

# CEC 2007

## Postgres performance bootcamp for Oracle DBAs

Marcus.Heckel@Sun.com

Glenn.Fawcett@Sun.com

Strategic Applications Engineering

<http://blogs.sun.com/glennf>

<http://blogs.sun.com/mheckel>



**SHIFT**—OUR UNIVERSE. OUR WORLD. YOUR MOVE.

# Goal Statement

*Leverage Oracle experience with performance monitoring to help those beginning to use Postgres.*

# What are we going to cover?

- *Performance monitoring tools and techniques for Postgres at the DB and OS level.*
- *Some high-level tuning of Postgres will be covered, but the main goal is to get the data for analysis*
- *We are NOT going to cover*
  - > Postgres Administration*
  - > SQL Tuning*

# Overview

- DB performance statistics overview
- Basic performance metrics
- Resource Monitoring
  - > Memory, CPU, IO, Network
- Tools
  - > DTRACE, PGLogging, PGstatspack

# Basic performance questions

- **What is the throughput?**
  - > transactions/sec, orders/hr, ect..
- **What is the response time?**
  - > Orders took 100ms to process
- **How do we monitor consumption?**
  - > IO, CPU, Memory, and Network

# Oracle - DB statistics tables V\$

- Oracle uses “virtual” statistics tables “V\$” tables.
  - > There are 402 V\$ tables in 10gR2
  - > Load profile: txn/sec, LIO/sec, reads/sec
- Answer's questions like:
  - What SQL is consuming the most resources?
  - Which tables are doing the most IO?
  - What is the cache hit ratio?
  - What latches/locks are hot?
- Basis for Statspack / AWR report.

# Oracle Statspack / AWR reports

- 1<sup>st</sup> choice for Oracle performance statistics.
- Built on top of V\$ statistics tables
- Samples of the V\$ tables are taken at specific intervals and reports are generated from the data.
- Highlights of data
  - > Wait based info aggregated over all running sessions.
  - > Per SQL statement reports
    - Response time, CPU time, LIO/exec, reads/exec
  - > Locking/Latching breakdown
  - > Object based profiling... LIO, Physical IO, Locks, ect..

# Postgres - DB statistics tables

## “PG\_STAT\*”

- Postgres has the “PG\_STAT” and “PG\_STATIO” series of tables.
  - > Only 17 tables vs Oracle's 402 :)
  - > Mostly IO statistics of database objects.
  - > Can do Load profile: txn/sec, LIO/sec, reads/sec
  - > NO: per/SQL statistics, Locking, Memory usage
  - > No official “STATSPACK” equivalent report.
    - Roll-your-own

# PG\_STAT Collection

- Enabling collection of statistics
  - > stats\_start\_collector must be set at DB start time. On by default.
    - Extent to which stats are collected can be changed at anytime
  - > Other variables that influence extent of stat gathering
    - stats\_block\_level – default off
    - stats\_row\_level – default off
    - stats\_command\_string – default on

# PG Stat Views

- PG\_STAT\_DATABASE

- > # commits/rollbacks
- > # active server processes connected to DB
- > Total read and buffer hits

```
tpce=# select xact_commit,xact_rollback,blks_read,blks_hit
        from pg_stat_database where datname='tpce';
```

xact_commit	xact_rollback	blks_read	blks_hit
774721	3824	4491895	316191630

- PG\_STAT\_ACTIVITY

- > Current query, it's wait status, time query began execution, client address/port number.

# PG\_STAT\_VIEWS

- PG\_STAT\_[ALL,SYSTEM,USER]\_TABLES
  - > Sequential scans, IDX scans, live rows fetched by seq/idx scan, rows inserted/deleted/updated.
- PG\_STATIO\_\*
  - > # disk blocks read and buffer hits from tables, indexes, and sequences for system and user tables.
- Statistics Access Functions
  - > Yet another way to access postgres stats.

# PG\_STAT Views, What Am I Seeing?

- Counter behavior can be observed easily

- > Run `select count()` on a large table

```
tpce=# select count(*) from ONE_BIG_TABLE;
```

- > Query the `pg_statio_user_tables` while the query is running.

```
tpce=# select heap_blks_read, heap_blks_hit
       from pg_statio_user_tables
       where relname='trade';
```

heap_blks_read	heap_blks_hit
1800247	28136810
...	...
1800247	28136810

- > No update while the query is running (same as Oracle), but when it completes....

```
tpce=# select heap_blks_read, heap_blks_hit
       from pg_statio_user_tables
       where relname='trade';
```

heap_blks_read	heap_blks_hit
3088600	28699807

# Dtrace with Postgres!

- Currently there are 13 Dtrace probes in Postgres.
- Two main area's of focus
  - > Transaction control
    - Transaction start, commit, rollback
  - > Locking control
    - LockAcquire
    - LWLockAcquire, etc...
- DTRACE is the future of performance monitoring and Postgres is taking full advantage!

# Basic performance questions

- **What is the throughput?**
  - > transactions/sec, orders/hr, etc..
- **What is the response time?**
  - > Orders took 100ms to process
- **How do we monitor consumption?**
  - > IO, CPU, Memory, and Network

# Oracle - Measuring throughput

- Load Profile: found in “Statspack” or “AWR”.
  - > Transactions/sec, Logical reads/sec, ect...

Load Profile

~~~~~

|                      | Per Second   | Per Transaction |
|----------------------|--------------|-----------------|
|                      | -----        | -----           |
| Redo size:           | 206,183.11   | 4,179.61        |
| Logical reads:       | 228,047.93   | 4,622.84        |
| Block changes:       | 1,277.72     | 25.90           |
| Physical reads:      | 1,299.82     | 26.35           |
| Physical writes:     | 77.20        | 1.56            |
| User calls:          | 881.95       | 17.88           |
| Parses:              | 2,169.19     | 43.97           |
| Hard parses:         | 36.99        | 0.75            |
| Sorts:               | 215.60       | 4.37            |
| Logons:              | 0.29         | 0.01            |
| Executes:            | 7,941.87     | 160.99          |
| <b>Transactions:</b> | <b>49.33</b> |                 |



# Oracle - Measuring throughput

- V\$SYSSTAT table contains all sorts of statistics regarding database performance.
- To view the # of committed transactions:

```
SQL> select value from v$sysstat where  
name='user commits';
```

- This can be scripted sample at set intervals and compute the number of “commits/sec” or transaction throughput.
- Postgres uses a similar method as well.

# Measuring throughput - POSTGRES

- “pg\_stat\_database” table shows:
  - xact\_committed
  - xact\_rolled\_back
- > Use SQL to sample the table at a specific interval to compute the Transaction rate.

