Understanding Oracle CPU Consumption: The Missing Link

OraPub

Craig A. Shallahamer
OraPub, Inc.
craig@orapub.com

LinkedIn XING

Understanding Oracle CPU Consumption: The Missing Link

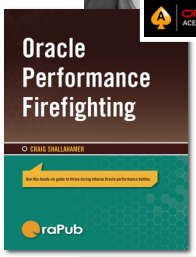
OraPub

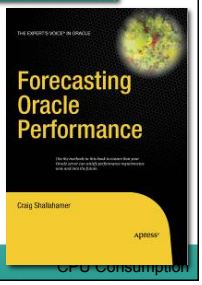
Craig A. Shallahamer
OraPub, Inc.
craig@orapub.com

LinkedIn XING

Who Am I?

- Studied economics, mathematics and computer science at Cal Polytechnic State University San Luis Obispo, California, USA.
- Started working with Oracle technology in 1989 as a Forms 2.3 developer on Oracle version 5.
- Soon after started performance firefighting daily.
- Co-founded both Oracle's Core Technology and System Performance Groups.
- Left Oracle to start OraPub, Inc. in 1998.
- Authored 24+ technical papers and worked in 31 countries.
- Author two books: Oracle Performance Firefighting and Forecasting Oracle Performance.
- Teaches performance analysis around the world.
- Oracle ACE Director.
- Blogs performance research: A Wider View





XING LinkedIn

raPub

focusing exclusively on Oracle systems performance analysis

stori

Find, Understand & Solve
Oracle Performance Problems

www.storifree.com
www.stori.orapub.com

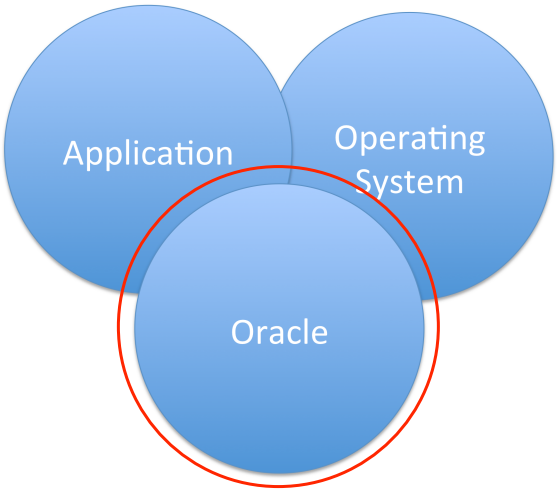
Resources

- Research Blog
- Free Tools
- Free Presentations
- Free Papers
- Books
- Consulting
- Training

Agenda

- Oracle Time Based Analysis (TBA)
- Wait Time
- CPU Time
- Merging Wait and CPU Time
- Seeing is believing
- Next Steps

Holistic Performance Analysis



WORKLOAD REPOSITORY report for

Top 5 Timed Events

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
CPU time		3,641		66.3	
db file sequential read	489,550	587	1	10.7	User I/O
db file scattered read	12,142	565	47	10.3	User I/O
direct path read temp	34,932	470	13	8.6	User I/O
log file parallel write	6,253	235	38	4.3	System I/O

Main Report

- Report Summary
- Wait Events Statistics
- SQL Statistics
- Instance Activity Statistics
- IO Stats
- Buffer Pool Statistics
- Advisory Statistics
- Wait Statistics
- Undo Statistics
- Latch Statistics
- Segment Statistics
- Dictionary Cache Statistics
- Library Cache Statistics
- Memory Statistics
- Streams Statistics
- Resource Limit Statistics
- init.ora Parameters

Quantitative

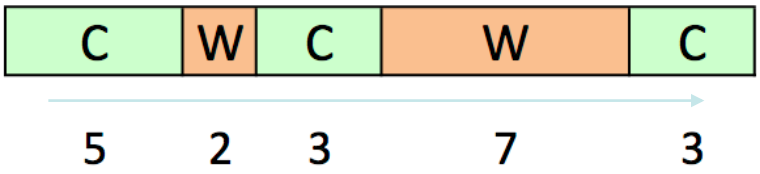
trustworthy, repeatable, demonstrable

Oracle **T**ime **B**ased **A**nalysis

- Modern Oracle performance analysis is about meeting user and business requirements, in large part by reducing *time*.
- *Time* to “run” a SQL statement, a batch job, a module, or business function.
- Batch jobs: Focus on reducing duration.
- OLTP: Focus on reducing elapsed time.
- **U**nit of **W**ork **T**ime **B**ased **A**nalysis focuses on the time it takes to process a single unit of work and allows the unification of an Oracle Time Based Analysis with Operations Research.

Elapsed time from an Oracle time perspective.

Elapsed Time = CPU Time + Wait Time




```
CPU   = 11
Wait =  9
---------
E     = 20
```

Single serial session

Oracle wait time over an interval.

Wait Events

- s - second
- cs - centisecond - 100th of a second
- ms - millisecond - 1000th of a second
- us - microsecond - 1000000th of a second
- ordered by wait time desc, waits desc (idle events last)

Event	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
db file sequential read	489,550	0.00	587	1	252.35
db file scattered read	12,142	0.00	565	47	6.26
direct path read temp	34,932	0.00	470	13	18.01
log file parallel write	6,253	0.00	235	38	3.22
db file parallel write	4,029	0.00	176	44	2.08
direct path write temp	30,705	0.00	160	5	15.83
log buffer space	416	0.00	46	111	0.21
log file sync	1,938	0.00	38	20	1.00

NI Wait time = 587 + 565 + 470 + 235 +176 + 180 + other = 1848 sec

```
base view:
wait: v$system_event
cpu : v$sys_time_model
```

