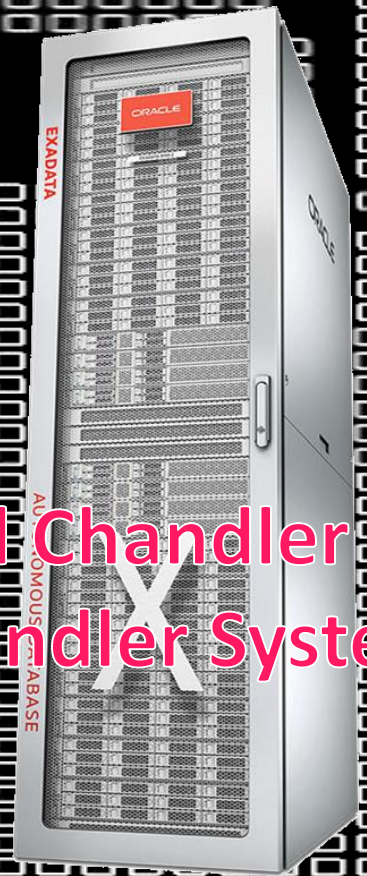
# GET ME OFF THAT F*CKING EXADATA



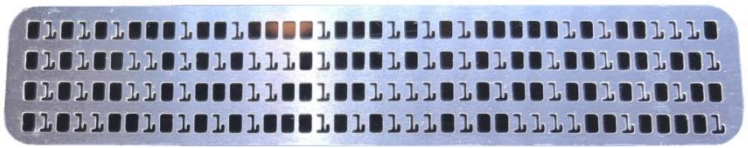

Neil Chandler
Chandler Systems

ITOUG
ITALIAN ORACLE USER GROUP

# GET ME OFF THAT F*CKING EXADATA


**Neil Chandler** Oracle ACE Director

**Chandler Systems**

MASH PROGRAM

https://mashprogram.wordpress.com

Talk relates to 19C and later versions

SYM42

https://sym42.org

# GET ME OFF THAT EXADATA

## Exadatas are Magic

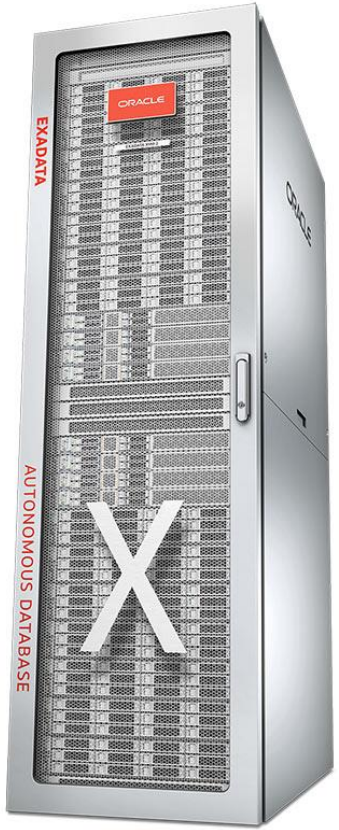
Me? In the pub? With Martin?
About 2009… around the release of **11.2**

Peter Scott
Data Warehouse Expert

"I'm working on a Exadata

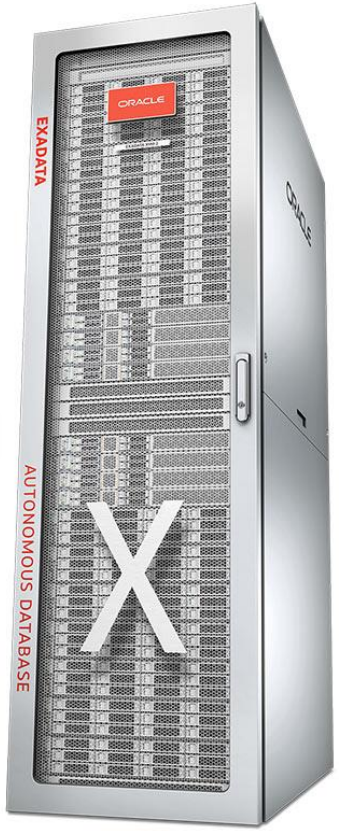
I table scanned 2,000,000,000 rows, about ½ TB….

… in 2 seconds
on a table with no indexes!



