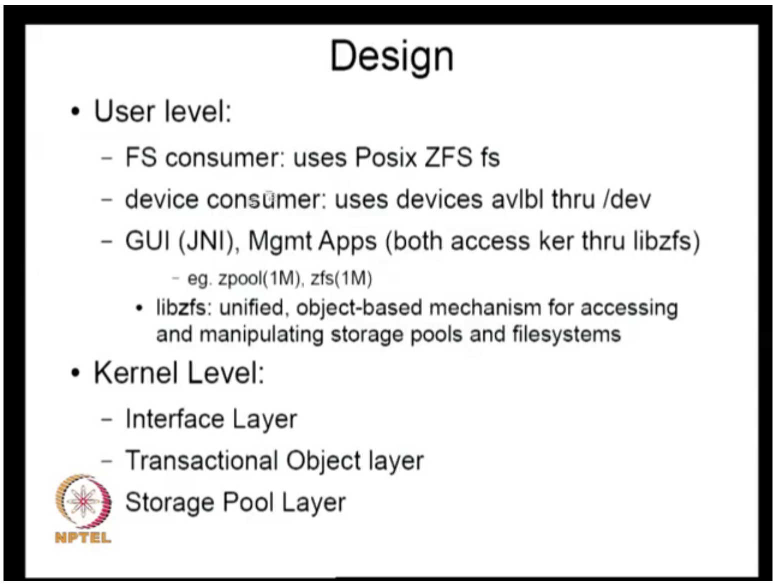
# Storage Systems

**Dr. K. Gopinath**
**Department of Computer Science and Engineering**
**Indian Institute of Science, Bangalore**

**Storage Filesystem Design / Storage Reliability, Performance, Security**
**Lecture – 25**
**Filesystem Design Part 5: ZFS continued, Implementing RAID in Filesystems (Basic concepts)**

Welcome again to the NPTEL course on strong systems. Last time; we were looking at some aspects of the ZFS filesystem. I will just elaborate it a bit more today and then continue with some discussion about how redundancy mechanisms can help you in handling the kinds of errors that typically represent in consistence and storage devices.

(Refer Slide Time: 00:59)



## Design

- User level:
  - FS consumer: uses Posix ZFS fs
  - device consumer: uses devices avlbl thru /dev
  - GUI (JNI), Mgmt Apps (both access ker thru libzfs)
    - eg. zpool(1M), zfs(1M)
    - libzfs: unified, object-based mechanism for accessing and manipulating storage pools and filesystems
- Kernel Level:
  - Interface Layer
  - Transactional Object layer
  - Storage Pool Layer

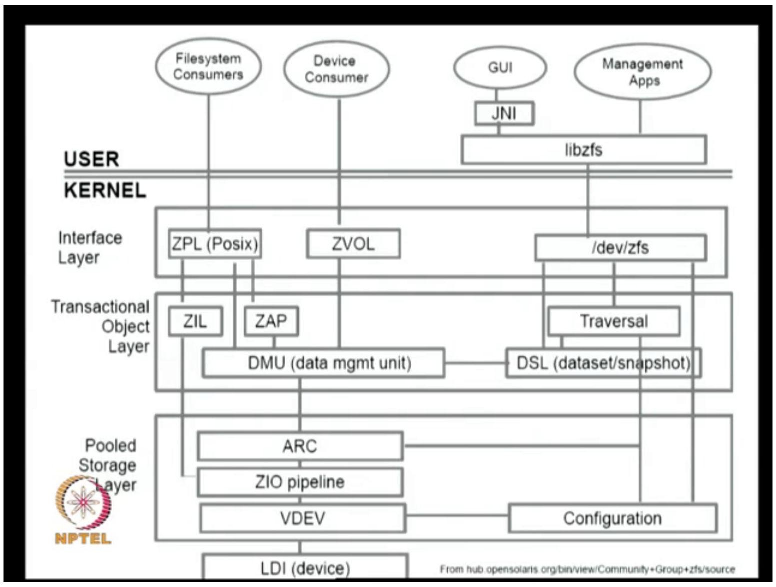
So, again just to recapitulate; the ZFS design has a user level and a kernel level at the user level you are you have the Posix interface. The ZFS interface essentially the file system interface because this is a combined design of both; it is a file system and a volume manager; together, it also can provide certain devices from the storage pool multiple disks are there. So, can provide some block devices. So, we can directly access some of those things and there are some in any complex system there is always what is called a management request; management request by which you can manage the device; it is complex at request management