```
tpce=# select xact_commit
      from pg_stat_database
      where datname='tpce';
```

- > Simple script can be created to display throughput.

# Postgres - Measuring Throughput

- DTRACE to compute txn/sec.
  - > Can target all pids or specific session.
  - > ALL Statements result in a commit
    - SELECT, INSERT, UPDATE, DELETE are all included

```
#!/usr/sbin/dtrace -qs
BEGIN
{
  commits = 0;
}

postgresql*:::transaction-commit
{
  commits++;
}

tick-1sec
{
  printf("%d commits per second\n", commits);
  commits = 0;
}
```

```
root@bigleaf> tps.d
186 commits/sec
276 commits/sec
270 commits/sec
270 commits/sec
247 commits/sec
248 commits/sec
318 commits/sec
347 commits/sec
283 commits/sec
```

# Measuring Response Time - ORACLE

- SQL profile in AWR/Statspack
- “Some” math required (Elapsd/Executions)

| Buffer Gets | Executions | Gets per Exec | %Total | CPU Time (s) | Elapsd Time (s) | Hash Value |
|-------------|------------|---------------|--------|--------------|-----------------|------------|
| 81,351,001  | 29,670     | 2,741.9       | 39.6   | 2148.09      | 2109.75         | 601790983  |

SELECT \* FROM ARTE\_BRP\_ERROR ER WHERE ER.BRPERR\_LCREF = :B3 AND  
 ER.BRPERR\_MJINSREF = :B2 AND ER.BRPERR\_RELSTAGE <= :B1 AND ER.BR  
 PERR\_FLAG = 'L'

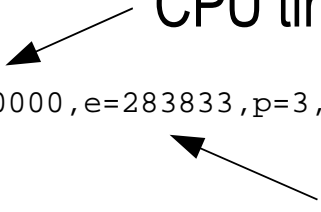
So in this case the AVERAGE response time is:

$$2109.75 / 29670 = 0.0711 \text{ (seconds/txn)}$$

# Oracle - Response Time via Trace

- Using the SQL TRACE interface, you can get a full profile of running transactions.
  - > Tracing is done on a **per session** basis
  - > All statements response time is logged... not just the average.
  - > Can produce histograms, 90<sup>th</sup> percentile, ...
- Analyzers include: HOTSOS, ORASRP, or TKPROF

CPU time in microseconds  
 EXEC #93:c=260000,e=283833,p=3,cr=7,cu=0,mis=1,r=0,dep=1,og=1,tim=2242233800187  
 Elapsed time in microseconds



# Oracle - Response Time profiling

- Analysis by Method R
  - > Focuses methodically on the components that make up the response time.

## Response Time Components Summary

| Response Time Component            | Duration           |               | # Calls        | - Duration per Call - |                |                 |
|------------------------------------|--------------------|---------------|----------------|-----------------------|----------------|-----------------|
|                                    |                    |               |                | Avg                   | Min            | Max             |
| <u>buffer busy waits</u>           | 575.4031s          | 55.5%         | 118,757        | 0.0048s               | 0.0000s        | 3.0079s         |
| <u>CPU service</u>                 | 262.9600s          | 25.4%         | 361,837        | 0.0007s               | 0.0000s        | 38.7000s        |
| <u>latch free</u>                  | 132.8807s          | 12.8%         | 19,788         | 0.0067s               | 0.0000s        | 0.1755s         |
| <u>log buffer space</u>            | 47.1814s           | 4.6%          | 3,214          | 0.0147s               | 0.0000s        | 0.1391s         |
| <u>enqueue</u>                     | 40.2380s           | 3.9%          | 1,744          | 0.0231s               | 0.0000s        | 2.4316s         |
| <u>free buffer waits</u>           | 1.8683s            | 0.2%          | 6              | 0.3114s               | 0.0258s        | 0.8240s         |
| <u>log file sync</u>               | 0.3367s            | 0.0%          | 1              | 0.3367s               | 0.3367s        | 0.3367s         |
| <u>L1 validation</u>               | 0.1184s            | 0.0%          | 10             | 0.0118s               | 0.0000s        | 0.0195s         |
| <u>wait list latch free</u>        | 0.0186s            | 0.0%          | 1              | 0.0186s               | 0.0186s        | 0.0186s         |
| <u>SQL*Net message from client</u> | 0.0028s            | 0.0%          | 3              | 0.0009s               | 0.0006s        | 0.0012s         |
| <u>SQL*Net message to client</u>   | 0.0000s            | 0.0%          | 3              | 0.0000s               | 0.0000s        | 0.0000s         |
| <u>buffer deadlock</u>             | 0.0000s            | 0.0%          | 17             | 0.0000s               | 0.0000s        | 0.0000s         |
| unaccounted-for                    | -25.1366s          | -2.4%         |                |                       |                |                 |
| <b>Total</b>                       | <b>1,035.8714s</b> | <b>100.0%</b> | <b>505,381</b> | <b>0.0020s</b>        | <b>0.0000s</b> | <b>38.7000s</b> |

# Postgres - Measuring response time

- Building on DTRACE for txn/sec, response time is easily added.