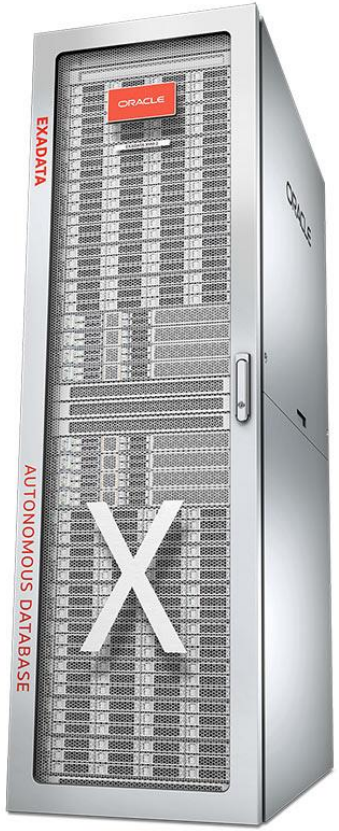
# GET ME OFF THAT EXADATA

## Exadatas are Magic



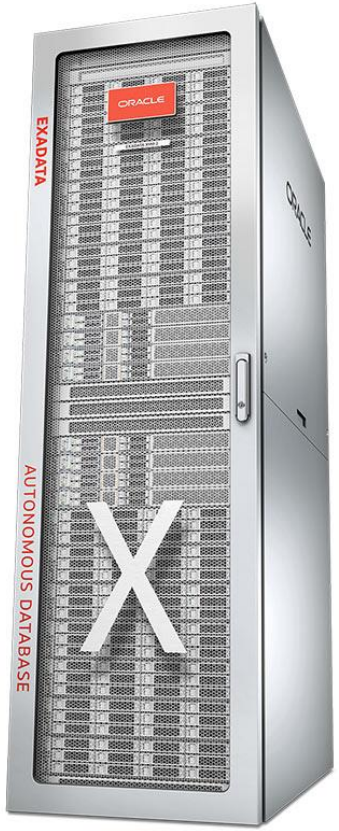
# GET ME OFF THAT EXADATA

## Exadatas are Magic

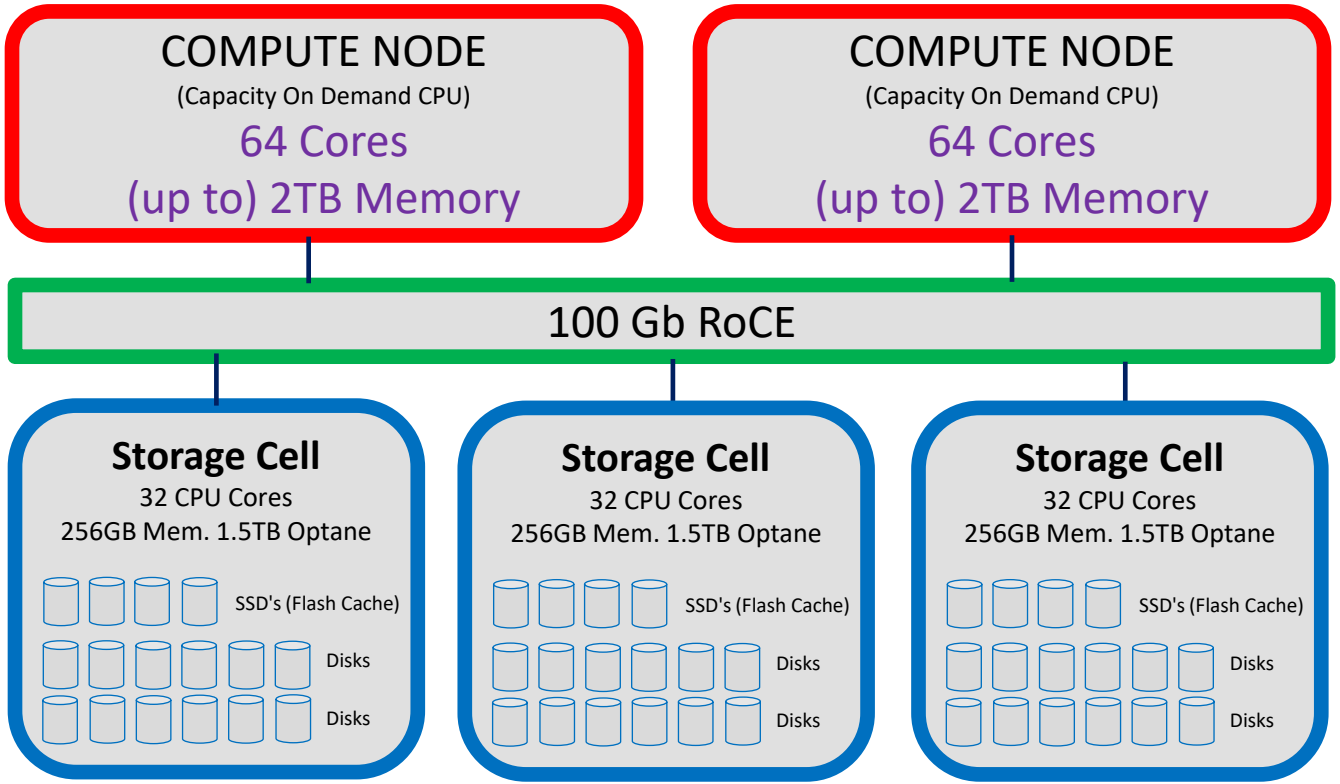
# GET ME OFF THAT EXADATA

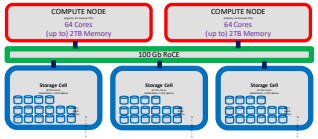
## What is an Exadata?



# GET ME OFF THAT EXADATA

Typical ¼ Rack X9M



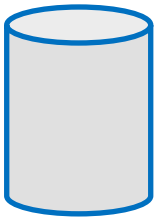



# MAGICAL DISKS

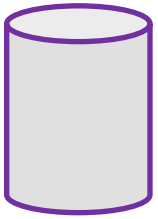
**Storage Cell (HC)**
32 CPU Cores
256GB Mem. **1.5TB Optane**
SSD's (Flash Cache)
Disks
Disks

**Storage Cell (EF)**
8x SSD's, no standard disks

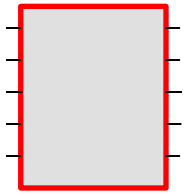
**Storage Cell (XT)**
12x disks, no flash, no Optane, no magic



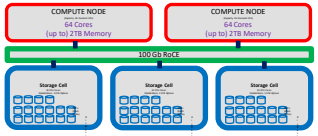
Disk - high capacity (18TB). 1,000 IOPS
(slow - several ms)



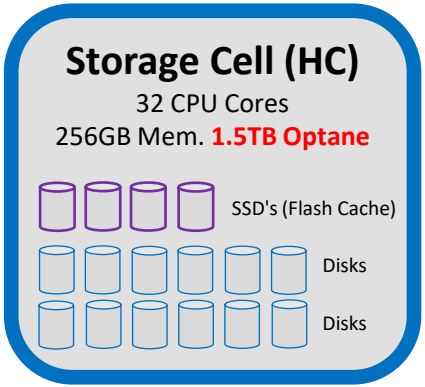
SSD - 0.5ms (6TB). 100,000 IOPS
(Oracle say 0.2ms)



Optane - 150µs (0.15ms). 1,600,000 IOPS
(Oracle say less than 19µs - 0.019ms)




# MAGICAL DISKS




**Storage-Based Filtering**
- **Minimises data returned to DB nodes**
- **Processing of predicates / simple joins**

**Automatic Storage-Level Indexes**

**Offload of HCC Uncompress & Scanning Encrypted Data**

**Lots of other bits and pieces (some useful, some trivial but nice)**

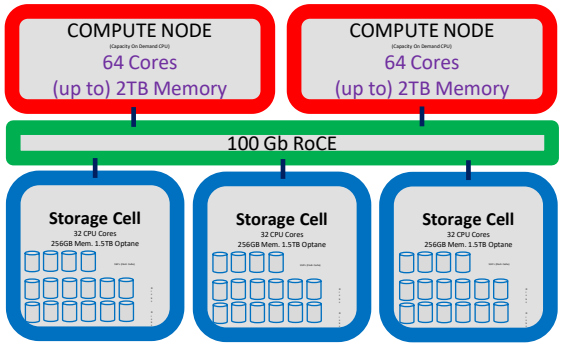
**Exadata and Database Software Features – Analytics**
- Unique Automatic Parallelization and Offload of Data Scans to storage
- Unique Filtering of Rows in Storage based on 'where' clause
- Unique Filtering of Rows in Storage based on columns selected
- Unique Storage Offload of JSON and XML Analytic Queries
- Unique Filtering of rows in Storage based on Join with other Table
- Unique Hybrid Columnar Compression
- Unique Storage Index Data Skipping
- Unique I/O Resource Management by User, Query, Service, DB, etc.
- Unique Automatic Transformation to Columnar Format in Flash Cache
- Unique Smart Flash Caching for Table Scans
- Unique Storage Offload of Index Fast Full Scans
- Unique Storage Offloads of Scans on Encrypted Data, with FIPS compliance
- Unique Storage Offload for LOBs and CLOBs
- Unique Storage Offload for min/max operations
- Unique Bloom Filter Offload to Storage
- Unique Reverse Offload to DB servers if Storage CPUs are Busy
- Unique Automatic Data Columnarization
- Unique Automatic Conversion of Data to In-Memory Formats when Loading into Flash Cache

**Exadata and Database Software Features – OLTP**
- Unique Persistent Memory Data Accelerator
- Unique Persistent Memory Commit Accelerator
- Unique Database Aware PCI Flash
- Unique Exadata Smart Flash Caching
- Unique Exadata Smart Flash Logging
- Unique Smart Write-back Flash Cache
- Unique I/O Prioritization by cluster, workload, DB or user to ensure QOS
- Unique Exafusion Direct-to-Wire Protocol
- Unique Database Intelligent Network Resource Management
- Unique Exachk full-stack validation
- Unique Full-stack security scanning
- Unique Database scoped security
- Unique Cell-to-Cell Rebalance preserving Flash Cache and Storage Index
- Unique Full-Stack Secure Erase
- Unique Instant Data File Creation
- Unique Smart Fusion Block Transfer
- Unique Control of Flash Cache Size per Database
- Unique In-Memory OLTP Acceleration
- Unique Undo-Block Remote RDMA Read
- Unique Support for 4000 Pluggable Databases per Container Database with Multitenant Option

