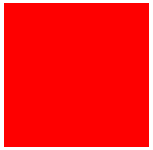
ORACLE®

ASH Outliers:
Detecting Unusual Events in Active Session History

John Beresniewicz, Oracle USA



Agenda

- ASH Fundamentals
- Wait Events and ASH
- Event Histograms
- Event Significance Levels
- SQL to Find Outliers in ASH
- Handling the GV$ problem

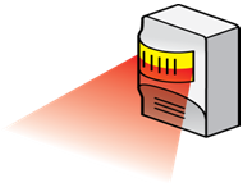
Motivating Use Case

- Unusually long wait events (outliers) suspected to trigger cascade-effect performance incidents
- RAC performance experts claim EM Top Activity not helpful as aggregation masks outlier events
- Can we see if ASH has sampled any such events?
  - If none observed does not mean they have not happened
    - However ASH sampling is biased to longer events

ASH Fundamentals

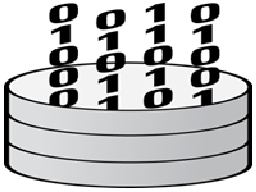
All 'Active' sessions captured every second

- Foregrounds and backgrounds are sampled
- Active foregrounds contribute to DB Time



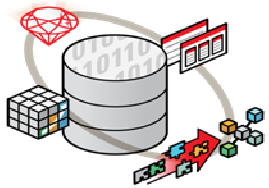
In-memory: *V$ACTIVE_SESSION_HISTORY*

- Sampling interval = 1 second
- Circular SGA buffer with latchless query access



On-disk: *DBA_HIST_ACTIVE_SESS_HISTORY*

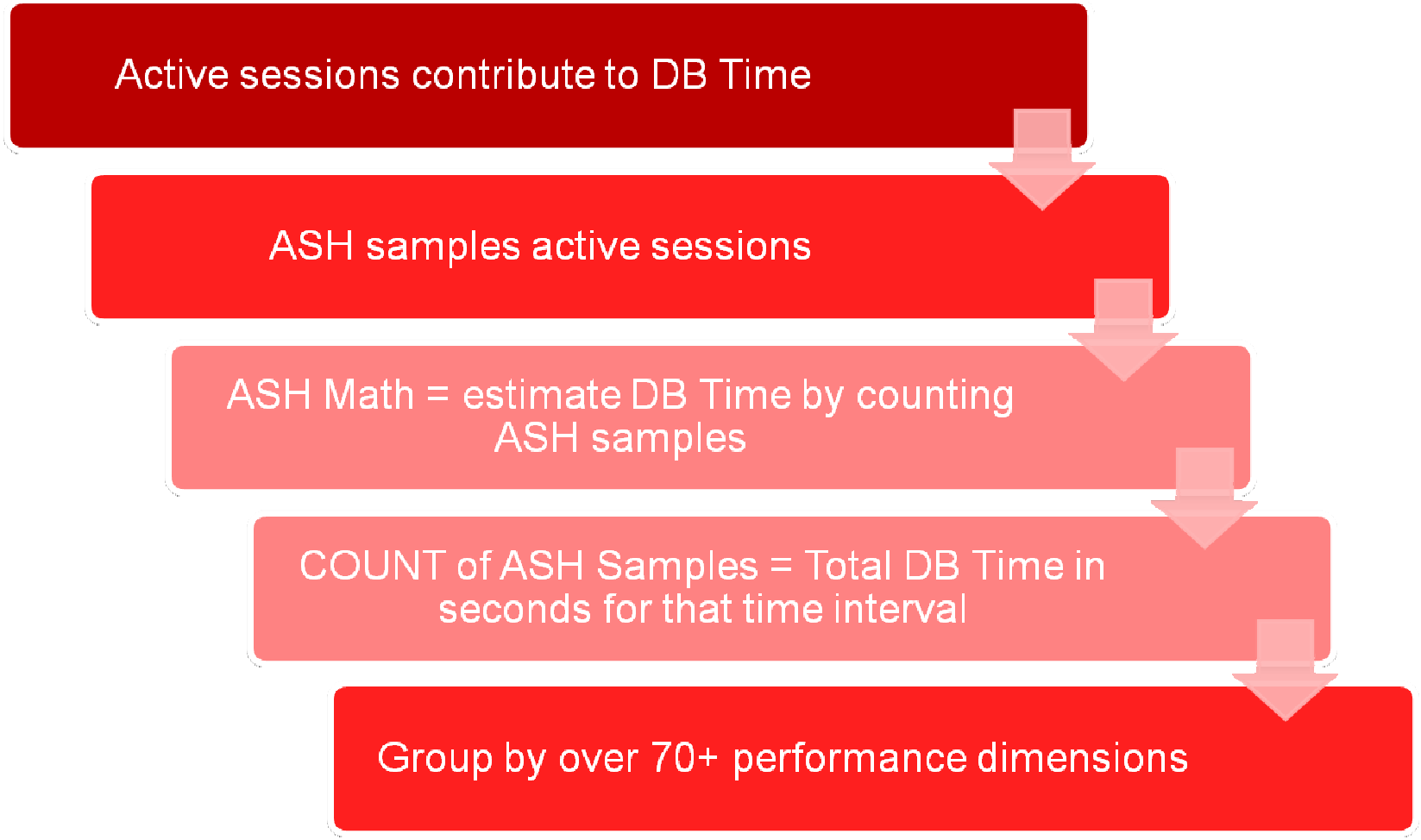
- Sub-sampling interval = 10 seconds



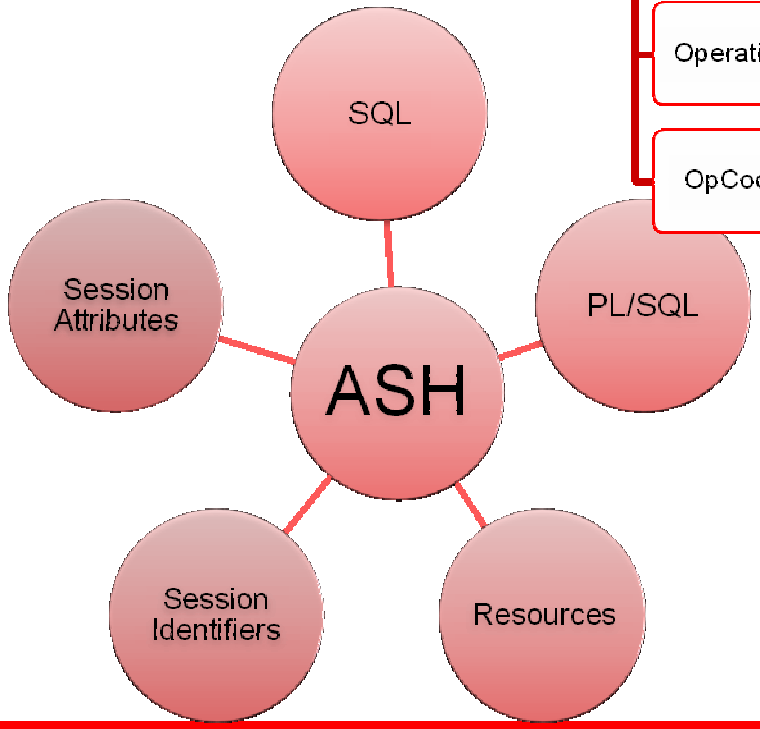
ASH is a system-wide record of database activity

- A **FACT** table with multiple dimensions that help diagnose performance issues

