

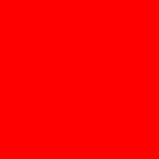
ORACLE®



ORACLE®

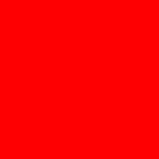
ZFS Performance with Databases

Ulrich Gräf
Principal Sales Consultant
Hardware Presales
Oracle Deutschland B.V. und Co. KG



The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.

The development, release, and timing of any features or functionality described for Oracle's products remain at the sole discretion of Oracle.



The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

In addition, the following is intended to provide information for Oracle and Sun as we continue to combine the operations worldwide. Each country will complete its integration in accordance with local laws and requirements. In the EU and other non-EU countries with similar requirements, the combinations of local Oracle and Sun entities as well as other relevant changes during the transition phase will be conducted in accordance with and subject to the information and consultation requirements of applicable local laws, EU Directives and their implementation in the individual members states. Sun customers and partners should continue to engage with their Sun contacts for assistance for Sun products and their Oracle contacts for Oracle products.

Hardware and Software

ORACLE®

Engineered to Work Together

ZFS Performance with Databases

Agenda:

- ZFS Introduction
 - Necessary for understanding of tuning
 - Can be skipped, when ZFS principles are known
- General view on the subject
 - Including Do's and Don't's
- ZFS and Oracle
- Some thoughts about the current disks

ZFS Performance with Databases

Agenda:

- **ZFS Introduction**
 - Necessary for understanding of tuning
 - Can be skipped, when ZFS principles are known
- General view on the subject
 - Including Do's and Dont's
- ZFS and Oracle
- Some thoughts about the current disks

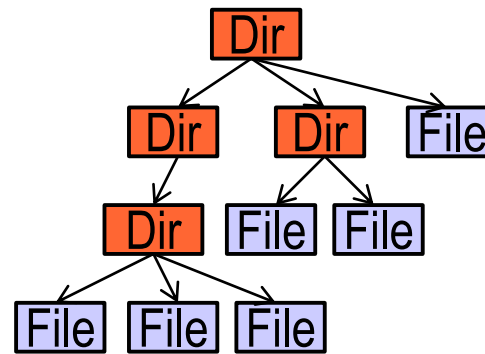
ZFS – Design Decisions

- Elements
 - Disk pools
 - Integrated volume manager
 - Large amounts of data (128 Bit addressing)
 - Only transactional changes of the disk
 - Fast snapshots
 - Checksums everywhere
 - Integrated self healing
 - Easy administration with inheritance principles
- All elements existed before ...
-

... But: Not in this combination!

Volmgr. + FS / ZFS:

Filesystem:



Posix-Interface:

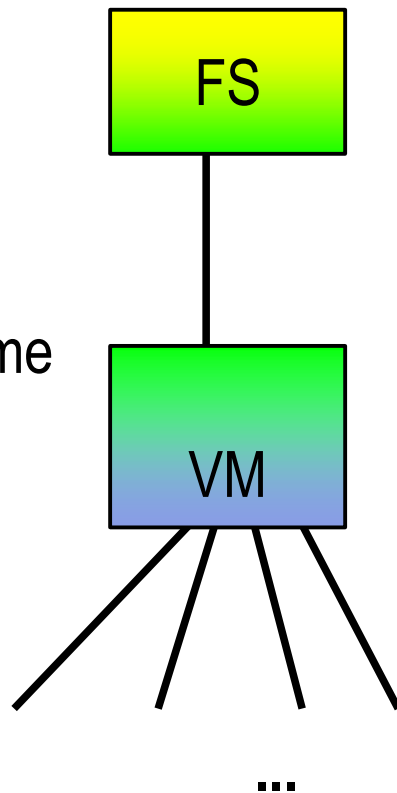
UFS:

Bit-Optimizing

Abstraction

Hiding

Volume



ZFS:

Admin-Optimization

Checksums

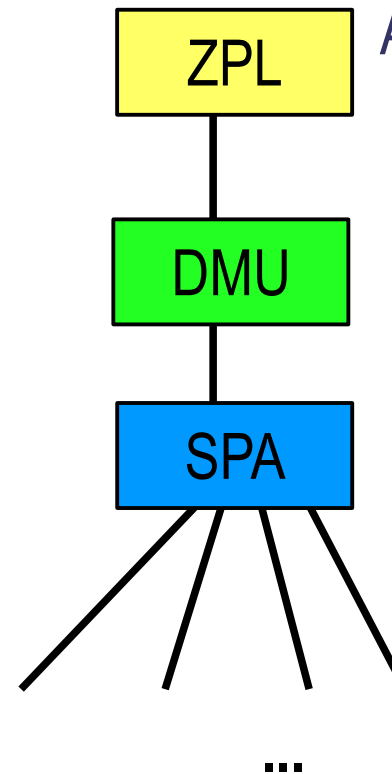
Transactions

Dynamic Striping

Compression

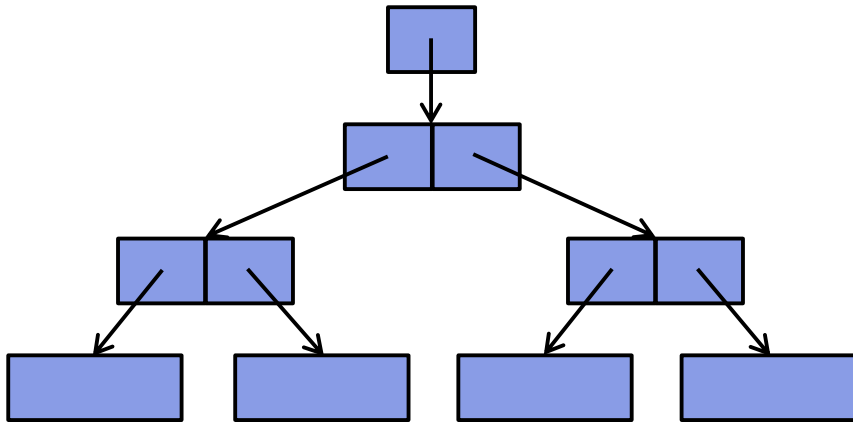
Self-Healing

HW-Optimization

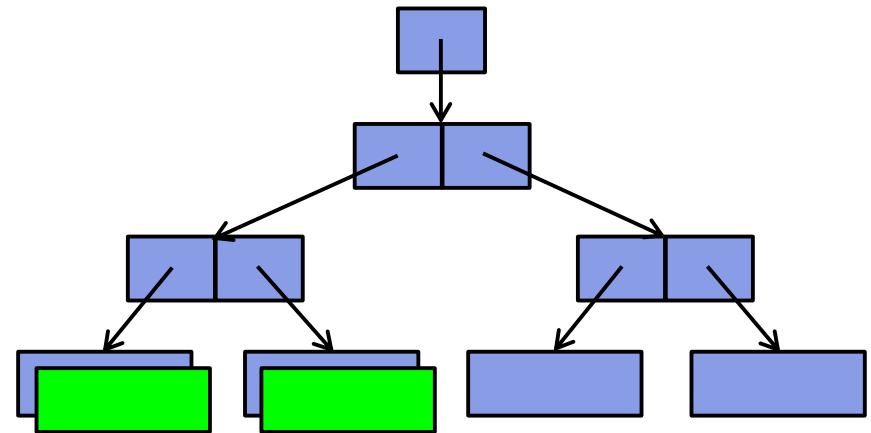


Consistency: Copy-On-Write Transactions

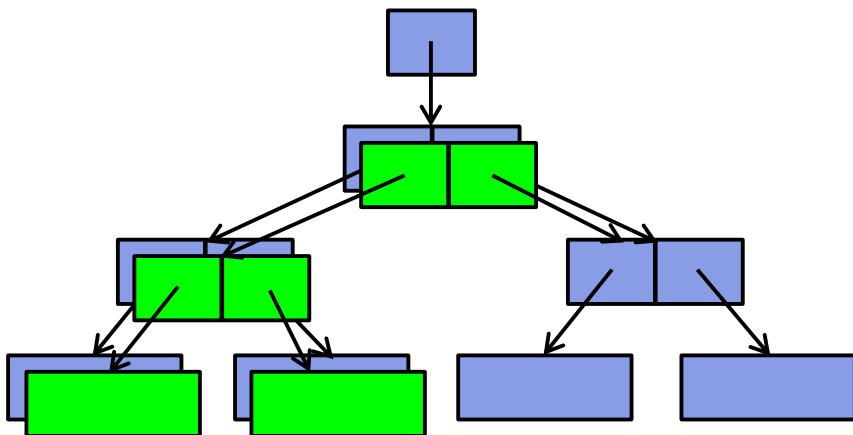
1. Consistant state



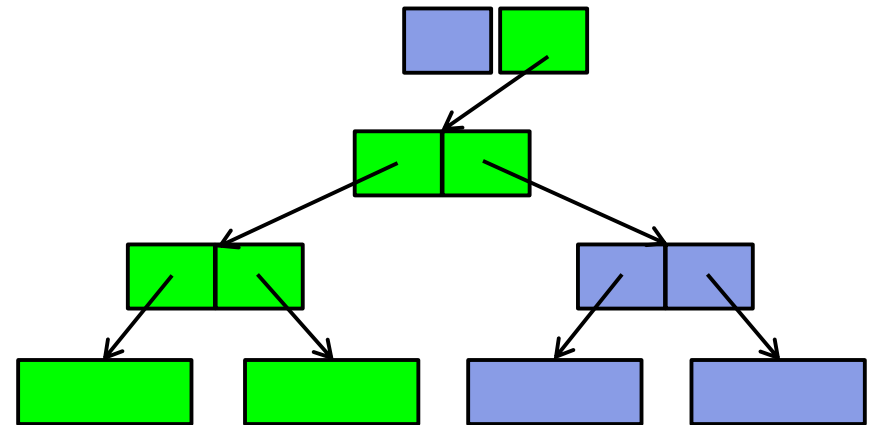
2. Write data to memory



3. Add Metadata, write new blocks

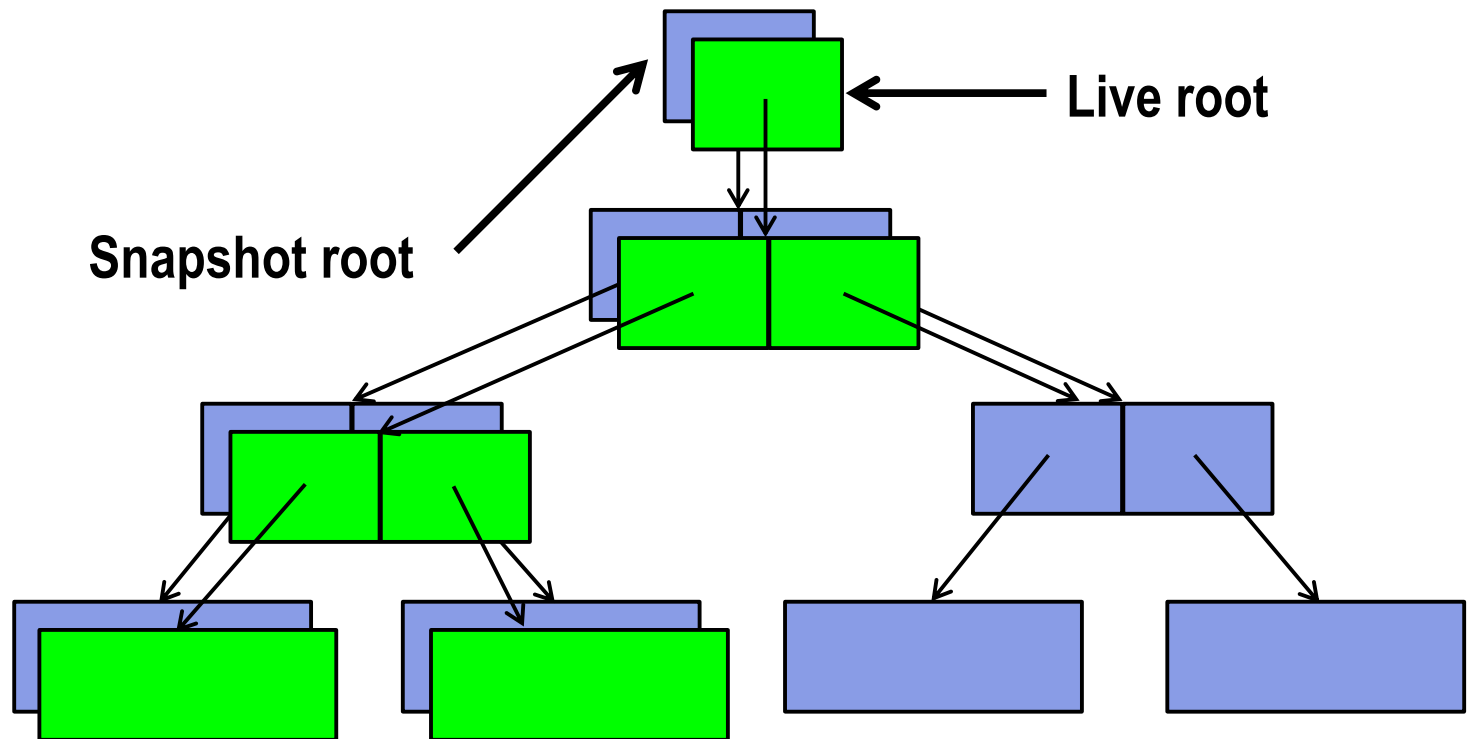


4. Write *uberblock* (= commit)



ZFS Snapshot:

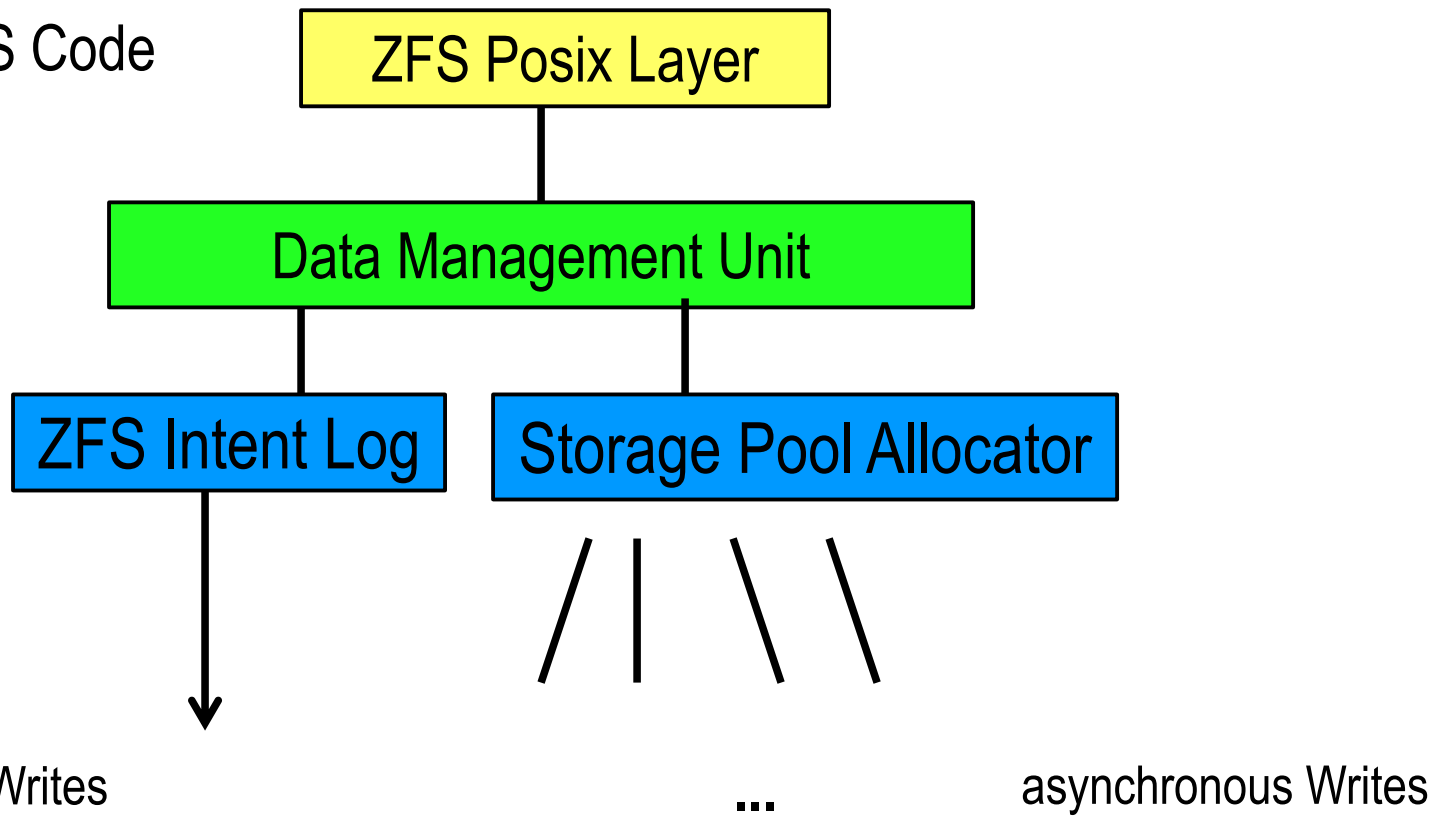
- Save the reference to the filesystem as a snapshot
- Take care in freeing blocks



Sync Writes: ZFS Intent Log

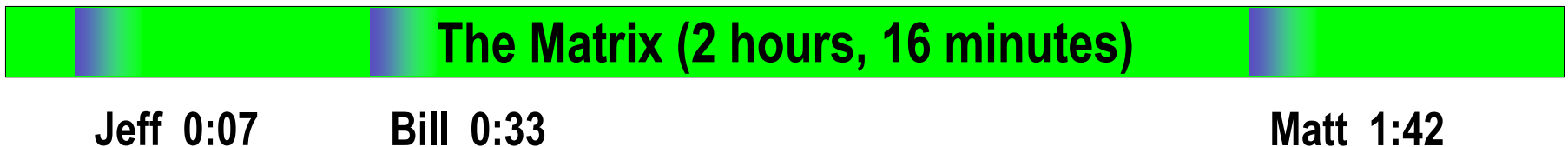
`fwrite()` mit `O_SYNC`, `O_DSYNC`, `fsync()`, `sync()`)

ZFS: FS Code



Intelligent Prefetch

- Multi User Prefetch
- For Streaming Service Provider



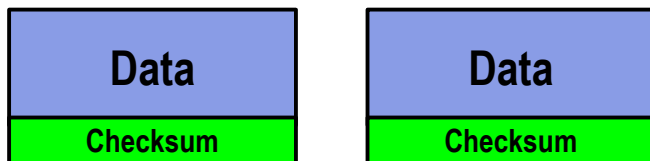
- Detects sequential and linear (length + stride)
- Prefetch in old model (volume + filesystem):
 - Prefetch only on volume level
 - Conflicts with multiple mechanisms
 - Moiré-effects
 - No user context in volume manager

**The Matrix
(10M rows,
10M columns)**

End-to-End Data Integrity

disk block checksums

- Small checksum stored at data block
- Checking part of checksum is small (8 or 16 bit)
- Some errors stay undetected

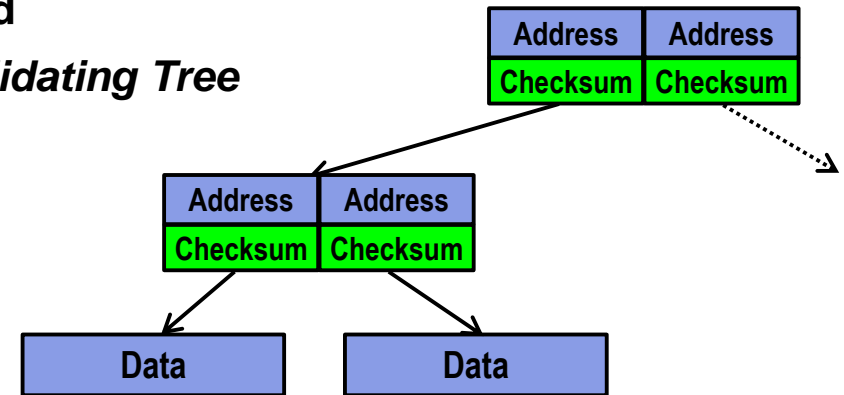


Only media errors can be found

- ✓ Bit rot
- ✗ Phantom writes
- ✗ Misdirected reads and writes
- ✗ DMA parity errors
- ✗ Driver bugs
- ✗ Accidental overwrite

ZFS structural integrity

- Large checksum stored at address
- Correct block can be identified
- *Self-Validating Tree*

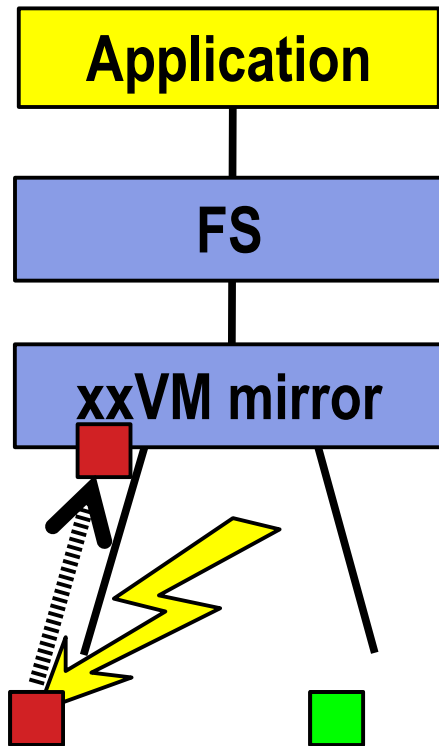


ZFS validates all blocks

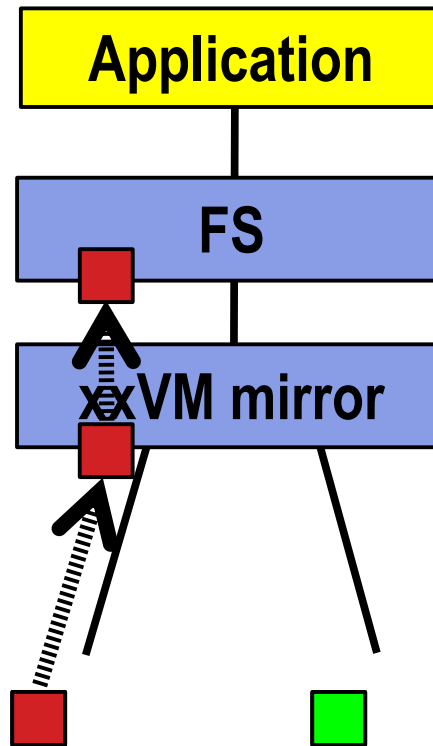
- ✓ Bit rot
- ✓ Phantom writes
- ✓ Misdirected reads and writes
- ✓ DMA parity errors
- ✓ Driver bugs
- ✓ Accidental overwrite

