Performance Tuning Web Applications

NYC Metro Area
Oracle Users Group Meeting
September 10, 2008

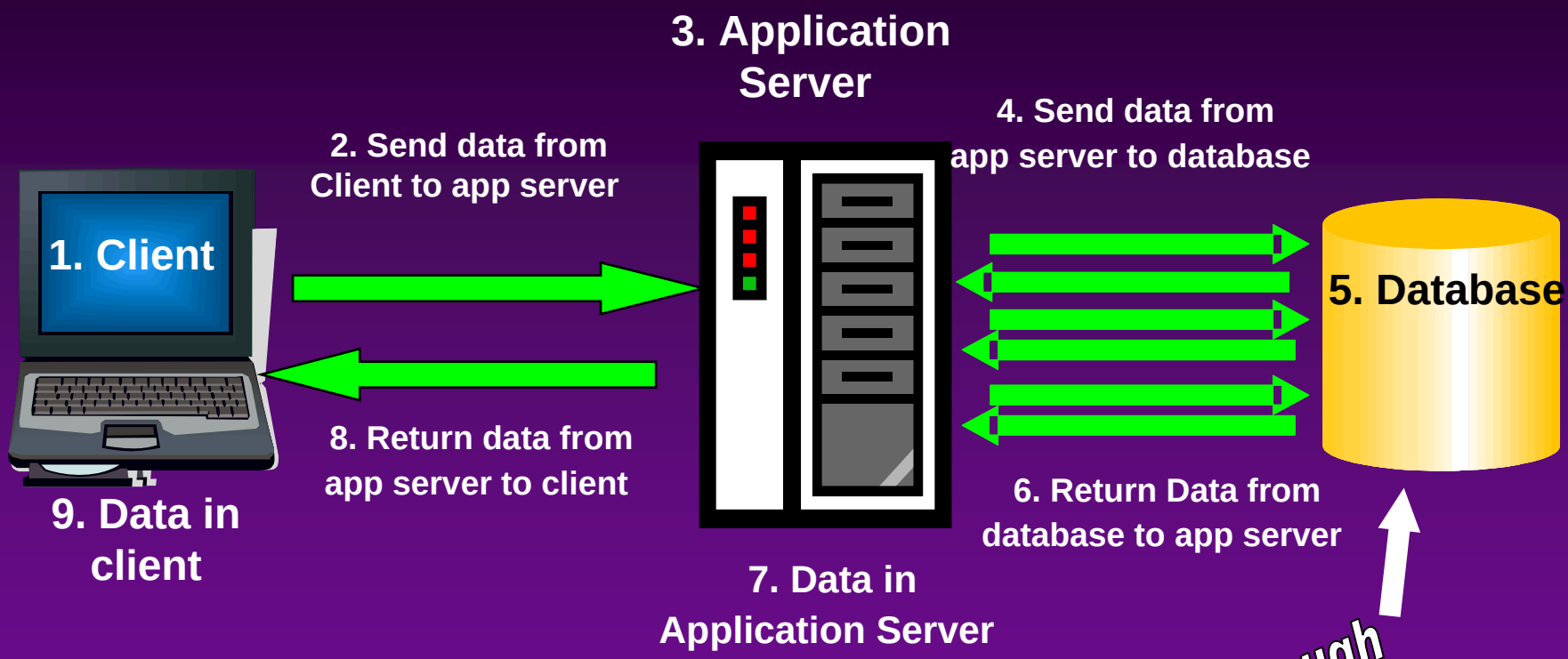
Dr. Paul Dorsey & Michael Rosenblum
Dulcian, Inc.
www.dulcian.com