ASH and DB Time



ASH Fact Table Dimensions

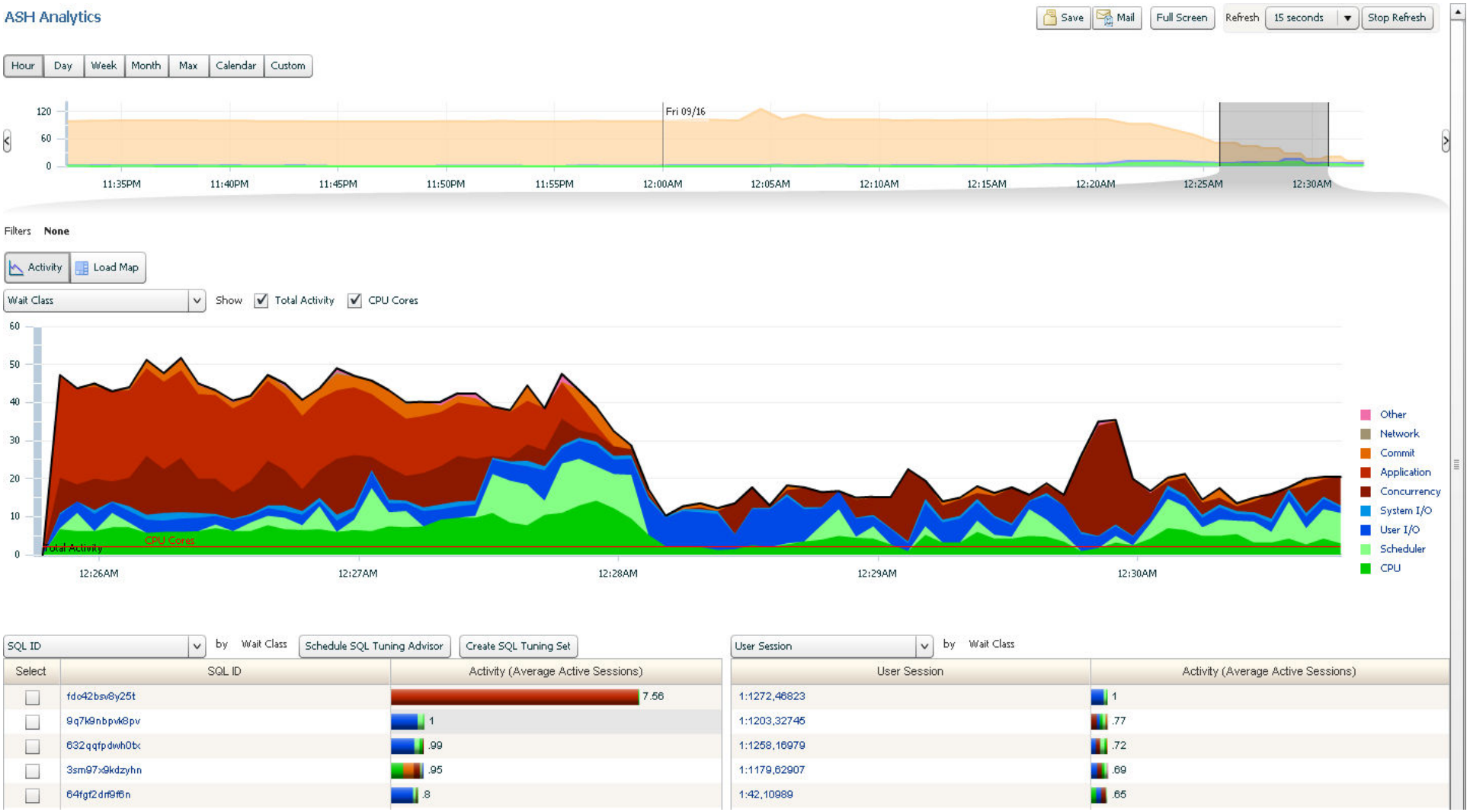


- **SQL**
  - SQL ID
  - Plan Hash
  - Operation
  - OpCode
- **PL/SQL**
  - PL/SQL
  - Top Level PL/SQL
- **Resources**
  - Wait Class
  - Wait Event
  - Object
  - Blocking Session
- **Identifiers**
  - Instance
  - Services
  - User Session
  - Parallel Process
  - Program
  - Session Type
- **Attributes**
  - Cons. Group
  - Module
  - Action
  - Client
  - Trans. ID