Data Errors in Traditional Model (vol + fs)

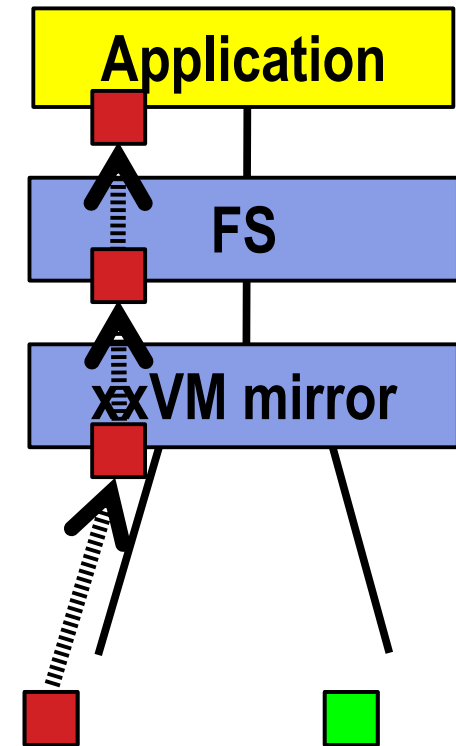
1. read() gets a bad block



2 a) Wrong metadata:
perhaps filesystem code finds
wrong data (pointer)
=> panic!

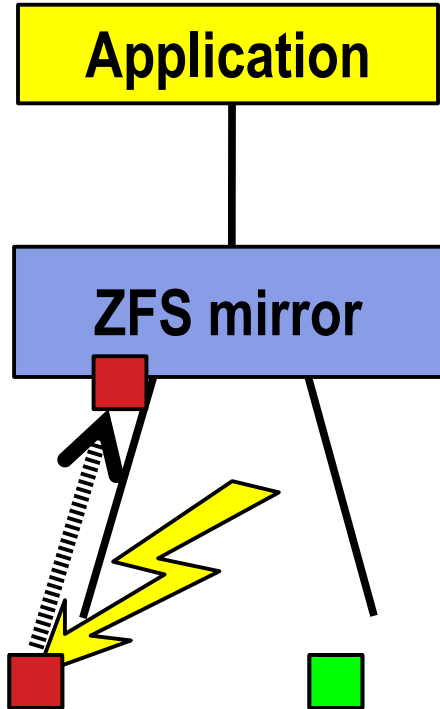


2 b) Wrong data:
Application reads wrong data,
gets problems or wrong results
pass unnoticed!

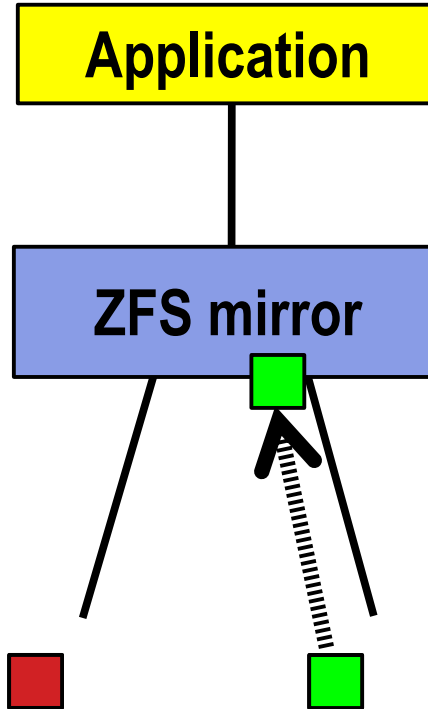


Data Errors in ZFS: Self Healing

1. read() gets a bad block

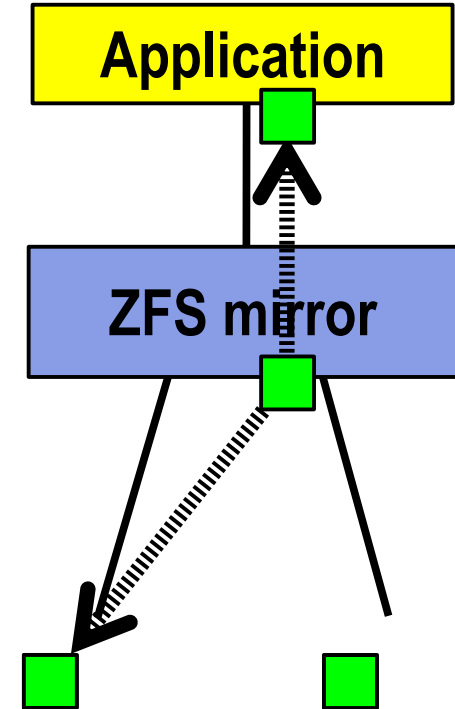


2. ZFS computes checksum, discovers unusable block, is able to get/compute the alternative block from a mirror/raidz redundant config



3. ZFS delivers only correct data to ZFS code and the Application

Plus: The defective block will be corrected automatically!



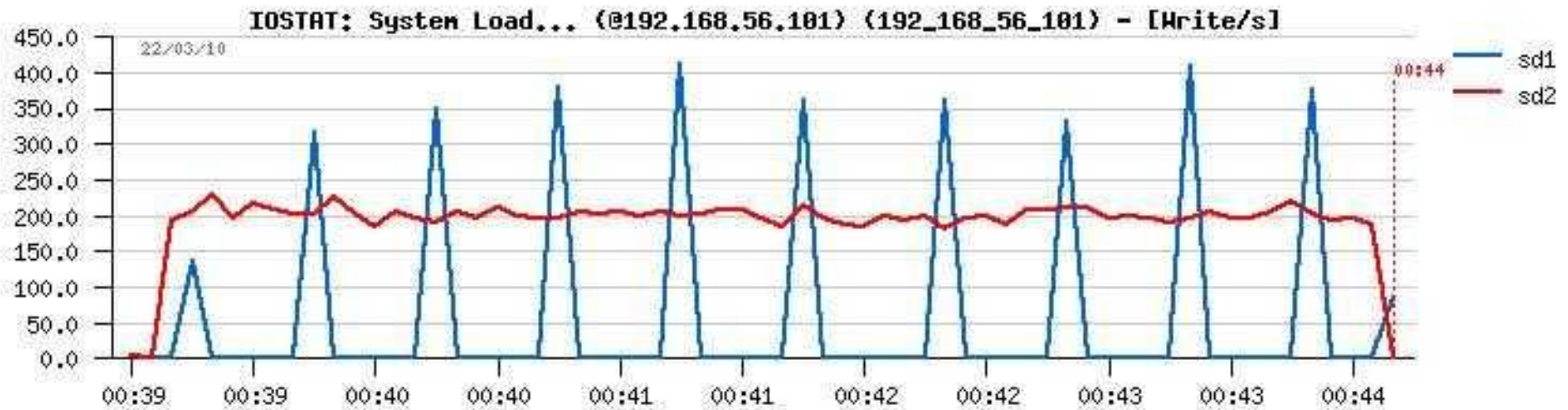
ZFS Self Healing

- Redundant configurations
 - Mirror: other side of mirror is used
 - RAIDZ / RAIDZ2 / RAIDZ3: correct data is reconstructed with the corresponding RAID algorithm.
 - No defective block is used in ZFS code or application
 - ZFS code and application work only with correct blocks
 - Error probability is very low because of 256 bit checksum
- Non-redundant configurations
 - Errors are detected; but Self Healing not possible (except when ZFS attribute copies was set to 2 or 3)
 - Defective blocks are not used by application or ZFS code
 - Errors are indicated to the application (I/O error)

