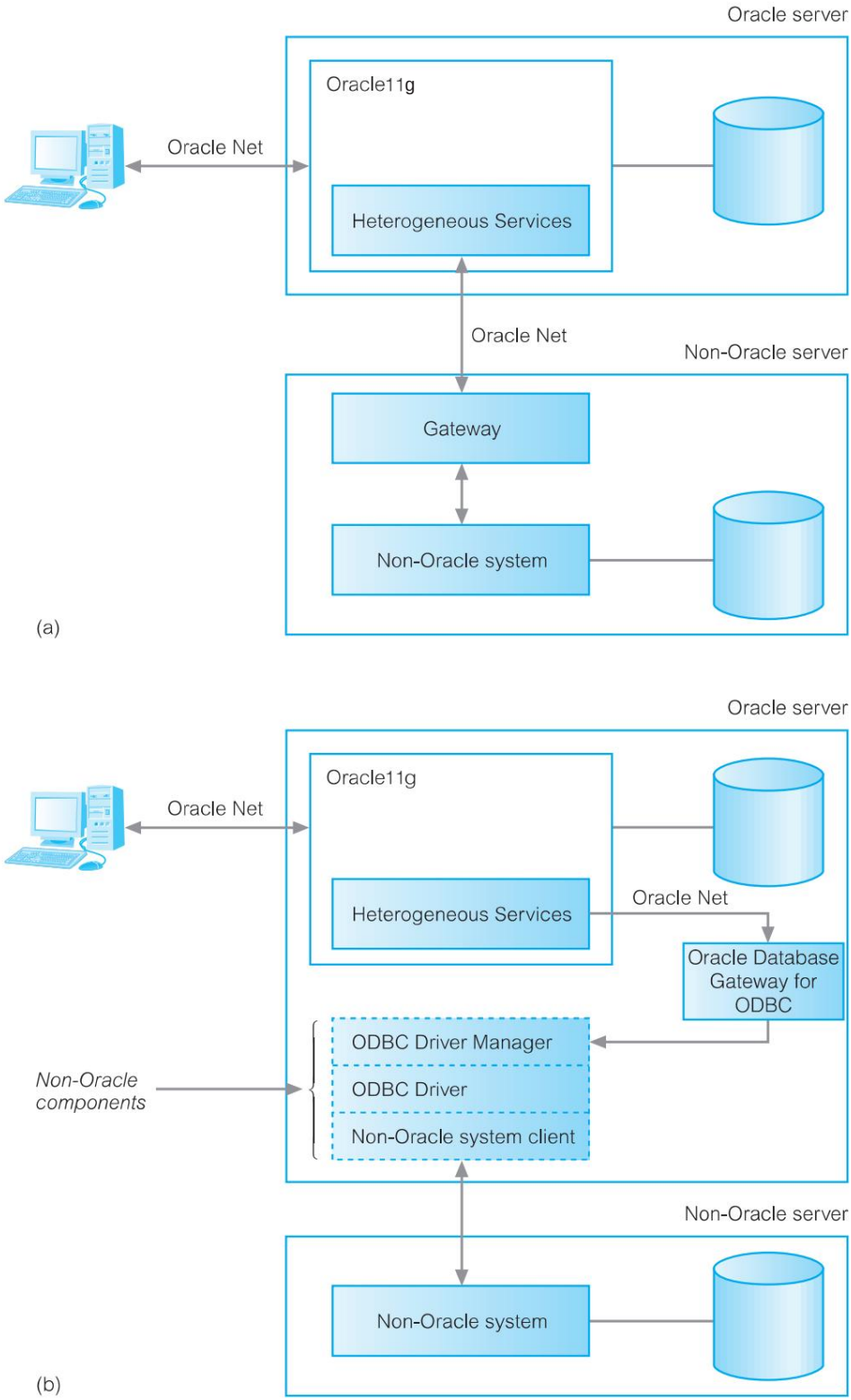


Illustration 25.23 Oracle Heterogeneous Services: (a) using a Oracle Database Gateway on the non-Oracle system; (b) using Oracle Database Gateway for ODBC.

