Systems Architecture

Client-Server Systems

Client/Server

- In general, any application where multiple clients connect to a single server.

- one client program (most typical)
  or
- multiple client programs
Relational Databases

- Most common client/server program is where the server is a relational database server.
  - warning: some use the term client/server to refer to this usage exclusively (we won’t).

Relation Database Implementation

- Client1, Client2, Client3 connect to the RDBMS Server which accesses disks.
IPC

- “Inter-Process Communications”
  - How processes will communicate and synchronize with one-another.
  - communications mechanisms:
    - shared memory
      - very fast
      - can’t use over a network
        - well, you can
    - message passing
      - can use over a network
      - slower
        - well, not always
  - will consider only message passing (most important)

IPC Protocols

- Basic message-passing mechanisms provide for a byte-stream only.

- Must implement various protocols on top of this
  - sockets
  - RPC (remote procedure call)
  - DO (distributed objects)
Sockets code example

```java
public class Server {
    public static void main(String[] args) throws Exception {
        ServerSocket server = new ServerSocket(1234);
        Socket client = server.accept();
        BufferedReader fromClient = new BufferedReader(
            new InputStreamReader(client.getInputStream()));
        System.out.println(fromClient.readLine());
    }
}
```

```java
public class Client {
    public static void main(String[] args) throws Exception {
        Socket server = new Socket("penny", 1234);
        DataOutputStream toServer = new DataOutputStream(
            server.getOutputStream());
        toServer.writeBytes("hello server");
        server.close();
    }
}
```

Performance

- **Latency**
  - The time to go back and forth

- **Bandwidth**
  - The amount of data that can be sent

- **Analogy from ocean lines**
  - Bandwidth of QE2 is high (can carry a lot)
  - Latency is bad (takes a long time for a round trip).
Test System

- Windows 2000 Java Server
  - Network
    - 100 Mbit/s ethernet
  - CPU
    - dual 1GHz processors
  - Memory
    - 1 GByte
- Windows 98 Java Client
  - Network
    - 100 Mbit/s ethernet
  - CPU
    - 366 MHz
  - Memory
    - 96 MByte

Java/Windows Performance Measures

- Latency: Sending “hello server\n” back and forth
  - Local method calls
    - .13 usec/2call
  - Socket on local machine
    - 70 usec / 2call (x500)
  - Socket on remote machine
    - 320,000 usec /2call (x5,000 , x2,500,000)
- Bandwidth
  - Sending “hello server\n” to server repeatedly
    - 1400 usec / 2call (x10,000 , x230)
Performance

<table>
<thead>
<tr>
<th></th>
<th>In Process</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency</td>
<td>1</td>
<td>2,500,000</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>1</td>
<td>10,000</td>
</tr>
</tbody>
</table>

C/Windows Performance Measures

- Latency: Sending “hello server\n\n” back and forth
  - Local method calls
    • .01 usec / 2call (10x Java)
  - Socket on local machine
    • 12 usec / 2call (6x Java)
  - Socket on remote machine
    • 840 usec / 2call (380x Java)
Performance

<table>
<thead>
<tr>
<th>Latency</th>
<th>In Process</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>84,000</td>
</tr>
</tbody>
</table>

Performance Implications

- Do as few calls as possible over the net
- Prefer asynchronous approaches
  - problem: success/failure indications
  - send lots of stuff, then synchronize
- Use bigger transactions
- Prefer one call with lots of data to many calls with the same amount of data
  - but not by much
- Send as little data as possible
Relational Databases

- Most common type of client/server software is where the server is an RDBMS server:
  - Oracle
  - SQL Server
  - Sybase
  - Informix

Database Access

- Access using SQL (Standard Query Language)
  - select itemname, quantity
    - from
    - orderitems, items
  - where
    - orderid = 239
    » and
    - orderitems.itemid = items.itemid

<table>
<thead>
<tr>
<th>query result</th>
</tr>
</thead>
<tbody>
<tr>
<td>itemname</td>
</tr>
<tr>
<td>bread</td>
</tr>
<tr>
<td>sugar</td>
</tr>
</tbody>
</table>
Programmatic Database Access