**Exadata and Database Software Features – High Availability**
- Unique Instant Detection of Node or Cell Failure
- Unique In-Memory Fault Tolerance
- Unique Sub-second Failover of I/O on stuck disk or Flash
- Unique Offload backups to storage servers
- Unique Exadata Data Validation (extended H.A.R.D.)
- Unique Prioritize Recovery of Critical Database Files
- Unique Automatic Repair of Corrupt Disk Data By Reading Other Storage Servers
- Unique Avoidance of Read I/Os on Predictive failed disks
- Unique Confinement and power cycle of temporarily poor performing drives
- Unique Shutdown Prevention If Mirror Storage Server is Down
- Unique Detection and Disabling of Unreliable Network Links
- Unique Preservation of Storage Index on Rebalance

# GET ME OFF THAT EXADATA

Typical ¼ Rack X9M




# What's the Problem?

**The cost… € £ $ zł**

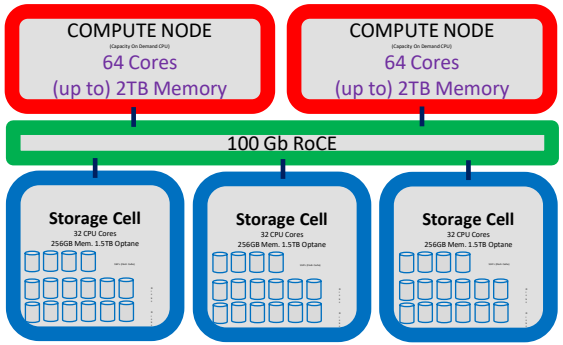
| | | |
|---|---|---|
| Hardware | : | **[€0.5m]** |
| EE DB | : €44,175 x64 Cores | **[€2.8m]** |
| RAC | : €21,390 x64 Cores | **[€1.4m]** |
| Storage Cell licenses | : € 9,300 **x36 disks** | **[€0.3m]** |

Total: **€5.0m**
+ 22% per annum Support

And you really should have
Prod, DR and Non-Production, so…

# GET ME OFF THAT EXADATA

Typical ¼ Rack X9M




# What's the Problem?

**The cost… *€ £ $ zł***

3 of these… **€15m**

And you probably need

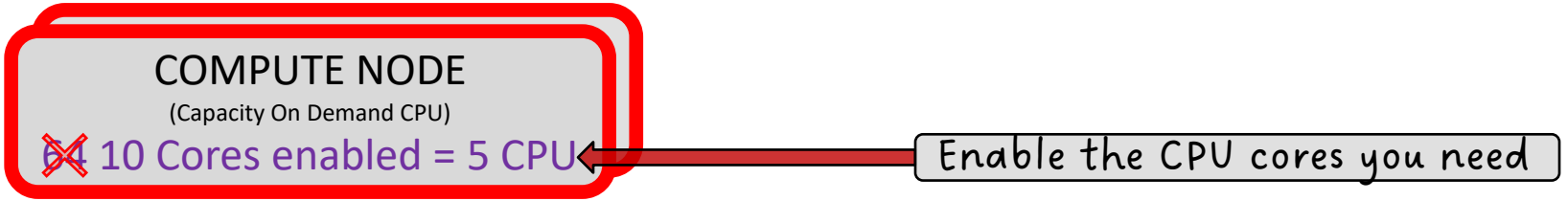
| | |
|---|---|
| Tuning | [€4,650 x 64 x 3] = **€0.8m** |
| Diagnostics | [€4,650 x 64 x 3] = **€0.8m** |
| Partitioning | [€10,695 x 64 x 3] = **€2.0m** |
| Advanced Security (TDE) | [€13,950 x 64 x 3] = **€2.6m** |
| Active Data Guard | [€10,695 x 64 x 3] = **€2.0m** |

**€23.2m** for 3 x fully licensed QUARTER RACKs with options!

*(but you will get a discount)*

# COST IS MAINLY LICENSES!

Capacity On Demand - can we reduce that **€23m** bill?




| | | |
|---|---|---|
| Hardware | | **€500k** |
| Storage Cell CPU licenses | [€ 9,300 x 36] | **€334k** |
| EE DB | [€44,175 x 10] | **€441k** |
| RAC | [€21,390 x 10] | **€213k** |
| Tuning | [€ 4,650 x 10] | **€ 46k** |
| Diagnostics | [€ 4,650 x 10] | **€ 46k** |
| Partitioning | [€10,695 x 10] | **€106k** |
| Advanced Security (TDE) | [€13,950 x 10] | **€139k** |
| Active Data Guard | [€10,695 x 10] | **€106k** |

Total: **€2131k x *40% discount* = €1278k x 3 = €3.8m**

+ 22% support on purchase price

# THAT’S STILL A LOT OF CASH!

**€3.8m!!!!!**



Multiple exclamation marks,' he went on, shaking his head, 'are a sure sign of a diseased mind.

# EXADATA REPLACEMENT

"We need to replace some end-of-life Exadata's"


How much?

F**K!!!!!

Give me some

f***ing options!

# OPTIONS

1. Stop using the apps and close the business?
2. Stay with Oracle, but move to commodity hardware
3. Migrate to PostgreSQL, on commodity hardware
4. Cloud *(AWS / Azure / OCI)*
5. Just buy the f**ing Exadatas!

We will sacrifice scalability, resilience, throughput, and/or performance if we **stop** using Exa's

# COMMODITY HARDWARE

to specify the commodity hardware
we need to understand what the Exadata is doing

3 Resource Metrics:

CPU - It's an easy** metric to understand *(if < 80%)*

MEMORY - what's allocated. Its mostly fixed size...

I/O - A little trickier. We need to work on this

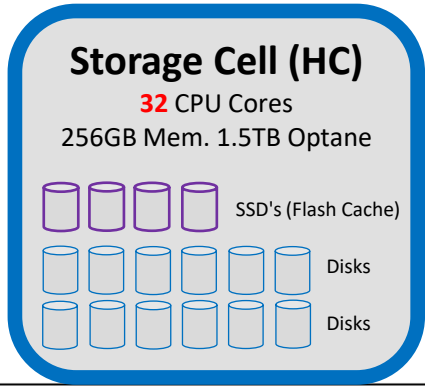
** the storage cells have CPU's in them…

# WHAT ABOUT CPU?

Each Storage Cell
has 32 CPU Cores

we must take that into account

*(why? Isn't that just I/O?)*

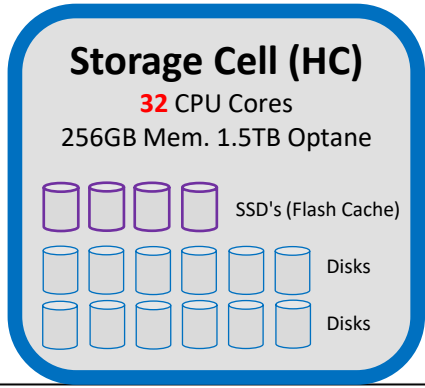


# WHAT ABOUT CPU?

The system I was measuring had 5 (relevant) storage cells.