```
#!/usr/sbin/dtrace -qs ##avg_tps.d

BEGIN
{
    commits = 0;
}
postgresql*:::transaction-start
{
    self->ts = timestamp;
}
postgresql*:::transaction-commit
/self->ts/
{
    @time = avg((timestamp - self->ts) / 1000000);
    self->ts = 0;
    commits++;
}
tick-1sec
{
    printf("%d commits/sec ", commits);
    printa("%@d avg resp time (ms)\n", @time);
    commits = 0;
    clear(@time);
}
```

```
root@bigleaf> avg_tps.d
186 commits/sec  52 avg resp time (ms)
276 commits/sec  85 avg resp time (ms)
270 commits/sec  96 avg resp time (ms)
270 commits/sec  90 avg resp time (ms)
247 commits/sec 100 avg resp time (ms)
248 commits/sec  98 avg resp time (ms)
318 commits/sec  79 avg resp time (ms)
347 commits/sec  72 avg resp time (ms)
283 commits/sec  89 avg resp time (ms)
```

# Postgres - Response time histogram

```

root@bigleaf> qtz_tps.d
992 commits per second. Below a distribution of resp time in milliseconds
      value  ----- Distribution ----- count
          1 |                                     0
          2 | @@@@@@@@@@@@@@@@@@@@@@@@@@@@          559
          4 | @@@@@@@@@@@@@@@@@@@@@@@@@@@@          636
          8 | @                                     20
         16 |                                     0
  
```

- DTRACE to show a quantization of when transactions finish.

```

#!/usr/sbin/dtrace -qs

BEGIN
{
    commits = 0;
}

postgresql*:::transaction-start
{
    self->ts = timestamp;
}

postgresql*:::transaction-commit
/self->ts/
{
    @time = quantize((timestamp - self->ts) / 1000000);
    self->ts = 0;
    commits++;
}

