Introduction to Distributed Databases

▲Distributed System Concepts

(1) A distributed database system consists of loosely coupled sites that share no physical component;
[分布式数据库系统由松散耦合的站点组成，这些站点不共享物理组件]
(2) Database systems that run on each site are independent of each other;
[在每个站点上运行的数据库系统彼此独立]
(3) Transactions may access data at one or more sites.
[事务可以访问一个或多个站点的数据]
(4) The location of data on each individual sites impacts query optimization, concurrency control and recovery. [数据在每个站点上的位置影响查询优化、并发控制和恢复]
(5) Distributed data is governed by factors such as local ownership, increased availability, and performance issues. [分布式数据受本地所有权、增加的可用性和性能问题等因素的控制]
(6) Distributed Data Independence: Users should not have to know where data is located.
[分布式数据独立性:用户不需要知道数据的位置]
(7) Distributed Transaction Atomicity: Users should be able to write transactions that access and update data at several sites. [分布式事务原子性:用户应该能够编写访问和更新多个站点数据的事务]
(8) Transactions are atomic, all changes persist if the transaction commits, or rollback if transaction aborts.
[事务是原子级别的(最低一级)，如果事务提交所有更改都将保持，如果事务终止则回滚]
(9) If sites are connected by slow networks, these properties are hard to support efficiently.
[如果站点通过慢速网络连接，则很难有效地支持这些属性]
(10) Users have to be aware of where data is located, i.e. Distributed Data Independence and Distributed Transaction Atomicity are not supported.
[用户必须知道数据的位置，即不支持分布式数据独立性和分布式事务原子性]
(11) For globally distributed sites, these properties may not even be desirable due to administrative overheads of making locations of data transparent.
[对于全域分布的站点，由于会产生使数据位置透明的管理开销，这些属性甚至可能不需要]

▲Types of Distributed Databases

Homogeneous [同质] – data is distributed but all servers run the same DBMS software.
Heterogeneous [异质] – different sites run different DBMSs separately and are connected to enable access to data from multiple sites.
Gateway protocols [外部网关协议] - API that exposes DBMS functionality to external applications.
Examples: ODBC and JDBC

▲Architectures

Client Server [客户服务器] – a system that has one or more client processes and one or more server processes. Client sends a query to a server, and the server processes the query returning the result to the client.
Collaborating Server [协作服务器] – capable of running queries against local data and executes transactions across multiple servers.
Middleware [中间件] – one database server can manage queries and transactions spanning across multiple servers. A layer that executes relational operations on data from other servers but does not maintain any data.
分布式数据库系统体系结构_Sais_Z 的博客-CSDN 博客_分布式数据库结构图
Client Server → Middleware → Collaborating Server

🔺Distributed Data Storage

(1) Relations are stored across several sites. To reduce message-passing costs a relation maybe fragmented across sites. [为了减少消息传递成本，可能会在站点之间分散关系]

(2) Fragmentation: breaks a relation to smaller relations and stores the fragments at different sites.

Horizontal fragments (HF) [水平片段] - rows of the original data.

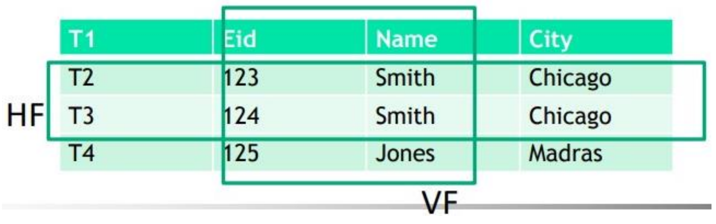
Selection queries, fragments by city

Disjoint union of the HF must be equal to the original relation.

Vertical fragments (VF) [垂直片段] - columns of the original data.

Projection queries, e.g. fragments of the first two columns

Collection of VF must be a loss-less join decomposition.

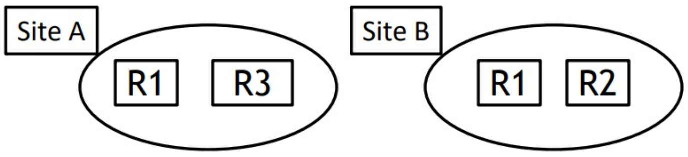


(3) Replication – storing several copies of a relation or fragment. Entire relation can be stored at one or more sites.

Increased Availability – If a site contains replicated data goes down, then we can use another site.

Faster Query Evaluation – Queries are executed faster by using local copy of a relation instead of going to a remote site.

