Best Practices for Developing Performant Applications
Luxi Chidambaran
Oracle
The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.
Overview

- Application Lifecycle
- Best Practices for Database Access Performance
- Performance Tools
- Use Cases
- Conclusion
Application Lifecycle

- Requirements
- Design
- Code
- Testing
- Deployment
- Production
- Maintenance

WHAT ABOUT PERFORMANCE?
Oops! What about Performance?

- Performance issues found once customer deploys
  - Often causes reactive fire-fighting
- Performance needs to be baked into the development process
Application Lifecycle

Production
- Performance Monitoring?

Deployment
- Deployment Guidelines for best Performance?

Testing
- Performance/Scalability Testing?

Code
- Are Performance Best Practices conformed to?

Design
- Does Design account for key Performance objectives?

Requirements
- Performance Requirements?

Maintenance
- Performance Regressions?
Focus of this presentation

• Not about SQL tuning
• Not about Oracle Database instance tuning
• It is about writing efficient Database access code in your application
• It is also about some of the performance tools that can help
Best Practices for Database Access Performance
Database Access Performance: Basic Rules

• Avoid repeated creation/destruction application/database-driver interface objects. Eliminates
  • Unnecessary code path
  • Excessive garbage collection

• Minimize contention for shared objects such as:
  • Persistent structures in the database
  • Resources (e.g. connections)

• Follow Database API Best Practices
  • Top Two out of “Top Ten Mistakes Found In Oracle Systems”:
    • Bad Connection Management
    • Bad Use of Cursors and the Shared Pool
Recurring Design Patterns

- Patterns in terms of Best Practices with respect to using Database APIs
- Enable an application developer to consistently apply established practices
- Enable a performance engineer to advise an application developer with respect to potential application changes
Follow Database API usage Best Practices

- Connection Pooling
- Bind Variables
- Statement Caching
- Turn off Auto Commits
- Reducing Roundtrips
  - Array DML
  - Array Fetching and Prefetching
  - PL/SQL and Java stored procedures
- Result Caching
- Secure Files
Performance Tools
AWR and ADDM

- Built into the Oracle Database 10g
- Automatic Workload Repository (AWR)
  - Evolution of statspack
  - Built-in repository
  - Captures performance statistics at regular intervals
- Automatic Database Diagnostic Monitor (ADDM)
  - Methodically analyses captured AWR stats
  - Generates recommendations
- Require Diagnostic Pack which requires a separate license
- Reports provide invaluable input for diagnosing performance issues
Getting ADDM/AWR Reports

- Create an AWR Snapshot
  ```sql
  BEGIN
  DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT ();
  END;
  ```
- Run your workload
- Create a second AWR Snapshot
  ```sql
  BEGIN
  DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT ();
  END;
  ```
- Generate reports
  ```sql
  @$ORACLE_HOME/rdbms/admin/addmrpt.sql
  @$ORACLE_HOME/rdbms/admin/awrrpt.sql
  ```
Use cases
Don’t create a storm
What is this AWR report stating?

<table>
<thead>
<tr>
<th></th>
<th>Per Second</th>
<th>Per Transaction</th>
<th>Per Exec</th>
<th>Per Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB Time(s):</td>
<td>11.9</td>
<td>84.0</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>DB CPU(s):</td>
<td>1.4</td>
<td>9.8</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Redo size:</td>
<td>103,299.0</td>
<td>732,704.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical reads:</td>
<td>2,321.0</td>
<td>16,462.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block changes:</td>
<td>633.0</td>
<td>4,490.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Physical reads:</td>
<td>0.2</td>
<td>1.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Physical writes:</td>
<td>9.3</td>
<td>66.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User calls:</td>
<td>1,429.5</td>
<td>10,139.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parses:</td>
<td>1,125.8</td>
<td>7,985.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard parses:</td>
<td>102.4</td>
<td>726.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W/A MB processed:</td>
<td>123,899.1</td>
<td>878,821.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logons:</td>
<td>51.1</td>
<td>362.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Executes:</td>
<td>1,077.9</td>
<td>7,645.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rollbacks:</td>
<td>0.0</td>
<td>0.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transactions:</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Time Model Statistics

- Total time in database user-calls (DB Time): 9748.2s
- Statistics including the word "background" measure background process
- Ordered by % of DB time desc, Statistic name