tick-1sec
{
    printf("%d commits per second.", commits);
    printa(" Below a distribution of resp time
           in milliseconds %@d \n", @time);
    commits = 0;
    clear(@time);
}
  
```

# Postgres Logging statistics

- Logging provides statistics about the life of each query, such as...
  - > Blocks read/written
  - > Buffer hit rates
  - > user/sys/total time spent working on query
  - > Context switching
- All data can be broken down to utilization within each step of query life.
  - > Parsing/ReWriter/Planner/Executor

# Postgres logging details

- Time spent in an individual query

11:46:04.797 PDT igen postgresSTATEMENT: insert into activity\_125(act\_type,trn\_type,id,tstamp) values('U','C',505948,'1-AUG-1935')

11:46:04.809 PDT igen postgresSTATEMENT: insert into activity\_125(act\_type,trn\_type,id,tstamp) values('U','C',505948,'1-AUG-1935')

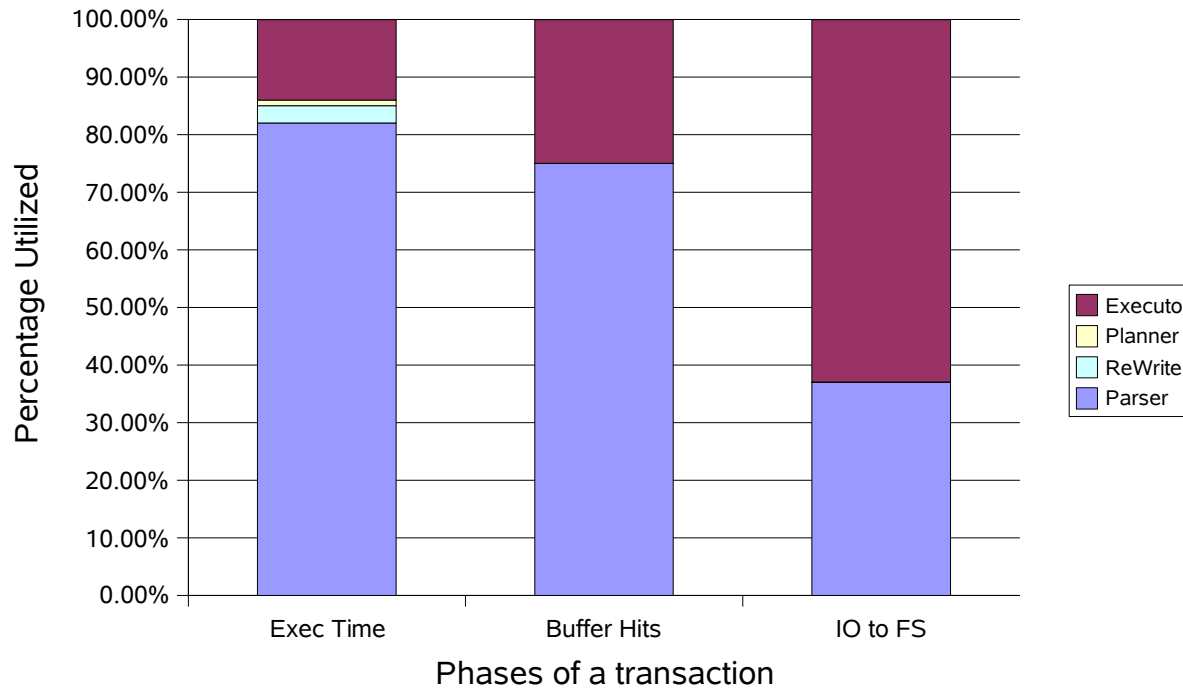
- **Total time is .012 of a second.**

|            |         |
|------------|---------|
| > Parser   | .005469 |
| > ReWriter | .000244 |
| > Planner  | .000028 |
| > Executor | .000956 |

- Only equals .006697, why only 1/2 the time of the timestamp? Time is lost while query traverses from Parser->ReWriter->Planner->Executor. This can be seen from the timestamps in the log trace file.

# Postgres - response time profile

Shows how a response time profile can be created using the Postgres log file.



# PGFOUINE

- Log Analyzer that generates reports based on postgres log files when trace is enabled.

## Slowest queries ^

| Rank | Duration (s) | Query                                                                                           |
|------|--------------|-------------------------------------------------------------------------------------------------|
| 1    | 4,777.68     | <b>UPDATE</b> accounts <b>SET</b> filler= <i>lower</i> ('teSt fiLIer') <b>WHERE</b> aid < 1000; |
| 2    | 3,949.61     | <b>UPDATE</b> accounts <b>SET</b> filler= <i>lower</i> ('teSt fiLIer') <b>WHERE</b> aid < 1000; |
| 3    | 3,763.44     | <b>UPDATE</b> accounts <b>SET</b> filler= <i>lower</i> ('teSt fiLIer') <b>WHERE</b> aid < 1000; |

## Queries that took up the most time (N) ^

| Rank | Total duration | Times executed | Av. duration (s) | Query                                                                                                                                                   |
|------|----------------|----------------|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1    | 1933h26m41s    | 23,387         | 297.62           | <b>UPDATE</b> accounts <b>SET</b> filler= <i>lower</i> ('') <b>WHERE</b> aid < 0;<br><a href="#">Show examples</a>                                      |
| 2    | 17h14m20s      | 23,387         | 2.65             | <b>UPDATE</b> branches <b>SET</b> filler= <i>upper</i> ('');<br><a href="#">Show examples</a>                                                           |
| 3    | 17m13s         | 23,387         | 0.04             | <b>SELECT</b> history.* <b>FROM</b> accounts, history <b>WHERE</b> accounts.aid=0 <b>AND</b> accounts.aid=history.aid;<br><a href="#">Show examples</a> |
| Rank | Times executed | Total duration | Av. duration (s) | Query                                                                                                                                                   |
| 1    | 70,161         | 1m6s           | 0.00             | <b>SELECT</b> * <b>FROM</b> tellers <b>WHERE</b> tid=0;<br><a href="#">Show examples</a>                                                                |
| 2    | 46,774         | 5m32s          | 0.01             | <b>SELECT</b> * <b>FROM</b> accounts <b>WHERE</b> aid=0;<br><a href="#">Show examples</a>                                                               |
| 3    | 23,387         | 1933h26m41s    | 297.62           | <b>UPDATE</b> accounts <b>SET</b> filler= <i>lower</i> ('') <b>WHERE</b> aid < 0;<br><a href="#">Show examples</a>                                      |

## Slowest queries (N) ^

| Rank | Av. duration (s) | Times executed | Total duration | Query                                                                                                                                                   |
|------|------------------|----------------|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1    | 297.62           | 23,387         | 1933h26m41s    | <b>UPDATE</b> accounts <b>SET</b> filler= <i>lower</i> ('') <b>WHERE</b> aid < 0;<br><a href="#">Show examples</a>                                      |
| 2    | 2.65             | 23,387         | 17h14m20s      | <b>UPDATE</b> branches <b>SET</b> filler= <i>upper</i> ('');<br><a href="#">Show examples</a>                                                           |