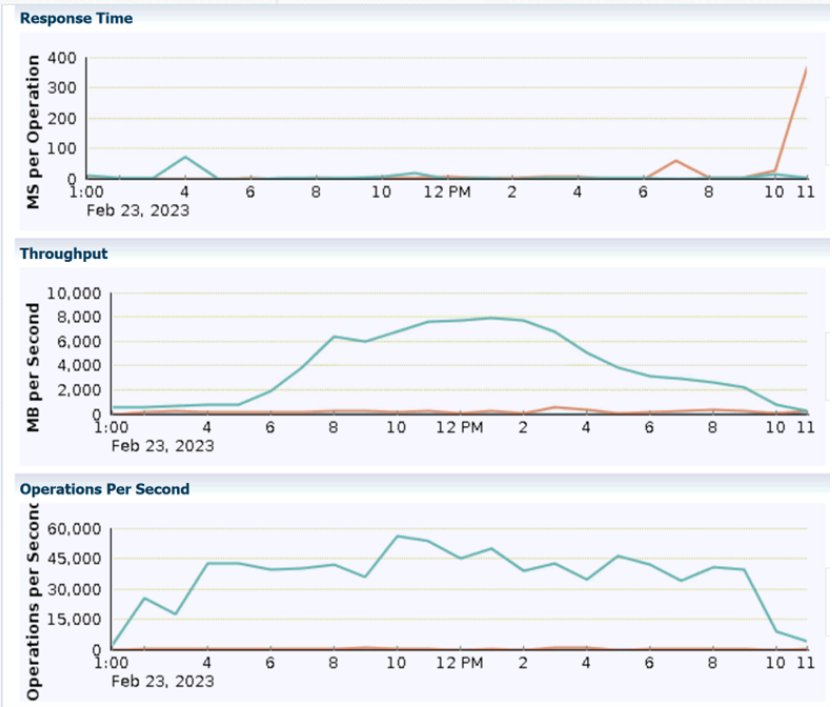
CPU utilisation averaged 35%

(32 * 5) * 35% = up to 58 CPUs



# MEASURING THE I/O

## Let's start with OEM



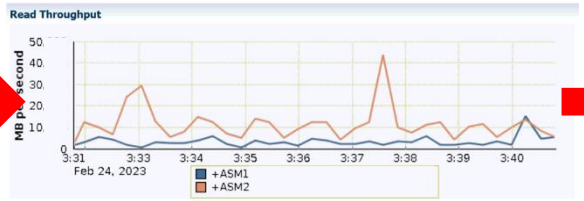

OEM has problems

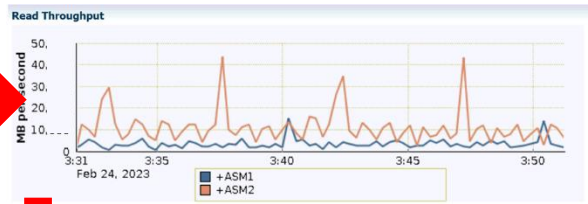
- The scale is dynamic and not controllable
- The granularity is low - hourly here
- which means potentially misleading averages

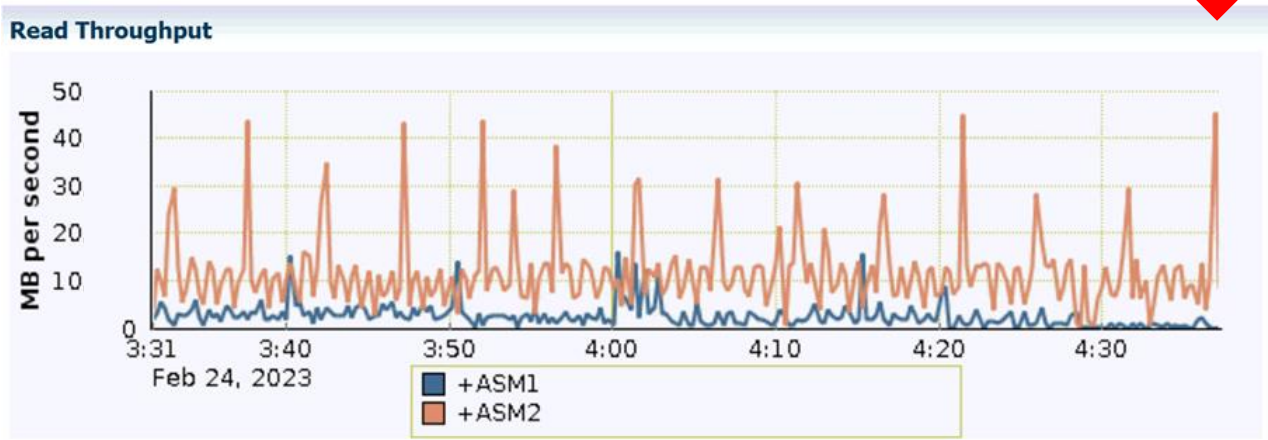
The average over an hour of just under 8GB/s could be 80GB/s for 5 important minutes and 1GB/s for 55 minutes