<table>
<thead>
<tr>
<th>Statistic Name</th>
<th>Time (s)</th>
<th>% of DB Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>connection management call elapsed time</td>
<td>7,892.78</td>
<td>80.97</td>
</tr>
<tr>
<td>parse time elapsed</td>
<td>3,951.02</td>
<td>40.53</td>
</tr>
<tr>
<td>hard parse elapsed time</td>
<td>1,195.05</td>
<td>12.26</td>
</tr>
<tr>
<td>DB CPU</td>
<td>1,138.28</td>
<td>11.68</td>
</tr>
<tr>
<td>sql execute elapsed time</td>
<td>985.46</td>
<td>10.11</td>
</tr>
<tr>
<td>repeated bind elapsed time</td>
<td>0.35</td>
<td>0.00</td>
</tr>
<tr>
<td>PL/SQL execution elapsed time</td>
<td>0.33</td>
<td>0.00</td>
</tr>
<tr>
<td>sequence load elapsed time</td>
<td>0.21</td>
<td>0.00</td>
</tr>
<tr>
<td>PL/SQL compilation elapsed time</td>
<td>0.08</td>
<td>0.00</td>
</tr>
<tr>
<td>hard parse (sharing criteria) elapsed time</td>
<td>0.01</td>
<td>0.00</td>
</tr>
<tr>
<td>hard parse (bind mismatch) elapsed time</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>DB time</td>
<td>9,748.21</td>
<td></td>
</tr>
<tr>
<td>background elapsed time</td>
<td>59.16</td>
<td></td>
</tr>
<tr>
<td>background cpu time</td>
<td>17.07</td>
<td></td>
</tr>
</tbody>
</table>
Finding 3: Session Connect and Disconnect
Impact is 9.59 active sessions, 80.97% of total activity.
Session connect and disconnect calls were consuming significant database time.

Recommendation 1: Application Analysis
Estimated benefit is 9.59 active sessions, 80.97% of total activity.

Action
Investigate application logic for possible reduction of connect and disconnect calls. For example, you might use a connection pool scheme in the middle tier.
Excessive Connection Activity

- Database Connections expensive to create
  - Spawn O/S process, network connection, several roundtrips
  - Associated database authentication and session creation
- Database Connections are expensive to tear down!
- Repeatedly Connecting/Disconnecting can be a huge scaling issue
- Turn on Connection Pooling
Java Implicit Connection Cache

Main Thread:

```java
// Set DataSource properties
ods.setURL(url);
ods.setUser(user);
ods.setPassword(password);
ods.setConnectionCachingEnabled(true);
ods.setConnectionCacheName("CPOOL");

Properties prop = new Properties();
prop.setProperty("MaxLimit", "+THREAD_COUNT");
prop.setProperty("MinLimit", "+THREAD_COUNT");
prop.setProperty("InitialLimit", "+THREAD_COUNT");
ods.setConnectionCacheProperties(prop);
```

Thread:

```java
// Obtain a connection
connection = dataSource.getConnection();
// run the workload
doWork(connection);

// close the connection when done
connection.close();
```
OCI Session Pool

Main Thread:
/* create a homogeneous session pool */
OCISessionPoolCreate(envhp, errhp, 
spoolhp, /* session pool handle */ 
(OraText **) poolName, poolNameLenp, 
(const OraText *) connstr, strlen(connstr), 
min, max, increment,/* pool size constraints */ 
(OraText *) "hr", strlen((char *) "hr"), 
(OraText *) apppassword, 
strlen((char *) apppassword), 
OCI_SPC_HOMOGENEOUS);

Thread:
OCISessionGet(envhp, errhp, 
&svchp, /* returned database connection */ 
authhp, /* initialized authentication handle */ 
/* connect pool name */ 
(OraText *) poolName, poolNameLen, 
/* session tagging parameters: optional */ 
NULL, 0, NULL, NULL, NULL, 
OCI_SESSGET_SPOOL);

do_workload(svchp, errhp, ptr);
OCISessionRelease(svchp, errhp, NULL, 0, OCI_DEFAULT);
Database Resident Connection Pool (DRCP) in Oracle Database 11g

- Fallback when there is no application tier connection pooling
- Also useful for sharing connections across middle tier hosts
- Supported only for OCI and PHP applications
- Scales to tens of thousands of database connections even on a commodity box
- Enable with `dbms_connection_pool.start_pool`
- Connect String
  - Easy Connect: `//localhost:1521/oowlab:POOLED`
  - TNS Connect String: `(SERVER=POOLED)`
Parsing is hard
What is the related AWR report stating?

<table>
<thead>
<tr>
<th>Load Profile</th>
<th>Per Second</th>
<th>Per Transaction</th>
<th>Per Exec</th>
<th>Per Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB Time(s):</td>
<td>10.4</td>
<td>43.1</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>DB CPU(s):</td>
<td>1.7</td>
<td>6.9</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Redo size:</td>
<td>11,793.8</td>
<td>49,001.1</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Logical reads:</td>
<td>6,588.8</td>
<td>27,375.1</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Block changes:</td>
<td>30.7</td>
<td>127.4</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Physical reads:</td>
<td>444.9</td>
<td>1,848.6</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Physical writes:</td>
<td>28.6</td>
<td>118.9</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>User calls:</td>
<td>11,032.4</td>
<td>45,837.5</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Parses:</td>
<td>5,988.3</td>
<td>24,880.3</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Hard parses:</td>
<td>920.2</td>
<td>3,823.4</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>W/A MB processed:</td>
<td>279,318.6</td>
<td>1,180,517.8</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Logons:</td>
<td>0.4</td>
<td>1.6</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Executes:</td>
<td>6,003.7</td>
<td>24,944.3</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Rollbacks:</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Transactions:</td>
<td>0.2</td>
<td>0.0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>
More from the same AWR report