| 3    | 0.04             | 23,387         | 17m13s         | <b>SELECT</b> history.* <b>FROM</b> accounts, history <b>WHERE</b> accounts.aid=0 <b>AND</b> accounts.aid=history.aid;<br><a href="#">Show examples</a> |

# Resource monitoring

- Focus on the basic resources
  - > IO
  - > CPU
  - > Memory
  - > Network

# Resource Monitoring - Oracle

- Most data comes from AWR/statspack with Oracle.
- OS tools can be used as well
- Can drill down with SQL\_TRACE to get more detailed.
- Oracle Enterprise Manager with plugin modules can be used to manage the workload
- Third party tools like Teamquest or Spotlight can be used as well.

# Resource Monitoring Breakdown

## Postgres

- IO
  - > DB: pg\_stat\* tables, PGLOGGING
  - > OS: iostat, DTRACE
- CPU
  - > DB: PGLOGGING, DTRACE
  - > OS: vmstat, prstat, mpstat, DTRACE
- Memory
  - > DB: -none-
  - > OS: pmap -xs, adb ::prtmem
- Network
  - > DB: -none-
  - > OS: nicstat, netstat

## Oracle

- IO
  - > DB: Statspack, AWR, SQL\_TRACE
  - > OS: iostat, DTRACE
- CPU
  - > DB: Statspack, AWR, SQL\_TRACE
  - > OS: vmstat, prstat, mpstat, DTRACE
- Memory
  - > DB: Statspack, AWR
  - > OS: pmap -xs, adb ::prtmem
- Network
  - > DB: Statspack, AWR, SQL\_TRACE
  - > OS: nicstat, netstat

# Resource Monitoring with DTRACE

- Gives OS perspective of system performance.
  - > Which users are consuming the most disk,cpu,mem...
  - > Where are all the TLBs coming from ... ect

“prustat” built on top of dtrace

| PID   | %CPU  | %Mem  | %Disk | %Net | COMM     |
|-------|-------|-------|-------|------|----------|
| 8639  | 12.13 | 64.62 | 17.11 | 0.00 | postgres |
| 5009  | 0.00  | 64.51 | 0.00  | 0.00 | postgres |
| 5011  | 0.01  | 64.47 | 0.02  | 0.00 | postgres |
| 10415 | 0.02  | 64.47 | 0.00  | 0.00 | postgres |
| 5013  | 0.00  | 64.48 | 0.00  | 0.00 | postgres |
| 5003  | 0.01  | 64.47 | 0.00  | 0.00 | postgres |

- Dtrace toolkit is a good place to start
  - > <http://www.brendangregg.com/dtrace.html>
  - > <http://www.sun.com/bigadmin/content/dtrace/>

# Additional monitoring and Tools

- Postgres statspack prototype.
  - > Example of how to use the pg\_stat\* tables to build your own statspack.
- Chime
  - > Graphical monitor built on top of DTRACE
  - > Utilizes Postgres probes
- PGADMIN3
  - > Admin and some basic access to perf tables
- PG fouine
  - > analyze postgres trace logs

# Postgres Statspacks Prototype

- Nothing “official” currently exists.
- Roll-Your-Own.
  - > PGSTATSPACK prototype
    - Works like statspack.
      - pg\_stat\* tables are sampled and stored in separate database tables.
      - Reports profile IO for each table/index.
    - Global txn/sec, cache\_hit, LIO/sec, Tuples/sec.
    - Kit, scripts and write-up at:
      - [http://blogs.sun.com/glennf/entry/pgstatspack\\_getting\\_at\\_postgres\\_performance](http://blogs.sun.com/glennf/entry/pgstatspack_getting_at_postgres_performance)

# Postgres Statspacks Prototype

- Once the stats and been captured, multiple reports can be created.
- Just a sample of the info you can derive...

## DATABASE THROUGHPUT

| database  | tps    | hitrate | lio_ps  | rd_ps   | rows_ps  | ins_ps | upd_ps | del_ps | rollbk_ps |
|-----------|--------|---------|---------|---------|----------|--------|--------|--------|-----------|
| igen      | 169.55 | 94.00   | 3909.70 | 211.15  | 23543.05 | 50.87  | 46.74  | 0.00   | 0.00      |
| tpce      | 0.04   | 0.00    | 2310.97 | 2307.90 | 0.65     | 0.01   | 0.00   | 0.00   | 0.00      |
| postgres  | 0.03   | 99.00   | 1.86    | 0.00    | 0.44     | 0.00   | 0.00   | 0.00   | 0.00      |
| template1 | 0.00   | 0.00    | 0.00    | 0.00    | 0.00     | 0.00   | 0.00   | 0.00   | 0.00      |
| template0 | 0.00   | 0.00    | 0.00    | 0.00    | 0.00     | 0.00   | 0.00   | 0.00   | 0.00      |

(5 rows)

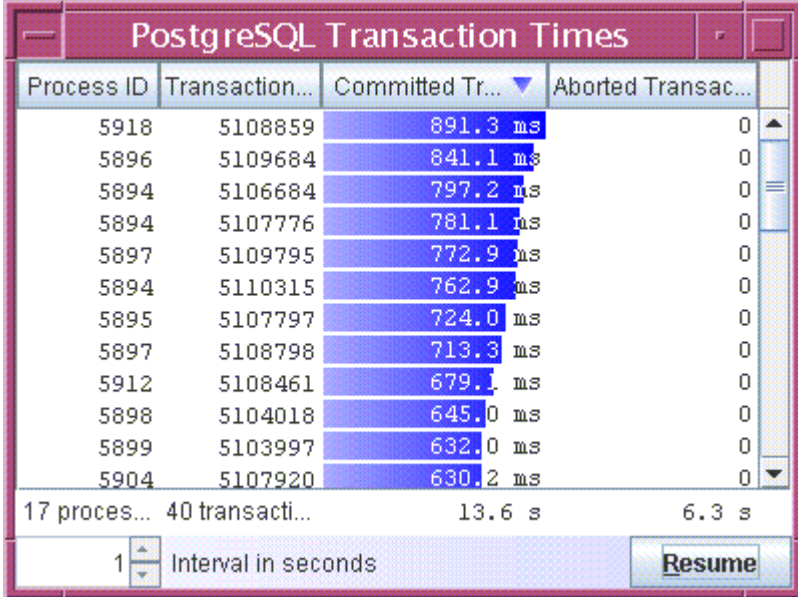
## MOST ACCESSED TABLES by pct of tuples: igen database

| table        | tuples_pct | tab_hitpct | idx_hitpct | tab_read | tab_hit | idx_read | idx_hit |
|--------------|------------|------------|------------|----------|---------|----------|---------|
| order_125    | 45         | 91         | 77         | 67566    | 698578  | 58050    | 202950  |
| product_125  | 42         | 99         | 99         | 82       | 120060  | 30       | 127345  |
| industry_125 | 10         | 99         | 0          | 1        | 22409   | 0        | 0       |
| customer_125 | 1          | 94         | 99         | 34978    | 657096  | 6858     | 1032477 |

# Utilizing Chime --- DEMO!

- Chime is a GUI front end for DTRACE probes.

We see the transaction number and the time in milliseconds for that transaction to be committed.

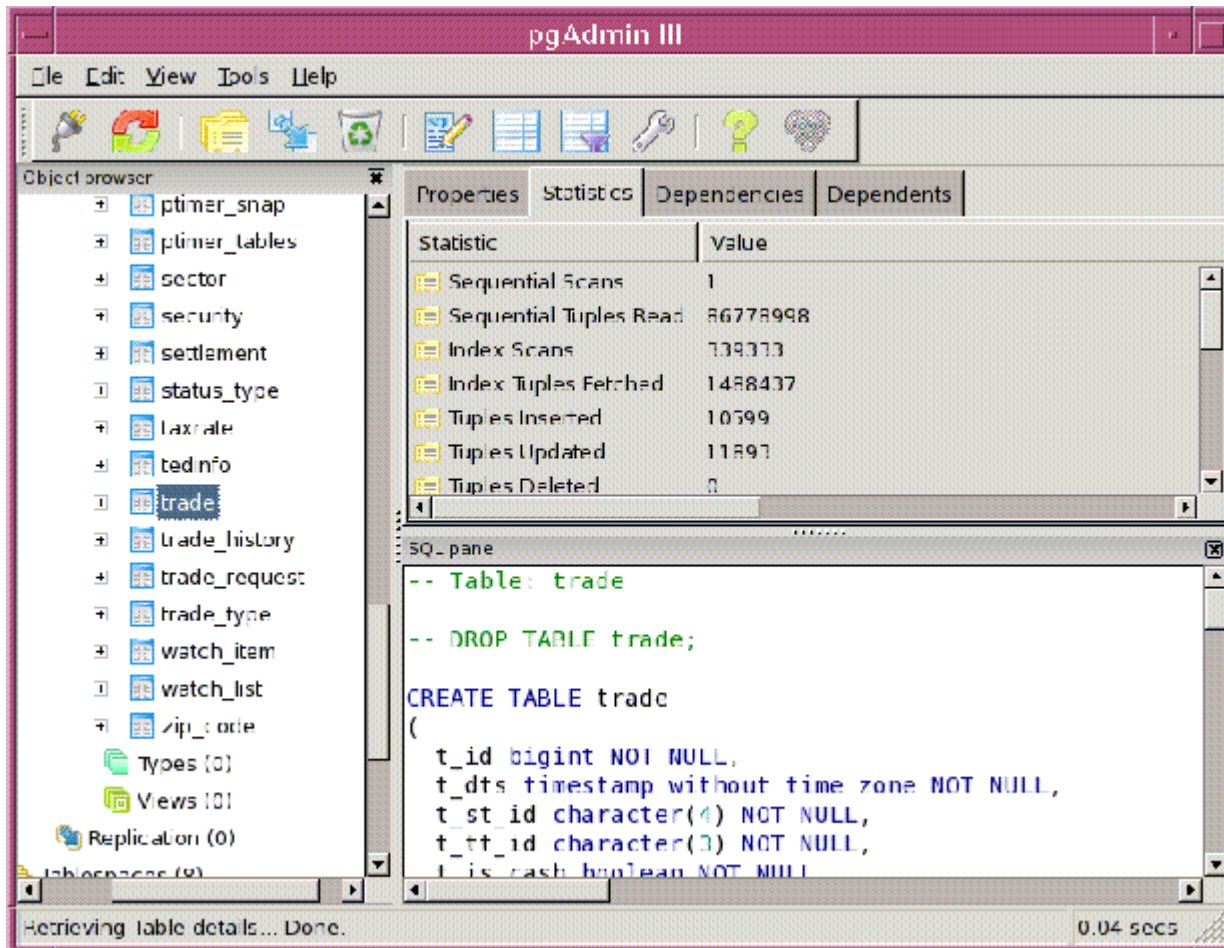