Why Performance Tuning Fails

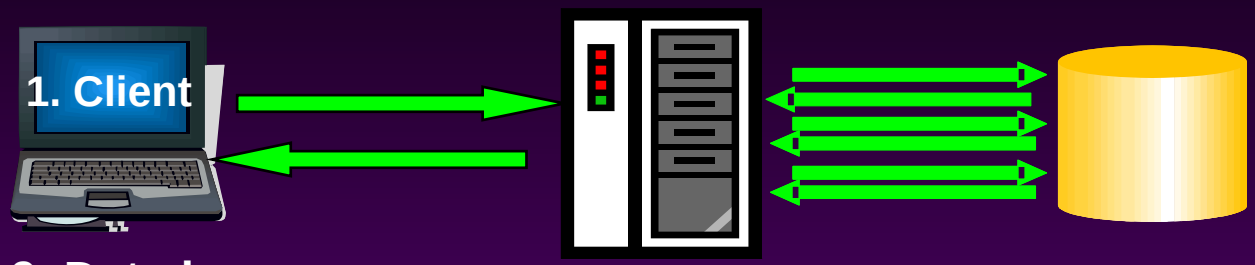
- We are solving the wrong problem.
- Tuning:
  - Usually makes the database run better.
  - Focuses on poorly running SQL.

- Web applications are frequently unaffected by these performance improvement approaches.
- Need to examine the entire system, not just the database.

Web Application Architecture



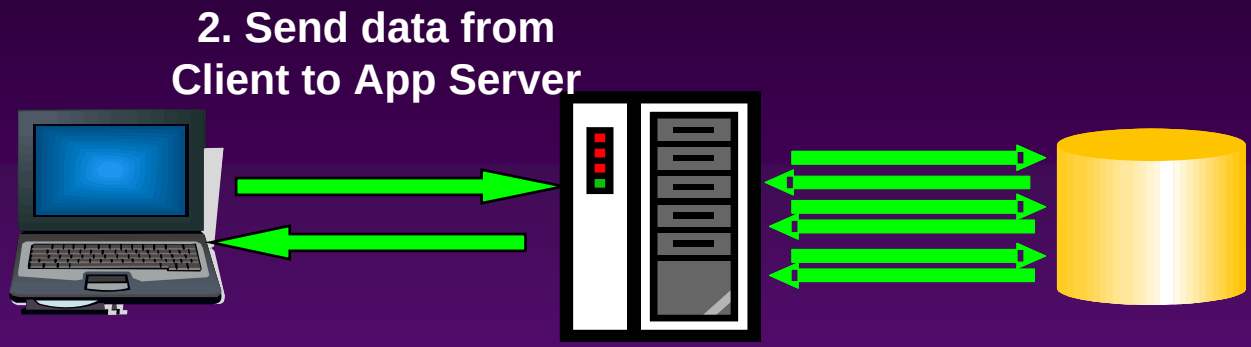
Steps 1 & 9 - Client



9. Data in client

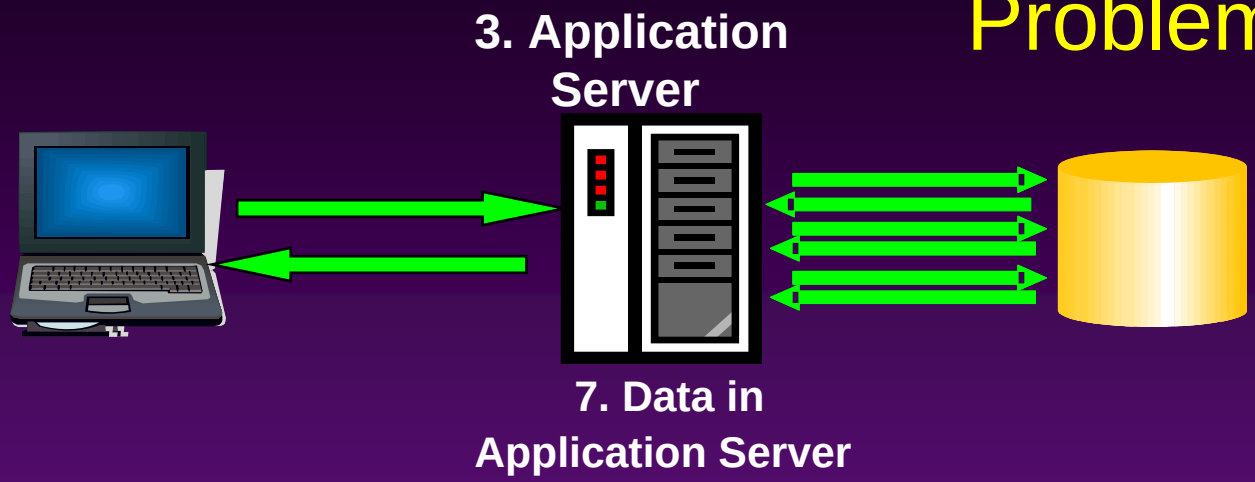
- Unlikely source of problems.
  - Should not be dismissed entirely.
  - Using AJAX architectures, it is possible to place so much code in the client that a significant amount of time is required before the request is transmitted to the application server.
- Beware of underpowered client machines with inadequate memory and slow processors.