(4) Two kinds of replication are Synchronous and Asynchronous replication below:



🔺Distributed Catalog Management [分布式目录管理]

Must keep track of how data is distributed across sites. [跟踪数据的分布]

Must be able to give a unique identifier to each replica of each fragment/relation. [提供唯一的标识符]

Global relation name- <local-name>, <birth-site>

Global replica name – replica id plus global relation name

Site catalog - Describes all objects (fragments, replicas) at a site and keeps track of replicas of relations created at this site. [点目录储存所有项目的相连数据的副本]

To find a relation look up its birth-site catalog [通过出生点查看相关性]

Birth-site never changes even if the relation is moved [出生点不变]

🔺Distributed Transactions

🔺Updating Distributed Data

Users should be able to update data without worrying where relations are stored.

Synchronous replication [同步复制] – all copies of a modified relation are updated before the modifying transaction commits.

[在提交修改的事务之前，更新修改关系的所有副本]

Asynchronous replication [异步复制] – copies of modified relation are updated over a period of time, and a transaction that reads different copies of the same relation may see different values.

[修改关系的副本在一段时间内更新，读取同一关系的不同副本的事务可能会看到不同的值，在如今的 DBMSs 中常用]

▲Synchronous Replication

Voting technique [投票技术] – a transaction must write a majority of copies to modify an object; read at least enough copies to make sure one of the copies is current.

[事务必须写大多数副本才能修改对象；阅读至少足够的副本，以确保其中一份副本是最新的]

(1) Each copy has a version number, the highest is the most current.

(2) Not attractive and efficient, because reading an object requires reading several copies.

(3) Objects are read more than updated.

e.g. 10 copies, 7 are updatable, 4 are read

Read-any-write-all technique – a transaction can read only one copy, but must write to all copies.

[事务只能读取一个副本，但必须写入所有副本]

(1) Reads are faster than writes especially if it's a local copy

(2) Attractive when reads occur more than writes

(3) Most common technique

Read-any-write-all cost - Before an update transaction can commit, it must lock all copies

(1) Transaction sends lock requests to remote sites and waits for the locks to be granted

(During a long period, it continues to hold all locks)

(2) If there is a site or communication failure then transaction cannot commit until all sites are recovered

(3) Committing creates several additional messages to be sent as part of a commit protocol

同步复制消耗昂贵，异步复制更受欢迎……

▲Asynchronous Replication

Allows modifying transactions to commit before all copies have been changed.

[允许在更改所有副本之前提交修改事务]

Users must be aware of which copy they are reading, and that copy may be out-of-sync for short period of time.

Two approaches: Primary Site and Peer-to-Peer replication.

(Difference lies in how many copies are "updatable" or "master copies")

Peer to Peer Asynchronous Replication - More than one copy can be designated as updateable (master copy). Changes to a master copy must be propagated to other copies somehow. Conflict resolution is used to deal with changes at different sites.

[可以指定多个副本为可更新副本，对主副本的更改必须以某种方式传播到其他副本，冲突解决用于处理不同站点的变化]

Each master is allowed to update only one fragment of the relation, and any two fragments updatable by different masters are disjoint. Updating rights are held by one master at a time.

[每个主节点只允许更新关系的一个片段，不同的主节点更新的任何两个片段是不相连的，更新权限一次由一个 master 持有]

Primary Site Asynchronous Replication

Primary site – one copy of a relation is the master copy.

[关系中的副本是原版拷贝]

Secondary site- replicas of the entire relation are created at other sites. They cannot be updated.

[关系中的副本是其它点的拷贝，不能被更新]

Method: Users register / publish a relation at the primary site and subscribe to a fragment of the relation at the secondary site.

Changes to the primary copy transmitted to the secondary copies are done in two steps:

[1] capture changes made by committed transactions [捕获已提交事务所做的更改]

[2] apply these changes [应用这些更改]

▲Capture

★Log Based Capture [基于日志的捕获] - the log maintained for recovery is used to generate a Change Data Table (CDT)

Procedural Capture [程序捕获] – A procedure that is invoked [调用] by the DBMS which takes a snapshot of the primary copy

Log based capture is generally better because it deals with changes to the data and not the entire database. However, it relies on log details which may be system specific.

▲Apply

The Apply process at the secondary site periodically obtains a snapshot of the primary copy or changes to the CDT table from the primary site, and updates the copy. (This is an action from capture)

Period can be timer or user's application program based.

Replica can be a view over the modified relation.

Log-Based Capture plus continuous Apply minimizes delay in propagating changes.

[基于日志的捕获加上连续的应用最小化了传播更改的延迟]

Procedural Capture plus application-driven Apply is the most flexible way to process updates.