| Process ID                   | Transaction... | Committed Tr... | Aborted Transac... |
|------------------------------|----------------|-----------------|--------------------|
| 5918                         | 5108859        | 891.3 ms        | 0                  |
| 5896                         | 5109684        | 841.1 ms        | 0                  |
| 5894                         | 5106684        | 797.2 ms        | 0                  |
| 5894                         | 5107776        | 781.1 ms        | 0                  |
| 5897                         | 5109795        | 772.9 ms        | 0                  |
| 5894                         | 5110315        | 762.9 ms        | 0                  |
| 5895                         | 5107797        | 724.0 ms        | 0                  |
| 5897                         | 5108798        | 713.3 ms        | 0                  |
| 5912                         | 5108461        | 679.1 ms        | 0                  |
| 5898                         | 5104018        | 645.0 ms        | 0                  |
| 5899                         | 5103997        | 632.0 ms        | 0                  |
| 5904                         | 5107920        | 630.2 ms        | 0                  |
| 17 proces... 40 transacti... |                | 13.6 s          | 6.3 s              |

1 Interval in seconds Resume

# Using PG\_Admin

- Useful tool for getting an overview of your DB



# Resources

- Postgres HOWTO guide for ADMIN
  - > <http://www.sun.com/software/solaris/howtoguides/postgresqlhowto.jsp>
- MDE's Postgres Twiki
  - > <http://twiki.sfbay.sun.com/pub/MDE/PostgreSQL>
- Postgres Official site
  - > <http://postgresql.org/>
- SAE website: <http://sae.central/>
- PAE website: <http://pae.eng>
- Solaris Internals and Performance
  - > <http://www.sun.com/software/solaris/howtoguides/postgresqlhowto.jsp>
- Postgres statspack prototype
  - > [http://blogs.sun.com/glennf/entry/pgstatspack\\_getting\\_at\\_postgres\\_performance](http://blogs.sun.com/glennf/entry/pgstatspack_getting_at_postgres_performance)
- Blogs:
  - > Google: “site:blogs.sun.com postgres”
  - > Josh Berkus: <http://blogs.ittoolbox.com/database/soup>

# Summary

- Leveraging DTRACE to expand the performance monitoring tools of Postgres.
  - > 13 probes currently
  - > More probes being evaluated and added.
  - > STAY TUNED!!
- Basic DB stats available via database tables
  - > pg\_stat\*
- Tracing/Profiling via Postgres logging.

# CEC 2007

## QUESTIONS???????

[Marcus.Heckel@Sun.com](mailto:Marcus.Heckel@Sun.com)

[Glenn.Fawcett@Sun.com](mailto:Glenn.Fawcett@Sun.com)

SAE: <http://sae.central.sun.com/>

<http://blogs.sun.com/glennf>

<http://blogs.sun.com/mheckel>

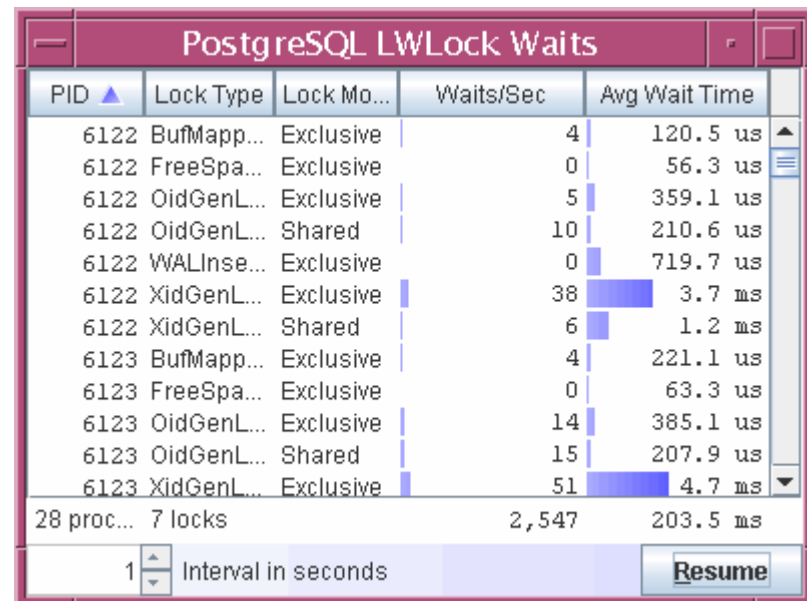
**SHIFT**—OUR UNIVERSE. OUR WORLD. YOUR MOVE.



# Utilizing Chime

- Looking at lock activity and hold times.

Useful to observe what type of lock might be causing problems, how often it's held and for how long. Each column is sortable from low-high or high-low.



| PID ▲              | Lock Type  | Lock Mo... | Waits/Sec           | Avg Wait Time |
|--------------------|------------|------------|---------------------|---------------|
| 6122               | BufMapp... | Exclusive  | 4                   | 120.5 us      |
| 6122               | FreeSpa... | Exclusive  | 0                   | 56.3 us       |
| 6122               | OldGenL... | Exclusive  | 5                   | 359.1 us      |
| 6122               | OldGenL... | Shared     | 10                  | 210.6 us      |
| 6122               | WALInse... | Exclusive  | 0                   | 719.7 us      |
| 6122               | XidGenL... | Exclusive  | 38                  | 3.7 ms        |
| 6122               | XidGenL... | Shared     | 6                   | 1.2 ms        |
| 6123               | BufMapp... | Exclusive  | 4                   | 221.1 us      |
| 6123               | FreeSpa... | Exclusive  | 0                   | 63.3 us       |
| 6123               | OldGenL... | Exclusive  | 14                  | 385.1 us      |
| 6123               | OldGenL... | Shared     | 15                  | 207.9 us      |
| 6123               | XidGenL... | Exclusive  | 51                  | 4.7 ms        |
| 28 proc... 7 locks |            |            | 2,547               | 203.5 ms      |
| 1                  |            |            | Interval in seconds |               |

# Postgres – Response time Profile via Postgres Trace

- `log_statement_stats`
  - > This will log the response time and runtime statistics of every running query... fairly heavy weight.
- For query optimizer:
  - > `log_parser_stats`, `log_planner_stats`, `log_executor_stats`  
– default off

# PGLOGGING EXAMPLES

- Example output