**Time Model Statistics**

- Total time in database user-calls (DB Time): 949s
- Statistics including the word "background" measure background processes
- Ordered by % or DB time desc, Statistic name

<table>
<thead>
<tr>
<th>Statistic Name</th>
<th>Time (s)</th>
<th>% of DB Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>parse time elapsed</td>
<td>869.36</td>
<td>90.55</td>
</tr>
<tr>
<td>hard parse elapsed time</td>
<td>758.85</td>
<td>79.96</td>
</tr>
<tr>
<td>DB CPU</td>
<td>150.92</td>
<td>15.90</td>
</tr>
<tr>
<td>sql execute elapsed time</td>
<td>50.81</td>
<td>5.35</td>
</tr>
<tr>
<td>connection management call elapsed time</td>
<td>0.13</td>
<td>0.01</td>
</tr>
<tr>
<td>hard parse (sharing criteria) elapsed time</td>
<td>0.10</td>
<td>0.01</td>
</tr>
<tr>
<td>PL/SQL execution elapsed time</td>
<td>0.06</td>
<td>0.01</td>
</tr>
<tr>
<td>PL/SQL compilation elapsed time</td>
<td>0.02</td>
<td>0.00</td>
</tr>
<tr>
<td>repeated bind elapsed time</td>
<td>0.01</td>
<td>0.00</td>
</tr>
<tr>
<td>sequence load elapsed time</td>
<td>0.01</td>
<td>0.00</td>
</tr>
<tr>
<td>hard parse (bind mismatch) elapsed time</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>DB time</td>
<td>949.03</td>
<td></td>
</tr>
<tr>
<td>background elapsed time</td>
<td>1.76</td>
<td></td>
</tr>
<tr>
<td>background cpu time</td>
<td>0.15</td>
<td></td>
</tr>
</tbody>
</table>
Finding 2: Hard Parse Due to Literal Usage
Impact is 8.32 active sessions, 79.74% of total activity.

SQL statements were not shared due to the usage of literals. This resulted in additional hard parses which were consuming significant database time.

Recommendation 1: Application Analysis
Estimated benefit is 8.32 active sessions, 79.74% of total activity.

Action
Investigate application logic for possible use of bind variables instead of literals.

Action
Alternatively, you may set the parameter "cursor sharing" to "force".

Rationale
At least 39 SQL statements with FORCE_MATCHING_SIGNATURE 5551823750033335619 and PLAN_HASH_VALUE 1833546154 were found to be using literals. Look in V$SQL for examples of such SQL statements.
Lack of Bind Variables

- Hard Parse is expensive
  - Creates shared cursor in SGA
  - Causes library cache latch contention
  - Causes shared pool contention
  - Causes scalability issues

- Use Bind Variables
  - Reduces hard parses on the server
  - Reduces risk of SQL Injection: potential security issue
Bind Variables in Java

• Instead of:

```java
String query = "SELECT EMPLOYEE_ID, LAST_NAME, SALARY FROM "
    +"EMPLOYEES WHERE EMPLOYEE_ID = "
    + generateNumber(MIN_EMPLOYEE_ID, MAX_EMPLOYEE_ID);
pstmt = connection.prepareStatement(query);
rs = pstmt.executeQuery();
```

• Change to:

```java
String query = "SELECT EMPLOYEE_ID, LAST_NAME, SALARY FROM "
    +"EMPLOYEES WHERE EMPLOYEE_ID = ?";
pstmt = connection.prepareStatement(query);
pstmt.setInt(1, n);
rs = pstmt.executeQuery();
```
Bind Variables in OCI

static char *MY_SELECT = "select employee_id, last_name, salary from \nemployees where employee_id = :EMPNO";

OCIBind *bndp1;
OCIStrmt *stmthp;
ub4 emp_id;

OCIStrmtPrepare2 (svchp, &stmthp, /* returned stmt handle */
errhp, /* error handle */
  (const OraText *) MY_SELECT,
  strlen((char *) MY_SELECT),
  NULL, 0, /* tagging parameters:optional */
  OCI_NTV_SYNTAX, OCI_DEFAULT);

/* bind input parameters */
OCIBindByName (stmthp, &bndp1, errhp, (text *) ":EMPNO",
-1, &(emp_id), sizeof(emp_id), SQLT_INT,
  NULL, NULL, NULL, 0, NULL, OCI_DEFAULT);
Literal Replacement

• Fallback if application cannot be changed to use binds
• init.ora parameter
  • CURSOR_SHARING={FORCE | SIMILAR | EXACT}
  • Default is EXACT
Soft things can hurt
What is this AWR report stating?