[程序捕获加上应用程序驱动应用是处理更新最灵活的方式]

Used in data warehousing [仓储] applications

▲Distributed Query Processing

▲Distributed Queries

Example query with a relation S (fragmented at Shanghai and Tokyo sites):

```
SELECT AVG(S.age)
FROM Sailors S
WHERE S.rating > 3 AND S.rating < 7
```

关于查询相关的各种情况：

（1）Horizontally Fragmented [横向分散]:

Tuples with rating < 5 at Shanghai, >= 5 at Tokyo. When calculating average, must compute sum and count at both sites. If WHERE contained just S.rating > 6, just one site.

（2）Vertically Fragmented [垂直分散]:

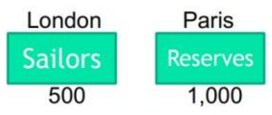
sid and rating at Shanghai, sname and age at Tokyo, tid at both. Joining two fragments by a common tid and execute the query over this reconstructed relation

（3）Replicated [重复的]

Since relation is copied to more than one site, choose a site based on local cost.

▲Distributed Joins

```
SELECT *
FROM Sailors S, Reserves R
WHERE S.sid = R.sid
```



Fetch as Needed [根据需要获取], i.e. ship blocks of Reserves to London and perform join with a join algorithm.

Ship to One Site [发送到一个站点] , i.e. Ship entire Reserves to London and perform join with a join algorithm.

Semi Joins and Bloom Joins: assume that some tuples in Reserves do not join with any tuples in Sailors.

Need to identify Reserve tuples that guarantee not to join with any Sailors tuples.

Tradeoff the cost of computing and shipping projection for cost of shipping full Reserves relation.

<u>Bloom Join is similar to Semi Join but there is a bit-vector shipped in the first step instead of a projection</u>

Online source:
Semi join and Bloom join are two joining methods used in query processing for distributed databases. When processing queries in distributed databases, data needs to be transferred between databases located in different sites. This could be an expensive operation depending on the amount of data that needs to be transferred. Therefore, when processing queries in a distributed database environment, it is important to optimize the queries to minimize the amount of data transferred between sites.
Concepts: Semi join and bloom join are two methods that can be used to reduce the amount of data transfer and perform efficient query processing.
Difference: Even though both semi join and bloom join methods are used to minimize the amount of data transferred between the sites when executing queries in a distributed database environment, bloom join reduces the amount of data (number of tuples) transferred compared to semi join by utilizing the concept of bloom filters, which employ a bit vector to determine set memberships. Therefore using bloom join will be more efficient than using semi join.
Reference: Difference Between Semi Join and Bloom Join | Compare the Difference Between Similar Terms

Optimization of distribution query:

Consider all plans, pick cheapest; similar to centralized optimization.

Query site constructs a global plan, with suggested local plans describing processing at each site. If a site can improve suggested local plan, free to do so.

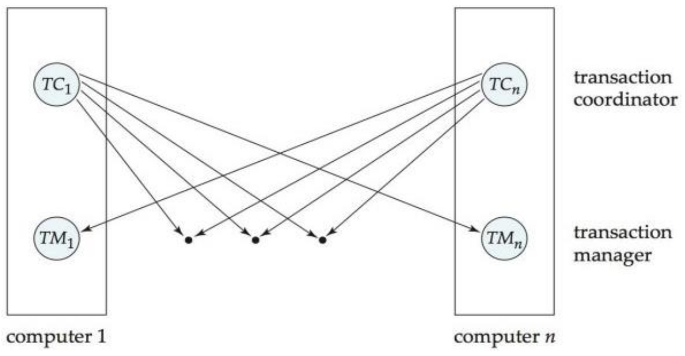
Concurrency Control in Distributed Databases

Distributed Transactions [分布式事务]

(1) Transaction is submitted at one site but can access data at other sites.
(2) Each site has its own local transaction manager and transaction coordinator.
(3) Function of the transaction manager is to ensure the *ACID* properties of those transactions that execute at that site [事务管理器的功能是确保在该站点上执行的事务的 ACID 属性]

1. Maintaining a log for recovery purposes.
2. Participating in an appropriate concurrency-control scheme to coordinate the concurrent execution of the transactions executing at that site.
   [参与适当的并发控制方案，以协调在该站点执行的事务的并发执行]
3. Failure recovery.

Transaction coordinator [事务协调器] is responsible for:
(1) Starting the execution of the transaction;
(2) Breaking the transaction into a number of sub-transactions;
(3) Distributing these sub-transactions to the appropriate sites for execution;
(4) Coordinating the termination of the transaction, which may result in the transaction being committed at all sites or aborted at all sites.