```
[53-1]2007-09-24 11:46:04.907 PDT igen postgresLOG: execute <unnamed>: update customer_125 set total_sales = $1 where custid = $2
```

```
[53-2] 2007-09-24 11:46:04.907 PDT igen postgresDETAIL: parameters: $1 = '-1335096830', $2 = '804491'
```

```
[54-1] 2007-09-24 11:46:04.910 PDT igen postgresLOG: EXECUTOR STATISTICS
```

```
[54-2] 2007-09-24 11:46:04.910 PDT igen postgresDETAIL: ! system usage stats:
```

```
[54-3] !      0.003001 elapsed 0.001024 user 0.000759 system sec
```

```
[54-4] !      [0.062416 user 0.095100 sys total]
```

```
[54-5] !      0/4 [44/115] filesystem blocks in/out
```

```
[54-6] !      0/0 [0/0] page faults/reclaims, 0 [0] swaps
```

```
[54-7] !      0 [0] signals rcvd, 0/0 [9/564] messages rcvd/sent
```

```
[54-8] !      4/0 [699/3] voluntary/involuntary context switches
```

```
[54-9] ! buffer usage stats:
```

```
[54-10] !      Shared blocks:      4 read,      0 written, buffer hit rate = 89.47%
```

```
[54-11] !      Local blocks:      0 read,      0 written, buffer hit rate = 0.00%
```

```
[54-12] !      Direct blocks:     0 read,      0 written
```

# Postgres – What's running?

- Using `pg_stat_activity` view we can see currently running sql.
  - > Columns like start times, waiting can give some insight as to the life of the queries listed.

```
tpce=# select procpid,current_query,waiting
from pg_stat_activity;
```

| procpid | current_query                                               | waiting |
|---------|-------------------------------------------------------------|---------|
| 6642    | select procpid,current_query,waiting from pg_stat_activity; | f       |
| 6659    | select count(t_id) from trade;                              | f       |

(2 rows)

# Postgres – Ustr CPU Dtrace “hotuser”

- Shows which routines are using the most CPU.
  - > Stats collection shows 24.3% in this sample.

```

root@bigleaf> hotuser -p 7031
Sampling... Hit Ctrl-C to end.
^C
FUNCTION                                COUNT    PCNT
libc_psr.so.1`memcpy                    1        2.7%
postgres`oid_hash                        1        2.7%
libc.so.1`_pollsys                       1        2.7%
postgres`pgstat_get_db_entry            1        2.7%
libc.so.1`_so_recv                      1        2.7%
postgres`PgstatCollectorMain            1        2.7%
libc.so.1`poll                           2        5.4%
libsocket.so.1`0xffffffff7f40f500      2        5.4%
libc.so.1`__pollsys                     2        5.4%
libc_psr.so.1`memcpy                    2        5.4%
libsocket.so.1`recv                     2        5.4%
libc.so.1`_fwrite_unlocked              2        5.4%
postgres`hash_search                    2        5.4%
postgres`hash_seq_search                 2        5.4%
postgres`hash_search_with_hash_value    6       16.2%
postgres`pgstat_recv_tabstat            9       24.3%

```

# Postgres – Ustr CPU Dtrace “hotuser”

- Shows which routines are using the most CPU.
  - > Locking (pg\_atomic\_cas) shows 16.4% in this sample