- Can access database by
  - typing commands at an sql command prompt
  - by running a GUI tool
  - programmatically
    - ODBC
      - Open Database Connectivity – Microsoft standard API
      - ANSI/ISO CLI is ODBC level1 compliant (Call Level Interface)
        » (see also DAO, OLE DB and ADO)
    - JDBC
      - very similar to ODBC
    - Various embedded SQL hacks

JDBC

- All sorts of possible configurations of client-side & server-side drivers

```
  +---------------------+  +---------------------+  +---------------------+
  | App                |  | JDBC               |  | ODBC                |
  | JDBC               |  | ODBC                |
  +---------------------+  +---------------------+  +---------------------+  +---------------------+
  | ODBC                |
  +---------------------+  +---------------------+  +---------------------+
  | server              |
  +---------------------+
  | RDBMS               |
```
```java
import java.sql.*;
public class Main {
    private static final String query = 
        "select itemname,quantity " + 
        "from orderitems,items " + 
        "where orderid=1 and orderitems.itemid=items.itemid";
    public static void main(String[] args) throws Exception {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection c = DriverManager.getConnection("jdbc:odbc:grocery");
        Statement s = c.createStatement();
        if( s.execute(query) ) {
            ResultSet r = s.executeQuery();
            printResults(r);
        }
    }

    private static void printResults(ResultSet r) throws Exception {
        final int nC = printHeadings(r);
        printRows(nC, r);
    }

    private static int printHeadings(ResultSet r) throws Exception {
        ResultSetMetaData m = r.getMetaData();
        final int nC = m.getColumnCount();
        for(int c = 1; c <= nC; c++) {
            System.out.print(m.getColumnName(c));
            System.out.print("\t");
        }
        System.out.println();
        return nC;
    }
}
```
Database Access from Java

private static void printRows(int nC, ResultSet r) throws Exception {
    while (r.next()) {
        for (int c = 1; c <= nC; c++) {
            System.out.print(r.getString(c));
            System.out.print("\t");
        }
        System.out.println();
    }
}

Without ODBC

Class.forName(
    "org.gjt.mm.mysql.Driver"
);

Connection c = DriverManager.getConnection(
    "jdbc:mysql://penny.dhcp.cs.toronto.edu/grocery"
);
Performance

- localhost
  - JDBC:ODBC
    - 850 us/query
  - JDBC:MYSQL
    - 500 us/query
- over network
  - JDBC:ODBC
    - 3,800 us/query
  - JDBC:MYSQL
    - 1,600 us/query
- local Java method call
  - 0.13 us/query
- C socket over network
  - 840 us/query

Data Compatibility

- Issue with any sort of system is how to support changes in data format from release to release of the software:
  - backwards compatible
    - newer releases of the software can open older datasets
  - forwards compatible
    - older releases of the software can open newer datasets
- General approach
  - have some sort of flexible header format
  - for backwards compatibility:
    - encode a current data version number
  - for forwards compatibility
    - store the oldest data version number such that
      - older software that uses that data version can still use this data
RDBMS Compatibility Advantages

- RDBMS's have 2 advantages w.r.t compatibility:
  - The data is not highly fragile.
    - e.g., in a binary file format, one small change somewhere can screw up the whole file
    - in SQL the schema can change considerable yet the data can still be accessed
  - RDBMSs support schema evolution
    - SQL
      - CREATE TABLE
      - MODIFY TABLE
      Can work on in-place databases

```
private void updateDatabase() {
    int version = getDataVersion();
    if( version < 1 )
        die("DB consistency error: Version number must be 1 or greater");
    switch(version) {
        case 1:
            updateDatabaseToVersion2();
            // fall-through
        case 2:
            updateDatabaseToVersion3();
            // fall-through
        case 3:
            o.println("<Database is up-to-date>");
            break;
        default:
            die("Database was created with newer version of software");
            break;
    }
}
```
private void updateDatabaseToVersion2() {
    o.println(<Converting database from version 1 to version 2>);
    try {
        sqlup("ALTER TABLE Coders ADD COLUMN w REAL");
        sqlup("UPDATE Coders SET w = 0.6");
        sqlup("UPDATE Version SET version = 2");
        sqlcommit();
    } catch(Exception e) {
        try {
            sqlrollback();
        } catch(Exception e2) {
            die("Error converting database to version 2: " + e.getMessage());
        }
    }
}