Oracle process CPU consumption

Time Model Statistics

- Total time in database user-calls (DB Time): 5488.7s
- Statistics including the word "background" measure background
- Ordered by % or DB time desc, Statistic name

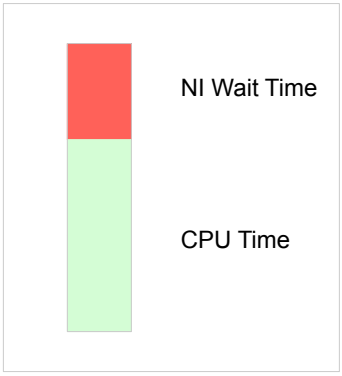
Statistic Name	Time (s)	% of DB Time
sql execute elapsed time	5,385.61	98.12
DB CPU	3,641.47	66.34
parse time elapsed	139.84	2.55
hard parse elapsed time	132.46	2.41
PL/SQL execution elapsed time	12.18	0.22
PL/SQL compilation elapsed time	9.35	0.17
connection management call elapsed time	9.21	0.17
hard parse (sharing criteria) elapsed time	5.65	0.10
sequence load elapsed time	1.19	0.02
failed parse elapsed time	0.76	0.01
repeated bind elapsed time	0.46	0.01
hard parse (bind mismatch) elapsed time	0.31	0.01
DB time	5,488.72	
background elapsed time	534.09	
background cpu time	84.49	

DB CPU = 3641

Total CPU =
3641 + 84 = 3725

```
base view:
v$sys_time_model
```


Database time



"DB time" does NOT include background process CPU consumption, though all OS processes compete for CPU resources. You decide.

NI Wait time = 587 + 565 + 470 + 235 +176 + 180 + other = 1848 sec
CPU time = 3641 sec = DB CPU = Oracle foreground process CPU time
Big Bar Time = DB time = 5489 sec = NI Wait time + CPU time

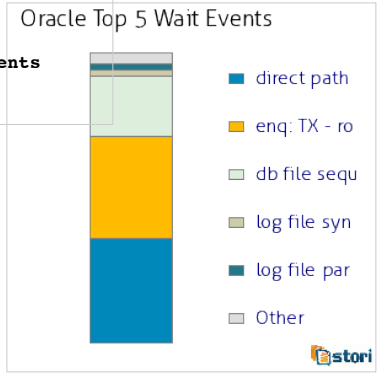
We can see the wait breakdown.

```
How can I help you? get oracle wait time by event top 6

direct path read,2530.741517
enq: TX - row lock contention,2432.46223
db file sequential read,1445.357118
log file sync,168.80679
log file parallel write,154.141394
read by other session,85.438134

How can I help you? get chart bigbar oracle events

work/1376145486_oracle_event_x_BB.png
```



How Oracle gets wait time

- Does this system call timing really work?
- Wait time: set the time.
- Wait time: wait for an opportunity to continue
- Wait time: time the call.

Does this really work?

```
$ cat syscall.c
...

void
main() {
    fd_set rfds;
    int i;
    struct timeval tv, xv;
    struct timespec start;

    FD_ZERO(&rfds);
    FD_SET(0,&rfds);

    for (i=5;i>0;i--) {

        clock_gettime(CLOCK_MONOTONIC, &start);
        gettimeofday(&xv, 0);

        tv.tv_sec = 2;
        tv.tv_usec= 0;
        select(1, &rfds, NULL, NULL, &tv);
    }

    clock_gettime(CLOCK_MONOTONIC, &start);
    gettimeofday(&xv, 0);
}
```

What time is it?

2 sec delay

Does this really work? Let's look!

```
 $ strace -r ./a.out
...
     0.000067 clock_gettime(CLOCK_MONOTONIC, {2511, 257235340}) = 0
     0.000041 gettimeofday({1382036695, 441289}, NULL) = 0
     0.000038 select(1, [0], NULL, NULL, {2, 0}) = 0 (Timeout)
     2.002513 clock_gettime(CLOCK_MONOTONIC, {2513, 259913758}) = 0
     0.000243 gettimeofday({1382036697, 444166}, NULL) = 0
     0.000242 select(1, [], NULL, NULL, {2, 0}) = 0 (Timeout)
     2.002479 clock_gettime(CLOCK_MONOTONIC, {2515, 262880212}) = 0
     0.000229 gettimeofday({1382036699, 447141}, NULL) = 0
     0.000323 select(1, [], NULL, NULL, {2, 0}) = 0 (Timeout)
     2.002547 clock_gettime(CLOCK_MONOTONIC, {2517, 265977190}) = 0
     0.000304 gettimeofday({1382036701, 450297}, NULL) = 0
     0.000312 select(1, [], NULL, NULL, {2, 0}) = 0 (Timeout)
     2.002518 clock_gettime(CLOCK_MONOTONIC, {2519, 269119872}) = 0
     0.000319 gettimeofday({1382036703, 453457}, NULL) = 0
     0.000325 select(1, [], NULL, NULL, {2, 0}) = 0 (Timeout)
     2.002559 clock_gettime(CLOCK_MONOTONIC, {2521, 272323512}) = 0
     0.000321 gettimeofday({1382036705, 456662}, NULL) = 0
     0.000314 exit_group(0)                  = ?
$
```

Notice the two second "select" and the two second "what time is it?" calls.

(c)2013 OraPub, Inc.

Time the call.