Résumé du chapitre

- The objectives of distributed transaction processing are the same as those of centralized systems, although more complex because the DDBMS must ensure the atomicity of the global transaction and each subtransaction.
- If the schedule of transaction execution at each site is serializable, then the calendrier global (the union of all local schedules) is also serializable, provided that local serialization orders are identical. This requires that all subtransactions appear in the same order in the equivalent serial schedule at all sites.
- Two methods that can be used to guarantee distributed serializability are verrouillage de l'horodatage deux In verrouillage de phase (2PL) a transaction acquires all its locks before releasing any. Two-phase locking protocols can use centralized, primary copy, or distributed lock managers. Majority voting can also be used. With horodatage, transactions are ordered in such a way that older transactions get priority in the event of conflict.
- Blocage distribué involves merging local wait-for graphs together to check for cycles. If a cycle is detected, one or more transactions must be aborted and restarted until the cycle is broken. There are three common methods for handling deadlock detection in distributed DBMSs: centralisé, hiérarchique , and distribué deadlock detection.
- Causes of failure in a distributed environment are loss of messages, communication link failures, site crashes, and network partitioning. To facilitate recovery, each site maintains its own log file. The log can be used to undo and redo transactions in the event of failure.
- The validation en deux phases (2PC) protocol comprises a voting and decision phase, in which the coordinator asks all participants whether they are ready to commit. If one participant votes to abort, the global transaction and each subtransaction must be aborted. Only if all participants vote to commit can the global transaction be committed. The 2PC protocol can leave sites blocked in the presence of sites failures.
- A non-blocking protocol is validation en trois phases (3PC), which involves the coordinator sending an additional message between the voting and decision phases to all participants asking them to pre-commit the transaction.
- X/Ouvrir DTP is a distributed transaction processing architecture for a distributed 2PC protocol, based on OSI-TP. The architecture defines application programming interfaces and interactions among transactional applications, transaction managers, resource managers, and communication managers.
- Distributed query processing can be divided into four phases: query decomposition, data localization, global optimization, and local optimization. Décomposition des requêtes takes a query expressed on the global relations and performs a partial optimization using the techniques discussed in Chapter 23. Localisation des données takes into account how the data has been distributed and replaces the global relations at the leaves of the relational algebra tree with their algorithmes de reconstruction. Optimisation globale takes account of statistical information to find a near-optimal execution plan. Optimisation locale is performed at each site involved in the query.
- The cost model for distributed query optimization can be based on coût total (as in the centralized case) or réponse temps, that is, the elapsed time from the start to the completion of the query. The latter model takes account of the inherent parallelism in a distributed system. Cost needs to take account of local processing costs (I/O and CPU) as well as networking costs. In a WAN, the networking costs will be the dominant factor to reduce.
- When the main cost component is communication time, the Semijoin operation is particularly useful for improving the processing of distributed joins by reducing the amount of data transferred between sites.

Questions de révision

25.1 In a distributed environment, locking-based algorithms can be classified as centralized, primary copy, or distributed. Compare and contrast these algorithms.

25.2 There can be many types of failures in a distributed environment. Discuss how failure due to network partition is handled by the distributed DBMS.

25.3 Outline two alternative two-phase commit topologies to the centralized topology.

25.4 Describe the term "non-blocking" and explain how it is related to two-phase and three-phase commit protocols.

25,5 Describe the protocol used to recover two-phase and three-phase commit in a distributed environment.

25.6 Specify the layers of distributed query optimization and detail the function of each layer.

25,7 Discuss the costs that need to be considered in distributed query optimization and discuss two different cost models.

25,8 Describe the distributed query optimization algorithms used by R* and SDD-1.

25,9 Briefly describe the distributed functionality of Oracle 11g.

Des exercices

25.10 You are the systems analyst for Maison de rêve. One of your responsibilities is to ensure that every business transaction performs to the desired standards at all the sites. You have currently received complaints from site managers that customers are experiencing difficulties in accomplishing their transactions. Enough details about the nature of the failure of their transactions have not been provided to you. You are therefore required to prepare an investigation plan that shall be approved by your manager. Indicate in your plan the possible cause of problems and how you are going to approach the problem. Your plan should also guarantee that such problems will not occur again.

25.11 Give full details of the centralized two-phase commit protocol in a distributed environment. Outline the algorithms for both coordinator and participants.

25.12 Give full details of the three-phase commit protocol in a distributed environment. Outline the algorithms for both coordinator and participants.

25.13 Analyze the database application deployed at an organization of your choice and discover whether it is distributed or centralized. Advice accordingly.

25.14 Consider five transactions T_1, T_2, T_3, T_4 and T_5 with:

- T_1 initiated at site S_1 and spawning an agent at site S_2
- J_2 initiated at site S_3 and spawning an agent at site S_1
- J_3 initiated at site S_1 and spawning an agent at site S_3
- T_4 initiated at site S_2 and spawning an agent at site S_3
- J_5 initiated at site S_3.

TRANSACTION	ÉLÉMENTS DE DONNÉES VERROUILLÉS PAR TRANSACTION	LA TRANSACTION DES ÉLÉMENTS DE DONNÉES EST ATTENDRE	SITE IMPLIQUÉ EN OPÉRATIONS
T_1	x_1	x_8	S_1
T_1	x_6	x_2	S_2
J_2	x_4	x_1	S_1
J_2	x_5		S_3
J_3	x_2	x_7	S_1
J_3		x_3	S_3
T_4	x_7		S_2
T_4	x_8	x_5	S_3
J_5	x_3	x_7	S_3

The locking information for these transactions is shown in the following table.

(un) Produce the local wait-for graphs (WFGs) for each of the sites. What can you conclude from the local WFGs?

(b) Using the example transactions, demonstrate how Obermarck's method for distributed deadlock detection works. What can you conclude from the global WFG?