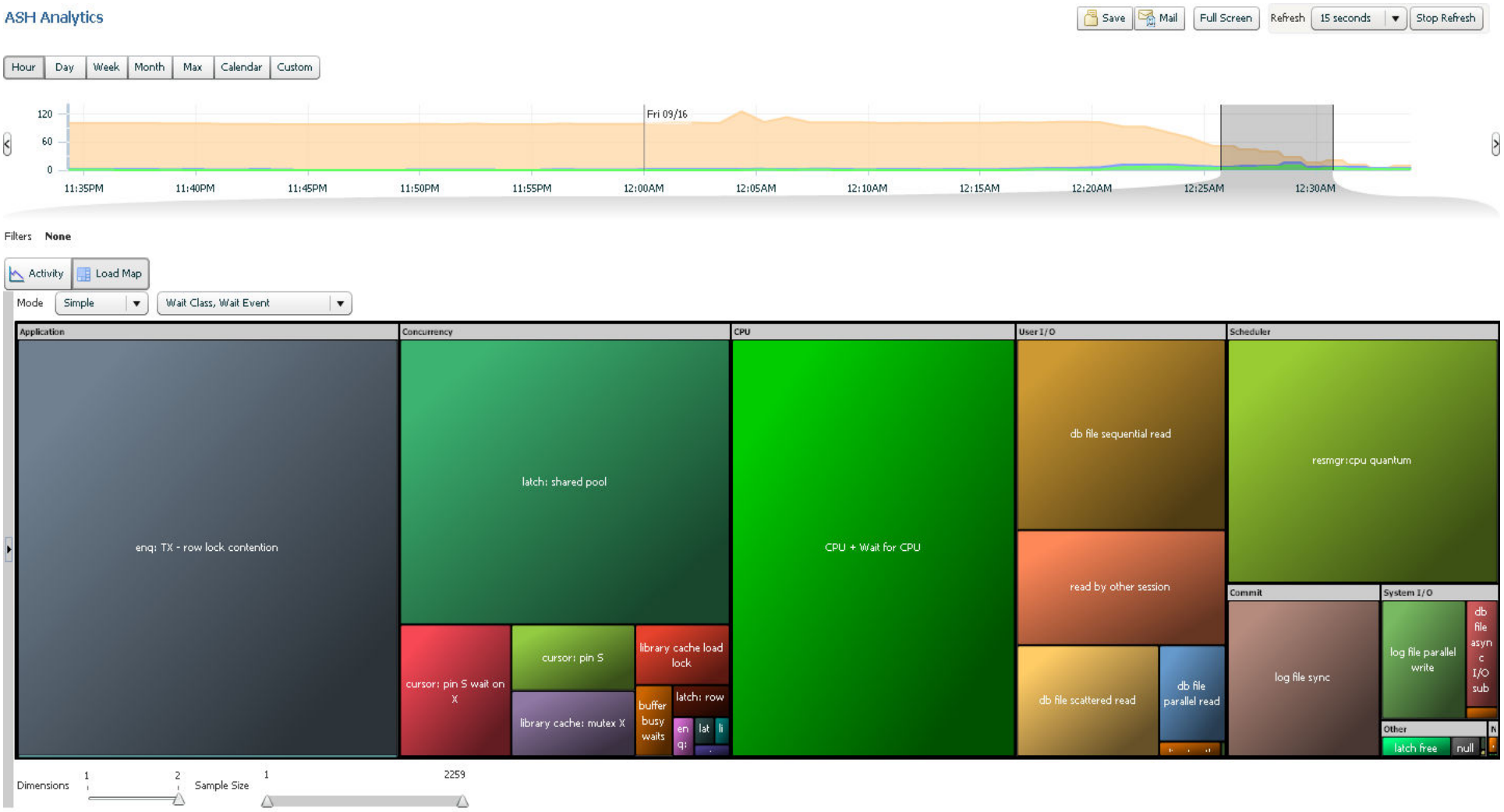
ASH Math

- COUNT(*) = DB TIME (secs)
  - Basic in-memory (V$ASH) formula
- The Kurtz Construct is nice
  - SUM(1) = DB Time (secs) for in-memory
  - SUM(10) = DB TIME (secs) for on-disk
- DB Time Method analysis:
  - Dimensional GROUP BY over COUNT(*)