# MEASURING THE I/O

## Real-Time Refresh in OEM has benefits


- NOT reproducible
- Scale Changes
- Hit F5 and its gone

so

Real-time is NOT reliable/repeatable

Historic Views are NOT granular

# MEASURING THE I/O

So, do it yourself, using **V$ASM_DISK_IOSTAT** *ON EACH NODE!*

Shows cumulative information, so capture, store and calc

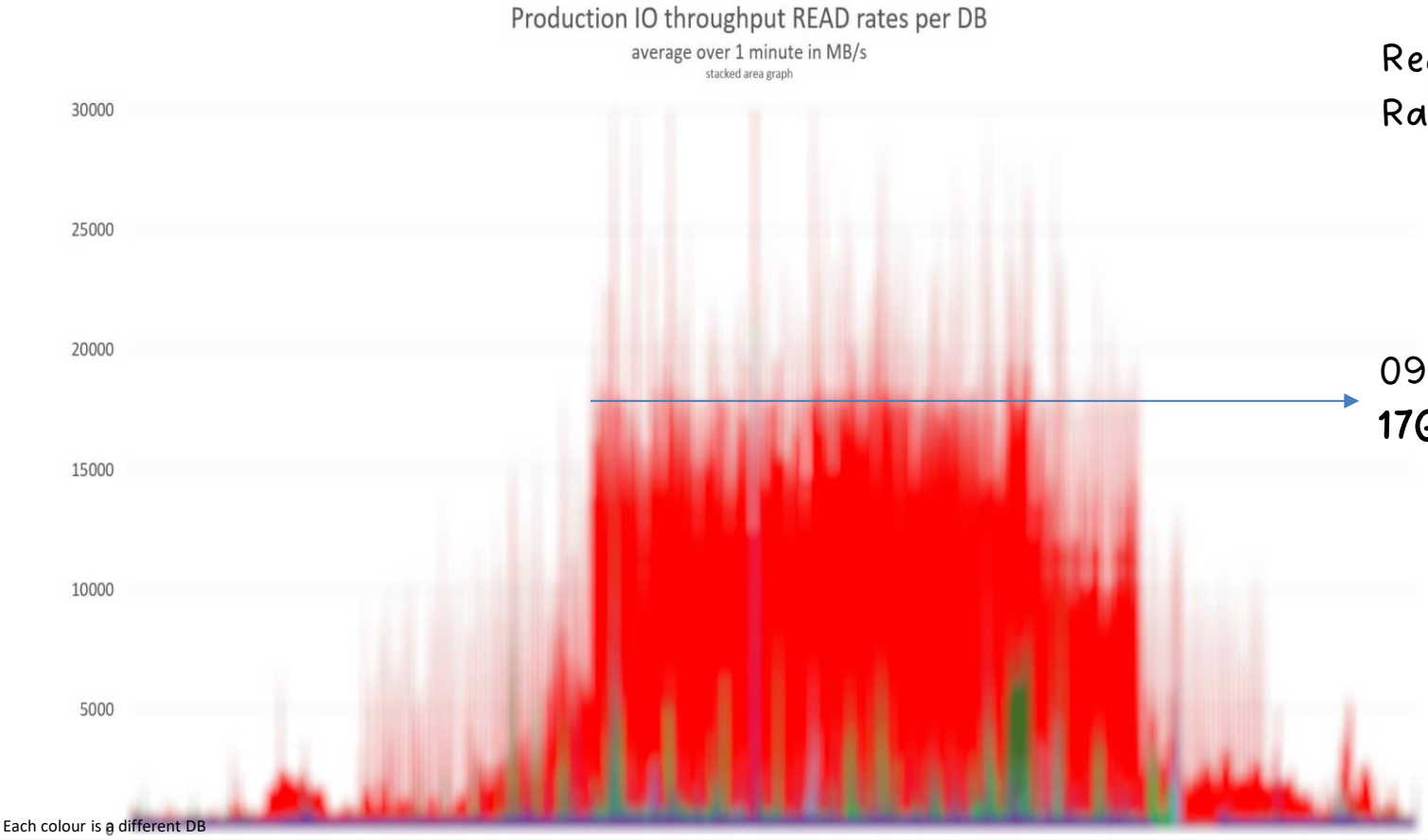
```
SELECT dbname||':'||to_char(max(sysdate),'YYYY-MM-DD HH24.MI.SS')
            ||':'||round(sum(BYTES_READ)/1024/1024)
            ||':'||round(sum(BYTES_WRITTEN)/1024/1024)
  FROM v$asm_disk_iostat
 WHERE dbname = 'ORCL'
 GROUP BY dbname;
```

**Get initial bytes**
**loop**
**Wait**
**Get new bytes**
**Calc: (new-old)/wait and save it**
**End-loop**
**Graph the output in your tool of choice**

Example output:

```
[grid@                asm_activity]$ head asm_activity.ORCL.out
DT:DB-ORCL:Read MB/s averaged over 60:Write MB/s averaged over 60
2023-02-07 14.04.42:ORCL:[illegible]
2023-02-07 14.05.42:ORCL:6726:246
2023-02-07 14.06.42:ORCL:5896:233
2023-02-07 14.07.42:ORCL:6793:135
2023-02-07 14.08.42:ORCL:6217:104
2023-02-07 14.09.42:ORCL:6105:104
2023-02-07 14.10.42:ORCL:6697:281
2023-02-07 14.11.43:ORCL:7078:1301
2023-02-07 14.12.43:ORCL:5675:1671
[grid@                asm_activity]$
```

# MEASURING THE I/O




Regular Peaks: **30+GB/s**
Rare Peaks: **55-60GB/s**

09:00-18:00 Average: **17GB/s**

# MEASURING THE I/O

This is the MINIMUM requirement for a commodity storage array

we must take into account Exadata disk **magic**

# MEASURING the MAGIC

High-level measurements first, from AWR:

`DBA_HIST_SYSSTAT`

What I/O are we avoiding, on average, every day?

# MEASURING the MAGIC

```
WITH dhs_values
    as (SELECT /*+ MATERIALIZE */ to_char(dhsnap.begin_interval_time,'YYYY-MM-DD') begin_date,to_char(dhsnap.begin_interval_time,'DY') begin_day,dhs.STAT_NAME ,sum(dhs.value) value
  FROM dba_hist_sysstat  dhs INNER JOIN
       dba_hist_snapshot dhsnap ON (dhs.snap_id = dhsnap.snap_id and dhs.dbid=dhsnap.dbid and dhs.instance_number=dhsnap.instance_number)
 WHERE dhs.stat_name IN ('cell physical IO bytes eligible for predicate offload'
                        ,'physical read total bytes'
                        ,'cell physical IO bytes saved by storage index'
                        ,'cell physical IO interconnect bytes returned by smart scan'
                        ,'cell IO uncompressed bytes'
                        ,'physical write bytes')
   -- ensure we select only runs in the first minute of the day
   -- e.g. 2023-01-12 00:00:23 will match below ensuring we only get 1 run in the figures for the day!
   AND trunc(dhsnap.begin_interval_time) = trunc(dhsnap.begin_interval_time,'MI')
 GROUP BY to_char(dhsnap.begin_interval_time,'DY'),to_char(dhsnap.begin_interval_time,'YYYY-MM-DD'), dhs.STAT_NAME
ORDER BY 1,2),

dhs_change as (select dv.begin_day,dv.begin_date,dv.stat_name,dv.value-LAG(dv.value) OVER (PARTITION BY dv.stat_name ORDER BY dv.stat_name,dv.begin_date,dv.begin_day) value_change  from dhs_values dv order by begin_date,stat_name)

select sys_context('userenv','db_unique_name') DB
      ,begin_day
      ,begin_date
      ,round(physical_reads/1024/1024/1024)                                                 PHYS_READ_GB
      ,round(eligible_offload/1024/1024/1024)                                               eligiBLE_OFFLOAD_GB
      ,round(interconnect_returned_ss/1024/1024/1024)                                       INTERCONNECT_RETURNED_GB
      ,round((eligible_offload/decode(physical_reads,0,1,physical_reads))*100)              eligiBLE_OFFLOAD_PCT
      ,round(100-(interconnect_returned_ss/decode(eligible_offload,0,1,eligible_offload))*100) OFFLOAD_SAVED_PCT
      ,round(sindx_saved/1024/1024/1024)                                                    SAVED_BY_STORAGE_INDEX
      ,round(IOU/1024/1024/1024)                                                            UNCOMPRESSED_GB
      ,round((interconnect_returned_ss/decode(iou,0,1,iou))*100)                            RETURN_UNCOMPRESSED_PCT
      --,round(PW/1024/1024/1024)                                                          PHYS_WRITE_GB
 from dhs_change dc
PIVOT
(min(dc.value_change) for (stat_name) in
 (
 'cell physical IO bytes eligible for predicate offload'      eligible_offload,
 'physical read total bytes'                                  physical_reads,
 'cell physical IO bytes saved by storage index'              sindx_saved,
 'cell physical IO interconnect bytes returned by smart scan' interconnect_returned_ss,
 'cell IO uncompressed bytes'                                 iou,
 'physical write bytes' PW
 )
)
where begin_date >= '2023-03-01'
order by begin_date
/
```