Step 2 - Client to Application Server



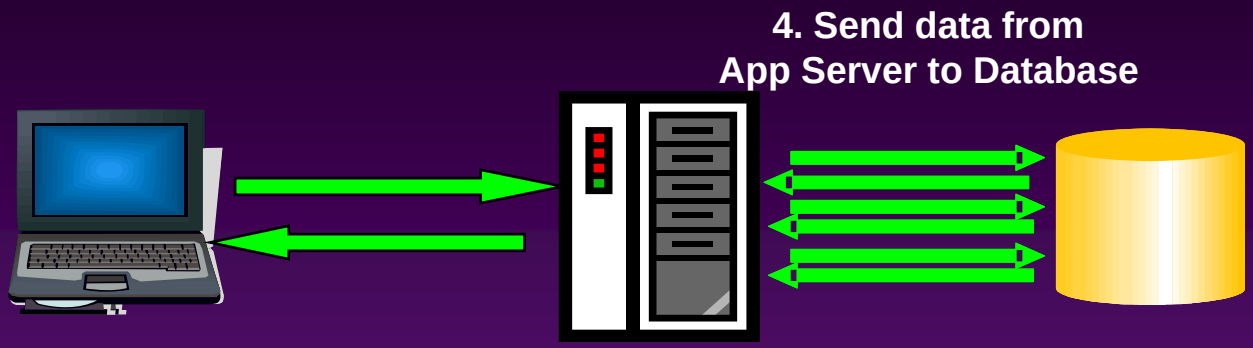
- Less common cause of performance problems
- Transmitting large amounts of information over the Internet may cause problems.
  - Uploading large files
  - Transmitting a large block of data

Steps 3 & 7 Application Server Processing Performance Problems



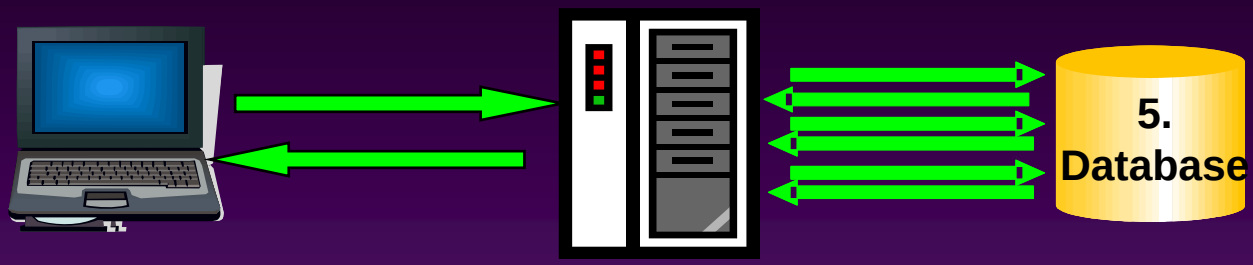
- Processing can be resource-intensive.
- Java programmers minimize database application logic in the middle tier.
- Complex data manipulation can be handled much more efficiently with database code.
  - Thick database approach is the key to efficiently performing web applications.