So, it turns out you can either have GUI based management request or you can have imagine interface through a program likes write written in c language for example, right and the way you access the functionality of the basic primitives of management is, but there is a primit library called libzfs and this what you include and then you get some functionality with it; it define certain the structures and functionalities little bit.

For example you have ability to manage it using this management interface one m with zpool and ZFS things; for example, in ZFS or you can create a file system delete a file system mount a system these are all high level management operations. So, when I pool also you can add something to a pool delete something from a pool these are all storage devices for example, I had a new disc I want add to it. So, these are the kind of things that you can do here ok.

So, the most important thing to remember is it in any complex system, there has to be a management request that is only a; we can interact with it to set it up for the way they want use it without that its actually useless that is why it always says if you think carefully there has to be management request.

(Refer Slide Time: 03:10)



Now, finally, these are user level, then there is a kernel level and I think we saw the picture last time it is this kind of detailed diagram and I went through a bit of it last time I just add a few more comments as we look at it on more time.

So, as I mentioned these are the consumers on the top and this is the functionality packaged or library for management purposes. So, this is a management side of it this is the actual user side of it. So, this libzfs in turn uses some internal device names the reason why you need all this stuff is because this particular system has an ability to do what is called snapshots and clones and other things.

So, the snapshots and clones also can be seen as different devices these are all internal devices and they are named using this particular interface this; say for example, if you have a particular clone its identical to some other let us say version of the file system on particular point in time and I can do what is called a right clone; that means, I have an exact copy of the file system as it exist in a particular point in time and I want to be able to write to it; now that will be given as a new device and then on top of the device, I will have a the regular file system structures on top of it.

So, now when I write to it; it will go through that particular the regular file system call, but we will actually use this device. So, say essentially what this does is it gives you some multiple internal names in the kernel which can be used to access snapshots clones and other kinds of let us say point and time versions of the file system; you need to able to name them you need to able to access them you need to able to manage them that is why all this stuff is sitting on this side ok.

So, for example, I can take a snapshot of my file system and then I can later say that I do not need the snapshot or I can let us say that there is snapshot, I want to access a particular file because I accidentally deleted it my regular directory, I want to access it right. So, when I want to do it; it may be that I need a mounting operation; I need to about to mount that file system.

So, I need to have some kind of capabilities of naming them accessing them managing them. So, all this functionality is exported through libzfs at this point, but also through some device names the traversal is basically I can have a snapshot of a snapshot of a snapshot or I can have a let us say a version of my file system as of last year I want to keep it intact and then I modify something there that is a new version.

I can take a snapshot of that one, then I can take a snapshot later of some modified things. I can have a. So, I need to able to find or traverse and find out which is the if I talk about a file is it the unmodified file of the 2 years back or is it something it has been modified after I took a snapshot or after that. So, I need to able to traverse those files. So, that is where the traverse is comes into picture.

So, let us look at; so, already mentioned about ZVOL is basically this is part; this part is; this part actually is basically nothing more than the regular file system that we know about; it has got the read write; open all those usual things regular Posix come index, then ZVOL is the thing which gives you certain devices because you can see that lot of devices out here, some of the devices normally what happens in a regular storage system with separation of volume manager and file system is that you talk to the volume manager and say give me some raw devices.

Here since an integrated system they are providing a capability by which you say certain portions I want to access it directly as a raw device as a block device and that is what this ZVOL is able to do again Linux. For example, you can have is something called md right and if you give it some devices like slash dev slash; SDA 1, SDA 2; it will provide you with a concatenated device of which you can use part of it for mounting a file system and rest of it can be used by partitioning it, I can use it as a separate device which can be accessed in the using a raw interface without going to block interface ok.

