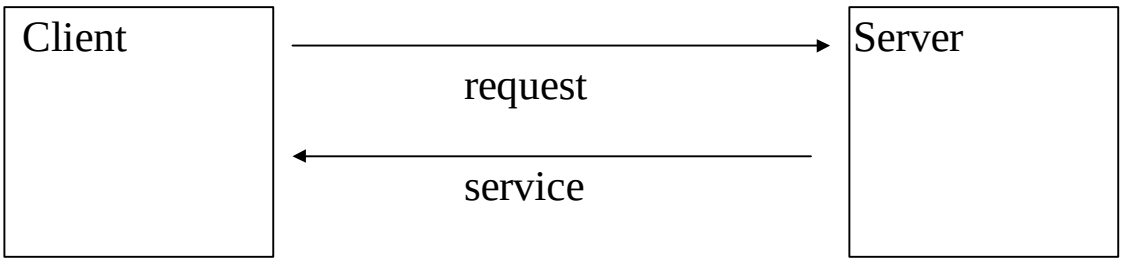
Client Server Architecture

Key concepts:

- client server architecture
- Functional requirements in the 2-tier structures
- Functional distribution in the 2-tier structures
- Implementation of Business Logic at the Server
- Requirements of an Open OLTP System
- Benefits and Limitations of 2-tier Client/Server Architecture
- The 3-tier Structures
- Middleware
- Component Software Model
- Database Middleware
- Transaction Processing and Integration Middleware
- Transaction Processing Monitors
- Performance Issues in TM
- Two-Phase Commit Protocol in TM
- Message Sensitive Routing
- Lifekeeper Clusters
- Repositories in Client-Server Environment
- 4GL Application Development Environment
- Communication Models
- The main standards for open transaction processing

Client - Server Architecture [Salem 1992]

The data processing is split into distinct parts. A part is either requester (client) or provider (server). The client sends during the data processing one or more requests to the servers to perform specified tasks. The server part provide services for the clients.



This basic structure is called *2-tier structure*

The main basic communication modes:

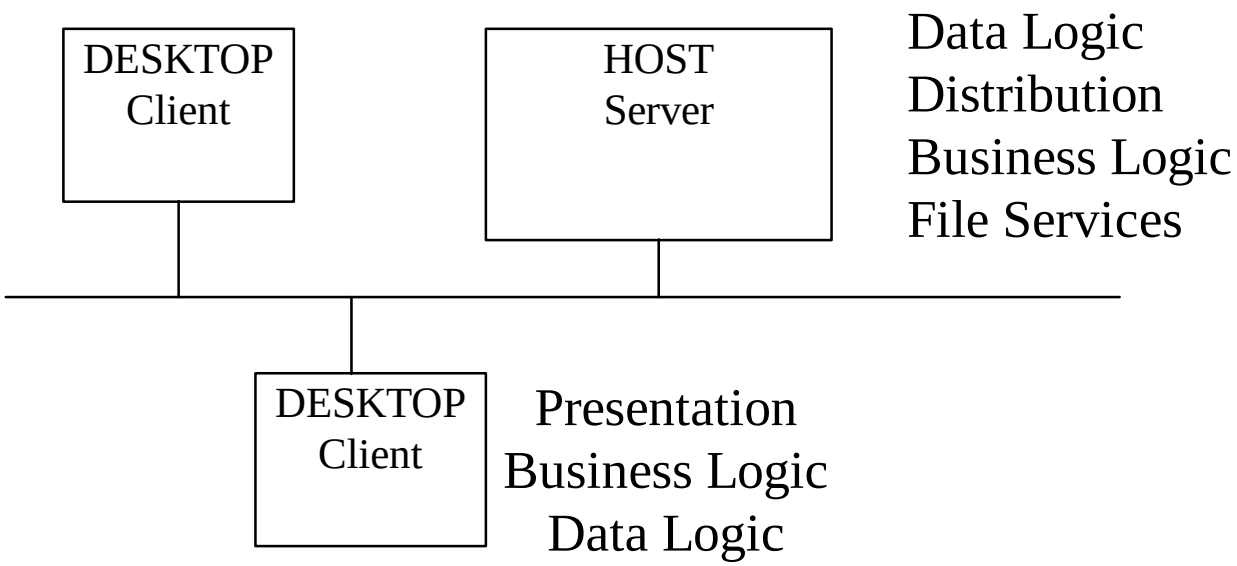
- RPC, remote procedure call
- Message-based

- The client and server parts may reside on the same node or on different nodes
- A part can play the roles of a server of a service and a client of an another service at the same time
  - A client can be connected to several servers

Functional requirements in the 2-tier structures

The applications may be divided into the following logical functional components:

- Presentation Services
  user interface, dialog control
- Presentation Logic
  user interaction, simple validation
- Business Logic
  control flow
- Distribution Services
  communication management
- Database Logic
  integrity, data manipulations
- Database Services
  security, transaction management
- File Services
  file sharing



Functional distribution in the 2-tier structures

Fat clients: Most of the functional modules of the application are performed on the clients

File server

Lite clients: Only few functional modules of the application are performed on the clients

Remote Data Client-Server Architecture

Client		Server
Presentation		Data services
Business logic	⇔	File services
Data logic		