Step 4 - Application Server to Database



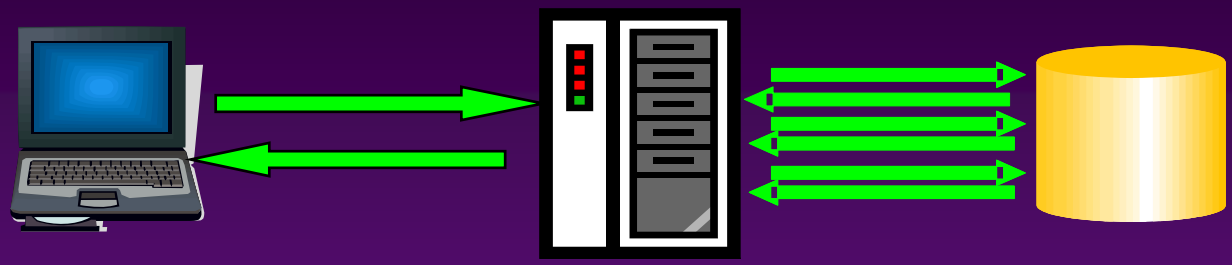
- Not instantaneous (but really fast)
- High number of transmission requests are the #1 cause of performance problems
- Database-independence is not a good idea.
  - Single request from a client may require many requests from the application server to the database in order to fulfill.
- Examine and measure the number of round-trips from application server to database.

Step 5 - Database Performance Problems



- Use traditional tuning.
- Beware of stateless implementation.
  - Information pertaining to a particular session must be retrieved at the beginning of every request and persistently stored at the end of every request.
  - Single table may generate massive I/O
    - Redo logs
    - Block contention

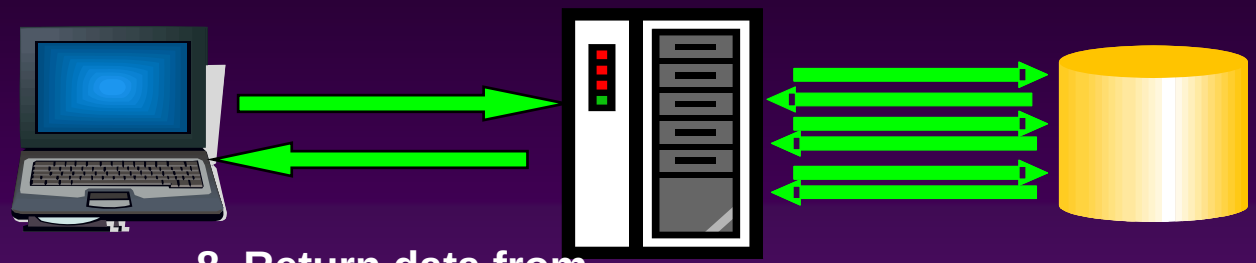
Step 6 - Database to Application Server Transmission Problems



6. Return Data from DB to App Server

- Rare problem
- Beware of unnecessary data movement.
  - One record is needed and the whole table is sent

Step 8 - Application Server to Client Transmission Problems

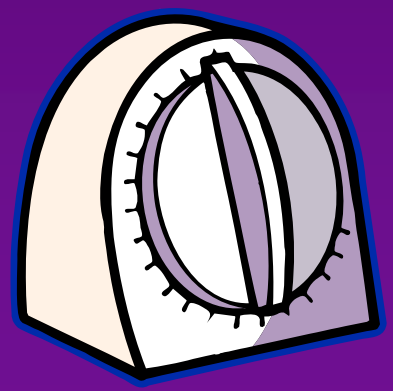


8. Return data from App Server to client

- #2 cause of performance problems
- Keep pages small.
  - Not too many fields
  - Not too much AJAX or JavaScript
  - Not too big a tree
  - Not too much data in a scrolling block
  - No images, or other unnecessary information
- Measure size of page

Locating Slow Performance Causes

- Embed timers into a system to detect where in the nine possible steps the application performance is degrading.
- Strategically placed timers will indicate how much time is spent at any one of the steps in the total process.