So, those kind of things are also possible here except that all of this is pooled into once again infrastructure right. Now I think I already mentioned ZAP is basically the way in which you want to provide some ability to incorporate directories in system basically I have a way of mapping names to certain objects that is what this ZAP is attribute basically you want to talk about mappings that mapping is basically is ZAP and this have to be transactionally done because you know that when I create a file or create directory it touches multiple places in the disk right. So, it has to be transactional. So, that is why this is a transactional unit.

The ZIL comes in because often times what happens is that database is want something to be as soon as I have finished writing or reading they want that update of your account or whatever transaction have done to be immediately make persistent you cannot wait for it to be left in memory and rest the chance of lost in case there is a power failure or things are what can.

So, you want something which as soon as the read or write finishes especially write you want it to be committed to disk immediately. So, that is that part of it here this part of it is basically a thing which is an intent log which is forced to discriminately that is why this thing some of these structures can be sitting in memory and there will be drained out through the cache to natural storage infrastructure in case you cannot afford to wait for it will be drained out to the normal mechanisms of flushing, then you want a direct path from here to here we can use the direct path. So, that I know that it has to be made persistent and waiting a system during synchronous reads or writes. So, I want to send out the updates I commit, I do transactions and then I log it into the intent log ok.

So, now once it is in intent log it is assistive persistent I might lose power etcetera, but the intent log keeps track of the transaction and in case something bad happens I can recover from the log that is why the ZIL is here. Now good thing about this also is that ZIL. Since it is a log I can put different types of devices more suited to logging purposes.

For example nowadays flash is coming in if we have enterprise flash you can do lot more IO operations by incorporating some amount of flash here. Since it is a sorry; since it is a logging device which is separate it can separate from its regular disk which are out here ok.

So, this is I think I mentioned the there are certain snapshot the all those functionalities traversing all those things are available to here. So, if you want to do copy on write essentially right you have to figure out this will interact with again the transactional layer this is the transactional layer it will interact with it. So, that it will do the copy on write all those things are working the copy on write part of when you update something etcetera all those things are dag in have to be done through transactions ok.

Because finally, what you have to do is when you do copy on write you are going to create a new object which has the same set of objects as the previous version except modified in one or 2 places where you made to copy on write right so; that means, that yeah this is not this again is not a atomic operations it is not; it cannot be done atomically on a disk; that means, you have to do a transaction for that also that is why this stuff actually depends on the transactional unit and with that when I mentioned transactional layers all this things are typically first done in a memory log.

You Cavite in memory log and then you try to drain it out every because you want to batch it again we are talking about disks very slow. So, you cannot do it every time you want to you cannot push out transactions immediately because we do it then you have to suffer the latency of the disk. So, what you want to do is to keep the memory based transactions and drain them out by doing clustering and then once they make it to the home locations then we can release the space along that is that is basically the idea that is the standard.

So, basically this is the part which actually does all the logging in memory and then this is being is being cached here this is what it is called adaptive replacement cache this is a slightly. Let us say modified version of regular buffer cache basically, what happens in a buffer cache is that it usually tries to optimize with respect LRU least recently used block.

Now, it turns out that often times LRU is not useful if you are streaming workloads. So, you want to be able to figure out how to a portion the amount of buffer space that is available to LRU kind of workloads and to streaming workloads. So, that is why this is an adaptive model which will look at the way the it has certain self-correcting abilities it looks at the kinds of misses are taking place and then decides that this is closer to a streaming workload and then gives it more space for streaming.

This is basically it in some sense it has an notion of current available let us say space for streaming versus LRU kind of workloads and then it essentially simulates what would happen if you have a slightly extra space this way or that way depending on the losses it can figure out that this whatever is coming in should be given slightly more streaming buffer space rather than what is currently possible. So, it actually has a correct self-correcting ability that is what this arc.