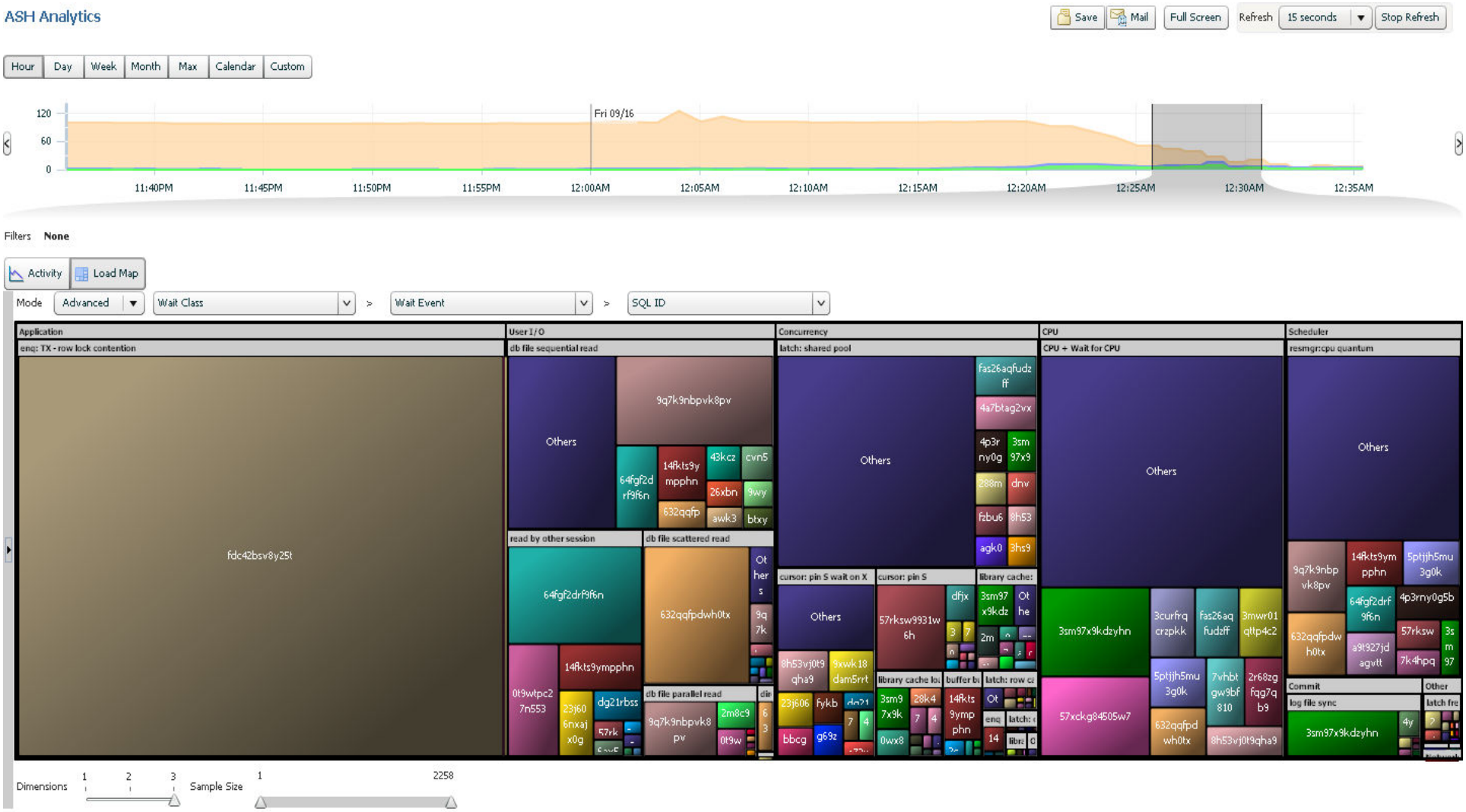
Good ASH Math: ASH Analytics



Group by Wait Event within Wait Class

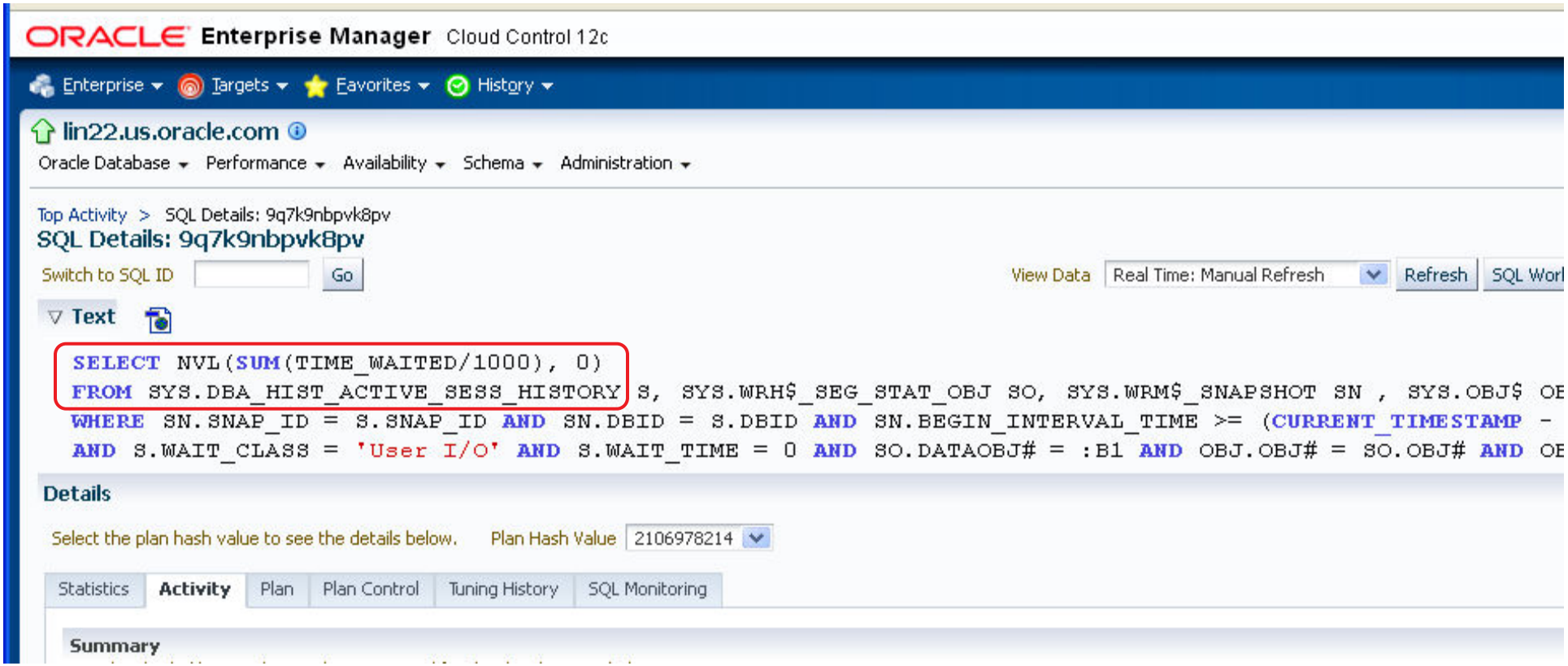


Add SQL_ID Grouping Dimension



Bad ASH Math

- SQL observed using 9 secs of CPU every 10 secs



Wait Events and ASH

- ASH samples actively waiting sessions
  - Wait time is unknown at sample time
  - The “fix up” writes actual wait time into TIME_WAITED
- ASH is a biased sampler of wait events
  - Longer events have higher probability of being sampled
- Avoid the temptation of TIME_WAITED
  - AVG(time_waited) DOES NOT estimate avg wait times
  - MIN and MAX do not work either
    - Except when MAX exceeds 1-second

The ASH “fix up”

- ASH columns may be unknown at sampling time
  - TIME_WAITED: session is still waiting
  - PLAN_HASH: session is still optimizing SQL
  - GC events: event details unknown at event initiation
  - Certain time model bit vector columns
- ASH “fixes up” missing data during subsequent sample processing
  - TIME_WAITED fixed up in last event sample
- Querying current ASH may return un-fixed rows
  - Should not be a problem generally

ON CPU and ASH

- ASH row status “ON CPU” derived, not observed
  - Session is in a database call
  - Session is NOT in a wait event (idle or non-idle)
- Un-instrumented waits => “ON CPU”
  - These are bugs and should be rare, but have happened
- Session on run queue may be WAITING or ON CPU
  - Depends on state prior to going onto run queue

V$EVENT_HISTOGRAM

- Histogram buckets of event wait times
- Captures statistical distribution of wait times
- All events since instance startup counted in some bucket
- Exponential time bucketing scheme captures long-tail distributions efficiently