Linux, 12.1, seq reads

```
$ strace -rp 2518
...
     0.000324 clock_gettime(CLOCK_MONOTONIC, {504, 52586559}) = 0
     0.000040 clock_gettime(CLOCK_MONOTONIC, {504, 52625324}) = 0
→    0.000040 pread(257, "\6\242\0\f\0"..., 8192, 427270144) = 8192
     0.000047 clock_gettime(CLOCK_MONOTONIC, {504, 52712996}) = 0
     0.000044 clock_gettime(CLOCK_MONOTONIC, {504, 52757393}) = 0
     0.000329 clock_gettime(CLOCK_MONOTONIC, {504, 53086771}) = 0
     0.000040 clock_gettime(CLOCK_MONOTONIC, {504, 53125505}) = 0
→    0.000040 pread(257, "\6\76 [y\f\0"..., 8192, 427278336) = 8192
     0.000047 clock_gettime(CLOCK_MONOTONIC, {504, 53213583}) = 0
     0.000040 clock_gettime(CLOCK_MONOTONIC, {504, 53253021}) = 0
     0.000327 clock_gettime(CLOCK_MONOTONIC, {504, 53580561}) = 0
     0.000040 clock_gettime(CLOCK_MONOTONIC, {504, 53619199}) = 0
→    0.000040 pread(257, "\6\273\f\0"..., 8192, 427286528) = 8192
     0.000047 clock_gettime(CLOCK_MONOTONIC, {504, 53706779}) = 0
     0.000040 clock_gettime(CLOCK_MONOTONIC, {504, 53752611}) = 0
```

0.000047672 sec = ((52712996-52625324)/1000000000)-0.000040

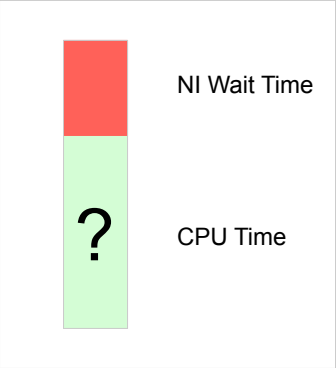
Wait for an opportunity to continue.

Linux, o12.1, CBC

```
$ strace -rp 32873
Process 32873 attached - interrupt to quit
...
     0.000027 getrusage(0x1 /* RUSAGE_??? */, {ru_utime={217, 423946}, ...
     1.991878 clock_gettime(CLOCK_MONOTONIC, {81915, 22092045}) = 0
     0.000058 getrusage(0x1 /* RUSAGE_??? */, {ru_utime={218, 96844}, ...
     0.000039 clock_gettime(CLOCK_MONOTONIC, {81915, 22165643}) = 0
...
... then bamb!
...
     0.000026 getrusage(0x1 /* RUSAGE_??? */, {ru_utime={225, 612701}, ...
     0.678239 clock_gettime(CLOCK_MONOTONIC, {81935, 710441832}) = 0
     0.000056 gettimeofday({1382121114, 896822}, NULL) = 0
→    0.000036 semop(1933315, {{52, -1, 0}}, 1) = 0
     0.010648 clock_gettime(CLOCK_MONOTONIC, {81935, 721164886}) = 0
     0.000034 gettimeofday({1382121114, 907540}, NULL) = 0
     0.000027 gettimeofday({1382121114, 907569}, NULL) = 0
→    0.000059 semctl(1933315, 46, SETVAL, 0x1) = 0
→    0.099170 semctl(1933315, 60, SETVAL, 0x7f4000000001) = 0
     0.000064 clock_gettime(CLOCK_MONOTONIC, {81935, 820517786}) = 0
     0.000033 gettimeofday({1382121115, 6892}, NULL) = 0
→    0.000029 semop(1933315, {{52, -1, 0}}, 1) = 0
     0.009552 clock_gettime(CLOCK_MONOTONIC, {81935, 830133485}) = 0
     0.000035 gettimeofday({1382121115, 16509}, NULL) = 0
 ...
```

There were no “selects”...

Focus: CPU time



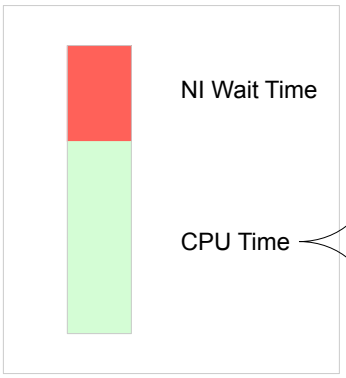
Q: What about the CPU time?

Q: Is that important?

Q: What is Oracle doing with all that CPU?

NI Wait time = 1848 sec
CPU time = 3641 sec = DB CPU = Oracle foreground process CPU time
Big Bar Time = DB time = 5489 sec = NI Wait time + CPU time

What about the CPU breakdown?



Time (s)	Function
1500	abc
1400	def
600	ghi
141	jkl

My dream...

```
PID: 28497  SID: 404  SERIAL: 6185

                                                                    Time
Time Component                                                      secs       %
------------------------------------------------------------ ---------- -------
wait: log file parallel write                                      25.017   83.41
wait: log buffer space                                              1.632    5.44
cpu : [?] sum of funcs consuming less than 2% of CPU time           1.500    5.00
wait: log file switch completion                                    0.559    1.86
cpu : [.] kcbchg1_main                                              0.227     .76
cpu : [.] kcrfw_redo_gen_ext                                        0.123     .41
cpu : [.] kcbget                                                    0.121     .40
cpu : [.] __intel_ssse3_rep_memcpy                                  0.095     .32
cpu : [.] kcrfw_copy_cv                                             0.090     .30
cpu : [.] kdkcmp1                                                   0.090     .30
cpu : [.] kcoapl                                                    0.089     .30
cpu : [.] kcbgcur                                                   0.074     .25
cpu : [.] kdiins1                                                   0.074     .25
cpu : [.] kauupd                                                    0.068     .23
cpu : [.] ktuchg2                                                   0.062     .21
cpu : [.] kduovw                                                    0.061     .20
cpu : [.] updrow                                                    0.057     .19
cpu : [.] kdimod0                                                   0.056     .19
```


Is CPU detail useful?