Now, once you are in the arc this is basically the caching; sub; sub layer and then you have to so often, you will be flushing it to disk and that is where the ZIO; pipeline comes into picture and the this intent log also is connected to this because if you want to flush anything with disk immediately you all can go through this.

If there is some reason by which you have to really make it sit in disk as I mentioned this is already logged here, but if you want to make it flush the log. So, that actually the sets in home locations actually is where it should I finally, build then this can directly go from here to this set because this has got the reason why you see the multiple entry points to ZIO pipeline some of it can come like this or some of it can go through all these things and come here and ok.

So, now notice that there is also some other complication that occur here you notice that you might also have some redundancy management going on, but if some might have what is called read one that is mirroring or read file all that kinds of things and that also all these are managed out here I will come to that soon, I will just talk a bit about this, but basically the important thing to notice is that at this point in time everything is a virtual block.

Because basically what this the for example, let us say the file system is what we are using right you thinks that it has got a block device and this block device is of some arbitrary size and with arbitrary liability properties; that means, it can have a it is been replicated 3 times or replicated twice or it has got a particular type of redundancy that is abstract notion that a file system is got.

So, the file system basically just knows that it is writing onto a block device and the redundancy part is have been handled by somebody also, but it and that part of it is being done by this part. So, when since it has got only the notion of abstract device this device actually could be pieced together with lots of disks out here lots of disks for example, there could be hundred disks out here ok.

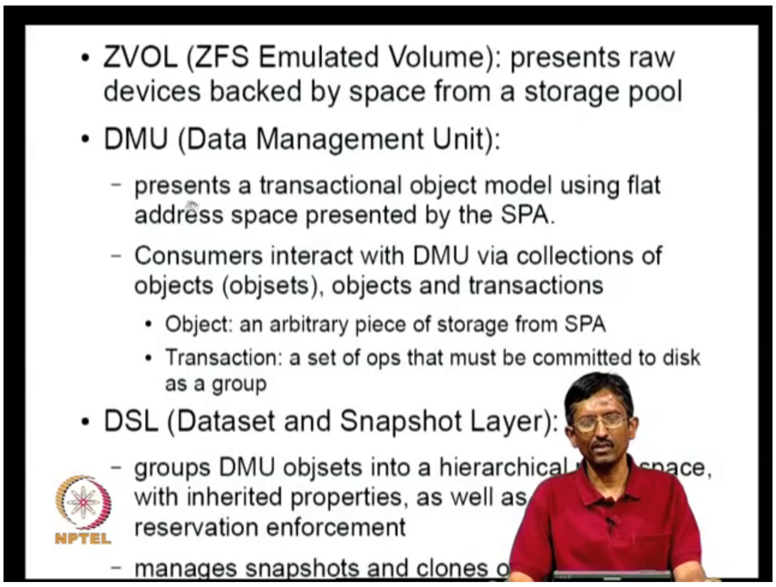
If somebody has crawled out; out of this hundred disks someone terabyte block device for example, or something all it can and then that one terabyte disk itself could be mirrored or it could be an RAID 5 configuration read 6 configuration. So, what we have is certain virtual devices which have internal structure this virtual device itself could be coming from multiple physical devices and the virtual device itself could be mirrored and it could be spanning 2 disks.

So, to make all this reasonably easy to do they have the notion of virtual device and then there is some kind of a tree like structure by which you will say that this virtual device is composed of this part of this one terabyte disk and this part of this other one terabyte disk and this itself is going to be mirrored, I get some amount of displace by combining these 2 and I am giving you some mirrored capability; that means, that how the thing is here and how the thing is here in terms of one half of the mirrors this side one of the mirror this side or combination of this all this things are left open to the device configuration through the management request ok.

So, at this point in time the Vdev is basically; what it does is it translates whatever things that had to be written it goes through all this multiple of indirection figures out exactly in which device its actually talking about because it could be that it is the virtual device that we have is concatenated from 2 terabyte disk and then that device itself is for redundancy purposes is going through mirroring and; that means, that it is using for mirroring purposes that using some part from this disk and some part from this disk or combination of various parts here and there.