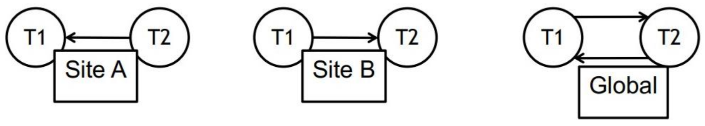


Lock management [锁管理器] can be distributed across many sites:
(1) Single-lock manager (Centralized) – One site does all the locking; vulnerable if one site goes down.
(2) Primary Copy – Only one copy of each object is designated a primary copy, requests to lock/unlock are handled by lock manager at the primary site regardless where the copy is stored.
(3) Distributed lock manager – Requests to lock/unlock a copy of an object stored at a site are handled by the lock manager at the site where the copy is stored.

▲Distributed Deadlock

Each site maintains local waits-for graph, and a cycle in a local graph indicates a deadlock.
A global deadlock might exist even if the local graphs contain no cycles.
Three algorithms of distributed deadlock detection

Centralized [集中式] – send all local graphs to one site that is responsible for deadlock detection.
Hierarchical [层级式] – organize sites into a hierarchy and send local graphs to parent in the hierarchy.
Timeout [超时式] – abort transaction if it waits too long.



▲Failure Recovery in Distributed Databases

Recovery in distributed DBMSs is more complicated than in centralized DBMSs
e.g. failure of communication links; failure of a remote site at which a sub transaction is executing.
All of the sub transactions must commit or not commit at all. This must be guaranteed despite link failures.
Need a commit protocol – the most common one is Two-Phase Commit protocol.
A log is maintained at each site, as in a centralized DBMS, and commit protocol actions are additionally logged.

Two Phase Commit (2PC)

The site at which the transaction originated is the coordinator. Other sites at which sub transactions are executed are subordinates. [发起事务的站点是协调器，执行子事务的其他站点是从属站点]
When a user decides to commit a transaction, the commit command is sent to the coordinator for the transaction. This initiates the 2PC:
Phase 1: Coordinator sends a prepare message to each subordinate; When subordinate receives a prepare message, it decides to abort or commit its sub transaction; Subordinate forces writes an abort or ready log record and sends a no or yes message to coordinator accordingly.
Phase 2: If coordinator receives a yes message from all subordinates, force writes a commit log record and sends commit message to all subordinates. Else force-writes an abort log record and sends an abort message; Subordinates force-write abort or commit log record based on the message they receive.
(In some implementations, after the two phases, subordinates send acknowledgement message [技术认可文电] to coordinator; after coordinator receives ack messages from all subordinates it writes an end log for the transaction)

▲Failure of a participating site [参与站点的故障]

(1) Examine its own logs, if there is commit or abort log record for transaction T, then redo/undo T respectively.
(2) The log contains a <ready T> record, repeatedly contact the coordinator or other active sites to find the status of T, then performs redo/undo accordingly and write commit/abort log records depending on

coordinator's response;

(3) The log contains no control records (abort, commit, ready) concerning T, undo.

▲Failure of the coordinator [协调器的失败]

(1) Participating sites must decide the fate of T.

(2) If an active site contains a <commit T>/<abort T> record in its log, then T must be committed/aborted.

(3) If some active site does not contain a <ready T> , preferable to abort T.

(4) If active sites have a <ready T> record in their logs, but no additional control records (such as <abort T> or <commit T>), it is impossible to determine if a decision has been made and what that decision is, until the coordinator recovers. (in-doubt transaction)

If locking is used, an in-doubt transaction T may hold locks on data at active sites. Such a situation is undesirable [不可取], because it may be hours or days before coordinator is again active. This situation is called the **blocking problem**, because T is blocked pending the recovery of coordinator site.

Solution is in the next page:

(1) Use <ready T, L> log record, where L is a list of all write locks held by the transaction T when the log record is written. At recovery time, after performing local recovery actions, for every in-doubt transaction T, all the write locks noted in the <ready T, L> log record (read from the log) are reacquired. (双重日志)

(2) Can also be solved using the 3-phase commit protocol (under certain situations) (嵌套回溯)

▲Recovery from Network Partitions [从网络分区恢复]

The coordinator and all its participants remain in one partition.

In this case, the failure has no effect on the commit protocol.

The coordinator and its participants belong to several partitions.

From the viewpoint of the sites in one of the partitions, it appears that the sites in other partitions have failed.

Sites that are not in the partition containing the coordinator simply execute the protocol to deal with failure of the coordinator. [不在包含协调器的分区中的站点只需执行协议来处理协调器的故障]

The coordinator and the sites that are in the same partition: the coordinator follow the usual commit protocol, assuming that the sites in the other partitions have failed.

[协调器和位于同一分区中的站点：协调器遵循通常的提交协议，假设其他分区中的站点已经失败]