- Confirm the performance story. Example: If there is a parsing issue, I expect some type of SP/LC related function.
- During my research, it can help me understand what Oracle is doing “under the hood.”
- Answering the question, “So what is Oracle doing with all that CPU?!” Especially when lots of CPU relative to wait time.
- CPU with little wait time. Therefore, no wait time clues.
- Potential Oracle bug detection.

Bad News

Oracle does not provide CPU consumption details through the v$ views.

raPub

Bad News

Oracle does not provide CPU consumption details through the v$ views.

Good News

OS tools do help solve the CPU consumption mystery.

Our OS Options...

- **strace/truss**. No. Only system calls and their elapsed time. Not CPU consumption.
- **dtrace**. Yes, but:
  - must be closely connected with Oracle
  - there is a port for Linux and Oracle
  - It is more of a "step by step" analysis/ debugger tool
- **gdb**. Perhaps, but it's more of a debugger than a profiler.
- **perf**. Yes!

What is "`Perf`"?

- **`Perf`** is:
  - Linux process profiling tool
  - Based on counting specific events:
    - instructions, cache-misses, cycles, etc.
  - Provides these counters for a function call
  - Provides a call tree graph
  - Has a set of tools for displaying, recording, and reporting.
- It's free and was on my Linux distribution.

`https://perf.wiki.kernel.org/index.php/Main_Page`

The "perf" tools

- `perf stat`. Obtain event counts
- `perf record`. Record events for later reporting.
- `perf report`. Break down events by process, function, etc.
- `perf annotate`. Annotate assembly or source code with event counters
- `perf top`. See live event count


```
$ which perf
/usr/sbin/perf

$ ps -eaf | grep oracleprod35
oracle   28497 28496  8 09:58 ?        00:31:36 oracleprod35 (DESCRIPTION=(LOCAL=YES
oracle   42265 28425  0 15:51 pts/1    00:00:00 grep oracleprod35

$ perf stat -p 28497 sleep 5

 Performance counter stats for process id '28497':

       560.686866 task-clock                #    0.112 CPUs utilized
              32 context-switches          #    0.000 M/sec
               1 CPU-migrations            #    0.000 M/sec
               0 page-faults               #    0.000 M/sec
   1,522,513,670 cycles                    #    2.715 GHz                      [83.
     952,309,947 stalled-cycles-frontend   #   62.55% frontend cycles idle     [83.
     568,827,423 stalled-cycles-backend    #   37.36% backend  cycles idle     [66.
   1,333,413,297 instructions              #    0.88  insns per cycle
                                           #    0.71  stalled cycles per insn [83.
     249,397,610 branches                  #  444.807 M/sec                    [82.
       4,478,117 branch-misses             #    1.80% of all branches          [83.

      5.000696361 seconds time elapsed
```

```
$ perf top -p 28497 -d 30


PerfTop: 89 irqs/sec  kernel: 5.6%  exact: 0.0% [1000Hz cycles],(target_pid: 28497)
-------------------------------------------------------------------------------

             samples  pcnt function                    DSO
             _______ _____ ____________________________ ____________________________

              205.00  7.6% kcbchg1_main                /home/oracle/base/product/12.1
              148.00  5.5% kcbget                      /home/oracle/base/product/12.1
              124.00  4.6% kcrfw_redo_gen_ext          /home/oracle/base/product/12.1
              114.00  4.2% kdkcmp1                     /home/oracle/base/product/12.1
              100.00  3.7% __intel_ssse3_rep_memcpy    /home/oracle/base/product/12.1
               99.00  3.7% kcrfw_copy_cv               /home/oracle/base/product/12.1
               89.00  3.3% kcoapl                      /home/oracle/base/product/12.1
               84.00  3.1% kcbgcur                     /home/oracle/base/product/12.1
               77.00  2.9% kdiins1                     /home/oracle/base/product/12.1
               62.00  2.3% kduovw                      /home/oracle/base/product/12.1
               56.00  2.1% ktuchg2                     /home/oracle/base/product/12.1
               52.00  1.9% ktugur                      /home/oracle/base/product/12.1
               51.00  1.9% kdimod0                     /home/oracle/base/product/12.1
               50.00  1.9% updrow                      /home/oracle/base/product/12.1
               46.00  1.7% kauupd                      /home/oracle/base/product/12.1
               42.00  1.6% qertbFetch                  /home/oracle/base/product/12.1
               38.00  1.4% kdudcp                      /home/oracle/base/product/12.1
```



```
$ perf record -e cycles -p 28497 sleep 30

[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.093 MB perf.data (~4057 samples) ]

$ perf report

# Events: 2K cycles
#
# Overhead          Command      Shared Object                        Symbol
# ........  ...............  ..................  ............................
#
     7.91%  oracle_28497_pr  oracle              [.] kcbchg1_main
     4.87%  oracle_28497_pr  oracle              [.] kcbget
     4.07%  oracle_28497_pr  oracle              [.] kcrfw_redo_gen_ext
     3.86%  oracle_28497_pr  oracle              [.] kdkcmp1
     3.42%  oracle_28497_pr  oracle              [.] __intel_ssse3_rep_memcpy
     3.37%  oracle_28497_pr  oracle              [.] kcoapl
     3.23%  oracle_28497_pr  oracle              [.] kcrfw_copy_cv
     3.08%  oracle_28497_pr  oracle              [.] kcbgcur
     2.56%  oracle_28497_pr  oracle              [.] kdiins1
     2.22%  oracle_28497_pr  oracle              [.] ktuchg2
     2.03%  oracle_28497_pr  oracle              [.] ktbgwl
     2.00%  oracle_28497_pr  oracle              [.] kauupd
     1.88%  oracle_28497_pr  oracle              [.] kduovw
     1.78%  oracle_28497_pr  oracle              [.] kdimod0
     1.56%  oracle_28497_pr  oracle              [.] kdudcp
```

```
$ perf record -g -e cycles -p 28497 sleep 3

[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.081 MB perf.data (~3558 samples) ]

$ perf report -g

# Events: 212  cycles
#
# Overhead          Command      Shared Object                       Symbol
# ........  ...............  ..................  ............................
#
     6.47%  oracle_28497_pr  oracle              [.] kcbchg1_main
            |
            --- kcbchg1_main
                kcbchg1
                ktuchg2
                ktbchg2
               |
               |--50.94%-- kdimod0
               |           kauupd
               |           updrow
               |           qerupFetch
               |           updaul
               |           updThreePhaseExe
               |           updexe
               |           ...
               |
               |--28.36%-- kddchg
               |           kduovw
```