So, somebody has to map what this guy is talking about to which parts of the disk it will be talking work that is what this all this term what is done here and then once it is done then we can do the actual IO operation this is basically the actual device driver ok.

(Refer Slide Time: 19:39)



- ZVOL (ZFS Emulated Volume): presents raw devices backed by space from a storage pool
- DMU (Data Management Unit):
  - presents a transactional object model using flat address space presented by the SPA.
  - Consumers interact with DMU via collections of objects (objsets), objects and transactions
    - Object: an arbitrary piece of storage from SPA
    - Transaction: a set of ops that must be committed to disk as a group
- DSL (Dataset and Snapshot Layer):
  - groups DMU objsets into a hierarchical space, with inherited properties, as well as reservation enforcement
  - manages snapshots and clones o

So, I can; I will just run through it one more time, just to make sure it is clear. So, the ZVOL is basically presents a raw device and it is backed by space from a storage pool multiple disks also there.

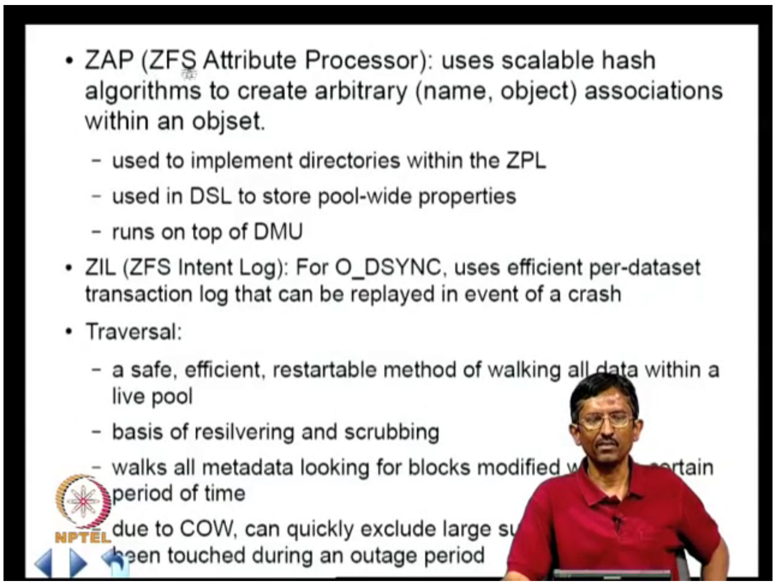
Data management unit is basically it is a transactional it provides a transactional model this the flat address space we are talking about is basically all the devices that are available for example, in Linux it will be the standard slash dev is da one is da 2 without virtua memory virtue without a volume manager whatever the devices are

And the consumer the guys up stair guys interact with DMU via collections objects and transactions sets a objects; objects and transactions. So, a transaction is basically a set of operation that must be committed to disk as a group. So, as I mentioned to you this snapshot layer basically depends on a base file system to get its space and it acts a certain other additional copy on write objects right.

So, there is a hierarchy that is there now because you can say that one snapshot it is based on one file system and this snapshot also itself could actually support some other later snapshots. So, there is hierarchal space and there could be some inherited properties with respective for example, most file systems if my snapshot would have lots of same facilities as the previous it might for example, I might mount this particular file system in particular way probably the snapshot of thing also will have the same kind of parameters ok.

So, unlikely that I have a file which I am doing let us say I am doing what is called data and metadata logging, but the snapshot thing I will only do metadata logging usually that going to happen if you are doing snapshot typically all the things that I have snapshot have the same property that is why there are inherited properties and similarly, there are all this usual things about quota and management also rolled in this point.