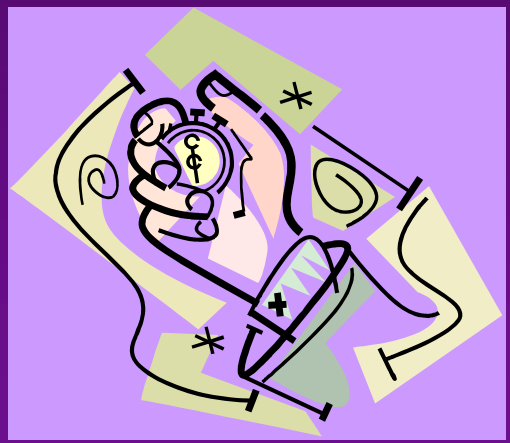
DULCIAN INC.

Common Causes of Performance Problems

- The most common causes of slow system performance are:
  - 1. Excessive round-trips from the application server to the database
  - 2. Large pages sent to the client
  - 3. Performing operations in the application server that should be done in the database
  - 4. Poorly written SQL and PL/SQL routines

DULCIAN INC.™

Measuring Performance

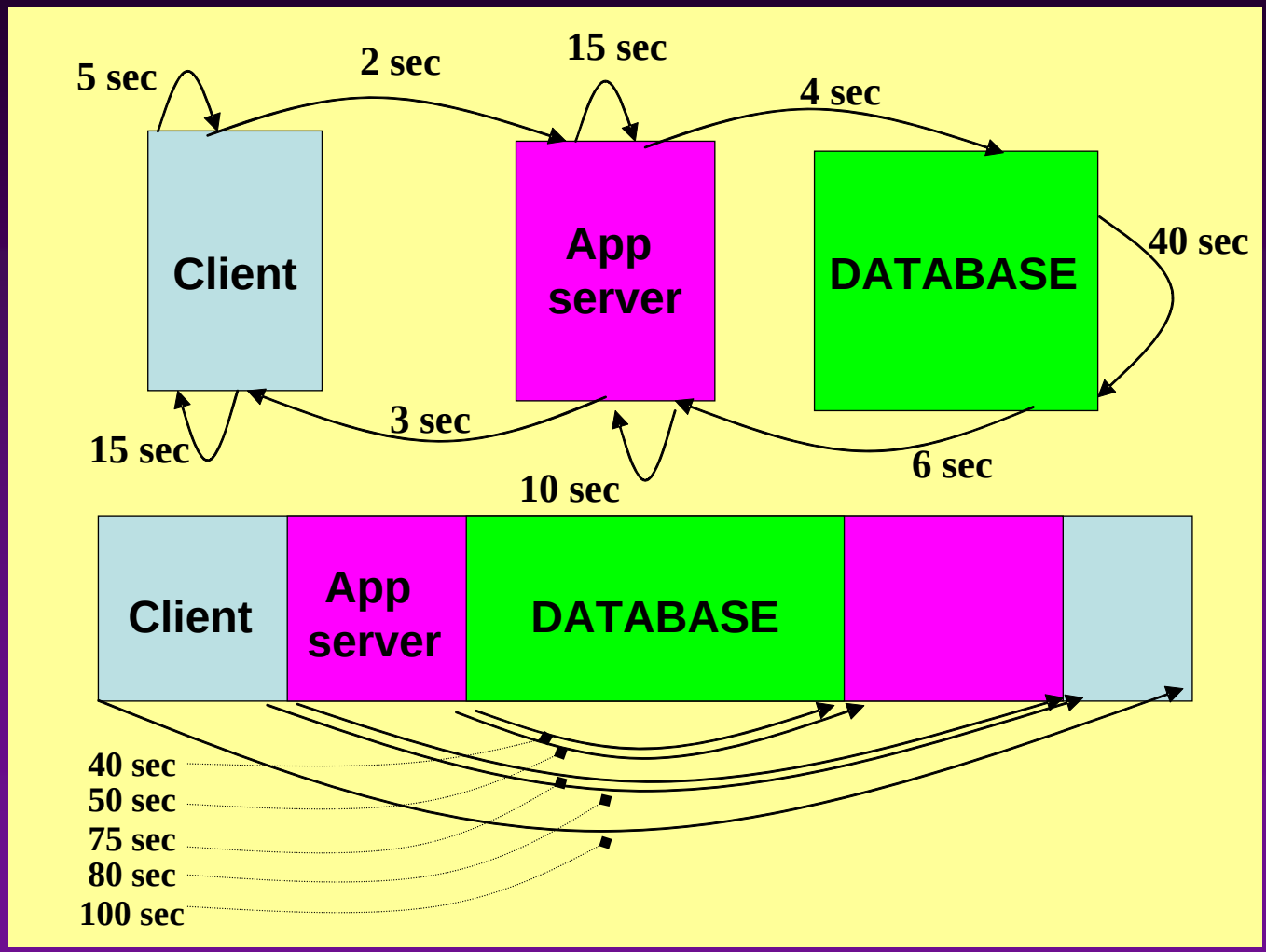


Timing Language Elements

- *Command: A*tomic part of the process (any command on any tier)
- *Step:* Complete processing cycle in one direction (always one-way)
  - Can either be a communication step between one tier and another, or a set of steps within the same tier.
  - Step consists of a number of commands.
- *Request:* Action consisting of a number of steps. A request is passed between different processing tiers.
- *Round-trip:* Complete cycle from the moment the request leaves the tier to the point when it comes back with some response information.

System Tuning for 3-tier Application (with numbers!)

9-step or 5 round-trip structure



Actions in 5 Round-Trip Structure

Client Level

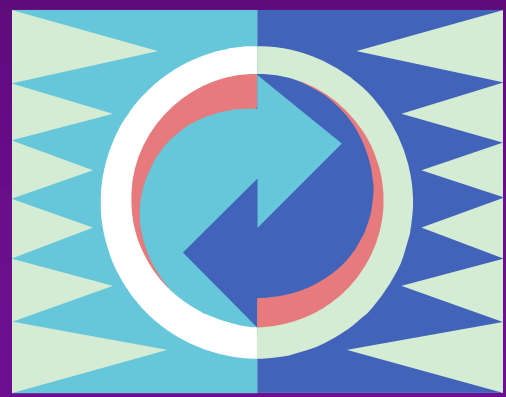
- 1. From request initiation to end of processing
  - User clicks button
  - Response is displayed
- 2. From request to application server to response receipt
  - Start of servlet call
  - End of servlet call

Application Level

- 3. From request acceptance to moment it is sent back