<table>
<thead>
<tr>
<th>Load Profile</th>
<th>Per Second</th>
<th>Per Transaction</th>
<th>Per Exec</th>
<th>Per Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB Time(s)</td>
<td>10.4</td>
<td>43.1</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>DB CPU(s)</td>
<td>1.7</td>
<td>6.9</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Redo size</td>
<td>11,793.8</td>
<td>49,001.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical reads</td>
<td>6,588.8</td>
<td>27,375.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block changes</td>
<td>30.7</td>
<td>127.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Physical reads</td>
<td>444.9</td>
<td>1,848.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Physical writes</td>
<td>28.6</td>
<td>118.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User calls</td>
<td>11,832.4</td>
<td>45,937.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parses</td>
<td>5,988.3</td>
<td>24,880.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard parses</td>
<td>920.2</td>
<td>3,823.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W/A MB processed</td>
<td>279,318.6</td>
<td>1,160,517.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logons</td>
<td>0.4</td>
<td>1.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Executes</td>
<td>6,003.7</td>
<td>24,944.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rollbacks</td>
<td>0.0</td>
<td>0.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transactions</td>
<td>0.2</td>
<td>0.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Finding 3: Soft Parse
Impact is 1.1 active sessions, 10.59% of total activity.

Soft parsing of SQL statements was consuming significant database time.

Recommendation 1: Application Analysis
Estimated benefit is 1.1 active sessions, 10.59% of total activity.

Action
Investigate application logic to keep open the frequently used cursors. Note that cursors are closed by both cursor close calls and session disconnects.
Excessive Soft Parsing: Lack of Statement Caching

- **Soft Parsing**
  - Session executes a statement that exists in shared pool
  - Creates session specific cursor context
  - Repeats metadata processing

- **Use Statement Caching**
  - Keeps frequently used session cursors open
  - Reduces soft parses on the Server
    - Not only faster but more scalable
  - Cuts repeated metadata processing
  - Consumes less network bandwidth
  - Cuts code path in driver/application tier
Statement Caching in Java

// Obtain a connection
connection = dataSource.getConnection();

// Enable statement caching
((OracleConnection) connection).setStatementCacheSize(20);
((OracleConnection) connection).setImplicitCachingEnabled(true);
Statement Caching in OCI

• Initialize the OCI Session Pool with statement cache
  
  ```
  ub4       stmt_cachesize = 20;
  /* set the statement cache size for all sessions in the pool */
  OCIAttrSet(spoolhp, OCI_HTYPE_SPOOL, &stmt_cachesize, 0,
             OCI_ATTR_SPOOL_STMTCACHESIZE, errhp);
  /* create a homogeneous session pool */
  OCISessionPoolCreate(envhp, errhp,
                        spoolhp, /* session pool handle */
                        . . ., OCI_SPC_HOMOGENEOUS|
                        OCI_SPC_STMTCACHE);
  /* modes */
  ```

• Use new flavors of prepare/release calls
  - OCIStmtPrepare2(), OCIStmtRelease()
Session Cached Cursors in the Database

- Fallback if you cannot change the application to use statement caching
- `session_cached.Cursors = X`
  - Defaults have changed in various releases
  - Oracle Database 11g Default = 50
Wrong Default
<table>
<thead>
<tr>
<th>Load Profile</th>
<th>Per Second</th>
<th>Per Transaction</th>
<th>Per Exec</th>
<th>Per Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB Time(s):</td>
<td>17.5</td>
<td>0.0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>DB CPU(s):</td>
<td>1.4</td>
<td>0.0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Redo size:</td>
<td>2,808,219.6</td>
<td>483.7</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Logical reads:</td>
<td>31,140.1</td>
<td>5.4</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Block changes:</td>
<td>23,285.0</td>
<td>4.0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Physical reads:</td>
<td>0.4</td>
<td>0.0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Physical writes:</td>
<td>78.8</td>
<td>0.0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>User calls:</td>
<td>6,974.3</td>
<td>1.2</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Parses:</td>
<td>9.2</td>
<td>0.0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Hard parses:</td>
<td>0.2</td>
<td>0.0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>W/A MB processed:</td>
<td>214,134.3</td>
<td>36.9</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Logons:</td>
<td>0.4</td>
<td>0.0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Executes:</td>
<td>6,976.7</td>
<td>1.2</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Rollbacks:</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Transactions:</td>
<td>5,806.3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
AWR Reports indicate excessive transaction activity