Remote Presentation Client-Server Architecture

Client		Server
Presentation		Business logic
	⇔	Data logic
		Data & file services

Split Logic Data Client-Server Architecture

Client		Server
Presentation		Business logic
Business logic	⇔	Data logic
Data logic		Data & file services

Implementation of Business Logic at the Server

Active elements in the Databases
- triggers
- stored procedures

Trigger:
stored DBMS procedures that are executed when there is change in database.
- triggering event
- response

It is stored in the DBMS
Centralized management and access control

Oracle SQL:
CREATE TRIGGER name AFTER | BEFORE INSERT | UPDATE | DELETE ON table FOR EACH ROW
BEGIN
PL/SQL block
END

Stored procedures:
collection of SQL statements and procedural language statements that control the flow of the procedure
It is stored in the DBMS
Centralized management and access control

It provides faster execution
Mostly pre-compiled

Requirements of an Open OLTP System

Transactions
Logical unit of data processing operations
- Atomicity
- Consistency
- Isolation
- Durability

OLTP
On-Line Transaction Processing
Characteristics:
- Short time transactions
- Several concurrent transactions
- Read-write transactions
- Database stores the actual state of the problem domain

Requirements of open OLTP systems
(based on X/Open DTP model)

- Vendor independence
- Application portability
- Distribution transparency
- Modularity
- Scalability
- Reliability
- Reconfigurability
- Monitoring

Benefits and Limitations of 2-tier Client/Server Architecture

Benefits

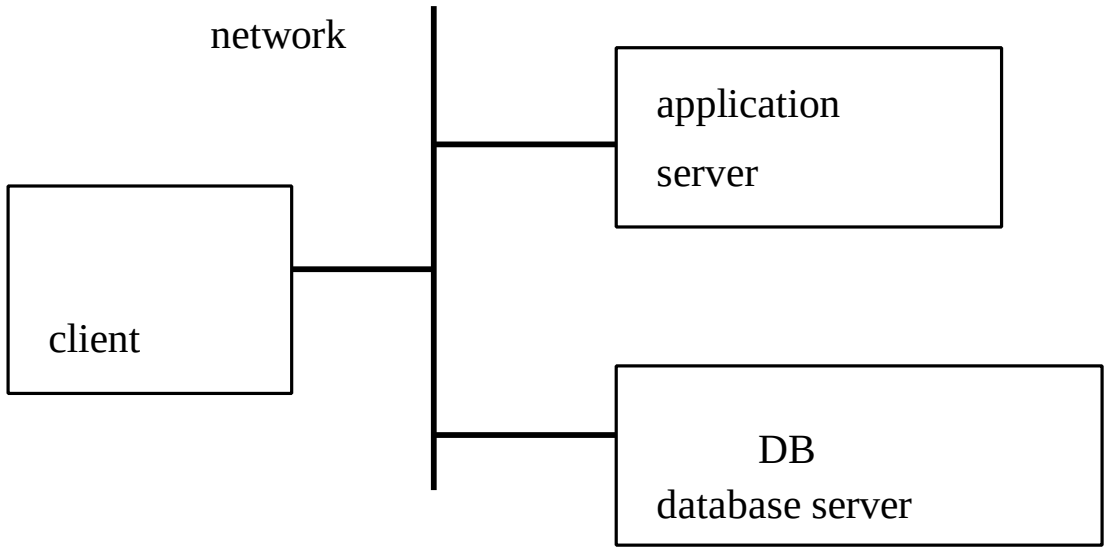
- Interoperability
- Portability
- Integration
- Transparency
- Security

Limits

- The client and server are tightly coupled, the client should use the interface protocol of the server.
- Network traffic is handled less efficiently, because it clients have a direct connection to the server.
- The network traffic increases significantly when high volumes of data and messages are shipped on the network.
- Asynchronous activity. - when a request is sent and an immediate reply is not required - is not supported.
- The connections are limited to a single resource per transaction.
- The application administration is difficult.
- One cannot tune the response times and there is no capability to perform load balancing
- The resources are scaled on the number of connected users not on the throughput
- It consumes many operating system processes and network connections in order to support a given set of users

The 3-tier Structures

Distribution of the different functional modules of the application on three different sites



Usual distribution:

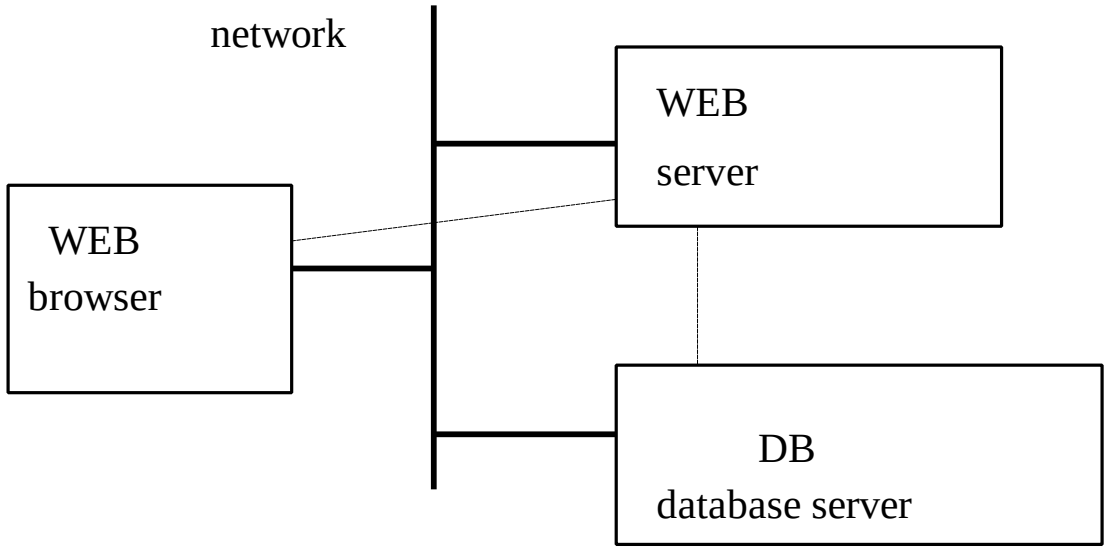
1. tier: Presentation logic, lite client
2. tier: Business logic, application server
3. tier: Database logic, database server

- less software on the client
- increased security
- more power, efficiency
- higher scalability
- lower support cost
- complex structure
- problem of heterogeneous data sources

The 3-tier structure can be extended to n-tier structure, containing several special application servers

Internet-based 3-tier architecture

Open, standardized interface



The WEB servers can access the databases through CGI interface

Interfaces among the components:

1./4. WEB browser ⇔ WEB server

HTML API

<FORM …. ACTION= URL>

….

</FROM>

2. WEB server ⇒ DBMS

CLI (usually SQL)

SQLEXEC(SQL-statement)

2. DBMS ⇒ WEB server

result written to standard output

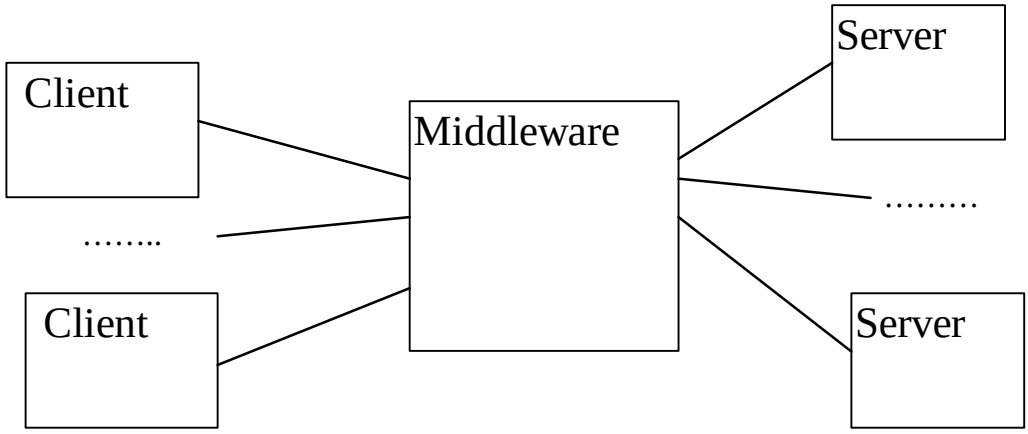
HTP.PRINT(HTML-statement)

Middleware

In order to couple the server and client parts from heterogeneous environments in an efficient way, the basic client server architecture is extended by a new component.

Middleware
An integrating resource between the clients and servers. It performs the following main activities:

- translation between the different protocols
- optimization of the load-balancing
- security control
- management of the connections



The middleware may contain several components.
The components may reside on the server node, on the client node or on a new middleware node.

The different types of middleware

- database
- network
- application cooperation

Component Software Model

The clients invoke managed server components, the named services. These services implement the core business routines. This structure is based on the three-tiers model:

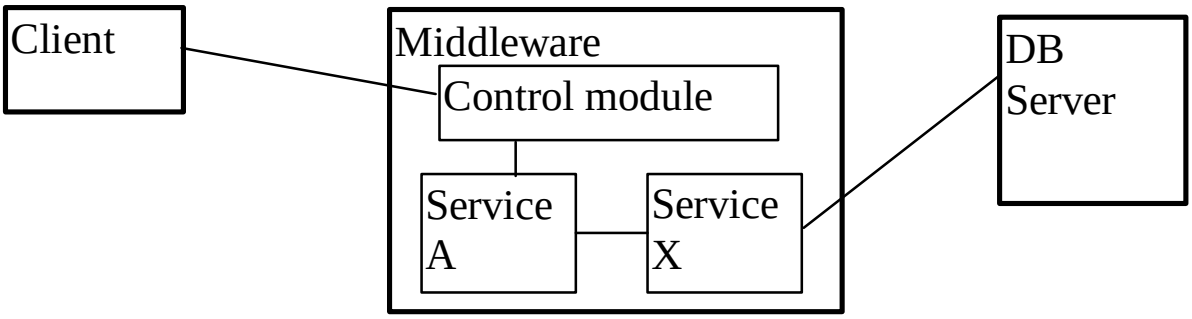
client: it performs GUI and invokes server components

middle tier: it is composed of server components to be invoked by the clients

- control module with control data
- *named services*

resource manager: database management

Named services are managed dynamically, as it offers a better optimization, starting and stopping the services depending on the system load. These services may invoke other services.



software pipelining:

A service processes only a part of the request and forwards it to an another service for further processing. If the first service will be free it can start to process the next request waiting for it. Thus the requests are processed in a pipeline. If the request does not need further processing steps, the actual service will return the result to the client. A request will touch several services.