ZFS Behaviour when Writing

- “lazy write batches”
 - Data: written delayed, peak at the COW
 - Log: written constantly

Example: Database on 2 Disks (**Data: sd1**, **Log: sd2**)



ZFS und Flash SSD Storage:

Hybrid Storage Pool

- A dedicated device can be used for the ZIL (*Logzilla*)
 - Use an SSD with a high number of I/Os per second and fast writes
 - Not every SSD is working (*Enterprise Grade* is required!)
 - Works similar to a Storage Subsystem with Cache
 - Cheaper than everything on SSD, SSD is used at the sweet spot
- ZFS is able to use a Flash SSD for reading (*Readzilla*)
 - Use an SSD with Capacity
 - No disk head movements are required
 - Contains a copy of the recently used data on disk
 - Faster response time, when data is used repeatedly
 - Be aware: *Cache Warming* is required, needs some time

ZFS Caches

- ARC Cache (always)
 - In main memory
 - Data (= blocks of the file content)
 - Metadata (directories, pointer blocks, file attributes)
 - Default: ((memory – 1GB) on systems > 4 GB)
 - Adapts automatically to available memory
- L2ARC (optional)
 - External device for extending the ARC cache
 - Strongly recommended: Flash SSD Storage

Both Caches use the ARC Algorithm

ARC – Adaptive Replacement Cache (1)

Well known algorithms for freeing up memory for OS use:

- FIFO – free the page, which was used first
 - This is easy to implement
 - Used for simple computers (your mobile?)
 - Good, when pages are used only once
- LRU – free least recently used page
 - Difficult to implement (needs action for each memory access)
 - Flavors of this are used in general purpose operating systems (Solaris, Windows, Linux, ...)
 - Good when page references are equally distributed

But: Typical Computers are different from that

- A page is referenced **once** or **very often** (no flat distribution)

ARC – Adaptive Replacement Cache (2)

Solution: Adaptive Replacement Cache

- ARC is a combination of FIFO and LRU
 - If a page is used once it is put into a (smaller) FIFO cache
 - If the page is used a second time it is put into the LRU
 - The sizes of FIFO and LRU are adapted automatically
- Result:
 - Pages which are used once (f.e. database load) are freed quickly
 - Pages which are used often (programs, libraries, database data, ...) stay undisturbed in main memory

=> ARC is much better than LRU for real world computers

ZFS Performance with Databases

Agenda:

- ZFS Introduction
 - Necessary for understanding of tuning
 - Can be skipped, when ZFS principles are known
- **General view on the subject**
 - Including Do's and Dont's
- ZFS and Oracle
- Some thoughts about the current disks

ZFS and a Database

What is a database?

- Reliable and consistent changes in a set of data
- Transactions (all-or-nothing) (ACID criteria)

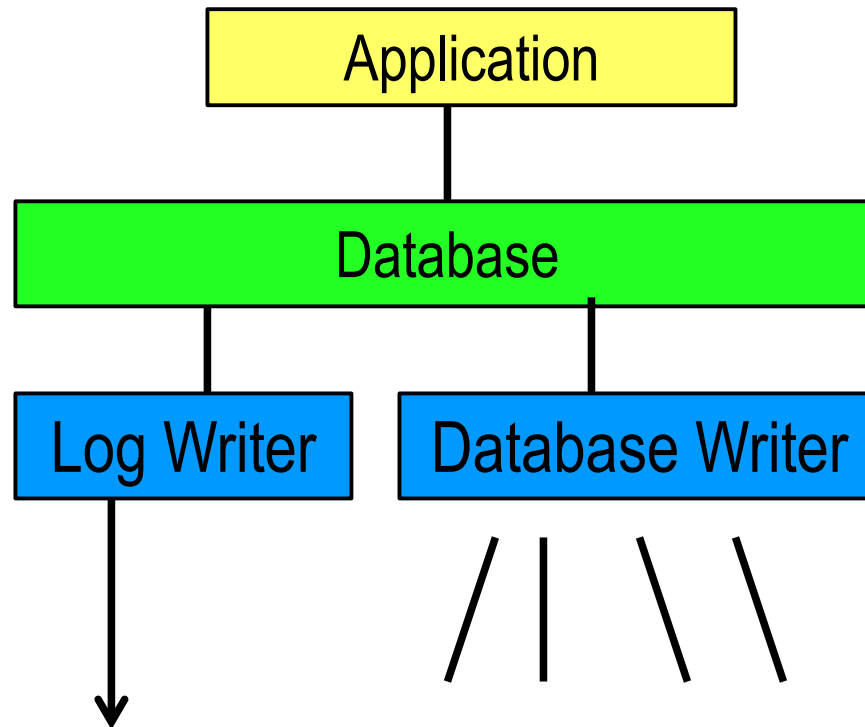
How?

- Changes are written to a log
- Then the client is notified
- Files of the database are changed in the background
- Later: the log is freed
(different mechanisms, f.e. checkpoint)

ZFS works similar

Writing in Databases

Database:



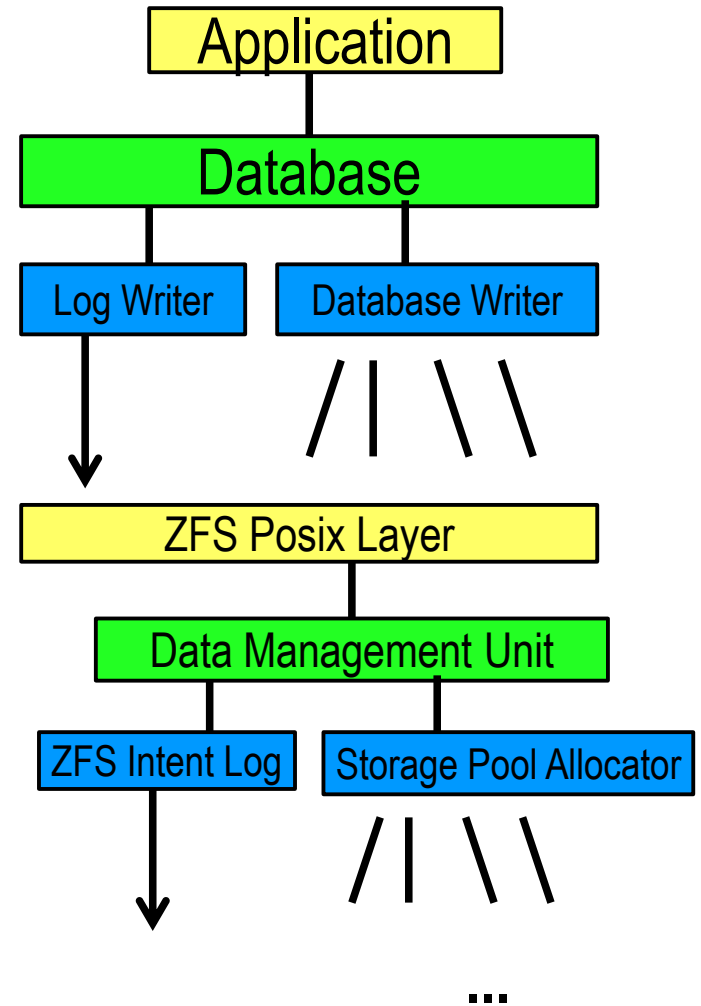
synchronous writes
(Integrity)