V$EVENT_HISTOGRAM

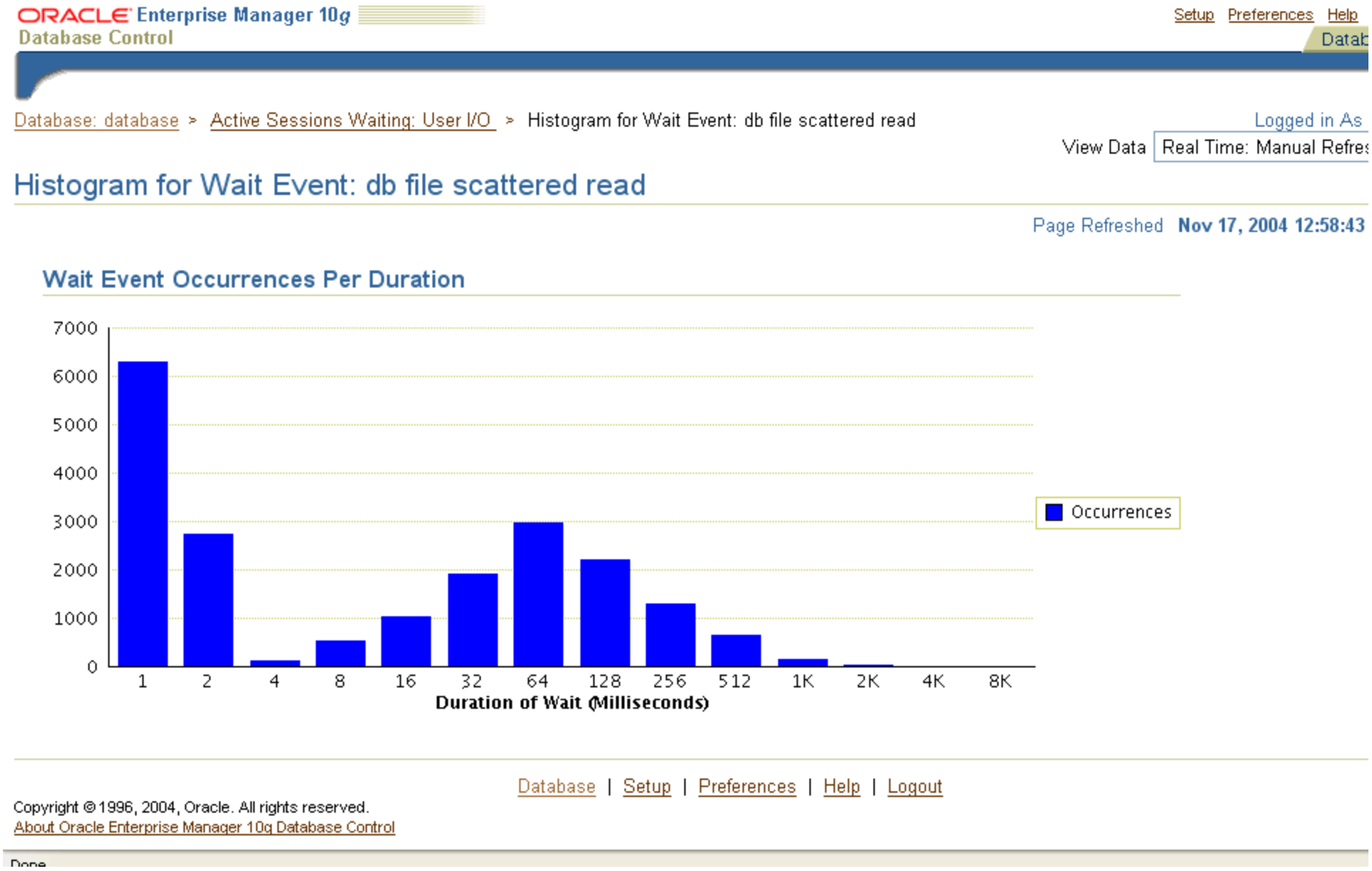
```
SQL> desc v$event_histogram
 Name                           Type
 ------------------------------ ------------------------------
 EVENT#                         NUMBER
 EVENT                          VARCHAR2(64)
 WAIT_TIME_MILLI                NUMBER
 WAIT_COUNT                     NUMBER
 LAST_UPDATE_TIME               VARCHAR2(64)
```

Event Histogram Time Buckets

```
SQL> select distinct
            wait_time_milli
           ,log(2,wait_time_milli)
     from
          v$event_histogram
     order by 1;
```