Control Module and Control Data

Function components of the *control module*:

- naming services
- message routing services
- load balancing services
- configuration management services
- transaction management services
- security services

Main *system configuration data*:

- system resources, security level, load balancing level,…
- participating server definitions
- accessible services
- location information of the services, servers
- runtime repository of application statistics

Data dependent routing:
The service request is mapped to a specific server group based on a value contained in a designated field. It routes the request to a specific service/resource sets. The routing information are stored in BB (Bulletin Board)

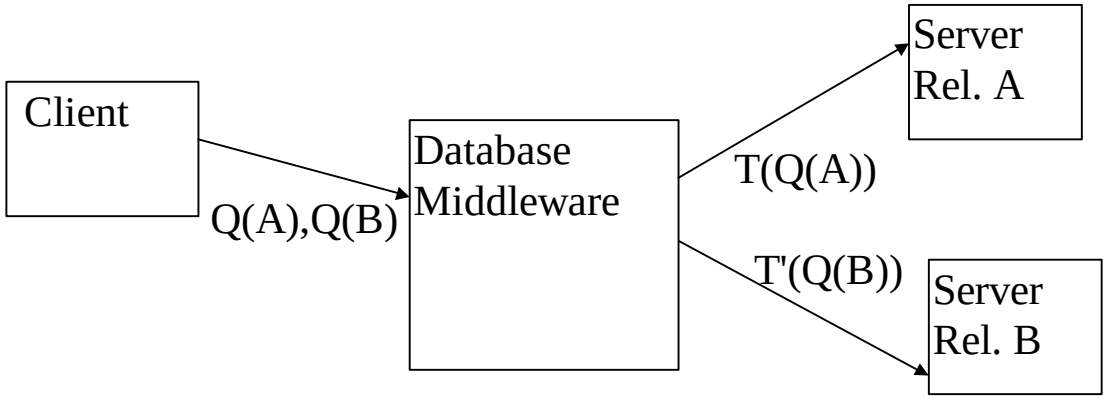
Security
It provides application service authentication , authorization and access control through an architected secury interface. The interface usually abstracts the Kerberos security model and allows the security systems to be integrated with the application. Access control lists may be used to protect services, queues, or events from unauthorized access.

Database Middleware

The two main services of the database middleware are the data management and the distribution services.

Data Management Service
It provides the transparent read and write operations to the server database systems. When it receives a request for data, this service translates the request to the appropriate database language and performs a query on the target database.
This service makes also the database replication transparent to the applications. It takes the responsibility of keeping all replicated databases synchronized.

Distribution Service
This service provides distribution or database location transparency for the applications. It unburden programmers from needing to know where applications and data are stored. The programmer has only to give the object name.
The real position of the object is determined by the directory service. The directory service keeps track of real addresses of the alias names. This enables scaling of applications by relocating logic or databases onto more powerful machines.



Transaction Processing and Integration Middleware

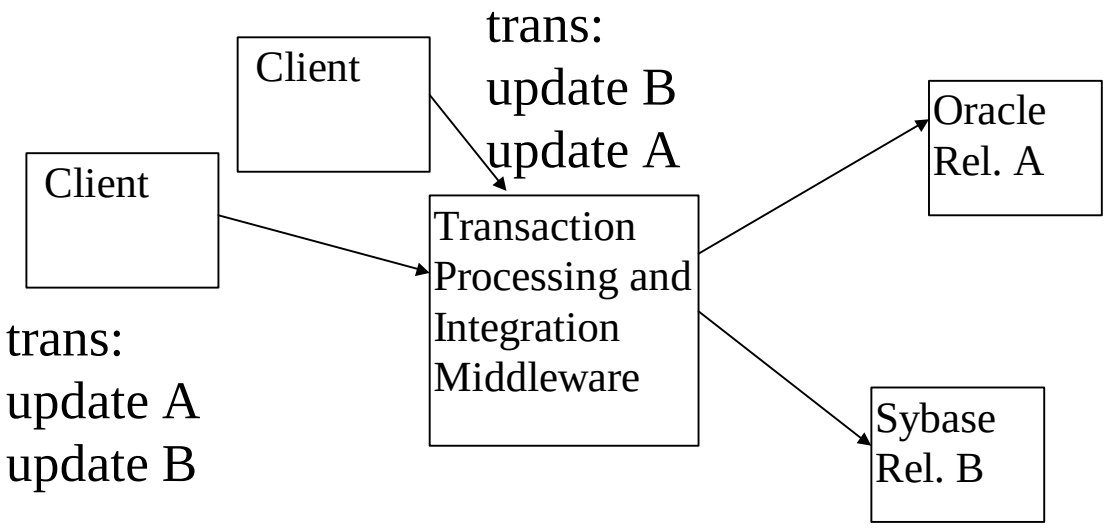
Transaction processing

The transaction processing (TP) is the automated, interactive processing of transactions to update the a shared database. A typical TP application features many end users interacting concurrently with a system to process business transactions.

