The Nine Lives of RAC and TAF

Mark Harrison
mh@pixar.com

client-side coding for:
- real application clusters
- transparent application failover

NoCOUG Fall Conference, Nov. 2009

About Me

- Not an Oracle Expert
- 80's: Started in Database World -- Applied Data Research
  - MS-DOS Expert!
- Worked in compilers, telecom
- 1997: Back to Databases
  - AsiaInfo: Chief Software Architect of China Internet
- 2001: Pixar Tech Lead
  - Oversee asset & data management, database, etc.
  - charter: find and retrieve all data over 50 year timeframe
  - "all the computers that don't have screens"

RAC + TAF =
Load balanced, (mostly) Transparent Application

- Automatic Migration and Reconnection on node failures
- Load Balancing, Connection Management (courtesy of RAC)
- Automatic Failover on queries
  - most queries don't need to do anything!
- Notifications on Connection, Transaction Errors
  - applications can easily respond to these notifications

Configuring TAF

- configure as database service or in tns
- service configuration overrides tns
- an entry can fail over to itself -- important for RAC!

```
(CONNECT_DATA =
  ...
  (FAILOVER_MODE =
    (BACKUP = tnsname)

    (TYPE = type)

    (METHOD = method)

    (RETRIES = x)
    (DELAY = y)
  )
)
```

fail over to this tns entry

session - select calls get an error
select - will automatically move select

basic - connect to backup on conn. failure
preconnect - connect to backup at conn. time

try reconnect x times,
with y second delay

TNS Example

```
TEMPLAR =
  (DESCRIPTION =
    (ADDRESS=(PROTOCOL=TCP)(HOST=trac101)(PORT=1521))
    (ADDRESS=(PROTOCOL=TCP)(HOST=trac102)(PORT=1521))
    (ADDRESS=(PROTOCOL=TCP)(HOST=trac103)(PORT=1521))
    (LOAD_BALANCE = yes)
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = templar)
      (FAILOVER_MODE =
        (BACKUP = templar)
        (METHOD = BASIC)
        (TYPE = SELECT)
        (RETRIES = 18)
        (DELAY = 1)
      )
    )
  )
```

tns entry fails over to itself.
that's OK, it's a RAC!

selects handled transparently

for RAC, use small delay with
more retries

Nine Application Cases to Consider

Idle Sessions

1. idle connections

Selecting

2. selecting from a table
3. selecting from dual
4. selecting from an xtable

Connecting

5. connecting to the database

Transactions

6. with periodic commits
7. mostly idle, periodic commits
8. don't check ("feeling lucky")
9. never commits

Four Ways to Code

No Extra Coding

1. idle connections
2. selecting from a table
3. selecting from dual
8. don't check ("feeling lucky")

Check Query Errors

4. selecting from an xtable

Check Connection Errors

5. connecting to the database

Check Transaction Errors

6. with periodic commits
7. mostly idle, periodic commits
9. never commits

1: idle connection

- Don't have to do anything.
- An idle connection (not executing query, DML, DDL) will fail over automatically on the next database interaction.

- [test: idle]

2,3: select from table, dual

- Don't have to do anything
- Must enable FAILOVER_MODE TYPE=SELECT
- When you fail over to the new node, your select will magically continue from where it left off.
- The magic:
  - OCI tells server:
    execute SQL statement #x as of SCN #y,
    skipping forword to row #z

- [tests: select, selectdual]

4. xtable - special case select, with error

- "XTable" -- special Oracle table type
- Maps Oracle internal data structures to table so that it can be inspected via select
- Specialized, more used for system tuning and troubleshooting than for applications
- xtable queries can't relocate, since they're looking at memory inside the instance
- Common hidden use: select from sys_context

- `test: [xtable]`

4. xtable - special case select, with error

```
try:
    curs.execute("""select sys_context('userenv','instance'),
                           sys_context('userenv','server_host')
                    from dual""")
    r=curs.fetchone()

except cx_Oracle.DatabaseError,e:
    # ORA-25401: can not continue fetches
    # ORA-25402: transaction must roll back
    # ORA-25408: can not safely replay call
    if e.message.code in [25401,25402,25408]:
        ###print 'ignoring(case 1):', e.message.message.strip()
        r=('unknown-instance','unknown-host')
    else:
        raise(e)
```