# MEASURING the MAGIC

## DBA_HIST_SYSSTAT

| DB | BEG | BEGIN_DATE | PHYS_READ_GB | eligiBLE_OFFLOAD_GB | INTERCONNECT_RETURNED_GB | eligiBLE_OFFLOAD_PCT | OFFLOAD_SAVED_PCT | SAVED_BY_STORAGE_INDEX | UNCOMPRESSED_GB | RETURN_UNCOMPRESSED_PCT |
|---|---|---|---|---|---|---|---|---|---|---|
| OLTP | MON | 2023-04-03 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | TUE | 2023-04-04 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | WED | 2023-04-05 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | THU | 2023-04-06 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | FRI | 2023-04-07 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | SAT | 2023-04-08 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | SUN | 2023-04-09 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | MON | 2023-04-10 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | TUE | 2023-04-11 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | WED | 2023-04-12 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | THU | 2023-04-13 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | FRI | 2023-04-14 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | SAT | 2023-04-15 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | SUN | 2023-04-16 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | MON | 2023-04-17 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | TUE | 2023-04-18 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | WED | 2023-04-19 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | THU | 2023-04-20 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | FRI | 2023-04-21 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | SAT | 2023-04-22 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | SUN | 2023-04-23 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | MON | 2023-04-24 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | TUE | 2023-04-25 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | WED | 2023-04-26 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | THU | 2023-04-27 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | FRI | 2023-04-28 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | SAT | 2023-04-29 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | SUN | 2023-04-30 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | MON | 2023-05-01 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| OLTP | TUE | 2023-05-02 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |

# MEASURING the MAGIC

## DBA_HIST_SYSSTAT

An additional 14% never happened due to storage cell indexes

*Edited down to 10 working days*

500-700TB reads per day

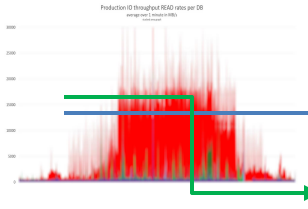
90+% *eligible for offload*

98% of that offloaded (processed on storage cell)

| PHYSICAL READ GB | ELIGIBLE OFFLOAD GB | INTERCONNECT RETURNED GB | ELIGIBLE OFFLOAD PERCENT | OFFLOAD SAVED PERCENT (of the eligible) | SAVED BY STORAGE INDEX GB | STORAGE INDEX PERCENT |
|---|---|---|---|---|---|---|
| 700,000 | 630,000 | 12,000 | 89 | 98 | 96,000 | 14 |
| 720,000 | 670,000 | 14,000 | 93 | 98 | 88,000 | 12 |
| 680,000 | 630,000 | 13,000 | 93 | 98 | 89,000 | 13 |
| 610,000 | 560,000 | 12,000 | 91 | 98 | 83,000 | 14 |
| 620,000 | 570,000 | 11,000 | 92 | 98 | 87,000 | 14 |
| 450,000 | 410,000 | 7,000 | 90 | 98 | 72,000 | 16 |
| 700,000 | 650,000 | 12,000 | 93 | 98 | 93,000 | 13 |
| 710,000 | 660,000 | 13,000 | 92 | 98 | 94,000 | 13 |
| 640,000 | 580,000 | 12,000 | 92 | 98 | 75,000 | 12 |
| 630,000 | 580,000 | 12,000 | 92 | 98 | 82,000 | 13 |

# MEASURING the MAGIC

## DBA_HIST_SYSSTAT



09:00-18:00 Average: **17GB/s**

**becomes**

**19.3GB/s (storage indexes)**

**+ all that network traffic & compute processing**

Plus the Regular Peaks at **30+GB/s**

add 14% storage indexes that "never happened" plus the offload…

Assume N/W of 100Gb/s
600TB=5,000,000 Gbits
=50,000seconds to transfer
=60% of a day

# MEASURING the MAGIC

Is this right?

Lets look another way

**GV$SQL**

```
with offload_data as (
       -- all sql where there is offload to the storage cells
       select sql_id, child_number, plan_hash_value phv, executions execs,
       IO_CELL_OFFLOAD_ELIGIBLE_BYTES,
       IO_INTERCONNECT_BYTES,
       PHYSICAL_READ_BYTES
  from gv$sql s
 where executions > 0
   and IO_CELL_OFFLOAD_ELIGIBLE_BYTES > 0
   and plan_hash_value <> 0
   and PARSING_SCHEMA_NAME not in (select username from dba_users where oracle_maintained='Y')),
     nonoffload_data as (
       -- all sql where there is no offload. show the io from the storage cells
       select sql_id, child_number, plan_hash_value phv, executions execs,
       IO_CELL_OFFLOAD_ELIGIBLE_BYTES,
       IO_INTERCONNECT_BYTES,
       PHYSICAL_READ_BYTES
  from gv$sql s
 where executions > 0
   and NVL(IO_CELL_OFFLOAD_ELIGIBLE_BYTES,0) = 0
   and plan_hash_value <> 0
   and PARSING_SCHEMA_NAME not in (select username from dba_users where oracle_maintained='Y'))
select sys_context('userenv','db_name')||' : SQL  with    offload ' info
       ,sum(IO_CELL_OFFLOAD_ELIGIBLE_BYTES)/1024/1024/1024 OFFLOADABLE_BYTES_GB
       ,sum(IO_INTERCONNECT_BYTES)/1024/1024/1024          IO_INTERCONNECT_BYTES_GB
       ,sum(PHYSICAL_READ_BYTES)/1024/1024/1024            PHYSICAL_READ_BYTES_GB
       ,100 - (100*(sum(IO_INTERCONNECT_BYTES) / sum(IO_CELL_OFFLOAD_ELIGIBLE_BYTES))) PCT_OFFLOADED
  from offload_data od
union all
select sys_context('userenv','db_name')||' : SQL  with no offload ' info
       ,sum(IO_CELL_OFFLOAD_ELIGIBLE_BYTES)/1024/1024/1024 OFFLOADABLE_BYTES_GB
       ,sum(IO_INTERCONNECT_BYTES)/1024/1024/1024          IO_INTERCONNECT_BYTES_GB
       ,sum(PHYSICAL_READ_BYTES)/1024/1024/1024            PHYSICAL_READ_BYTES_GB
       ,0 PCT_OFFLOADED
  from nonoffload_data od
```

| INFO | OFFLOADABLE_BYTES_GB | IO_INTERCONNECT_BYTES_GB | PHYSICAL_READ_BYTES_GB | PCT_OFFLOADED |
|---|---|---|---|---|
| OLTP : SQL  with    offload | 2,800,000 | 355,000 | 2,900,000 | 87.68 |
| OLTP : SQL  with no offload | 0 | 275,000 | 275,000 | 0.00 |

note: some sql will live in the shared pool for a long time, hence 2.8PB of offloadable bytes

# MEASURING the MAGIC

this is very useful, but "general"
It does not reflect individual SQL's…

# INDIVIDUAL SQL MAGIC

## GV$SQL

`physical_read_bytes`
`io_cell_offload_eligible_bytes` *>0 means it's eligible for offload!*
`io_cell_offload_returned_bytes` *and if it is, what volume of data was returned*