Transaction Processing and Integration Middleware

Integration software in distributed, heterogeneous environment. It provides:

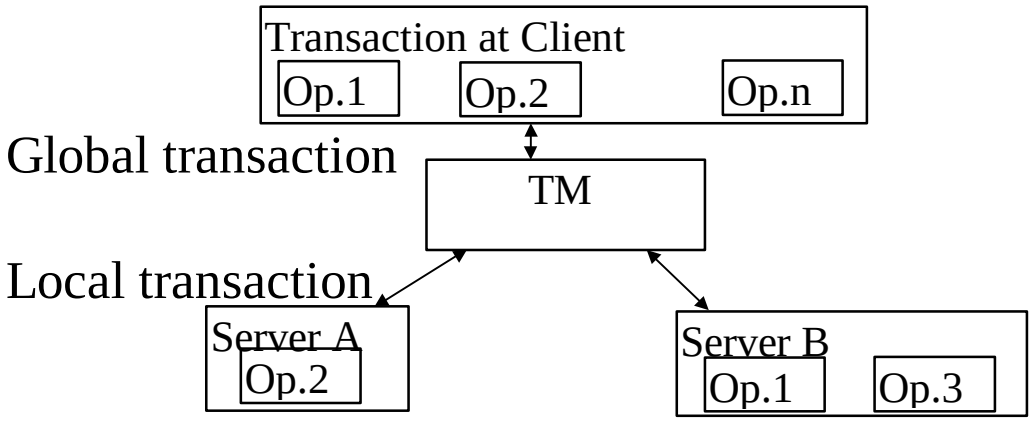
- Integration across heterogeneous platforms
- Reuse of existing applications and data
- Staging the activities involved in application and business process re-engineering



Transaction Processing Monitors

A Transaction Processing Monitor (TM) manages and coordinates the flow of transactions through the system. It guaranties the ensuring of the ACID concept, the integrity of data accessed across several sites or managed by different database products.

The TM coordinates distributed transactions to enable multi-site updates against a heterogeneous database on networked computers. It tracks transaction participants using global transactions and supervises a two-phase commit protocol. It also coordinates the recovery in the event of a site failure, network failure or resource deadlocks. It uses algorithms designed to minimize the cost of database accesses and network traffic.



Transactional RPC :
RPC's that carry transaction contexts along with their messages and data, and that provide the exactly once invocation semantics required for reliable distributed communications. It is extension with transactional syntax.

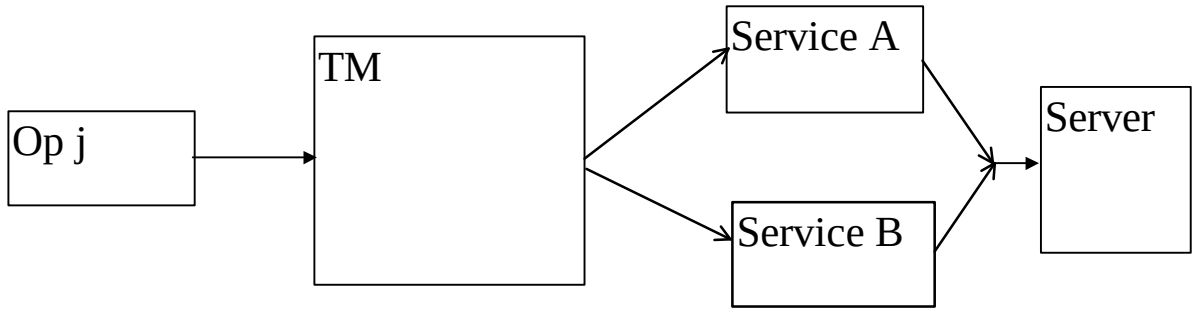
Performance Issues in TM

Load-balancing algorithms can spread work over machines and dynamically add more processes when hotspots of activities occur.

- static
- dynamic

To ensure maximum application throughput, the control module automatically performs load balancing and scheduling. It delivers a particular request to the server that can process the request most quickly. It calculates the load on a given server by totaling the load factors for the currently enqueued requests.
The assignment of the request is usually based on the actual load distribution of the system.
Other, more sophisticated methods may be applied that also take other aspects into consideration as

- network traffic
- task overlapping



TP monitors' *funnelling* capabilities allow large number of clients to share scarce system resources on nodes. This idea of sharing is based on the statistical fact that not all clients want to perform transactions exactly at the same time.

Two-Phase Commit Protocol in TM

In the case of distributed transactions, the global transaction contains several local transactions.
The global transaction is successful only then, when all of the local transactions are successful.
The Two-Phase Commit (2PC) protocol ensures this requirement.

One node among the participating transaction nodes should play the role of a coordinator node. The coordinator node is responsible for taking the final commit or abort decision.