The far left column does sum to 100%.

But how to integrate wait and CPU time

- Oracle provides wait time.
- Oracle provides total CPU consumption.
- Perf provides CPU cycle counters.
- How to integrate this into a time based analysis...

Suppose over a 30 second period...

- v$session_wait sum(time_waited)/100 is 10 seconds
- v$sess_time_model, db_cpu/1000000 is 20 seconds
- 60% of CPU counters are function ABC
- 30% of CPU counters are function DEF
- 10% of CPU counters are all other functions

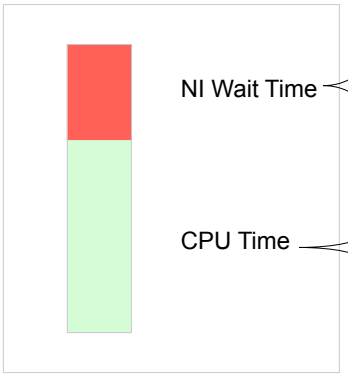
Can I then state...

- Total Wait time is 10 seconds
- Total CPU time is 20 seconds
- 12 sec of CPU for function ABC
- 6 sec of CPU counters for function DEF
- 2 sec of CPU counters for all other functions

Can I jump from counts to seconds?

- The default sample rate is 1000 cycles/sec.
- If a function is on CPU less than 1/1000 second, the count *could* be missed.
- This is also true for ASH's. And ASH's samples at only 1 cycle/sec.
- The more important a function is to us, the more likely it will be sampled.
- With this in mind, I'm comfortable going from counter to time.

Again... My Dream...



Total time over an N minute interval

Time (s)	Event
2540	direct path read
2432	enq: tx – row lock
1445	db file sequential read
700	other

Time (s)	Function
1500	abc
1400	def
600	ghi
141	jkl

Don't stop dreaming... This is closer to what I really want!

Time (s)	Type	Time Component
2540	wait	direct path read
2432	wait	enq: tx – row lock
1500	cpu	abc
1445	wait	db file sequential read
1400	cpu	def
700	wait	other
600	cpu	ghi
141	cpu	jkl

Total time over an N minute interval

Let's create a script.

Help user Identify the PID to profile
Initial setup
Loop
  get oracle CPU consumption (snap 0)
  get oracle wait times (snap 0)
  start capture oracle kernel cpu details
  sleep x
  get oracle CPU consumption (snap 1)
  get oracle wait times (snap 1)
  stop capture oracle kernel cpu details
  do some cool math and other neat stuff
  display results
End Loop

Introducing the “fulltime.sh” tool. Process level CPU and Oracle wait time reporting and analysis.

Periodically cycles like the tool `top`. The refresh rate and other details can be changed within the tool. You are intelligently prompted for process ID.

$ ./fulltime.sh

Full control at the command line.

$ ./fulltime.sh <pid> <duration> <cycles>

$ ./fulltime.sh 5486 30 1

Example: CBC – fulltime.sh

```
$ fulltime.sh 32873 120 1

PID: 32873  SID: 9  SERIAL: 13  USERNAME: OE2  at 18-Oct-2013 12:13:49
CURRENT SQL: SELECT COUNT(*) FROM ( SELECT SUM(OBJECT_ID) FROM ORDERS UNION SELECT


total time: 44.438 secs,  CPU: 41.611 secs (93.64%),  wait: 2.827 secs (6.36%)

                                                                  Time
Time Component                                                    secs       %
------------------------------------------------------------ ---------- -------
cpu : [.] kcbgtcr                                              29.714   66.87
cpu : [.] kdstf000010100001km                                   3.716    8.36
cpu : [.] lnxsum                                                3.541    7.97
cpu : [?] sum of funcs consuming less than 2% of CPU time       2.393    5.38
cpu : [.] kaf4reasrp0km                                         2.180    4.91
wait: PL/SQL lock timer                                         2.100    4.73
wait: latch: cache buffers chains                               0.727    1.64
```