WAIT_TIME_MILLI	LOG(2,WAIT_TIME_MILLI)
1	0
2	1
4	2
8	3
16	4
32	5
64	6
128	7
256	8
512	9
1024	10

I/O Event Histogram



Latch Wait Event Histogram

ORACLE Enterprise Manager 10g
Database Control

Setup Preferences Help
Datab

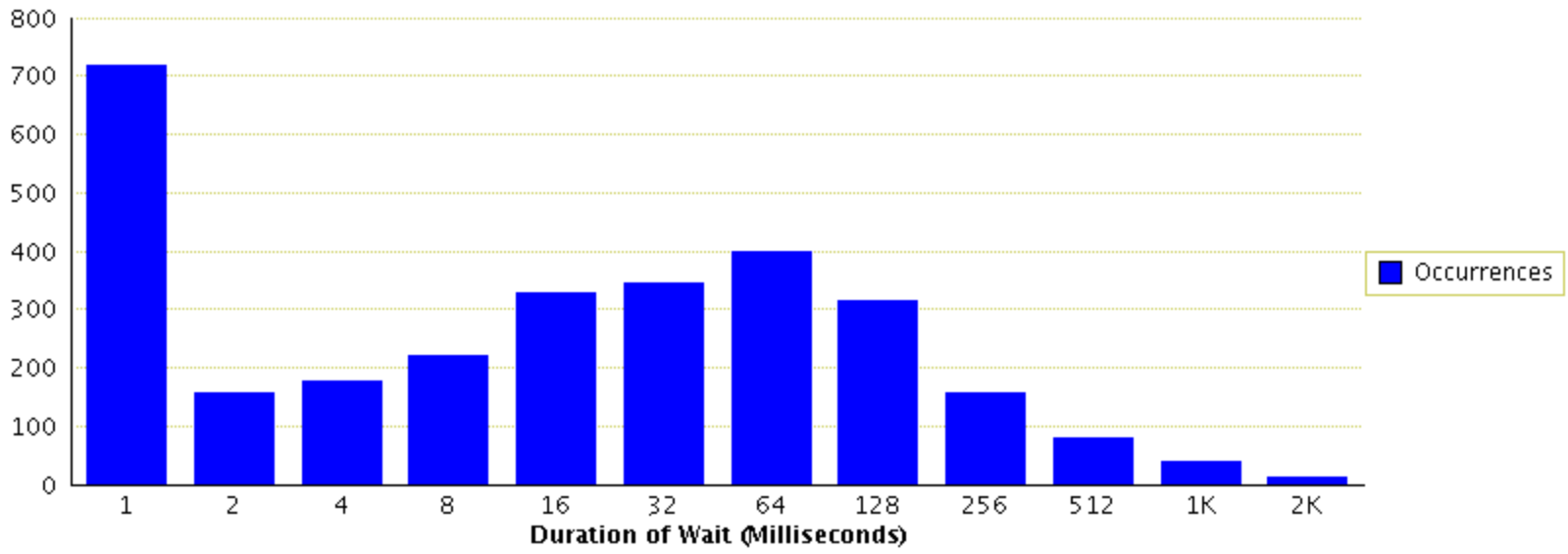
Database: database > Active Sessions Waiting: Concurrency > Histogram for Wait Event: latch: library cache

Logged in As
View Data Real Time: Manual Refres

Histogram for Wait Event: latch: library cache

Page Refreshed **Nov 17, 2004 1:01:36**

Wait Event Occurrences Per Duration



Database | Setup | Preferences | Help | Logout

Copyright © 1996, 2004, Oracle. All rights reserved.
About Oracle Enterprise Manager 10g Database Control

Done

Histogram Math

- Histograms capture probability distribution of TIME_WAITED by event over this startup cycle

$$\Pr(time\_waited < bucket_N) = \sum_{bucket<N} WaitCount \Big/ \sum_{allbuckets} WaitCount$$

Significance of Histogram Buckets

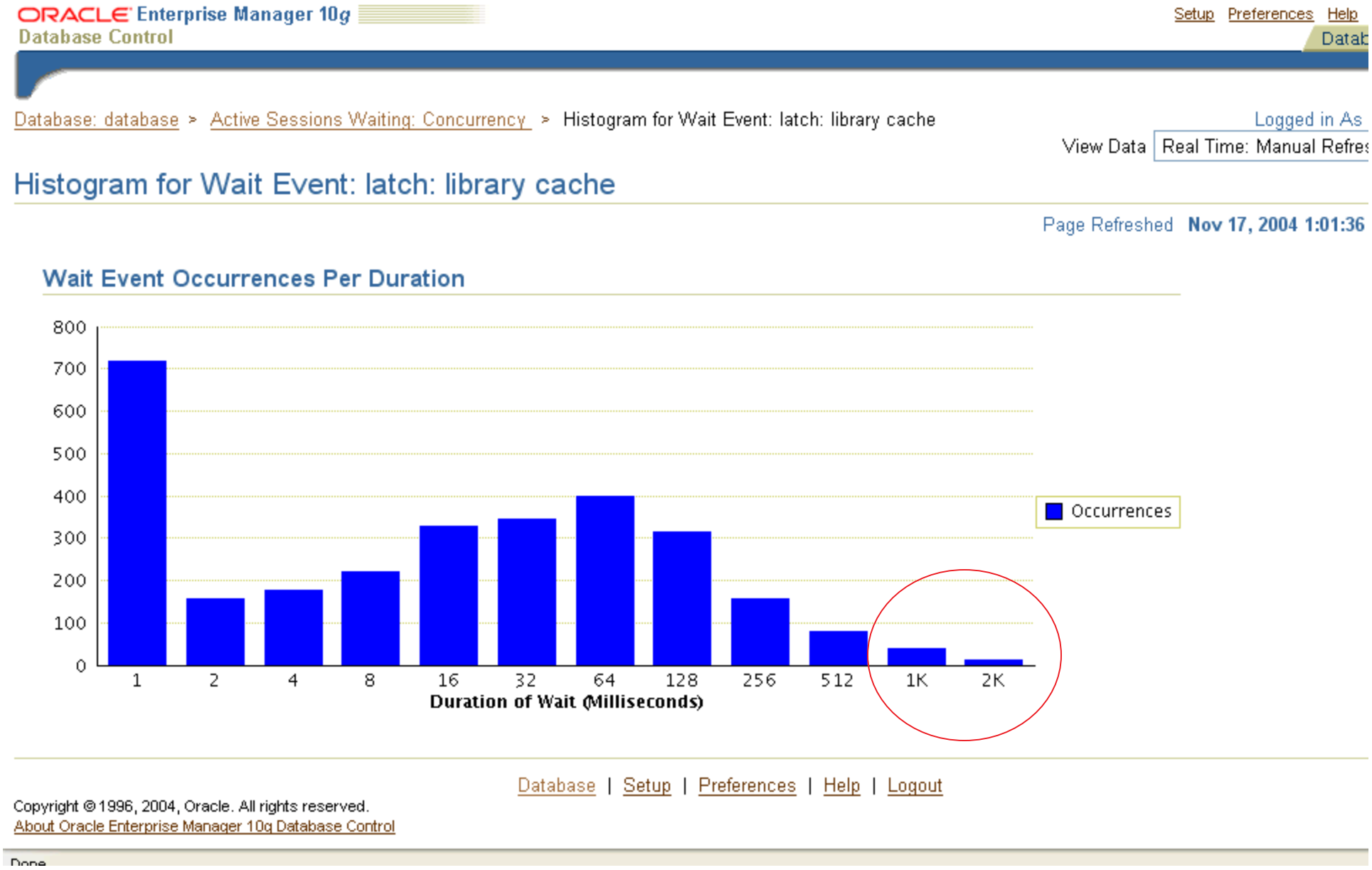
$$Significance_{bucketN} = 1 - \left(\sum_{bucket>=N} WaitCount \Big/ \sum_{allbuckets} WaitCount \right)$$

