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).

![Diagram of client/server architecture]

Relation Database Implementation

![Diagram of relation database implementation]

14 - Client/Server  
CSC407  
3

14 - Client/Server  
CSC407  
4
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” 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>In Process</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency</td>
<td>1</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

Stores data into tables

<table>
<thead>
<tr>
<th>order</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>orderid</td>
<td>orderdate</td>
<td>custid</td>
</tr>
<tr>
<td>239</td>
<td>Nov.13</td>
<td>2349</td>
</tr>
<tr>
<td>267</td>
<td>Nov.14</td>
<td>3903</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>custid</td>
</tr>
<tr>
<td>2394</td>
</tr>
<tr>
<td>3903</td>
</tr>
<tr>
<td>3453</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>orderitems</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>orderid</td>
<td>itemid</td>
<td>quantity</td>
</tr>
<tr>
<td>239</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>239</td>
<td>28</td>
<td>4</td>
</tr>
<tr>
<td>267</td>
<td>42</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>items</th>
</tr>
</thead>
<tbody>
<tr>
<td>itemid</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>28</td>
</tr>
<tr>
<td>42</td>
</tr>
</tbody>
</table>
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>

“stored procedure” if this is parameterized and the whole thing is named

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

```java
import java.sql.*;
public class Main {
    private static final 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);
    }
}
```

**Database Access from Java**

```java
import java.sql.*;
public class Main {
    private static final 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);
    }
}
```
Database Access from Java

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

```java
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