2PC has two stages:

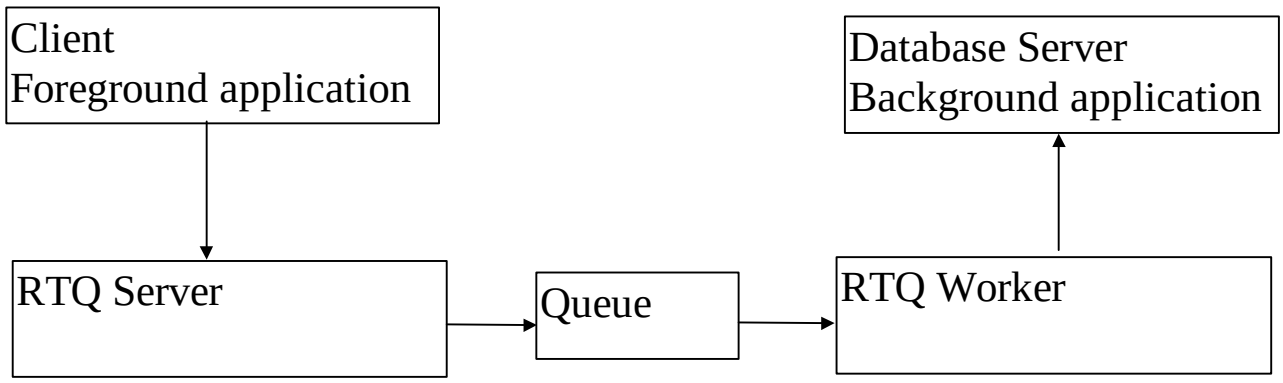
- a prepare phase
- a commit phase.

The purpose of the prepare phase is to allow the transaction manager to collect a "yes" or "no" vote on the transaction from all participating databases. If any "no" votes are received then the entire transaction is rolled back. If all "yes" votes are received then the transaction manager enters the second, commit, phase of the protocol.
In the second phase a request to complete the transaction is issued to each participating database. At the end of the second phase the transaction is completed on all databases.

This ensures that in a distributed transaction that all servers will either commit or rollback; never will one commit and the other rollback. This is very important in the account debit example cited earlier where one account must be credited and the other debited.

Recoverable Transaction Queuing, or RTQ accepts and processes transactions using the store-and-forward concept: client applications store transactions in a queue, and those transaction are later pulled from the queue and processed. RTQ supports various scheduling options for queued transactions

RTQ runtime management system architecture.



RTQ is used to process transactions that do not require immediate processing. A foreground application submits transactions to the RTQ Server and a background application processes queued transactions with the RTQ Worker. Queued transactions can be processed at a scheduled time or in a trickle-feed fashion. For example, to increase data protection an application can commit database changes to a regular database and to RTQ. RTQ will later forward these updates to an alternate (possibly off-site) database.

Message Sensitive Routing

Caching:
When a client application reads a database record, the DBMS executes an I/O to read that record from the shared database. When the I/O completes, the DBMS retains a copy of the record in memory. If any other application requests the record it can be simply forwarded from memory and does not need to do an I/O.

A good performance can be achieved by a good "cache hit rate"

The distribution in DDBMS can cause a decrease of the cache hit rate.

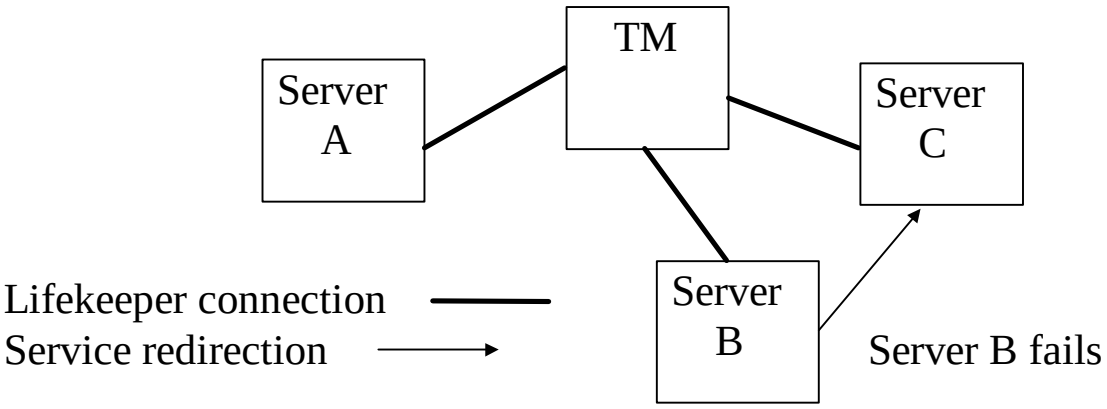
Pinging:
The same database record is updated at different sites, servers after each other. Updated records that are released to another server must first be removed or "flushed" from memory, as it is not ensured that this value in memory remains the actual value in the database due to the other server.
When pinging occurs frequently it can significantly reduce the OPS cache hit rate and overall system performance.

The Message Sensitive Routing (MSR):
The requests for selected data always get routed to the same DBMS server, thereby reducing pinging. MSR routes messages based on a value (or set of values) within the request itself.
MSR improves the cache hit rate.