### Top 5 Timed Foreground Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Waits</th>
<th>Time(s)</th>
<th>Avg wait (ms)</th>
<th>% DB time</th>
<th>Wait Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>log file sync</td>
<td>432,341</td>
<td>1,145</td>
<td>3</td>
<td>90.54</td>
<td>Commit</td>
</tr>
<tr>
<td>DB CPU</td>
<td></td>
<td>98</td>
<td></td>
<td>7.72</td>
<td></td>
</tr>
<tr>
<td>buffer busy waits</td>
<td>26,834</td>
<td>15</td>
<td>1</td>
<td>1.15</td>
<td>Concurrency</td>
</tr>
<tr>
<td>latch: In memory undo latch</td>
<td>6,880</td>
<td>2</td>
<td>0</td>
<td>0.19</td>
<td>Concurrency</td>
</tr>
<tr>
<td>SQL*Net message to client</td>
<td>504,409</td>
<td>2</td>
<td>0</td>
<td>0.17</td>
<td>Network</td>
</tr>
</tbody>
</table>
Finding 2: Commits and Rollbacks
Impact is 15.69 active sessions, 90.54% of total activity.

Waits on event "log file sync" while performing COMMIT and ROLLBACK operations were consuming significant database time.

Recommendation 1: Application Analysis
Estimated benefit is 15.69 active sessions, 90.54% of total activity.

Action
Investigate application logic for possible reduction in the number of COMMIT operations by increasing the size of transactions.

Rationale
The application was performing 345218 transactions per minute with an average redo size of 483 bytes per transaction.
Auto Commits

- Beware. Many database drivers (e.g. JDBC) have auto commit on
  - Causes more transactions, log flushes
  - Increases response time
  - Breaks atomicity of the transactions
- Use driver specific knob to turn off auto commits
  - e.g. JDBC
    - `conn.setAutoCommit(false);`
**Bulk up and speed up**
Array DML/Fetching/Prefetching

- Use array operations instead of single row operations
  - Single row DMLs/fetches incur excessive roundtrips
- Check array size is large enough
- Some drivers support prefetching instead of array fetching
- Identify tuning candidates via Enterprise Manager or V$SQL or AWR sql statistics tables.
Array Fetch size from Enterprise Manager

SQL Details: 512j5d0v34f6k

```sql
select h.rptno, h.subject, b.lineno, b.comments, h.do_by_release, h.release_id from rpthead h, rptbody b
where h.rptno = b.rptno and h.utility_version in ('4.0', '4.5', '5.0', '5.5', '6.0') and h.product_id in (1990, 1991, 1992, 1993, 1994, 1995, 2059, 2535) and b.comments like '%CHG: Fix%>' order by h.rptno, b.lineno
```

Details

Select the plan hash value to see the details below. Plan Hash Value 2304123955

<table>
<thead>
<tr>
<th>Shared Cursors Statistics</th>
<th>Execution Statistics</th>
<th>Other Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Parses 1</td>
<td>Executions</td>
<td>Executions that Fetched all Rows (%) 100.00</td>
</tr>
<tr>
<td>Hard Parses 0</td>
<td>CPU Time (sec)</td>
<td></td>
</tr>
<tr>
<td>Child Cursors 1</td>
<td>20.11</td>
<td></td>
</tr>
<tr>
<td>Invalidation 0</td>
<td>20.11</td>
<td></td>
</tr>
<tr>
<td>Largest Cursor Size (KB)</td>
<td>37.48</td>
<td></td>
</tr>
<tr>
<td>All Cursor Size (KB)</td>
<td>37.48</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Rows 7.955</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fetches 796</td>
<td></td>
</tr>
</tbody>
</table>
Array Fetch size from V$SQL example

```sql
SQL> select sql_text, executions, fetches, rows_processed from V$SQL where sql_text like 'select city from locations';
```

<table>
<thead>
<tr>
<th>SQL_TEXT</th>
<th>EXECUTIONS</th>
<th>FETCHES</th>
<th>ROWS_PROCESSED</th>
</tr>
</thead>
<tbody>
<tr>
<td>select city from locations</td>
<td>8800</td>
<td>26400</td>
<td>202400</td>
</tr>
</tbody>
</table>

- Looking at V$SQL
  - ROWS_PROCESSED/EXECUTION = 23
  - Bump up client side prefetch or array-fetch to 24
  - Fetches all rows in one roundtrip (instead of three)

- V$SQL information can get aged out
  - Same statistics available via persistent AWR tables
  - `DBA_HIST_SQLSTAT, DBA_HIST_SQLTEXT`
Array Fetching in Java