5. connecting

- Lots of things can go wrong while connecting
- (special case: hung connection)

- `testcase: [reconn]`

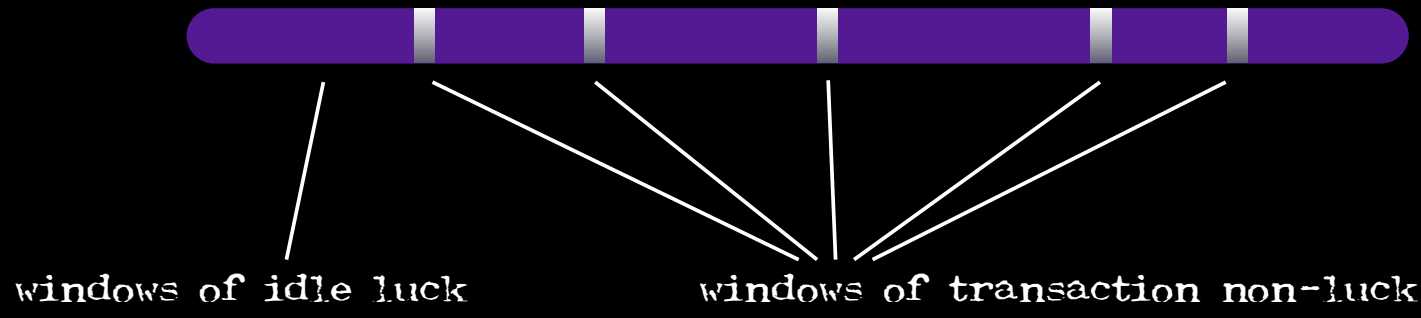
5. connecting

```
while True:
    try:
        conn=cx_Oracle.connect(*args,**args2)
    except cx_Oracle.DatabaseError,e:
        # ORA-01033: ORACLE initialization or shutdown in progress
        # ORA-12537: TNS:connection closed
        # ORA-12528: TNS:listener: all appropriate instances are
        #            blocking new connections
        # ORA-12521: TNS:listener does not currently know of instance
        #            requested in connect descriptor
        # ORA-12520: TNS:listener could not find available handler for
        #            requested type of server

        if e.message.code in [1033,12537,12528,12521,12520]:
            time.sleep(10)
            continue
            else:
                raise(e)
```

8. lucky transaction

- sometimes you get lucky, and your node downage will happen between transactions.
- in this case, either the select or idle cases apply, and you fail over without having to take any coding action
- "feeling lucky?" -- not best engineering approach?
  - but, luck comes at no extra cost over not using TAF!



- [test: luckytrans]

6,7,9: other transactions

- periodic - mostly in transaction, like many batch jobs
- short - mostly idle connection, like an interactive app
- open - never commits -- for testing transaction failover

- all handled the same, like luckytrans but with error checking
- all uncommitted DML is lost
- application needs to re-execute DML

6,7,9: other transactions

```
while True:
    try:
        #insert, update, etc
        conn.commit()
    except cx_Oracle.DatabaseError,e:
        # ORA-25401: can not continue fetches
        # ORA-25402: transaction must roll back
        # ORA-25408: can not safely replay call
        if e.message.code in [25401,25402,25408]:
            print 'TAF rollback, restarting transaction...'
            conn.rollback()
            redo insert, update, etc
            continue
        else:
            raise(e)
```

Timing out hung RAC connection attempts

- Sometimes connect() to the DB hangs
- IMHO, OCI runtime should handle this!
- Sadly it does not
- Causes:
  - TCP session interrupted
  - VIP not properly transferred from downered node to replacement node
  - evil space monkeys
  - ???

Timing out hung RAC connection attempts

- Method 1: via network switch
  - route client -> DB connections through L4 switch, have switch detect dead host and close client connection.
  - only for dead TCP circuit detection, NOT load balancing
  - good for inducing seizures in your DBA team!
- Method 2: time out client
  - when connecting:
    - set an interval timer for N seconds
    - set SIGALRM handler to raise a TimeOut exception
      - connect
      - cancel interval timer
    - catch Timeout exception, retry or fail

