System architecture

Maciej Grzenda
Warsaw University of Technology
Faculty of Mathematics and Information Science
M.Grzenda@mini.pw.edu.pl
http://www.mini.pw.edu.pl/~grzendam
Distributed databases – when?

• For mission-critical systems when stable connection to the remote server can not be provided
• To decrease response time
• To provide for autonomous maintenance of software systems
• When different software systems in an organisation require their separate DBMS
Distributed databases – sample architecture

Client applications

Database Server#1

LAN

Database#1

VPN

Database#n

Client applications

Database Server#n

LAN

...
Distributed transactions

- Transactions that require multiple databases or multiple DBMS to participate in
- If every database has its own private transaction log, then even a transaction that affects two different databases on the same database server requires a distributed transaction
What is a distributed transaction?

• DT requires significant overhead – 2PC (two phase commit protocol) is involved. Basically, there is a coordinating service required. Notice that the following problems must be addressed:
  – Some of participating DBMS may be not available,
  – Some of DBMS may not accept requested changes e.g. due to constraint violation
Distributed transaction - example

DELETE
KRAKOW_OFFICE.Sales.dbo.Customers
WHERE CustomerId='1020'

INSERT INTO
POZNAN_OFFICE.Sales.dbo.Customers
VALUES ('1020', 'Top Company', ...)

Maciej Grzenda
http://www.mini.pw.edu.pl/~grzendam
Example - discussion

• A DT may be submitted to one of participating DBMS servers
• The transaction coordinator must communicate with both DBMS to:
  – Confirm updates are possible e.g. there is no 1025 customer in POZNAN_OFFICE DBMS
  – Transactionally execute updates to both servers
## Data synchronisation in distributed databases

<table>
<thead>
<tr>
<th>Method</th>
<th>Short description</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed transactions</td>
<td>• Updates are made to all participating servers at the same time&lt;br&gt;• Transaction processing stopped whenever any of the servers is not available</td>
<td>• The only method that guarantees on-line consistency of participating databases</td>
</tr>
<tr>
<td>Replication</td>
<td>• The updates to monitored tables are transferred to other participating servers.&lt;br&gt;• Can be used to constantly update the data in stand-by servers.</td>
<td>• Accepts latency in data transfer&lt;br&gt;• Updates to participating databases are postponed, thus the system as a whole may accept user’s input even if one of the servers does not respond</td>
</tr>
</tbody>
</table>
Replication overview

Replication categories

- Transactional replication
  Updates to monitored tables are monitored. Complete SQL statements are transmitted to another DBMS.

- Snapshot replication
  Copies of table content are transmitted to another DBMS.

- Merge replication
  The content from two DBMS is merged on regular basis.

Ms SQL Server replication categories have been described.

Maciej Grzenda

http://www.mini.pw.edu.pl/~grzendam
Transactional replication

1. Transaction is submitted
2. The information on transaction is saved
3. The database is updated

Ms SQL Server solutions have been described

SQL statements

Log-reader agent

Distribution agent

DBMS#1

Transaction log

Database

DBMS#2

Transaction log

Database
Transactional replication

• Can be treated as a kind of add-on functionality to the core of DBMS:
  – Log reader agent monitors on-line all the changes to some of the tables (say: customers) by using transaction logs,
  – Distribution agent delivers the SQL statements (namely INSERT, UPDATE and DELETE made on Customers table) to the subscriber periodically (e.g. every night) or on-line
Transactional replication – case study

- Task: the head office must have a complete list of invoices issued by regional office by the end of the next business day
- Solution #1:
  - Transactionally i.e. using distributed transactions repeat all the updates made on regional server so as to reflect them on the central server
  - Drawback: no invoices can be issued once the central server gets unavailable
- Solution #2:
  - Set up transactional replication (also on-line)
  - In case central server is not available, all the changes to invoices tables are still gathered by the log-reader agent and delivered later – once the connection between the server is re-established
Snapshot and merge replication