Let's see it live (recorded)...

```
[oracle@sixcore perf]$ fulltime.sh
```


What does kcbchg1_main mean?

```
$ perf report

# Events: 2K cycles
#
# Overhead          Command      Shared Object                       Symbol
# ........  ...............  ..................  ..........................
#
     7.91%  oracle_28497_pr  oracle              [.] kcbchg1_main
     4.87%  oracle_28497_pr  oracle              [.] kcbget
     4.07%  oracle_28497_pr  oracle              [.] kcrfw_redo_gen_ext
     3.86%  oracle_28497_pr  oracle              [.] kdkcmp1
     3.42%  oracle_28497_pr  oracle              [.] __intel_ssse3_rep_memcpy
     3.37%  oracle_28497_pr  oracle              [.] kcoapl
     3.23%  oracle_28497_pr  oracle              [.] kcrfw_copy_cv
     3.08%  oracle_28497_pr  oracle              [.] kcbgcur
     2.56%  oracle_28497_pr  oracle              [.] kdiins1
     2.22%  oracle_28497_pr  oracle              [.] ktuchg2
     2.03%  oracle_28497_pr  oracle              [.] ktbgwl
     2.00%  oracle_28497_pr  oracle              [.] kauupd
     1.88%  oracle_28497_pr  oracle              [.] kduovw
     1.78%  oracle_28497_pr  oracle              [.] kdimod0
     1.56%  oracle_28497_pr  oracle              [.] kdudcp
     1.55%  oracle_28497_pr  oracle              [.] ktbgfi
     1.49%  oracle_28497_pr  oracle              [.] ktugur
     1.48%  oracle_28497_pr  oracle              [.] updrow
     1.26%  oracle_28497_pr  oracle              [.] kdxbrs1
```

?

Decoding Oracle kernel function names.

Do a Google search... no promises... no guarantees...

```
k (kernel)
   2 (two phase commit)
   a (iot changes)
   c (cache)
      b (buffer)
   d (data)
   f (file)
   g
      bt (btree)
      h (heap mgr)
      l (library cache)
      u (os proc)
      x (ip comm)
   j (RAC)
   k (realted to SP)
   l (DP loader)
   m (mts)
   n (streams, rep)
   o (obj)
   p
      p (parsing)
      u
         c (cusor mgt)
```

```
k q
      l (LC/SP mgt)
      r (resource mgr)
      r
         b (b&r)
         vg (DDL redo)
   t (space mgt, trx control)
      b (blk level)
      e (ext level)
      f (flashback)
      fb (file bitmap)
      p (parallel trx)
      r (read consis)
      s (checking/verifying)
      u (undo)
   w (AQ)
   z (security, encrypion)
```

```
o (oper out->in of Oracle)
p (plsql)
q (query)
r (row)
sel (select)
sr (sort)
up (update)
```


Some inferences from previous `perf` report output.

- kcbchg1main. kernel, cache, buffer, change
- kcbget. kernel, cache, buffer, get
- kcrfw_redo_gen_ext. kernel, cache, ..redo generation
- kdkcmp1. kernel, data, change

Example: FBW - Oracle

```
SQL> @rtpctx
Remember: This report must be run twice so both the initial and
final values are available. If no output, press ENTER about 20 times
Database: prod35                                              18-OCT-13 07:11
Report:   rtpctx.sql                OSM by OraPub, Inc.          Page
                         Response Time Activity (131 sec interval)

                                                      Avg Time         Time      W
Time Component                              % TT     % WT Waited (ms)        (sec) Count
----------------------------------- ------- ------- ----------- ----------- -----
free buffer waits                     40.71   41.59      21.540     809.460
enq: KI - contention                  30.65   31.31   33852.222     609.340
DLM cross inst call completion         6.03    6.16    7496.875     119.950
db file async I/O submit               6.02    6.15    1286.989     119.690
CPU consumption: Oracle SP + BG procs  2.12    0.00       0.000      42.235
write complete waits                   1.80    1.84    1194.333      35.830
log file redo write                    1.58    1.61      89.714      31.400
log buffer full - LGWR bottleneck      1.00    1.02      75.530      19.940
log file switch (private strand flush  0.59    0.60    1675.714      11.730
```


Example: FBW – PIO – Fulltime.sh

```
PID: 13967  SID: 268  SERIAL: 17  USERNAME: MG2  at 18-Oct-2013 07:14:11
CURRENT SQL: declare  i number; begin   dbms_application_info.set_module

total time: 16.382 secs,  CPU: .888 secs (5.42%),  wait: 15.494 secs (94.58%)
                                                                  Time
Time Component                                                    secs       %
----------------------------------------------------------------- ---------- -------
wait: free buffer waits                                         15.482   94.51
cpu : [?] sum of funcs consuming less than 2% of CPU time        0.584    3.56
cpu : [.] _intel_fast_memcmp                                     0.080     .49
cpu : [.] kcbgtcr                                                0.054     .33
cpu : [.] qerixGetKey                                            0.030     .18
cpu : [.] __intel_new_memset                                     0.027     .16
cpu : [.] expepr                                                 0.026     .16
cpu : [.] ktrgcm                                                 0.025     .15
cpu : [.] qerixStart                                             0.023     .14
cpu : [.] qerfxFetch                                             0.021     .13
cpu : [.] kdxlrs2                                                0.019     .11
wait: db file sequential read                                    0.010     .06
wait: db file scattered read                                     0.002     .01
wait: read by other session                                      0.000     .00
wait: latch: cache buffers chains                                0.000     .00
```