Timing out hung RAC connection attempts

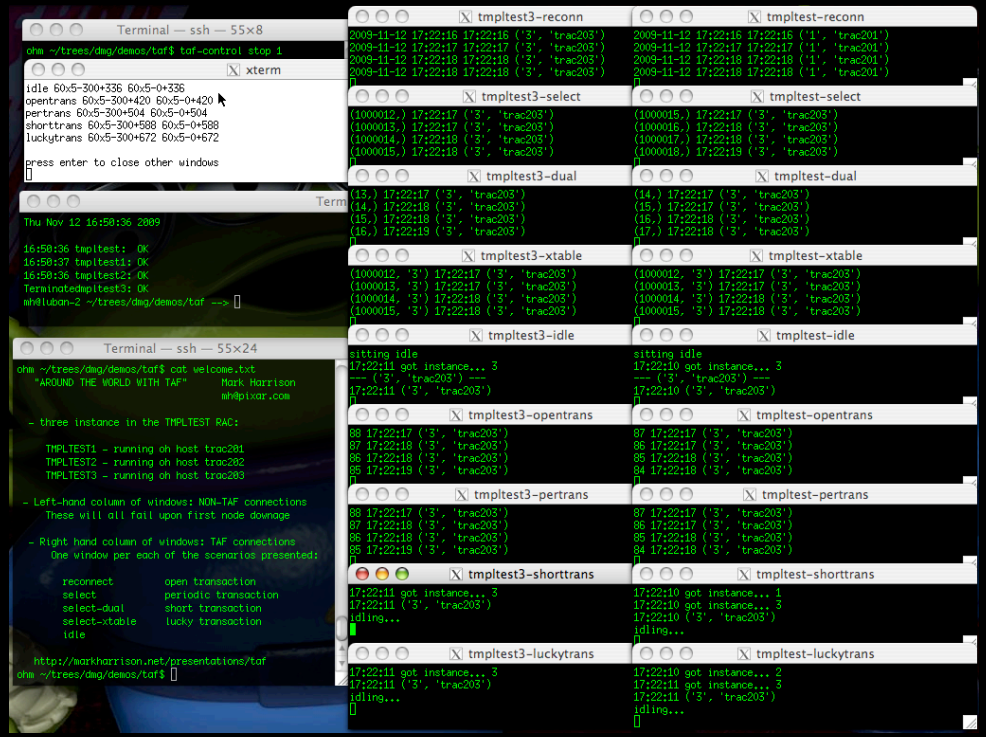
```
# Python example, should be similar logic in most languages

# boilerplate for raising Timeout exception
class TimeoutExc(Exception): pass
def alarmhandler(signame,frame): raise TimeoutExc()

while True:
    signal(SIGALRM, alarmhandler)
    signal.alarm(5)       # in 5 secs, raise exception TimeoutExc
    try:
        c=connect()       # if we connect,
        signal.alarm(0)   # cancel alarm
        break             # and continue normal processing
    except TimeoutExc:
        print 'timed out, retrying....'
```