# INDIVIDUAL SQL MAGIC

## GV$SQL

```
spool offloadsql.out append
select  sys_context('userenv','db_unique_name') db
        ,sql_id
        ,sum(executions) executions
        ,round(sum(io_cell_offload_eligible_bytes)/1024/1024/1024) offload_eligible_GB
        ,round(sum(io_cell_offload_returned_bytes)/1024/1024/1024) offload_returned_GB
        ,round((sum(io_cell_offload_eligible_bytes) - sum(io_cell_offload_returned_bytes))/1024/1024/1024) actual_gb
        ,round(((sum(io_cell_offload_eligible_bytes) - sum(io_cell_offload_returned_bytes)) / decode(sum(io_cell_offload_eligible_bytes),0,1,(sum(io_cell_offload_eligible_bytes))))*100) io_saved_pct
        ,round(sum(elapsed_time)/1000000,2) elapsed_time_sec
        ,round((sum(physical_read_bytes)-sum(io_cell_offload_returned_bytes))/1024/1024/1024) gb_saved
        ,round((sum(io_cell_offload_eligible_bytes)-sum(io_cell_offload_returned_bytes))/1024/1024/1024) gb_eligible_saved
        ,round((sum(io_cell_offload_returned_bytes))/decode(sum(executions),0,1,sum(executions))/1024/1024/1024) gb_returned_per_exec
        ,round((sum(io_cell_offload_eligible_bytes)-sum(io_cell_offload_returned_bytes))/decode(sum(executions),0,1,sum(executions))/1024/1024/1024) gb_saved_per_exec
        ,round(sum(elapsed_time)/1000000/decode(sum(executions),0,1,sum(executions)),2) secs_per_exec
        ,round(((sysdate) - (to_date(min(first_load_time),'YYYY-MM-DD/HH24:MI:SS'))) * 86400 / sum(executions)) secs_since_exec
        ,(to_date(min(first_load_time),'YYYY-MM-DD/HH24:MI:SS')) first_load_time
        --,sql_text
from gv$sql
where 1=1
  -- exclud oracle accounts
  and PARSING_SCHEMA_NAME not in (select username from dba_users where oracle_maintained='Y')
  and executions>0
  and io_cell_offload_eligible_bytes > 0
--  and sql_id='&&sql_id'
group by sys_context('userenv','db_unique_name') ,sql_id -- all PHV's for a SQL_ID so we get the good and the bad in here, averaged
order by gb_eligible_saved desc
/
```

# INDIVIDUAL SQL MAGIC

## GV$SQL

| SQL_ID | EXECs | OFFLOAD_ELIGIBLE_GB | OFFLOAD_RETURNED_GB | IO_SAVED_PCT | GB_SAVED | GB_RET_PER_EXEC | GB_SAVED_PER_EXEC |
|---|---|---|---|---|---|---|---|
| **7x....** | **2000** | **550,000** | **200** | **100** | **549,900** | **0** | **200** |
| 8c.... | 80000 | 400,000 | 11000 | **97** | 399,000 | 0 | 5 |
| a0.... | 43000 | 220,000 | 8000 | **96** | 219,000 | 0 | 5 |
| b4.... | 43000 | 220,000 | 8000 | **96** | 219,900 | 0 | 5 |

.
.
.

"7x" saved 540TB of data transfer from I/O subsystem to DB node for this SQL

These 4 SQL's avoid 1PB of data transfer over their existence in the shared pool

# INDIVIDUAL SQL MAGIC

So what to do with these SQL's?

Lets see how we get on without the Exadata Functionality:

1. Capture the SQL
2. Run the SQL and get the metrics from GV$SQL
3. Purge the Cursor
4. Switch off Exdata Functionality
5. Run the SQL and get the metrics from GV$SQL

# INDIVIDUAL SQL MAGIC

## Exadata Functionality GV$SQL Metrics

```
SQL_ID                          7x..........
IO_CELL_OFFLOAD_ELIGIBLE_GB       204
IO_CELL_OFFLOAD_RETURNED_GB         0
GB_SAVED                          204
IO_SAVED_PCT                      100%
ELAPSED_TIME_SEC                  13sec
```

## GV$SQL Metrics without Exadata

```
SQL_ID                          7x..........
IO_CELL_OFFLOAD_ELIGIBLE_GB         0
IO_CELL_OFFLOAD_RETURNED_GB         0
GB_SAVED                            0
IO_SAVED_PCT                        0
ELAPSED_TIME_SEC                  608sec
```

```
alter session set "CELL_OFFLOAD_PROCESSING"=false
alter session set "_KCFIS_STORAGEIDX_DISABLED"=true
alter session set "_BLOOM_FILTER_ENABLED"=false
alter session set "_BLOOM_PRUNING_ENABLED"=false
alter system flush shared_pool
```

*In this case, the execution plan was using a bloom filter*

# INDIVIDUAL SQL MAGIC

Is this OK?

Does this SQL have a time-critical aspect in the application?

Does 13 secs to 10 mins matter?

## GV$SQL Metrics without Exadata

```
SQL_ID                          7x.....
IO_CELL_OFFLOAD_ELIGIBLE_GB           0
IO_CELL_OFFLOAD_RETURNED_GB           0
GB_SAVED                              0
IO_SAVED_PCT                          0
ELAPSED_TIME_SEC                608sec
```

What about the other 300 SQL's with offload in this database?

How many key DB's are on this Exadata Cluster?

[of the dozen DB's on this cluster, 50% are "small/medium" <1TB - and won't care about having no Exadata Magic]

# SYSTEM PERFORMANCE TESTING

## So what next?

Do you have the capability to switch off the Exadata functionality on a full-sized copy of the DB and test the system?

Do you have Real Application Testing?

Do you have the ability to replay or test Production-level volumes?

```
alter system set "CELL_OFFLOAD_PROCESSING"=false
alter system set "_KCFIS_STORAGEIDX_DISABLED"=true
alter system set "_BLOOM_FILTER_ENABLED"=false
alter system set "_BLOOM_PRUNING_ENABLED"=false
```

# COMMODITY HARDWARE

## Servers

any will do. Plenty of CPU and Memory is cheap.
(You will probably need to run Oracle Linux Virtualisation Manager (OLVM) to control licenses). Just don't use VMWare.

## Storage

Are you using Hybrid Columnar Compression (HCC)

1 DB on this cluster is; 10TB of HCC "never-accessed" data stored at 20x compression [Compress for Archive High]

Commodity H/W will need to have space for expanding the HCC data… 200TB+

However, all decent commodity H/W includes some native compression (4x)

# COMMODITY HARDWARE

## Storage

Is there a (commodity) SAN which can cope with your workload?


*A fully loaded DELL EMC VMAX 950F will do 150GB/s and 6.7m IOPS.*

*But it's hard to call that "commodity"*

# COMMODITY HARDWARE

**Storage**

Is there a (commodity) SAN which can cope with your workload?


A Pure //XL170 will run at 20GB/s

2 x //XL170's synchronously linked will deliver 35GB/s seamlessly with encryption and compression

so you will need 5 or 6… (Prod/DR/Non-Prod)

5 x //XL170's (with sufficient storage) costs more than 3 Exadatas (in initial costs)…

# POSTGRESQL

## PostgreSQL

Removal of licensing costs!

**PL/SQL rewrite, and care needed in some areas like long running transactions XID wraparound and vacuuming, partitioning challenges (no global indexes), NULL behaviours

It's going to be cheaper for that reason… same major commodity storage issue though
But can PostgreSQL cope?

## PG is *very* capable

migrating most small-to-medium sized systems [up to, say, 1TB] will work fine**

Partitioning, Vacuuming Issues, and On-Line capabilities mean **VLDB's** are still more successful on Oracle, ***at the moment.***
Native Encryption in the DB is Oracle-only (unless you buy EDB's new offering).
*Storage-level encryption is inherently less secure for data at rest*

# CLOUD

Cloud is perfect for elasticity, startups and irregular workloads.