Example: inserts – fulltime.sh

```
PID: 13698  SID: 258  SERIAL: 9  USERNAME: MG2  at 18-Oct-2013 07:02:55
CURRENT SQL: create table custome

total time: 8.036 secs,  CPU: 3.078 secs (38.3%),  wait: 4.958 secs (61.7%)
                                                                    Time
Time Component                                                      secs        %
------------------------------------------------------------ ---------- -------
wait: direct path read                                             4.048   50.37
cpu : [.] qerandvRop                                               0.768    9.56
cpu : [?] sum of funcs consuming less than 2% of CPU time          0.718    8.94
cpu : [.] kgghash2                                                 0.716    8.90
wait: direct path write                                            0.453    5.63
cpu : [.] kdblailb                                                 0.443    5.52
wait: events in waitclass Other                                    0.200    2.49
wait: control file parallel write                                  0.135    1.68
cpu : [.] qesaFastAggNonDistSSOnSfn                                0.127    1.58
cpu : [.] kdblai                                                   0.115    1.44
cpu : [.] qerltFRop                                                0.109    1.36
cpu : [.] kaf4reasrp0km                                            0.080    1.00
wait: db file single write                                         0.045     .56
wait: Disk file operations I/O                                     0.044     .55

Samples remaining: 998
Gathering next 10 second sample...
To see the Call Graph, press ENTER or to exit press CNTRL-C.
```



```
# Events: 1K cycles
#
# Overhead          Command      Shared Object                              Symbol
# ........  ...............  ...................  ...................................
#
    23.42%  oracle_13698_pr  oracle             [.] qerandvRop
            |
            --- qerandvRop
               |
               |--99.77%-- kdstf000010100000km
               |          kdsttgr
               |          qertbFetch
               |          qerandvFetch
               |          rwsfcd
               |          qerltFetch
               |          ctcdrv
               |          opiexe
               |          opiosq0
               |          kpoal8
               |          opiodr
               |          ttcpip
               |          opitsk
               |          opiino
               |          opiodr
               |          opidrv
               |          sou2o
               |          opimai_real
               |          ssthrdmain
               |          main
               |          __libc_start_main
                --0.23%-- [...]

    22.21%  oracle_13698_pr  oracle             [.] kgghash2
            |
            --- kgghash2
               |
               |--98.34%-- qerandvRop
               |          kdstf000010100000km
               |          kdsttgr
               |          qertbFetch
               |          qerandvFetch
               |          rwsfcd
               |          qerltFetch
```

Example: inserts – Call Graph

Example: LIO – Single – Fulltime.sh

```
PID: 28758  SID: 168  SERIAL: 49  USERNAME: MG2  at 18-Oct-2013 10:28:49
CURRENT SQL: SELECT COUNT(*) FROM DBA_EXTENTS, DBA_EXTENTS, DBA_EXTENTS


total time: 1.996 secs,  CPU: 1.996 secs (100%),  wait: 0 secs (0%)

                                                                  Time
Time Component                                                    secs       %
----------------------------                                 ---------- -------
cpu : [.] smbget                                                 0.642   32.16
cpu : [.] sorgetqbf                                              0.628   31.46
cpu : [.] qersoFetchSimple                                       0.480   24.05
cpu : [.] rworupo                                                0.140    7.02
cpu : [.] qeaeCn1Serial                                          0.101    5.04
cpu : [?] sum of funcs consuming less than 2% of CPU time        0.005     .26
```

sort, merge, buffer gets

To create a single LIO focused session:
select count(*) from dba_extents, dba_extents, dba_extents;


```
# Events: 597  cycles
#
# Overhead          Command      Shared Object               Symbol
# ........  ...............  ..................  ...................
#
    31.32%  oracle_28758_pr  oracle              [.] smbget
            |
            --- smbget
                |
                |--93.12%-- sorgetqbf
                |           qersoFetchSimple
                |           qersoFetch
                |           qerjotFetch
                |           qergsFetch
                |           opifch2
                |           opiefn0
                |           opipls
                |           opiodr
                |           rpidrus
                |           skgmstack
                |           rpiswu2
                |           rpidrv
```

Example: LIO, Single Session, Call Graph

Example: PIOR

```
PID: 2518  SID: 35  SERIAL: 47  USERNAME: MG2  at 17-Oct-2013 11:42:52
CURRENT SQL: SELECT SUM(OBJECT_ID) FROM ALL_OBJECTS

total time: 37.129 secs, CPU: 28.879 secs (77.78%), wait: 8.25 secs (22.22%)
                                                                  Time
Time Component                                                    secs       %
---------------------------------------------------------------- ---------- -------
cpu : [?] sum of funcs consuming less than 2% of CPU time         19.640   52.90
wait: direct path read                                             7.294   19.64
cpu : [.] kaf4reasrp1km                                            4.577   12.33
cpu : [k] copy_user_generic_string                                 2.108    5.68
cpu : [.] kdstf1100101000000km                                     1.019    2.75
wait: events in waitclass Other                                    0.856    2.30
cpu : [.] sxorchk                                                  0.791    2.13
cpu : [.] kcbgtcr                                                  0.638    1.72
wait: db file sequential read                                      0.101     .27
wait: db file scattered read                                       0.000     .00

Samples remaining: 0
Gathering next 30 second sample...
```

kernel, iot changes

Example: Pin S - Oracle

```
SQL> @rtpctx
Remember: This report must be run twice so both the initial and
final values are available. If no output, press ENTER about 20 times.

Database: prod35                                                18-OCT-13 06:4
Report:   rtpctx.sql              OSM by OraPub, Inc.              Page
                        Response Time Activity (130 sec interval)

                                                      Avg Time        Time
Time Component                             % TT    % WT Waited (ms)   (sec) Cour
---------------------------------------- ------- ------- ----------- ----------- ----
CPU consumption: Oracle SP + BG procs     92.74    0.00       0.000     775.073
cursor: pin S                              6.89   94.91       8.357      57.560
control file parallel write                0.12    1.71      24.186       1.040
db file async I/O submit                   0.06    0.77      10.217       0.470
log file redo write                        0.06    0.76       9.388       0.460
```

```
$ fulltime.sh 124545 15 1

PID: 12545  SID: 168  SERIAL: 9  USERNAME: SYSTEM  at 18-Oct-2013 06:40:13
CURRENT SQL: SELECT COUNT(*) FROM DBA_OBJECTS WHERE 1=0

total time: 14.468 secs,  CPU: 13.239 secs (91.51%),  wait: 1.229 secs (8.49%)

                                                                 Time
Time Component                                                   secs       %
---------------------------------------------------------- ---------- -------
cpu : [?] sum of funcs consuming less than 2% of CPU time        10.938   75.60
cpu : [.] __intel_new_memset                                      1.252    8.66
wait: cursor: pin S                                               1.229    8.49
cpu : [.] opiexe                                                  0.436    3.01
cpu : [.] audsel                                                  0.367    2.53
cpu : [.] kxsxsi                                                  0.270    1.87

Samples remaining: 0
Gathering next 15 second sample...
```