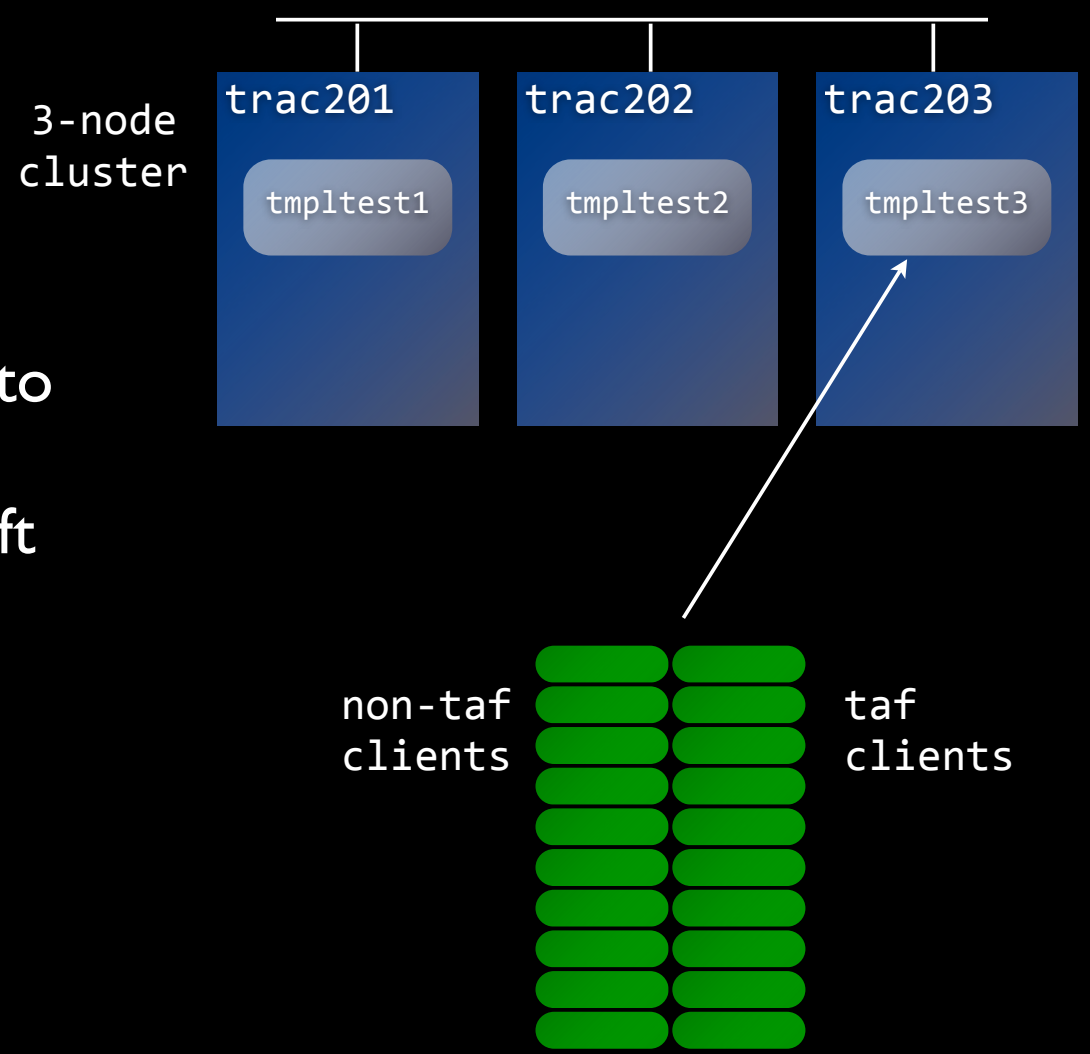
Pixar TAF Testbed

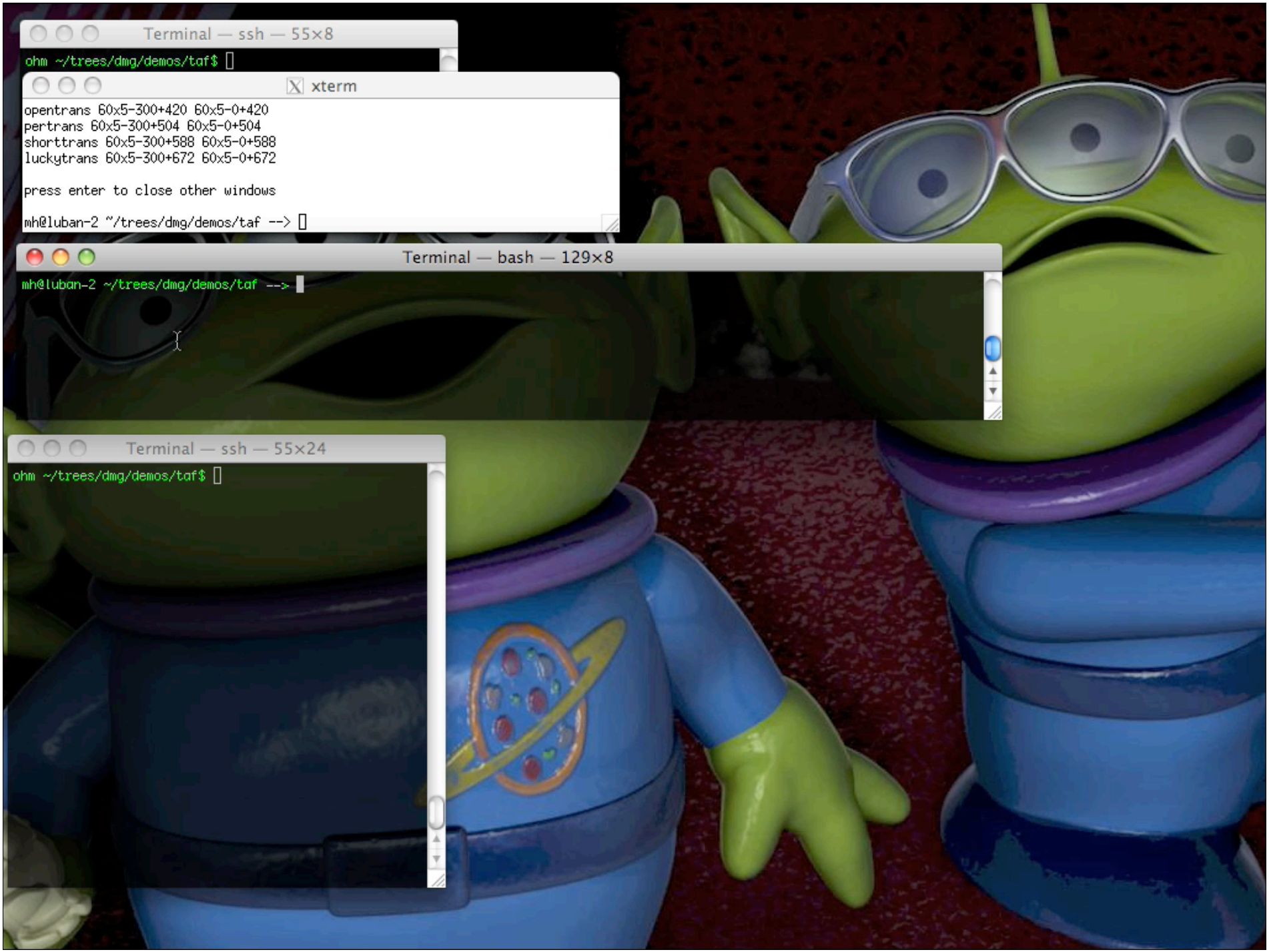
- open source, cross platform
- good testbed for
  - verifying your RAC setup
  - testing your TAF application code
- contains example code for all nine scenarios
- node control programs



Demo

- 10.2g RAC on linux
- clients on Mac
- start off connected to node 3
- non-taf clients on left
- taf clients on right
- move applications from node to node




Terminal — ssh — 55×8
ohm ~/trees/dmg/demos/taf$
xterm
opentrans 60x5-300+420 60x5-0+420
pertrans 60x5-300+504 60x5-0+504
shorttrans 60x5-300+588 60x5-0+588
luckytrans 60x5-300+672 60x5-0+672
press enter to close other windows
mh@luban-2 ~/trees/dmg/demos/taf -->
Terminal — bash — 129×8
mh@luban-2 ~/trees/dmg/demos/taf -->
Terminal — ssh — 55×24
ohm ~/trees/dmg/demos/taf$

Demo Takeaways

- Any users would not have notice being bounced from node to node
  - sometimes a 1-2 pause when a node goes down hard

- constant Reconnecting = Just Say No!
- queries -- all the benefits, none of the work!
- "xtables", cleverly disguised as sys_context
- transactions -- need to catch and reapply DML
  - easy for some applications, hard for others
- But notice, the Feeling Lucky app worked fine in all cases!

Summary

- If you use RAC, use TAF!!
- configure TAF either in TNS or Service Description
- TAF documentation not great
- Put TAF error checking in a lib for all your apps
- Most select functionality is free
- Relatively small coding changes for connect, transactions
- Exercise your RAC extensively, reboot and restart instances like madmen until you are 100% confident in your RAC
- 10G has an important RAC VIP patch -- apply it!

- Thanks to Rich Headrick, Terry Sutton, and Iggy Fernandez of Database Specialists for their great support in this effort.
- Thanks to Pixarians for their patience as we figured this out.

Resources

- Net Services Administrator Guide
- OCI Programmer's Guide
- JDBC Developer's Guide and Reference
- Data Provider for .NET Developer's Guide

- this presentation and TAF testbed:
  http://markharrison.net/taf

HELP WANTED!!

- Systems Programmer
- Data Management Group
- C++, Python, Linux, Mac
- Familiarity with Oracle +++

- http://careers.pixar.com
- job #406, Systems Software Engineer
- mention this talk
- cc: mh@pixar.com

<?/>