String query = "SELECT EMPLOYEE_ID, LAST_NAME FROM EMPLOYEES "
   + " WHERE EMPLOYEE_ID > ? "
   + " ORDER BY EMPLOYEE_ID";

pst = connection.prepareStatement(query);
pst.setInt(1, generateNumber(MIN_EMPLOYEE_ID, MAX_EMPLOYEE_ID));
pst.setFetchSize(20);
rs = pstmt.executeQuery();
ResultSetMetaData rsmd = rs.getMetaData();
int columnCount = rsmd.getColumnCount();
while (rs.next()) {
    for(int i = 1; i <= columnCount; ++i)
        System.out.println(rsmd.getColumnName(i) +"[" +rsmd.getColumnTypeName(i) +"]": " +rs.getString(i));
}
#define ARRAY_SIZE 20 /* Fetch 20 row at a time */
/* Ensure that OCIDefine calls are done with big enough buffers */
while (!done)
{
    status = OCIStmtFetch(stmthp, errhp, ARRAY_SIZE,
                           OCI_FETCH_NEXT, OCI_DEFAULT);

    if ((status == OCI_SUCCESS) || (status == OCI_NO_DATA))
    {
        if (status == OCI_SUCCESS)
            rows = ARRAY_SIZE;  /* all rows asked for were obtained */
        else if (status == OCI_NO_DATA)
        {
            /* might have gotten fewer rows */
            OCIAttrGet(stmthp, OCI_HTYPE_STMT, &rows, (ub4 *) NULL,
                        OCI_ATTR_ROWS_FETCHED, errhp);
            done = TRUE;
        }
    }
    else
    /* handle error */
String dml = "UPDATE EMPLOYEES SET SALARY = ?"  
  + " WHERE EMPLOYEE_ID = ?";
psmt = connection.prepareStatement(dml);
((OraclePreparedStatement)pstmt).setExecuteBatch(UPDATE_COUNT);
for(int i = 0; i < UPDATE_COUNT; ++i)  
{  
  pstmt.setInt(1, generateNumber(MIN_SALARY, MAX_SALARY));
  pstmt.setInt(2, generateNumber(min, max));
  pstmt.executeUpdate();
  completedOp++;
}
Array DML in OCI

```c
static char *MY_DML = (char *) "update employees set salary = :sal \n    where employee_id = :id";

OCIBind *bndp1, *bndp2;
OCIStmt *stmthp;

OCIStmtPrepare2 (svchp, &stmthp, . . .);    /* returned stmt handle */

/* do parameter bindings */
OCIBindByPos (stmthp, &bndp1, errhp, 1,
            (void *) array_on_client,
            (sword) sizeof(array_element),
            SQLT_INT, NULL, NULL,
            NULL, 0, NULL, OCI_DEFAULT);
/* bind other parameters */

/* execute the statement and commit */
OCIStmtExecute (svchp, stmthp, errhp, update_num, 0,
            (OCISnapshot *) NULL, (OCISnapshot *) NULL,
            OCI_COMMIT_ON_SUCCESS);
```
Don’t be chatty
Stored Procedures

- Bundle multiple SQL statements in one call
  - Use anonymous blocks or stored procedures
  - Eliminates roundtrips to database
  - Eliminates moving data between database and client
- Can improve performance dramatically
- Monitor roundtrips and bytes transferred stats
  - High values may indicate optimization opportunities
## Network Stats in AWR

### Instance Activity Stats

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Total</th>
<th>per Second</th>
<th>per Trans</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMON posted for undo segment shrink</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>SQL*Net roundtrips to/from client</td>
<td>126,066</td>
<td>5,646.60</td>
<td>3.00</td>
</tr>
<tr>
<td>TBS Extension: files extended</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>TBS Extension: tasks created</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>TBS Extension: tasks executed</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>active txn count during cleanout</td>
<td>3,910</td>
<td>175.13</td>
<td>0.09</td>
</tr>
<tr>
<td>application wait time</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>auto extends on undo tablespace</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>background checkpoints completed</td>
<td>1</td>
<td>0.04</td>
<td>0.00</td>
</tr>
<tr>
<td>background checkpoints started</td>
<td>2</td>
<td>0.09</td>
<td>0.00</td>
</tr>
<tr>
<td>background timeouts</td>
<td>88</td>
<td>3.94</td>
<td>0.00</td>
</tr>
<tr>
<td>branch node splits</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>buffer is not pinned count</td>
<td>295,524</td>
<td>13,236.76</td>
<td>7.03</td>
</tr>
<tr>
<td>buffer is pinned count</td>
<td>596</td>
<td>26.70</td>
<td>0.01</td>
</tr>
<tr>
<td>bytes received via SQL*Net from client</td>
<td>13,343,923</td>
<td>597,685.34</td>
<td>317.64</td>
</tr>
<tr>
<td>bytes sent via SQL*Net to client</td>
<td>16,504,021</td>
<td>739,228.75</td>
<td>392.86</td>
</tr>
</tbody>
</table>
Cash in with Result Caching
Benefits

• Easy to Use Cache
  • Add /*+ result_cache */ hint in SQL

• Frees application developers from building custom caches

• Extends server-side result caching to client side memory
  • Leverages cheaper client-side memory
  • Each application has its working set cached locally