• **Snapshot replication:**
  – Can be used to periodically replace the content of subscriber’s table(s)
  – Example: publication of price list from the company’s headquarter

• **Merge replication:**
  – Can be used to periodically merge the content of two different databases
  – Example: table of customers existing on the central server and mobile workstations of sales representatives. Central database is merged with new customers’ data from mobile computers.
High availability (HA)

- In case of IT systems means that the system is available all the time (24x7),
- Typical HA architectures typically involve:
  - Redundant devices:
    - Redundant power supplies,
    - RAID (Redundant Array of Inexpensive Disks),
    - Multiple network interface cards,
    - Redundant network connections,
  - Clusters of servers.
HA: clusters

- Cluster: two or more similar interconnected servers usu. sharing the same set of disks (disk array),
- Different strategies exist. Examples:
  - Active/passive: one of the servers is ready to accept the requests, but normally passive
  - Active/active: both servers (or more than 2) share the load
Cold failover cluster

Before failover:
- SQL statements
  - Server#1
  - Server#2
  - Database
- Up and running, but idle

After failover:
- Not functioning.
  - When recovered, still does not receive SQL statements
  - Server#1
  - Server#2
  - Database
  - SQL statements

Maciej Grzenda
http://www.mini.pw.edu.pl/~grzendam
### Hot failover cluster

**Before failover**

- **Server#1**
  - SQL statements

- **Server#2**

- **Database**

- All cluster servers are working and handle requests

**After failover**

- **Server#1**
  - Not functioning. When recovered, can join other servers to handle requests

- **Server#2**

- **Database**

- SQL statements

---

**Maciej Grzenda**

http://www.mini.pw.edu.pl/~grzendam
Multiple database instances handle one database. At least two servers are needed. HA is ensured: if some of the servers fail, the remaining servers can still handle the users. Still, not fully transparent for client applications in case of failover. The servers are connected by high-speed connection. Some operations have to be negotiated by the servers e.g. locking a record.
Client - server model

- client
- client
- ......
- client

SQL commands

DB Server

database
Multi-tier model

Client → Application Server

Application Server

... Communication with business objects...

Application Server

Connection pool

Application Server

SQL commands

Application Server

DB Server

Client

Database

Client
Multi-tier model

• +:
  – better data access through connection pool,
  – large number of clients supported,
  – business logic separated from the interface,
  – better work organisation in large web projects - project is subdivided into smaller subprojects:
    • presentation layer (e.g. JSP, servlets, XML, XSLT),
    • business layer (EJB)
    • database layer

• –:
  – steep learning curve,
  – complicated environment.
Database connections

- Client communicates with the server via a connection,
- In order to establish connection, authorisation is required, in case of SQL Server:
  - either SQL Authentication (user+password)
  - or Windows Authentication (domain user is a „trusted” user)
- Numerous connections from the client to the database server may exist (although not suggested - consumes server’s resources)
- connection is used to submit queries (SELECT, INSERT, UPDATE, DELETE) and other statements (EXEC Procedure, CREATE TABLE etc.)
Database connection
Sample code - Java

```java
Connection con = null;
try {
    // Load the Driver class file
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    // Make a connection to the ODBC datasource Movie Catalog
    con = DriverManager.getConnection("jdbc:odbc:Movie Catalog", "user", "password");

    // Create the statement
    Statement statement = con.createStatement();
    ResultSet rs = statement.executeQuery("SELECT * " + "FROM Titles");
    while ( rs.next() ) {
        // get the title_name, which is a String
        out.println(rs.getString("title_name"));
    }

    ...... 
    rs.close();
}
```

Maciej Grzenda
http://www.mini.pw.edu.pl/~grzendam
Writing a client-server

• **Preferred sequence:**
  – if something can be done on the server’s side - do it there!
  – Sample server side solutions: stored procedures, user defined functions, replication, import-export facilities

• **Make it a client-server:**
  – always filter the data before displaying it to the user,
  – avoid large processing loops on the client’s side,

• avoid user interaction during a transaction,

• determine appropriate transaction isolation level.