...

asynchronous database activity
(but also written synchronously)

A database implemented on ZFS

- Double Implementation
 - 2 Log mechanisms
 - DB writes data twice (log and db files)
 - ZFS writes all this twice (Log and COW)
- Optimized DB
 - DB maintains a model of disk and diskhead
 - Not applicable with ZFS
 - => performance loss
- Non-Optimized DB
 - ZFS implements the Optimization
 - Write consolidation (small to large)
 - ZFS Prefetch helps sequential loads



ZFS DB Performance: Configuration

- More capacity: raidz, raidz2, raidz3
 - ZFS writes RAIDZ rows always at the same time
 - Reading needs the involvement of multiple disks, because ZFS blocks are spread over multiple disks (ZFS is optimized for datarate)
 - Random reads (f.e. database index) produces additional IO requests
- More random read performance: use ZFS mirror
- Self Healing needs a redundant ZFS configuration
- Avoid extreme wide raidz stripes!
 - F.e. x4500: raidz2: 44+2
 - Sequential read and write is ok (backup/archive purposes)
 - Random read and write for databases is very slow

ZFS DB Performance: ZFS zvols

- Zvols are usable like raw devices but configured in ZFS
 - Visible in /dev/zvol/rdisk resp. /dev/zvol/dsk
 - Use as a device, export it via iSCSI or Fibre Channel (currently: supported in ZFS Storage Appliance, later: Solaris)
- Same mechanism used as if it is a file in a ZFS filesystem
 - Especially: no overwriting instead: intent log and COW
 - Data of zvol are not sequential on disk, zvol does not help here
- Use it:
 - For iSCSI, FC, device emulation (instead of lofi), ...
 - When thin-provisioning is needed
 - Not for DB:
 - Same performance as a file in ZFS, but worse to administer

ZFS DB Performance: Optimization in DB

Databases with an I/O optimization model:

Oracle, Informix, DB/2, Adabas C, ...

- Assumption: small change in file block number
= small movement of the disk head => fast
- Therefore: best performance for these DBs is on raw disks or a volume manager (also: Oracle ASM)
- See: Instructions how to make DB files contiguous in a filesystem
- ZFS is an additional layer which consumes some performance

Databases without a disk head optimization model:

Sybase, MySQL, PostGreSQL, MaxDB, SQLite, ...

- ZFS implements an optimization for these databases
 - Consolidation of lots of small writes into less large writes
 - Prefetch for sequential reads

Case Study: Sybase on ZFS at a Telco

Sybase

- Writes into its Log
- Does a lot of in-place updates in its database files
- No comprehensive IO reordering
 - Needs a lot of small writes, datarate is low

Customer moved from UFS to ZFS on the same storage

- “Sybase is now 8x faster than before”
- Analysis showed:
 - Disks get less writes, because of the write consolidation in ZFS
 - Reads were random anyway, sequential read was unimportant

ZFS DB Performance: General (1)

- Use a separate zpool for the Database Log
 - COW for DB data and DB log are then independent
 - Log cannot be disturbed by writes to database files => faster
- Configure ZFS attribute recordsize for Log and Data
 - Use DB blocksize for Log and a multiple of that for data
 - Do not use a too small blocksize in the database (bad idea anyway)
 - ZIL can be used better
- Configure ZFS attribute atime=off
 - Posix standard requires the update of last access date at each read
 - Last access data accuracy is 1/1000 of a second
 - Worst case: add 1000 writes per second and database file
 - Its not necessary for databases
=> switch the update of atime to off

ZFS DB Performance: General (2)

- Hybrid Storage Pool with SSD (Flash SSD)
 - Use a zpool with an appropriate Flash SSD
 - => More IOs / Second => better DB performance
 - Large databases can benefit from a L2ARC on SSD
- Data working set larger than space in main memory
- Not possible to increase the amount of main memory

- Tablespaces / Indices on Flash SSD
 - Tablespaces and/or Indices can be stored on an SSD
 - Can be a better choice than using a Hybrid Storage Pool
 - No support in a cluster for local Flash SSD

- ZFS Storage Appliance is supported in a Cluster

ZFS DB Performance: General (3)

- ZIL for a Hybrid Storage Pool should be large enough
 - Absolute minimum: Store the Datarate for 2 transactions
 - Datarate in MB/Second
 - COW Transactions are usually done every 5 Seconds (Default)
 - F.e. 200 MByte / Second write rate in the database to "disk"
 - $2 * 5 \text{ Seconds} * 200 \text{ MByte/Second} = 2 \text{ GB}$
 - **Please take care of enough reserve!**
ZFS will get very very (unacceptable!) slow, when the Log is full

ZFS DB Performance: General (4)

- Use only up to 80% of the space in the zpool
 - ZFS does Copy-on-Write, no block will be overwritten
 - Therefore writing needs additional free space
 - Probability of a fragmentation and gang chaining rises
 - ZFS works over a fill grade of 80%, but has less performance
 - Use ZFS attributes quota and refquota to limit usage
- Combination of ZFS with Storage Subsystems (w. cache)
 - ZFS knows more about the usage
 - Storage Subsystem offers out-of-line disk repair
 - Use a redundant configuration on both levels
 - Out-of-line disk repair is possible
 - ZFS Self Healing, Prefetch, Integrity Check are possible

ZFS DB Performance: Memory (1)

- ZFS uses a new cache algorithm: ARC
- Can use a lot of memory (Default: memory - 1GB)

Recommendation:

- Consider to buy more memory than with UFS/VxFS
- Use the most current Solaris 10 (9/10 / Update 9)
- Avoid the ARC adaption at sudden memory requirements
 - Cluster switch or process start is delayed
 - With lots of changes in the process structure
 - Clustered Systems (Solaris Cluster)
 - Limit the ARC in `/etc/system` (value is in bytes):

```
set zfs:zfs_arc_max = 0x780000000
```

ZFS DB Performance: Memory (2)

Metadata needs also space:

- Space for file attributes is negligible (2 KByte per file)
- Space for pointers (128 Byte) gets relevant, when filesystems are large
 - 1/1000 when recordsize is 128KByte
 - 1/500 when recordsize is 64 KByte
 - ...
 - $\sim 1/16 = 6.4\%$ when recordsize is 2 KByte (avoid this!)
- F.e. a database with 100 TB content and 128 KByte recordsize
- needs 100 GB for the metadata
 - Important: when the database does random reads using all data
- => Plan for enough main memory

ZFS DB Performance: Memory (3)

ZFS pool operations need main memory

- For example
 - Resilvering (means: (re-)create the mirror) scans all the blocks of the pool (data and metadata)
 - Scrubbing (~ online fsck) also scans all the blocks
 - Replication scans at least all metadata to transmit
- All scanning activity need an additional buffer
 - Up o 25% of main memory

=> Plan for enough memory (or its slow during production)

ZFS DB Performance: Memory (4)

Flash SSD as read cache for ZFS (aka L2ARC, Readzilla)