Lifekeeper Clusters

If there is a failure on server node in the DDBMS, in order to resume processing it is necessary to first restore the server to operational status and then restart the local DBMS and associated applications. When restarted, DBMS performs recovery to restore database integrity.

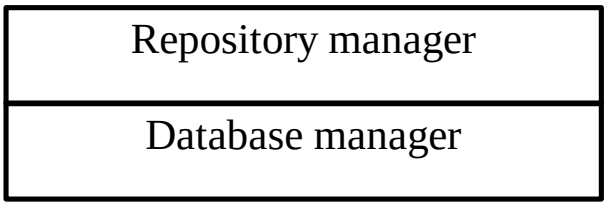
LifeKeeper:
is a software product that monitors two or more servers, detects server failures, and automatically restarts failed applications on an alternative server.
Because of its distributed service location transparency, the Transaction Monitor can route requests accordingly if LifeKeeper restarts failed applications on a different server.



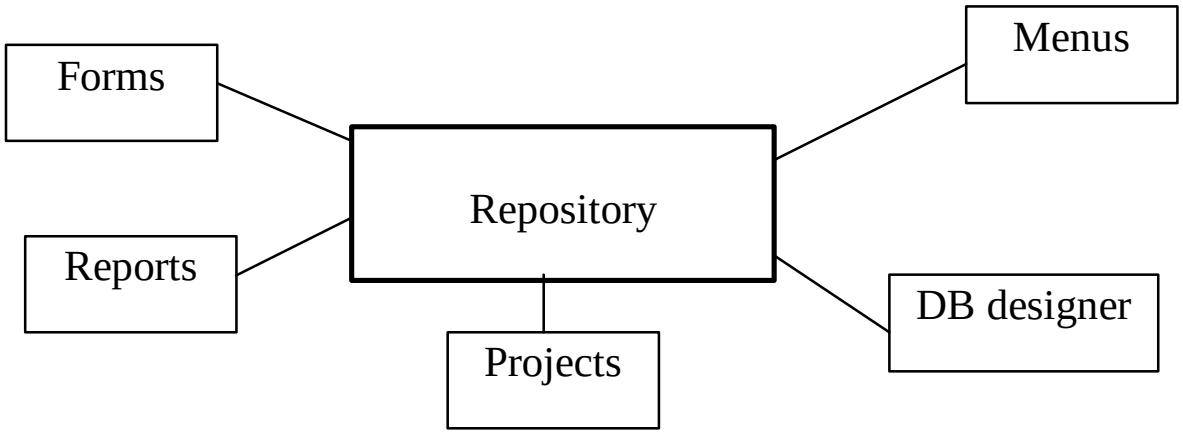
LifeKeeper service runs on each server and maintains a heartbeat connection so it can detect and respond to server failures. The TM replicates application logic within and across servers for scalability and ensures continued access to services from either server in case of component failures.

Repositories in Client-Server Environment

Repository:
a special database of information about engineered artifacts, that holds the description of objects stored in normal databases



Content of repositories:
- database objects definitions
- documents
- source code
- type definitions
- interface definitions



Benefits of shared repository
- centralized access
- only one instance is required
- minimal redundancy

Repository products

- repository manager
- browser
- information model
- scripting language
- data import and export utilities

The *repository manager* supports the operations on the repository including

- functions to manage change objects
- DB systems amenities

Some problems related to repositories

- managing changes
  manage parallel development
- managing relationships
  relationships among heterogeneous sources
- source and target type checking
  integrity checking
- propagation of changes
  operation on source propagates to target
- version management
  dependency among the versions
- team development
  parallel development, release management
- reusing
  searching, conversion, re-definition

4GL Application Development Environment

The Transaction Managers may be integrated with the participating DBMS's 4GL components.

Main 4GL components:

- Forms designer
- Menu designer
- Report designer

This integration has two levels:

- Interface module
- Generator module

TM Interface:
The TM Interface translates the TM APIs into DBMS native calls accessible to the 4GL developer. The developed applications benefit from the 4GL tools - provided features like GUI controls, modal and modeless windows.

TM Generator:
This reads TM distributed services information and generates DBMS 4GL objects. Once a service is selected, the default 4GL modules are generated with objects corresponding to the TM distributed service input and output parameters. The developer can personalize the generated 4GL modules using the standard 4GL components.

Conventional API of TEXUDO

The API provides interface for
- sending messages to a named service
- receiving results
- transaction demarcation

with four communication model:
- synchronous and asynchronous request/response
- conversation
- store and forward message queuing
- event based programming

TEXUDO : Distributed Transaction Monitor with high level API

Two basic types of API interface:
- library-based
- language-based

Library-based interface:
Library-based programming is supported via a set of 3GL (like C, Cobol) procedures. The set of these procedures is called as Application to Transaction Manager Interface (ATMI).ATMI is a superset of the X/Open's XATMI interface.

Language-based interface:
The language-based programming is a remote procedure call (RPC) facility called TxRPC. TxRPC defines a transactional extension to the OSF's DCE RPC.