Intel chip, new, set memory

Example: Pin S, fulltime.sh


```
# Events: 5K cycles
#
# Overhead          Command      Shared Object                          Symbol
# ........  ...............  ..................  ..............................
#
     9.61%  oracle_12545_pr  oracle              [.] __intel_new_memset ?
            |
            --- __intel_new_memset
                |
                |--91.02%-- opiexe
                |          opiodr
                |          rpidrus
                |          skgmstack
                |          rpiswu2
                |          rpidrv
                |          psddr0
                |          psdexn
                |          pevm_EXECC
                |          pfrinstr_EXECC
      ...
                |          sou2o
                |          opimai_real
                |          ssthrdmain
                |          main
                |          __libc_start_main
                |
                |--3.91%-- audStatement
```

Example: Pin S, Call Graph

I did a Google search for __intel_fast_memset

Optimize memcpy

High performance versions of commonly used functions may exist, but perhaps not readily usable as a "drop in" replacement. For instance, Intel® C++ Compiler provides optimized versions of **memcpy**, **memset**, **memcmp**, and **memmove** inside the run-time library called libirc.a, although these versions have different names: **_intel_fast_memcpy, _intel_fast_memset, _intel_fast_memcmp**. The Intel® C++ Compiler automatically generates calls to these functions for optimized builds, but for non-optimized (**-O0**) builds calls the versions in **libc.so**. That's great when using the Intel® C++ Compiler, but adopting a different compiler can be a large task.

http://software.intel.com/en-us/articles/optimizing-without-breaking-a-sweat

Example: CBC - Oracle

```
SQL> @rtpctx
Remember: This report must be run twice so both the initial and
final values are available. If no output, press ENTER about 20 times.

Database: prod35                                                  18-OCT-13
Report:   rtpctx.sql              OSM by OraPub, Inc.               Page
                        Response Time Activity (50 sec interval)

                                                      Avg Time        Time
Time Component                          % TT    % WT Waited (ms)     (sec) Cour
--------------------------------------- ------- ------- ---------- ---------- ----
CPU consumption: Oracle SP + BG procs    99.04    0.00      0.000    299.015
latch: cache buffers chains               0.39   40.69      7.564      1.180
db file async I/O submit                  0.26   26.90     11.304      0.780
control file parallel write               0.11   11.72     21.250      0.340
log file redo write                       0.07    6.90     11.765      0.200
oracle thread bootstrap                   0.04    3.79     55.000      0.110
os thread creation                        0.01    1.38     20.000      0.040
commit: log file sync                     0.01    0.69     10.000      0.020
target log write size                     0.01    0.69      5.000      0.020
```

Example: CBC – fulltime.sh

```
$ fulltime.sh 32873 45

PID: 32873  SID: 9  SERIAL: 13  USERNAME: OE2  at 18-Oct-2013 12:13:49
CURRENT SQL: SELECT COUNT(*) FROM ( SELECT SUM(OBJECT_ID) FROM ORDERS UNION SELECT


total time: 44.438 secs,  CPU: 41.611 secs (93.64%),  wait: 2.827 secs (6.36%)

                                                                  Time
Time Component                                                    secs       %
----------------------------------------------------------------- ---------- -------
cpu : [.] kcbgtcr                                                 29.714    66.87
cpu : [.] kdstf00001[illegible]00001km                             3.716     8.36
cpu : [.] lnxsum                                                   3.541     7.97
cpu : [?] sum of funcs cons[illegible] less than 2% of CPU time    2.393     5.38
cpu : [.] kaf4reasrp0km                                            2.180     4.91
wait: PL/SQL lock timer                                            2.100     4.73
wait: latch: cache buffers chains                                  0.727     1.64
```

kernel, cache, buffer, get, CR

Back to reality... Useful when:

- Confirm the performance story. Example: If there is a parsing issue, I expect some type of SP/LC related function.
- During my research, it can help me understand what Oracle is doing "under the hood."
- Answering the question, "So what is Oracle doing with all that CPU?!" Especially when lots of CPU relative to wait time.
- CPU with little wait time. Therefore, no wait time clues.
- Potential Oracle bug detection.

Want to dig deeper--wider?

- **Presentations**: OraPub search, “time”
- **Craig’s Blog** – A Wider View
- **Training** from OraPub
  - Oracle Performance Firefighting (I)
  - Adv Oracle Performance Analysis (II)
  - Seminar: Go Faster! Make Oracle Work For You
- **Tools** at www.orapub.com
  - fulltime.sh. OP search, “fulltime”
  - OSM Toolkit. OP search, “osm”
  - Firefighting Diagnostic Template. OP search “ff diag”
- **Stori.** Find, understand and solve Oracle performance problems
- **Books**
  - Oracle Performance Firefighting.
  - Forecasting Oracle Performance.

Irvine, CA USA
December 9-13

Munich, Germany
December 19-20

Madrid, Span
December 16-17

Thank you for attending.

Questions?
Contact Craig at
craig@orapub.com - www.orapub.com

Get StoriFree at
http://storifree.com

stori
Find, Understand & Solve
Oracle Performance Problems

East Coast Oracle Users Conference

The most recent version of this presentation is on-line at www.orapub.com. Do an OraPub search for "CPU".

Understanding Oracle CPU Consumption: The Missing Link

OraPub

Craig A. Shallahamer
OraPub, Inc.
craig@orapub.com

LinkedIn XING