```

root@bigleaf> hotuser -p 7996
FUNCTION                                COUNT    PCNT
...
...
postgres`MemoryContextAlloc             17      0.8%
postgres`slot_deform_tuple               17      0.8%
plpgsql.so`exec_stmt                     17      0.8%
postgres`ExecInitExpr                    19      0.9%
postgres`hash_any                         20      0.9%
postgres`AcquireExecutorLocks            20      0.9%
postgres`_bt_checkkeys                   20      0.9%
postgres`AllocSetFree                    21      1.0%
postgres`HeapTupleSatisfiesMVCC          21      1.0%
postgres`MemoryContextAllocZeroAligned   22      1.0%
plpgsql.so`exec_assign_value             24      1.1%
postgres`SearchCatCache                  27      1.2%
postgres`_bt_compare                      30      1.4%
postgres`TransactionIdPrecedes           33      1.5%
plpgsql.so`exec_eval_simple_expr         33      1.5%
postgres`hash_search_with_hash_value     41      1.9%
postgres`AllocSetAlloc                   43      2.0%
postgres`LWLockAcquire                   43      2.0%
postgres`LWLockRelease                   65      3.0%
postgres`GetSnapshotData                  75      3.5%
libc_psr.so.1`memcpy                     77      3.5%
postgres`pg_atomic_cas                   356     16.4%

```

# Oracle Trace Wait Interface

- Oracle tracing is a lot like truss or Dtrace for the database.
  - > What is a particular “shadow” process doing?
    - SQL statements, wait events, ...
  - > Trace produces \*.trc file in udump directory.
    - Post process with HOTSOS, TKPROF, ORASRP,...

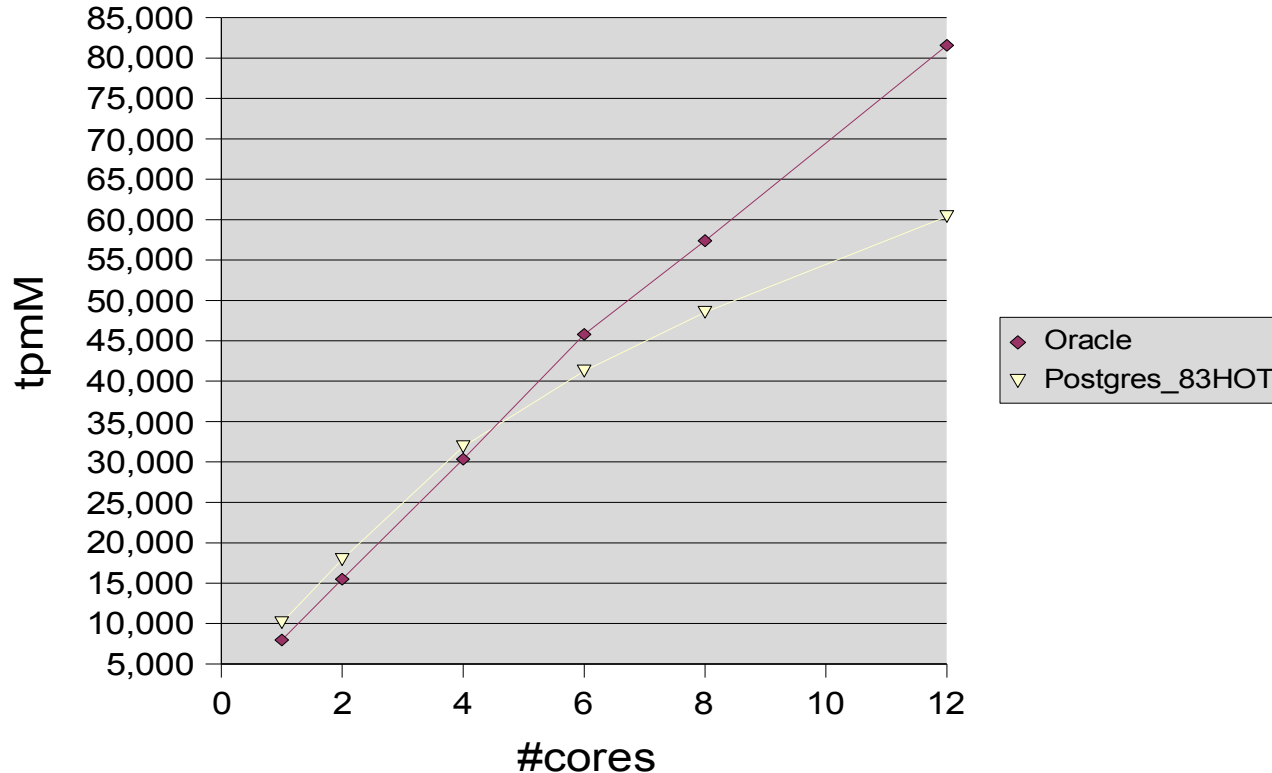
```

SQL> connect / as sysdba
SQL> oradebug setospid 5544
SQL> oradebug event 10046 trace name context forever, level 12
...
SQL> oradebug event 10046 trace name context off
...
EXEC #2:c=0,e=324,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=677730703911
WAIT #2: nam='db file sequential read' ela= 5954 p1=1 p2=15356 p3=1
WAIT #2: nam='db file sequential read' ela= 7235 p1=1 p2=14168 p3=1
FETCH #2:c=10000,e=13869,p=2,cr=3,cu=0,mis=0,r=1,dep=1,og=4,tim=677730717849
STAT #1 id=1 cnt=0 pid=0 pos=1 obj=6251 op='TABLE ACCESS FULL
SQLPLUS_PRODUCT_PROFILE (cr=3 r=0 w=0 time=121 us)'
STAT #2 id=1 cnt=1 pid=0 pos=1 obj=18 op='TABLE ACCESS BY INDEX ROWID OBJ#(18)
(cr=3 r=2 w=0 time=13829 us)'
STAT #2 id=2 cnt=1 pid=1 pos=1 obj=36 op='INDEX UNIQUE SCAN OBJ#(36) (cr=2 r=1 w=0
time=6350 us)'
WAIT #1: nam='SQL*Net message to client' ela= 3 p1=1650815232 p2=1 p3=0
WAIT #1: nam='SQL*Net message from client' ela= 256 p1=1650815232 p2=1 p3=0

```

# Some good news on performance!

iGenOLTP v1.6 125scale Scaling (Postgres vs Oracle)



Below 6 cores, Postgres performs better than Oracle on igenOLTP!!

# Where's the \*.ora files?

- Parameter files in \$PGDATA
  - > postgresql.conf =~ init.ora & listener.ora
    - Memory buffers, users, logging, ect... defined here.
    - Network listener defined here.... unlike Oracle in a separate file.
  - > pg\_hba.conf =~ sqlnet.ora
    - This file defines which connections are allowed to connect to the database.

# Basic file locations

- \$PGDATA
  - > Config files
  - > Base home for table and index data.
- Executables? Where do they live?
  - > Oracle has \$ORACLE\_HOME
  - > Postgres defaults: “/usr/local/bin/pgsql” but this is configurable.
- Logging?
  - > Oracle logs to alert.log files
  - > Postgres logs to stderr or syslog

# 9. Template – Sample Photo Layout



Everyone and Everything Participating on the Network