- Measures the cumulative distribution function of TIME_WAITED probabilities represented by the histograms (per bucket)
- Every event in the bucket has at least this significance

Defining “Outlier Events”

- Events with low probability of occurrence
- Events with high significance value
- Q: Has ASH sampled any such events?

“Outlier” = “Unusual”



Finding Outlier Events in ASH

- Which ASH rows (if any) represent wait events with significantly long TIME_WAITED against the event histogram record?

- Two step process:

1. Compute event histogram bucket significance
2. Join ASH to histograms and filter by significance

Step 1: Compute Bucket Significance

```
WITH EH$stats
as
(select
       EH.*
      ,ROUND(1 - (tot_count - bucket_tot_count + wait_count) / tot_count,6)
                                                          as event_bucket_siglevel
from
    (select event#
            ,event
            ,wait_time_milli
            ,wait_count
            ,ROUND(LOG(2,wait_time_milli))                 as event_bucket
            ,SUM(wait_count) OVER (PARTITION BY event#)    as tot_count
            ,SUM(wait_count) OVER (PARTITION BY event# ORDER BY wait_time_milli
                                                          RANGE UNBOUNDED PRECEDING)
                                                           as bucket_tot_count
       from  v$event_histogram
     ) EH
)
```

Step 2: Join ASH to Buckets and Filter

```
select
      EH.event_bucket
     ,ASH.sample_id
     ,ASH.session_id
     ,EH.event_bucket_siglevel as bucket_siglevel
     ,ASH.event
     ,ASH.time_waited/1000 ASH_time_waited_milli
     ,ASH.sql_id
 from
       EH$stats  EH
      ,v$active_session_history ASH
 where
      EH.event# = ASH.event#
  and EH.event_bucket_siglevel > &siglevel
  and EH.event_bucket = CASE ASH.time_waited
                            WHEN 0 THEN null
                            ELSE TRUNC(LOG(2,ASH.time_waited/1000))+1
                        END
order by
sample_id, event, session_id
;
```


DEMO?
SOFTWARE.
HARDWARE.
COMPLETE.
TAPE
AC
DC
STANDBY
XSCF
ORACLE

The GV$ Problem

- Motivating use case is for RAC
- Execute V$ query on all instances?
  - Too much effort
- Convert V$ query to GV$ query?
  - GV$ remote joins not optimized
    - QC will pull V$ASH and V$EVENT_HISTOGRAM and join locally
- Neither of these “solutions” is acceptable

GV$ Table Function

- Table function distributes V$ cursor across RAC nodes and marshals result sets
- Perfect solution for this use case

```
SELECT … FROM TABLE GV$( (CURSOR( SELECT FROM V$)))
```


DEMO?
SOFTWARE.
HARDWARE.
COMPLETE.
TAPE
AC
DC
STANDBY
XSCF
ORACLE

Comments and Caveats

- Highly significant events may not be sampled
- Works best for long-tailed distributions
  - Bi-modal, single-bucket, timeout events not well-behaved
- Exponential bucket sizing gets coarse quickly
  - Significance levels increase in big jumps
  - Important distinctions may hide inside large buckets
- An interesting and unusual application of ASH where TIME_WAITED is the key
  - Does it help with the motivating use case?