Communication Models

Library-based request/response mode
based on native functions
- tpcall(): calling service in synchronous case
- tpacall(): calling service in asynchronous case
- tpgetrply() : requesting the answer in in asynchronous case
- tpforward() : send the request to an another service

In *synchronous* case the caller resumes the execution until the answer is received. In *asynchronous* case the caller does not wait for the reply. It go on with the processing. The requested services may be executed parallel. The caller can by means of the tpgetrply() function synchronize the requested services.

```
{  tpinit(NULL);
   data1 = tpalloc(BUFTYPE, SUBTYPE, len);
   tpbegin();
   tpcall('DEPOSIT', data1, 0, &data1, &len, 0);
   tpcall('WITHDRAW', data1, 0, &data1, &len, 0);
   tpcommit(0);
   tpterm(); }
```

```
{  tpbegin();
   handle1 = tpacall("withdraw',…);
   handle2 = tpacall('deposit'..); //parallel execution of services….
   tpgetrply(&hand1,&r1…);
   tpgetrply(&hand2,&r2…);
   tpcommit();    tpterm(); }
```

Language based (IDL) request/response mode
It is based on TxRPC
RPC : remote procedure call, it uses function call formalism
The requested function code is located on a remote node and the execution is performed on that remote node.
It can be only synchronous. The client suspends its execution until the result is received.

```
{     tx_begin();
      ret1 = withdraw(account1, amount1, &balance);
      ret2 = deposit (account2, amount2, &balance);
      if (ret1 == SUCCESS && ret2 = SUCCESS)
            tx_commit; else tx_rollback(); }
```

Software pipelining
The requested service (function) may perform only a subpart of the work requested by the client. In this case, it sends a request to an another service t complete the task.

Client site calling a service: tpcall ('validate',data,…)
Server site the implementation:

```
validate(R * reques){
  data = request->data;
  if (check(data.id) == SUCCESS) {
            // forward to an another service
     tpforward('CUST_STATUS',data,… );
  } else {  raise_failure(..); tpreturn(failure,..);}}
```

Conversation-based mode

based on the following functions

- tpconnect() : client creates a connection to a service
- tpsend() : send a message to the partner
- tprecv() : reading the message from the partner

The server is called immediate, it uses a tprecv() to read the next message from the incoming request queue.

Client site:

```
rec>bc = 100; /* to request 100 records */
cd = tpconnect('GETRECORDS',…);
while(1) {
    tprecv(cd,&recods,…);
    num_cnt = process_records(records,..);
    if num_cnt < 100 break;
    tpsend(cd,….); /* requesting the next set of records */
}
```

Service code:

```
GETRECORDS( *msg){
data = msg->data;
while (act_cnt < data->request_cnt) {
   get_next_batch_of_records(…);
   tpsend(msg_cd,data,…);
      // wait for client that asks for more records
   tprecv(msg->cd,&data,…);
}
get_last_batch_of_records();
tpreturn (TPSUCCESS,..);}
```

Event-based mode

A server informs the clients about the happening of some events. The clients are ready to activate a service-module as response to the event.

The event-based communication can be:

- unsolicited : the clients register a service to each of their events. If an event occurs, the registered service should be called. Functions:
  - tpnotify() : send message to only one client
  - tpbroadcast():send message to one or more clients
- brokered : There is no direct connection between the even generator and event consumer. There is three types of roles:
  - subscriber: it subscribes at the broker for some events
  - event generator: generates the events
  - broker: it decides upon the subscription which clients should be informed about the event

The main functions:

- tppost(): inform the broker about an event
- tpsubscribe(): subscribe an event
- tpunsubscribe() : unsubscribe an event

Event-based mode

Example

client site:

```
tpsetunsol (atm_handle);
// it tells the server to invoke the atm_handle function when an unsolicited message arrives
```

server side:
the VALIDATE service

```
validate(R * reques){
  data = request->data;
  if (check(data.id) == SUCCESS) {
     tpforward('CUST_STATUS',data,… );
  } else {
          // notify the consol about the failure
     tpnotify(consol, data,FLAG,..);
     raise_failure(..);
     tpreturn(failure,..);
 }
```

Store and forward message queuing mode
based on the following functions

- tpenqueue() : place a message into the queue
- tpdequeue() : take a message out of the queue

It can be used in cases when the partners can not communicate with each other in on-line way. The message is stored in a queue until it can be processed.

Transaction demarcation interface
It is used to determine the boundaries of the transactions.

It is based on three routines:

- tpbegin
- tpcommit
- tpabort

Tpbegin : start of a global transaction. A new transaction ID is generated. It will be attached to every activity during the transaction. The Transaction Manager interacts with the participating resource managers to complete the transaction commitment or abort coordination protocol.
Tpcommit: commit the transaction, the result of updates is stored in the database.
TpAbort: abort the transaction, the updates should be rolled back.

The main standards for open transaction processing

- *X/Open XA distributed transaction processing (DTP)*
which is an architecture that allows multiple programs to share resources (e.g., databases and files) provided by multiple resource managers and allows their work to be coordinated. The architecture defines application programming interfaces and interactions among transactional applications, transaction managers, resource managers, and communications resource managers. The transaction manager and the resource manager communicate by means of the XA interface.

- *X/Open transactional remote procedure call (TxRPC)*:
which allows an application to invoke local and remote resource managers as if they were all local. TxRPC also allows an application to be decomposed into client and server components on different computers interconnected by means of remote procedure calls (RPCs).