• Achieves better performance by eliminating server roundtrips

• Improves server scalability by saving server resources

• Transparently maintains cache consistency with server side changes
Identifying Candidate Queries for Client Result Caching from AWR

**SQL ordered by Elapsed Time**

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100.
- Total DB Time (s): 0.078
- Captured SQL account for 85.3% of Total

<table>
<thead>
<tr>
<th>Elapsed Time (s)</th>
<th>CPU Time (s)</th>
<th>Executions</th>
<th>Elap Per Exec (s)</th>
<th>% Total DB Time</th>
<th>SQL ID</th>
<th>SQL Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.210</td>
<td>318</td>
<td>790,370</td>
<td>0.00</td>
<td>20.75</td>
<td>0ab3wv02at56a</td>
<td>driver_wu think@stad02 (TNS V1-VG) select employee_id, last_name...</td>
</tr>
<tr>
<td>1.198</td>
<td>310</td>
<td>779,472</td>
<td>0.00</td>
<td>28.59</td>
<td>a0b00v02z6a</td>
<td>driver_wu think@stad02 (TNS V1-VG) select employee_id, first_name...</td>
</tr>
<tr>
<td>1.053</td>
<td>300</td>
<td>790,320</td>
<td>0.00</td>
<td>25.83</td>
<td>0a6b0y02a6a</td>
<td>driver_wu think@stad02 (TNS V1-VG) select employee_id, email from...</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>16.24</td>
<td>0.40</td>
<td></td>
<td>016c6b56b2c4a</td>
<td>spplus@stad02 (TNS V1-VG) REGU_dng_cronad_adaploy...</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>13.69</td>
<td>0.34</td>
<td></td>
<td>16m56g34nrvy</td>
<td>select s.name, owc_name from...</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2.72</td>
<td>0.07</td>
<td></td>
<td>bsc6324kauz</td>
<td>insert into wh$_bg_event_summary...</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2.37</td>
<td>0.01</td>
<td></td>
<td>1ms614kosez</td>
<td>insert into WRM$ _SERVICE_STAT...</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1.79</td>
<td>0.04</td>
<td></td>
<td>4zk34v23z96df</td>
<td>insert into wh$ _system_event...</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1.35</td>
<td>0.03</td>
<td></td>
<td>0bwi2mz0s3aa</td>
<td>insert into wh$ _enqueue_summary...</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1.26</td>
<td>0.03</td>
<td></td>
<td>0wmi1ves95v6c</td>
<td>insert into wh$ _rowcache_summary...</td>
</tr>
</tbody>
</table>
### Identifying Candidate Queries for Client Result Caching from AWR

#### SQL ordered by CPU Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total is the CPU Time divided into the Total CPU Time times 100
- Total CPU Time is 1,162
- Captured SQL account for 70.5% of Total

<table>
<thead>
<tr>
<th>CPU Time</th>
<th>Plan Cache</th>
<th>Executions</th>
<th>CPU per Exec</th>
<th>% Total</th>
<th>% Total DB Time</th>
<th>SQL Id</th>
<th>SQL Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>215</td>
<td>1,213</td>
<td>790,376</td>
<td>0.60</td>
<td>26.40</td>
<td>26.75</td>
<td>00020_vw0926x5n</td>
<td>driver_w.Bsd6@sd40 (TNS V1-V0)</td>
</tr>
<tr>
<td>310</td>
<td>1,186</td>
<td>779,472</td>
<td>0.60</td>
<td>26.01</td>
<td>26.59</td>
<td>01020_vw0926x5n</td>
<td>driver_w.Bsd6@sd40 (TNS V1-V0)</td>
</tr>
<tr>
<td>300</td>
<td>1,062</td>
<td>790,326</td>
<td>0.60</td>
<td>26.08</td>
<td>26.39</td>
<td>01020_vw0926x5n</td>
<td>driver_w.Bsd6@sd40 (TNS V1-V0)</td>
</tr>
<tr>
<td>1</td>
<td>183</td>
<td>790,307</td>
<td>0.60</td>
<td>0.09</td>
<td>0.40</td>
<td>18110_vw0926x5n</td>
<td>ORACLE comma.plug@sd40 (TNS V1-V0)</td>
</tr>
<tr>
<td>1</td>
<td>14</td>
<td>790,358</td>
<td>0.62</td>
<td>0.04</td>
<td>0.24</td>
<td>18110_vw0926x5n</td>
<td>ORACLE comma.plug@sd40 (TNS V1-V0)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>790,366</td>
<td>0.14</td>
<td>0.01</td>
<td>0.01</td>
<td>18110_vw0926x5n</td>
<td>ORACLE comma.plug@sd40 (TNS V1-V0)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>790,367</td>
<td>0.10</td>
<td>0.01</td>
<td>0.04</td>
<td>18110_vw0926x5n</td>
<td>ORACLE comma.plug@sd40 (TNS V1-V0)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>790,368</td>
<td>0.08</td>
<td>0.01</td>
<td>0.02</td>
<td>18110_vw0926x5n</td>
<td>ORACLE comma.plug@sd40 (TNS V1-V0)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>790,369</td>
<td>0.04</td>
<td>0.00</td>
<td>0.00</td>
<td>18110_vw0926x5n</td>
<td>ORACLE comma.plug@sd40 (TNS V1-V0)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>790,370</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>18110_vw0926x5n</td>
<td>ORACLE comma.plug@sd40 (TNS V1-V0)</td>
</tr>
</tbody>
</table>