- ZFS set `secondarycache = on | metadata | off`
- Can be configured separately for each ZFS filesystem
- Usually only "on" or "off" make sense: to cache or not to cache
- `secondarycache=metadata` only in very special cases:
 - Metadata do not fit into main memory but fit into L2ARC
 - Then data blocks can be read with one disk access (and perhaps some cache accesses)
- Beware: L2ARC needs also main memory!
 - Each block (recordsize) needs 200 Byte main memory. This can be large!
 - Example: A system with 2 TB L2ARC and 16 KB recordsize:
 - 2^{41} Bytes space, @16 KB => 2^{27} blocks in L2ARC,
 - $2^{27} * 200 = 25.6$ GByte needed in main memory

ZFS DB Performance: Memory (5)

Plan your memory: a system with 64 GByte main memory

| Plan: | Default | Example |
|--|---------|---------|
| - OS in general: GByte | 1 GByte | 1 |
| - Segmap (not for S7xxx): GByte for conventional IO (read, write, ...) | 12 % | 8 |
| - Freespace for Solaris: GByte for memory requests by OS and apps | 1/32 | 2 |
| - Application: F.e. Oracle DB, 32 GB SGA GByte and 2 GB for program, PGA ... | | 34 |
| - Metadata for 4 TB (128 KB recordsize) | | 4 GByte |
| - Reserve (use at least 1 GB!): | | 2 GByte |

- Sum of Memory plan: 51 GByte => set ARC to less than 13 GByte!

ZFS and Direct IO

- UFS direct IO is a collection of optimizations
 - Traditional: write data to disk without caching in the main memory
- Can be set with a ZFS attribute: primarycache (not on S7xxx)
 - zfs set primarycache= on | metadata | off
 - Since: Solaris 10 10/09 (Update 8) or a current patchlevel
 - on (default): Everything (data and metadata) will be cached
 - metadata: metadata (pointer and attributes) are cache
 - off: no caching
- To get directio like in UFS: zfs set primarycache=metadata
 - No data goes into the cache, as the database has its own cache
 - Solaris remembers the pointers to the data on disk
 - Currently not possible with ZFS Storage Array (S7000)

ZFS: Configuration for throughput

- Databases write everything synchronously to disk
 - Log data are written synchronously because of consistency
 - Data file data is written synchronously to track the status of the block
 - A database, which writes data at a rate of 100 MB/Second creates a ZFS Log write rate of 200 MByte / Second
- `zfs set logbias= latency | throughput` (latency is the default)
 - "throughput" means: the data is written directly to the ZFS data area after that: only acknowledgements (no data) are written to the log
 - Datarate is much lower (Just over 100 MByte in the example)
 - More throughput with the same number of disks
 - Needs some disks for the ZFS data (>> 1 disk!)
 - Needs 2 consecutive writes (data and log)
 - Benefits a lot from a Flash SSD for the log

ZFS on SAN Storage Subsystems

- SAN Storage Subsystems are used from multiple Hosts
- Usually the vendors recommend throttling of the Hosts
 - Limiting the number of outstanding IOs per LUN
 - Solaris: `/etc/system: set sd:sd_max_throttle=8`
 - ZFS: `/etc/system: set zfs:zfs_vdev_max_pending=8`
(ZFS default is 35)
 - Numbers should be identical
 - Number should correspond to setting in SAN system
- Other Hints:
 - Configure enough LUNs to fulfill the necessary number of IOs
 - Let SAN department check filling degree of write cache
(should be < 80% and not constantly rising; else: use more disks)

ZFS and Storage Subsystems with Cache

Storage Subsystems with battery backed cache

- Writes are fulfilled in a very short time (usually < 1 ms)
- Data is written to cache
- Battery guarantees, that data is written after a power failure or crash

ZFS switches on the disk write cache (local and remote disks)

- Enhances parallelism when ZFS uses the whole disk
- Data is flushed to disk before a COW transaction commit
- SCSI command "Synchronize Cache" or SATA command "Flush Cache"

Some Storage Subsystems misinterpret the command

- Major Brand Storage Subsystems work ok: Oracle, LSI, HDS, EMC
- Write cache is flushed to disk despite the battery => slowdown
- Then flushing the cache can be disabled:

```
/etc/system: set zfs:zfs_noflushcache=1
```

Databases with Shared Storage Subsystems

- Storage Subsystems are used from multiple hosts
 - Despite defined throttles a bottleneck can exist
 - Projects which use the same storage are not coordinated
 - Throttling the hosts does not guarantee response times
- Recommendations:
 - Load tests with typical application use cases
 - Create performance profiles containing:
#(client requests), #(lun requests), lun response times
 - This allows verification, when other projects are active on the storage
 - Shows influence of other SAN users

Response times are only guaranteed with dedicated storage!

ZFS DB Performance: Prefetch

ZFS file level prefetch is switched on by default

- Full table scans benefit from prefetch

Certain Databases do not use the prefetched data

- But prefetched data use space from ZFS ARC cache
- F.e.: DB has an index for everything
- Read size / DB block size switch on prefetch, but data is not used
- Or ZFS prefetch + prefetch in Storage Subsystem collide
- Prefetch can be switched off:

```
/etc/system: zfs:zfs_prefetch_disable = 1
```

ZFS DB Performance: Switch off ZIL? **No!**

ZIL – ZFS Intent Log

- Applications and ZFS need some data on stable storage
- Synchronous writes are guaranteed to be written before the return
 - Applications: O_SYNC, O_DSYNC at file open()
 - Applications: fsync(), fclose() via NFS, sync() or umount()
- Synchronous writes are implemented with the ZFS Intent Log

Older tuning recommendations suggest switching off the ZIL

- Yes, it can get faster. But: At a crash:

All data in the pool is lost!

- **Better: use a fast device as a ZFS log**

- **Flash SSD (or DRAM SSD)**
- **Or use a SAN device on a Storage Subsystem with buffered Cache**

ZFS Hints

- Use current Solaris Patches
 - Attention: New features of ZFS are only available after an upgrade
 - Use:
 - `zpool upgrade pool`
 - `zfs upgrade fs`
 - Needs some IO activity and locking inside ZFS
 - Do not upgrade during your peak system load...

ZFS and Databases: Summary

Is it possible to use ZFS for Databases? Yes!

- ZFS is slightly worse compared to raw devices or ASM
- But offers a lot of enhancements for administration and production (snapshots, self healing, inheritance, ...)
- ZFS implements an optimization for simple databases
- ZFS runs well after some tuning with major databases

ZFS Performance with Databases

Agenda:

- ZFS Introduction
 - Necessary for understanding of tuning
 - Can be skipped, when ZFS principles are known
- General view on the subject
 - Including Do's and Dont's
- **ZFS and Oracle**
- Some thoughts about the current disks

Oracle DB Performance on Filesystems

Oracle DB is faster on UFS or Veritas FS?

- Yes (UFS and Veritas FS do not use transactions)
- ZFS: advantages for administration and production
 - Snapshots
 - Online check (scrubbing)
 - Self Healing
 - Large Filesystems (xxx TB filesystems)
 - Hierarchical administration
- These are not available for Veritas FS and UFS.
- Performance of Oracle on ZFS is good enough (after tuning)

Oracle DB on ZFS or on ASM ?

... when this is the question of the customer: Use ASM!