  - Start of processing in servlet
  - End of processing in servlet
- 4. From request sent to database
  - Start of JDBC call
  - End of JDBC call

Database Level

- 5. From request acceptance to sending back the response
  - Start block
  - End of block



Review

Topics Covered

1. Steps in web application process
2. Places where performance can suffer
3. Measuring performance

Still to discuss

1. SQL tuning
2. Application server / database communication tuning
3. Managing persistent layer

SQL Tuning: REMEMBER!!!

- 1. Use bind variables.
- 2. Use bind variables.
- 3. Use bind variables.
- 4. Use bind variables.
- 5. Use bind variables.
- 6. Use bind variables.
- 7. Use bind variables.

© Tom Kyte

- 1. Don’t build SQL in JAVA.
- 2. Don’t build SQL in JAVA.
- 3. Don’t build SQL in JAVA.
- 4. Don’t build SQL in JAVA.
- 5. Don’t build SQL in JAVA.
- 6. Don’t build SQL in JAVA.
- 7. Don’t build SQL in JAVA.

© M. Rosenblum

Simple Case

- The problem:
  - Value lists are explicitly hard-coded across the system
    - Difficult to determine what exactly is used
    - Hard to maintain
    - Data-dependent (cannot be cached)
- The solution – single point of tuning!
  - Universal Value List Builder

Universal Value List (1)

- Specify exactly what is needed as output and declare the corresponding collection:

```
Create type lov_oty is object
  (id_nr NUMBER,
    display_tx VARCHAR2(256));

Create type lov_nt
              as table of lov_oty;
```

Universal Value List (2)