Cloud is more expensive for known consistent workloads.
**unless you've really overprovisioned on-prem

**AWS / Azure***/ non-Oracle Cloud**

- double your license costs for the same CPU
- Enjoy paying the AWS 30% margin for the same compute power
- Storage at 35GB/s is hard to achieve and niche (as of today – tomorrow ?)

***today, there's OCI in Azure. Tomorrow? Who knows?

# CLOUD

## OCI incorporating Exadata Cloud At Customer

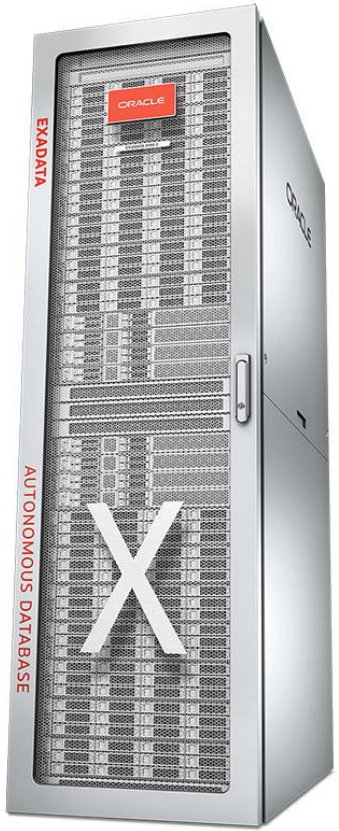
- has Exadata, so it's the same hardware
- more flexible, and probably the cheapest "large" cloud offering out there
- costs about 20% more than on-prem by my recent costings
  but! What are you getting for the 20%?
  - ✓ TDE
  - ✓ RAT
  - ✓ Patching
  - ✓ a nice GUI which dumbs down your DBA so they forget the basics
- You're probably not migrating to PG from there

Regardless of cloud provider, **beware of egress costs** and know what that means for you!

# GET ME OFF THAT EXADATA

What have we learned?

- Exadatas are complex
- They are also very very good
- You need to do some investigation to find out how much you are using them
- If you have a system doing a lot of work you may struggle to use commodity hardware / standard Cloud computing (to even match the minimum requirement)






That just goes to show that you never know, although what it is we never know I suspect we'll never know.

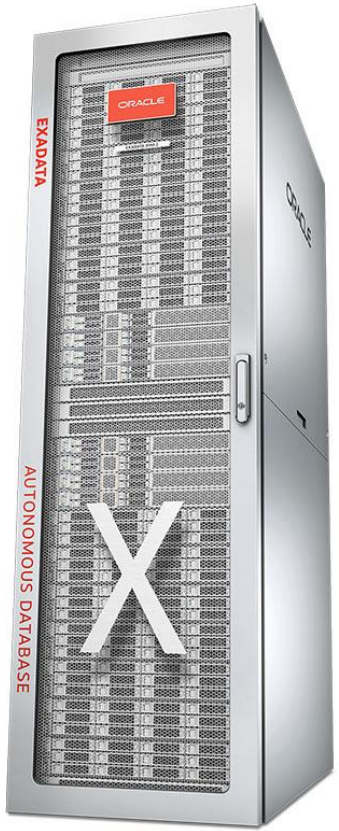
# GET ME OFF THAT EXADATA

There's a cost to do business.

Exadatas seem expensive

Sometimes that cost, despite seeming high, is the cheapest way forward

for now…

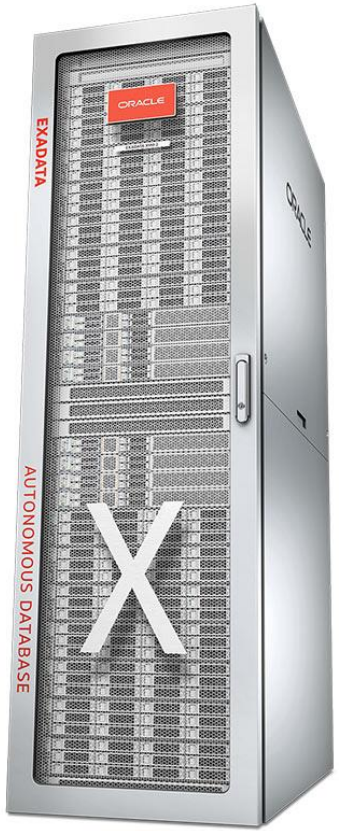





That just goes to show that you never know, although what it is we never know I suspect we'll never know.

# SO, WHAT HAPPENED?

1. Stop using the apps and close the business?
2. Stay with Oracle, but move to commodity hardware
3. Migrate to PostgreSQL, on commodity hardware
4. Cloud *(AWS / Azure / OCI)*
5. Just buy the f**ing Exadatas!




## They bought the f**ing Exadatas!



That just goes to show that you never know, although what it is we never know I suspect we'll never know.

# GET ME OFF THAT EXADATA

# THANK YOU

BLOG: http://chandlerdba.com
Twitter: **@chandlerDBA**
E: neil@chandler.uk.com

# asm_activity.sh

```
# asm activity - mbs read and write. nohup asm_activity.sh ORCL &
if [ ${#} -eq 0 ]
then
 echo "Please enter the database name as a parameter e.g. ORCL"
 exit
fi

db_name=$1

# only works for 1 DB cos the code is simple
l_ro_prev=0
l_rw_prev=0
l_ro_diff=0
l_rw_diff=0
delay=60
# 1440=1 day, 10080=1 week
loops=1440
count=1
echo "DT:DB-${db_name}:Read MB/s averaged over ${delay}:Write MB/s averaged over ${delay}" | tee -a asm_activity.${db_name}.out

while [[ ${count} -lt ${loops} ]]
do
let count=count+1

sqlplus -s / as sysdba << EOF
set head off feed off trimspool on pages 5000 lines 230
set termout off
spool asm_activity.${db_name}.tmp
select DBNAME||':'||to_char(max(sysdate),'YYYY-MM-DD HH24.MI.SS')
             ||':'||round(sum(BYTES_READ)/1024/1024)
                                                    ||':'||round(sum(BYTES_WRITTEN)/1024/1024)
  from V\$ASM_DISK_IOSTAT
 where 1=1
   and dbname = '${db_name}'
 group by dbname;
spool off
EOF

 l_db=`cat asm_activity.${db_name}.tmp | grep -v "^$" | awk -F: '{print $1}'`
 l_dt=`cat asm_activity.${db_name}.tmp | grep -v "^$" | awk -F: '{print $2}'`
 l_ro=`cat asm_activity.${db_name}.tmp | grep -v "^$" | awk -F: '{print $3}'`
 l_rw=`cat asm_activity.${db_name}.tmp | grep -v "^$" | awk -F: '{print $4}'`

 let l_ro_diff=(${l_ro} - ${l_ro_prev})/${delay}
 let l_rw_diff=(${l_rw} - ${l_rw_prev})/${delay}

 l_ro_prev=${l_ro}
 l_rw_prev=${l_rw}

 echo "${l_dt}:${l_db}:${l_ro_diff}:${l_rw_diff}" | tee -a asm_activity.${db_name}.out
 sleep ${delay}
done
```