- The best performance of Oracle DB is available with ASM
 - ASM is more integrated with the database
 - ASM is not hindered by filesystem semantics

Use ZFS, when

- The highest performance is not the main focus
- Multiple databases or other applications also run on this machine
- Lots of files need to be stored additionally
- ZFS functionality is needed (snapshots, self healing,)
- Administration considers file handling easier than ASM
- Organizational rules prevent that DB admins perform storage management

ZFS and Oracle DB: ZFS

- Use separate ZFS Pools for DB Redolog and DB datafiles
- Adjust the recordsize
 - 128 KByte / 64 KByte
 - For full table scans (OLAP, DWH, DataMart, non-std queries)
 - FRA and Online Redologs
 - Archivelog
 - Tablespaces for direct path sort
 - 8 KByte (or larger) for DB and OLTP
- Basic
 - Use Direct IO: `zfs set primarycache=metadata fs`
 - No atime recording: `zfs set atime=off fs`
 - Set `logbias` to throughput (needs: lots of disks, log on SSD)

ZFS and Oracle DB: System

- Limit the ARC Cache
 - Helps with Oracle DB restart and the use of a Failover Cluster
 - See memory calculations above
- Use current Solaris Patches
- Upgrade Zpool and ZFS

ZFS and Oracle DB: ZFS Log and Cache

- Use a fast device for the ZFS log
 - Flash SSD (up to 100 MByte per SSD, multiple can be used)
 - LUN on a Storage Subsystem with battery backed cache
- Use of a L2ARC is not recommended
 - Oracle 11gR2 has Database Smart Flash Cache
 - Extension of the main memory cache of the database (SGA)
 - Controlled by the database, therefore better performance
 - => Use "Database Smart Flash Cache" instead of L2ARC

ZFS and Oracle DB: Optimizing Writes

- See Metalink-Note: 147468.1

Since Oracle 9i `FAST_START_MTTR_TARGET` parameter is the preferred method of tuning incremental checkpoint target.

`FAST_START_MTTR_TARGET` enables you to specify the number of seconds the database takes to perform crash recovery of a single instance. Based on internal statistics, incremental checkpoint automatically adjusts the checkpoint target to meet the requirement of `FAST_START_MTTR_TARGET`.

- **`FAST_START_MTTR_TARGET = 100 ... 200`**
- The database writes are more continuous
 - No peaks in writing to ZFS, more performance
- Do not set it to 0: Each checkpoint induces a large write peak
- (Source: Oracle Germany STU)

ZFS and Oracle DB: FAQ

- ZFS and Oracle single instance: Yes
- ZFS and Oracle RAC: No
 - ZFS is not a shared filesystem, use ASM
- ZFS and Oracle DB with Direct IO Yes
 - use: primarycache=metadata
- ZFS: switch off single writer lock N/A
 - Only UFS has a single writer lock

ZFS Performance with Databases

Agenda:

- ZFS Introduction
 - Necessary for understanding of tuning
 - Can be skipped, when ZFS principles are known
- General view on the subject
 - Including Do's and Don't's
- ZFS and Oracle
- **Some thoughts about the current disks**

Current disks

- Disks got very fast ... 15000 RPM
- Disks have a high capacity ... 1 ... 2 TB
- Disks have a high transfer rate
 - On disk: 250 MByte / second
 - SATA: 600 MByte / second

True or False ?

Current disks

- Disks got very fast ... 15000 RPM
 - **NOT RECENTLY**, we had this 8 years ago
- Disks have a high capacity ... 1 ... 2 TB
 - Growth slowed down in the last 10 years
- Disks have a high transfer rate
 - On disk: 250 MByte / second
 - SATA: 600 MByte / second

True or False ?

Disks have short access time?

- 15000 RPM = revolutions per minute
= $15000 / 60 = 250$ revolutions per second
= 4 ms per revolution
- Random IO (current SAS disks):
 - Disk head has to get to the right track (0.2 ... 1.2 ms)
 - Disk has to rotate to the right sector (0 ... 4 ms)
 - => ~ 2 ms average access time (peak = not achievable)
 - Usually only 50% of this is achievable
- Sequential IO: No problem
 - Data can be read with physical read speed (~ 200 MByte/second)

How fast are disks with lots of IOs?

- Disks use tagged token queueing / command queueing
 - Commands are tagged (number is assigned)
 - Disk executes the commands in the optimal sequence for the disk
 - Parallel commands visible in: `iostat -xzn`, column: `actv`
 - The answers are identifiable through the tags
 - ZFS switches on the write cache, more than `actv` are active
- Response times indicate, how much random seeks are active:
 - 1 seek: needs 2 ms (above)
 - 2 seeks: the first needs 2 ms, the second 2 ms after that = 4 ms
a random request all the time hits a disk queue of 2
 - Current disks have 128 tags: up to 258 ms response time

When tags are exhausted, the IOs are queued in the OS
(column `wait`) then it is really too slow!

Recent Project at a customer

- Storage renewal
 - 200 disks with 73 GB were replaced with 50 x 300 GB disks
 - Old disks are 10k RPM, new disks 15k RPM
- Random IOs
 - Customer needs 15000 random IOs / second (Oracle AWR report gave this analysis)
- Situation:
 - The customer used only 20 of the new disks (sufficient for the capacity)
=> The performance was bad
 - With 20 disks they had only $20 \times 250 = 5000$ possible IOs!
 - Now the customer uses 80% of the disks and 20% of the capacity
Performance is ok, but some space will not be used ...

Disks get faster and faster?

- Comparison of disks
 - 1990: 424 MB SCSI disk, 12 ms average access time, 500 KB / second
 - 2010: 2 TB SATA disk, 4 ms average access time, 200 MB / second
 - $200 \text{ MB/second} / 500 \text{ KB/second} = 400 \text{ x faster? NO!}$
- Build a 2 TB storage system with the disks, how many IOs?
 - 424 MB disk: 80 per disk, 320 000 IOs (... needs 4000 disks)
 - 2 TB disk: 125 per disk and that means 125 per 2 TB
 - The capacity outgrows the IOs more than factor 2000 (320000 / 125)
- Time for a full disk backup:
 - 424 MB disk: ~ 1000 seconds ... ~ 20 minutes
 - 2 TB disk: ~ 10000 seconds ... ~ 3 hours

=> The development of the disk parameters is **not balanced !!!**

Disk Technology Development possible?

- Example: 3.5 inch disk
 - 3.5 inch diameter = ~ 8.75 cm diameter
 - ~ 28 cm circumference
 - With 250 revolutions per second (15k RPM): 250 x 28 cm/second speed
 - 250 x 28 cm/second = 7000 cm/second = 70 m/second
 - ~ 250 Km/h (= 155 mph)
- Is it possible to rotate faster? No, not with current technology!
 - Today's disks are Winchester disks
 - This technology uses a force to press the head towards the disk surface and the head glides on an air cushion created from the rotation
 - The air is faster when it moves around the head (aerodynamics)
A faster rotation probably creates shock waves which would destroy the disk

Disk Technology Development possible?

- There are other issues
 - Vibration created by the disk rotation
 - Centrifugal force rises quadratically with the RPMs
(15000 RPM => centrifugal force > 4000g !)
 - Positioning on the disk track is more difficult

Determine an IO Performance Problem

How to determine an IO performance problem:

- Eliminate CPU overload:
 - vmstat: idle >> 0%
 - mpstat: not a single CPU is overloaded and hinders the application
 - mpstat: enough threads active in application
- Eliminate memory overload:
 - vmstat: memory-free > 1/32 of the main memory
 - vmstat: sr = 0 (except for short times)
- IO performance problem, when response times are bad:
 - iostat -xzn 5: asvc_t > 20 ms (asvc_t = average service time)

Planning the IO Subsystem

- Take hints from the application
 - Run the application with concurrently: `iostat -xzn 5`
 - Or Oracle: Statspak, AWR report, ...
 - Information about the planned growth
- Now the parameters of the storage are known
 - Size of the necessary storage
 - Datarate
 - Number of IOs
 - The storage has to fulfill all the parameters



Questions?

ORACLE®