- Write a PL/SQL function to hide all required logic:

```
function f_getLov_nt
 (i_table_tx,i_id_tx,i_display_tx,i_order_tx)
return lov_nt is
  v_out_nt lov_nt := lov_nt();
begin
  execute immediate
    'select lov_oty('
          ||i_id_tx||','||i_display_tx||
                    ')'||
    ' from '||i_table_tx||
    ' order by '||i_order_tx
  bulk collect into v_out_nt;
  return v_out_nt;
end;
```

Universal Value List (3)

- Test SQL statement with the following code:

```
select id_nr, display_tx
from table(
        cast(f_getLov_nt
               (:1, -- 'emp'
                :2, -- 'empno'
                :3, --'ename||''-''||job'
                :4  -- 'ename‘
               )
             as lov_nt)
            )
```

DULCIAN INC.

Complex Case

- The problem:
  - Users upload CSV-files
    - Name of file defines type
    - Column headers map directly to table columns
    - One row of file could mean multiple inserts
- Wrong solution
  - Parse file in the middle-tier and build inserts
- Right solution:
  - Load file to the database as CLOB
  - Build all inserts in the database

Build Inserts

```
Declare
  type integer_tt is table of integer;
  v_cur_tt integer_tt;
Begin
for r in v_groupRow_tt.first..v_groupRow_tt.last loop
  v_cur_tt(r):=DBMS_SQL.OPEN_CURSOR;
  for c in c_cols(v_mapRows_tt(r)) loop
    for i in v_header_tt.first..v_header_tt.last loop
      if v_header_tt(i).text=c.name_tx then
        v_col_tt(i):=c;
        v_col_tx:=v_col_tx||','||v_col_tt(i).viewcol_tx;
        v_val_tx:=v_val_tx||',:'||v_col_tt(i).viewcol_tx;
      end if;
    end loop;
  end loop;
  v_sql_tx:='insert into '||v_map_rec.view_tx||
     '('||v_col_tx||') values('||v_value_tx||')';
  DBMS_SQL.PARSE(v_cur_tt(r),v_sql_tx,DBMS_SQL.NATIVE);
end loop;
```

Process Data

```
for i in 2..v_row_tt.count
loop
  for r in
  v_groupRow_tt.first..v_groupRow_tt.last
  loop
    for c in v_col_tt.first..v_col_tt.last
    loop
      if v_col_tt(c).id  = v_mapRows_tt(r) then
        DBMS_SQL.BIND_VARIABLE(v_cur_tt(r),
         ':'||v_col_tt(c).viewcol_tx,
         v_data_tt(c).text);
      end if;
    end loop;
    v_nr:=dbms_sql.execute(v_cur_tt(r));
  end loop;
end loop;
```

Application Server / Database

- Critical success factor – managing database sessions:
  - Almost impossible to have one session per connection
  - Cost of opening/closing sessions is high
- Opportunity:
  - Total number of physical sessions at any point in time is fairly small.
- Good idea:
  - Create connection pool with a fixed number of connections (using Autoextend option)
  - Serve them to incoming requests as needed
- Problems:
  - A single physical session can serve requests from different logical sessions at different points in time.
  - Cannot trust ANYTHING defined at the session level.

Connection Pooling (1)

- Packaged variables cleanup

```
begin
  dbms_session.reset_package;
  dbms_session.free_unused_user_memory;
end;
```

Connection Pooling (2)

- Temporary tables cleanup

```
procedure p_truncate is
   v_exist_yn varchar2(1);
Begin
  select 'Y' into v_exist_yn
  from v$session s, v$tempseg_usage u
  where s.audsid = SYS_CONTEXT('USERENV','SESSIONID')
  and   s.saddr = u.session_addr
  and   u.segtype = 'DATA'
  and rownum = 1;
  for c in (select table_name from user_table
            where temporary = 'Y‘
            and duration = 'SYS$SESSION') loop
    execute immediate 'truncate table '||c.table_name;
  end loop;
end;
```

Managing Persistent Layer

- Client/Server
  - Temporary table with supporting information (one row per session)
  - Read - from support area.
  - Write – via the engine:
    - Get action from the application
    - Modify support area
    - Send response to the application
- Reason
  - Eliminates about 75% of repeated requests