---

**ORACLE**
Identifying Candidate Queries for Client Result Caching from AWR

- Identify top SELECT statements
  - BY CPU
  - BY Elapsed Time
- Pick queries
  - On tables that are not updated often
  - With result sets can fit in available client memory
- Ideal candidates for /*+ result_cache */ hint
- init.ora parameter
  - CLIENT_RESULT_CACHE_SIZE
Niles Benchmark Performance Improvements

DB CPU Reduction:
Up to 600%

Response Time:
Up to 15-22% Faster
Dealing with large objects?
Secure Files Best Practices

• Data API
  • LOBs are handled like LONG or LONG RAW columns
  • Recommended for small LOBs
  • No extra roundtrips
• LOB API
  • Recommended for offset based access
  • Use for large LOBs (MBs)
  • Extra roundtrips (pre 11g) to get data, length, chunk-size
  • Oracle Database 11g Improvements for LOBs
    • OCI does Zero Copy network transfer
      • 15-20% improvement in CPU usage for LOBs over 100k
    • LOB Prefetch eliminates some extra roundtrips
Set SDU_SIZE for Large Data Transfers

- Controls SQL*Net packet size
- Default is 8k starting with Oracle Database 11g
- Set it up to 32k (max supported) if application does
  - Large Result set array fetches
  - Large Array DML operations
  - Large PL/SQL IN/OUT bind transfers
  - LOBs
  - Needs to be set on both client and server
- Monitor network stats in AWR
Performance Monitoring
Monitoring Performance by specific criteria

- AWR and ADDM provide instance level statistics
- An instance might be running several application modules
- Need ability to diagnose performance issues for a specific application or module
## End-to-end monitoring attributes

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLIENT_IDENTIFIER</td>
<td>Specifies the &quot;real&quot; end user</td>
</tr>
<tr>
<td>SERVICE</td>
<td>Specifies an application (configured at deployment)</td>
</tr>
<tr>
<td>MODULE</td>
<td>Specifies a functional area of an application</td>
</tr>
<tr>
<td>ACTION</td>
<td>Specifies an application action within a MODULE</td>
</tr>
</tbody>
</table>
Coding End-to-end monitoring attributes

• Instrument code for setting the end-to-end attributes
• Enables Database to track statistics based on these attributes
• Lookup your driver documentation for more details
  • Settings piggybacked on Database calls in JDBC and OCI
  • In iAS environments: DMS (Dynamic Monitoring Service) overrides JDBC user supplied values, if configured
Benefits of End-to-end Monitoring

- Turn stats on/off dynamically based on end-to-end attributes
  - **DBMS_MONITOR**
    - DBMS_MONITOR.CLIENT_ID_STAT_ENABLE('john.doe');
    - DBMS_MONITOR.SERV_MOD_ACT_STAT_ENABLE( 'APPS1','PAYROLL');
- Statistics available in
  - **V$CLIENT_STATS**
  - **V$SERVICE_STATS**
  - **V$SERV_MOD_ACT_STATS**
  - **V$ACTIVE_SESSION_HISTORY**
- Helps answer real questions in production:
  - Why is application user john.doe experiencing poor response times?
  - Why is a particular application slow?
  - Why is a particular application module slow?
  - Why is a particular application action slow?
Summary

- Overview of Performance in the Application Lifecycle
- Recurring patterns in diverse stacks
- Performance tools: AWR and ADDM
- Use cases
  - How the tools help identify potential issues early in the development cycle
  - Best Practices for Database API usage
- End-to-end Performance Monitoring
White Paper

• Building High Performance Drivers for Oracle Database 11g: OCI Tips and Techniques
Related Sessions

Session ID: S298816
Session Title: Hands-on Lab: Best Practices for Developing Performant Applications with Oracle Database 11g
Track: Oracle Develop: Database
Venue: Marriott
Room: Golden Gate B1
Date: 2008-09-22
Start Time: 16:00

Session ID: S298768
Session Title: Best Practices for Deployment, Performance, and Diagnosability of Oracle Net Services
Track: Database
Room: Rm 104
Date: 2008-09-25
Start Time: 10:30
The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.