- Web - idea
  - Create persistent table
    - Add session ID
  - Estimate system could slow down 3-5%
- Web - real life
  - 50%-200% slower (only at peak times)
  - Workload limit after which the whole system started to fall apart

Why is performance affected? (1)

- Database running in ARCHIVELOG
  - All DML against SUPPORT table recorded
  - Filled up about 85% of all logs!
- All support changes must be persistent.
  - Extra COMMITS occurred
  - LOG FILE SYNC wait event count skyrocketed
- Table had primary key (ID from a sequence)
  - Due to DML activity from hundreds of sessions, every 15 minutes, the database logged a deadlock
  - Very high contention on some index blocks

Why is performance affected? (2)

- Cumulative heavy I/O load
  - Individual requests take more time.
  - Sessions were not released from connection pool fast enough.
  - Total number of simultaneous sessions is 4 times more than estimated.
- Each session used more memory, more temporary segments, etc.
  - Slowed down the system even more
  - Especially true for I/O operations (since there were more simultaneous requests).
  - Quickly spirals into a slow-down and eventual stoppage of the system

Why is performance affected? (3)

- Database resources quickly became over-utilized just by making a table persistent with a session key.
- Two core issues:
  - 1. How to decrease I/O?
  - 2. How to resolve index contention?

DULCIAN INC.

Solution

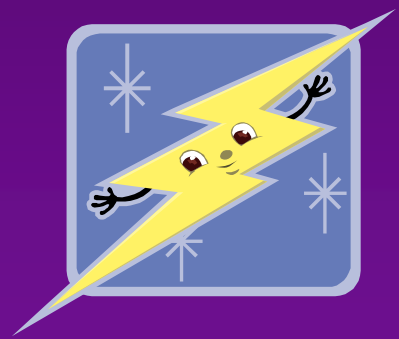
- Create a separate database instance
  - New instance runs in NOARCHIVELOG mode
  - New instance has only one schema.
  - That schema contains only one table: SUPPORT INFO
  - SUPPORT INFO table is hash-partitioned by session ID (1024 partitions)
  - All indexes are local.
- Main schema has a database link and synonym
  - Everything appears as though nothing has changed.
  - All requests to the support table must include session ID (to use local indexes).
  - Some rewrite was required to enforce this rule.

Result

- System ran as fast as originally predicted
  - Extra waits caused by data cases via DBLink were negligible (less than 0.01/request - average of 3000 requests/hour).
  - No time lost writing logs
  - Less I/O → less sessions → less resources used → less waits → faster response → less sessions …
- Using a large number of partitions, less chances of creating a “hot block”, since all indexes were local.
- Lessons learned:
  - In the Oracle environment, everything is linked together.
  - Any changes can lead to a “domino effect”

Conclusions

- Keep all nine of the potential areas for encountering performance problems in mind.
- Investigate each one carefully to discover ways in which performance can be improved.
- It is not just the database.



Share your Knowledge: Call for Articles/Presentations

- Submit articles, questions, … to

IOUG – The SELECT Journal

select@ioug.org

Reviewers wanted

ODTUG – Technical Journal

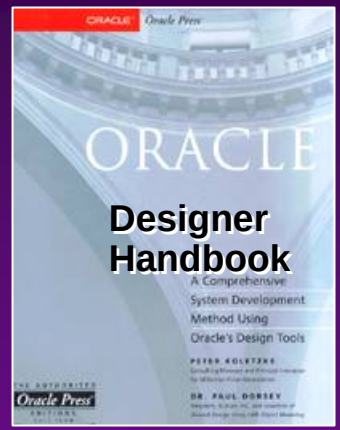
pubs@odtug.com

Dulcian's BRIM® Environment

- Full business rules-based development environment
- Includes FREE license for BRIM Web 3.0
- For Demo
  - Write "BRIM" on business card

Contact Information

- Dr. Paul Dorsey – paul_dorsey@dulcian.com
- Michael Rosenblum – mrosenblum@dulcian.com
- Dulcian website - www.dulcian.com



Latest book:
Oracle PL/SQL for Dummies