(Refer Slide Time: 21:54)



- ZAP (ZFS Attribute Processor): uses scalable hash algorithms to create arbitrary (name, object) associations within an objset.
  - used to implement directories within the ZPL
  - used in DSL to store pool-wide properties
  - runs on top of DMU
- ZIL (ZFS Intent Log): For O_DSYNC, uses efficient per-dataset transaction log that can be replayed in event of a crash
- Traversal:
  - a safe, efficient, restartable method of walking all data within a live pool
  - basis of resilvering and scrubbing
  - walks all metadata looking for blocks modified w[illegible]rtain period of time
  - due to COW, can quickly exclude large su[illegible] been touched during an outage period

As I mentioned earlier ZFS is basically a way to create name object associations that is how they implement directories and they use it for also keep interact of information about the pool wide properties because various different pools might have different attributes for example, one set of pool might have RAID one attribute some other pool objects together might have RAID 5 attribute or whatever.

So, all those things are also given and as I mentioned to you there is a intent log which basically is a specifically tailored for databases or when you want to make a data synchronous based on the disk you have to log it the data is going to be logged into this part. So, it is not lost in spite of whatever happens and this will be finally, since it is already logged it can be sent through this will interact later with the regular ZIO mechanism.

So, that it finally, goes and to the home locations and this is basically; this traversal part in basically a way to go through multiple versions of clones and snapshots for example,. So, all this is being supported on a single pool of storage. So, you need to have some way of traversing this things. So, it also helps into do what is the re-silvering is basically you have mirror one mirror fails then it is called re-silvering to copy it from the existing the remaining mirror; mirror remaining copy you want to copy it onto the new set of disks that are going to take its place it is called re-silvering.

Scrubbing is one in which usually what happens is that if you look at certain larger disks, if you do not keep accessing, it sometimes there are some errors called latent sector error that can develop. So, what you want to do is to every. So, often keep reading them and then in case you are not able to read them then you use your standard error correction mechanisms to rebuilt it again.

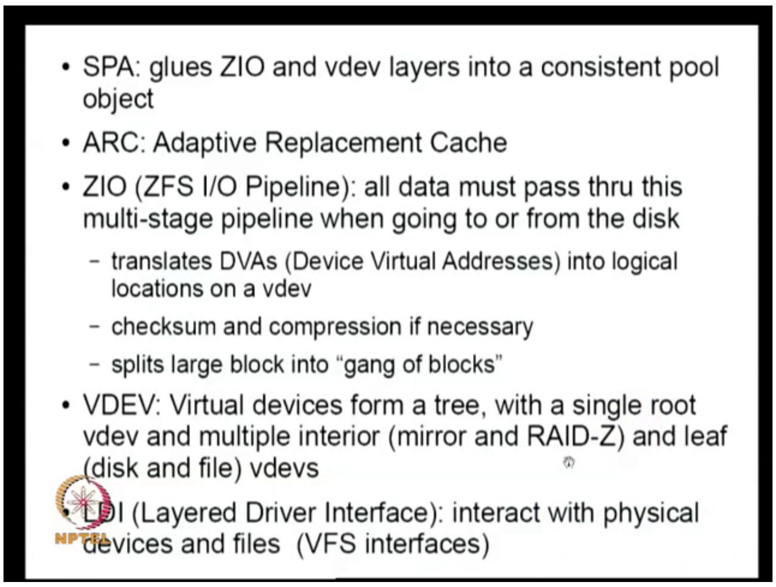
If you do not do this the time when you want it; it may be that you wanted to retain when you are already in an error situation you already in error situation and then you are trying to read it and then it guys comes back and see this is an error at that point you cannot anything because you do not have access to data which is used for fixing a problem because you cannot read it ok.

So, to avoid the situation you preferably try to keep reading all the disks memory. So, often hopefully in non-error situations, so that if there is some problem reading a disk the standard error mechanisms for recovery will be available that is the idea

So, there are various other the traverse are also gives a some other powerful capabilities if you want to find out what all has changed within particular period of time we can do it because usually there is a copy on write with respect snap snapshots and clones etcetera

So, because of this you are able to figure out what is the modified state right now without too much trouble sometime that is important for backup purposes or you want to restart something you want to know what has changed. So, the traversal mechanism because of the copy and write nature; it allows you to do some of this is quite quickly.

(Refer Slide Time: 25:36)

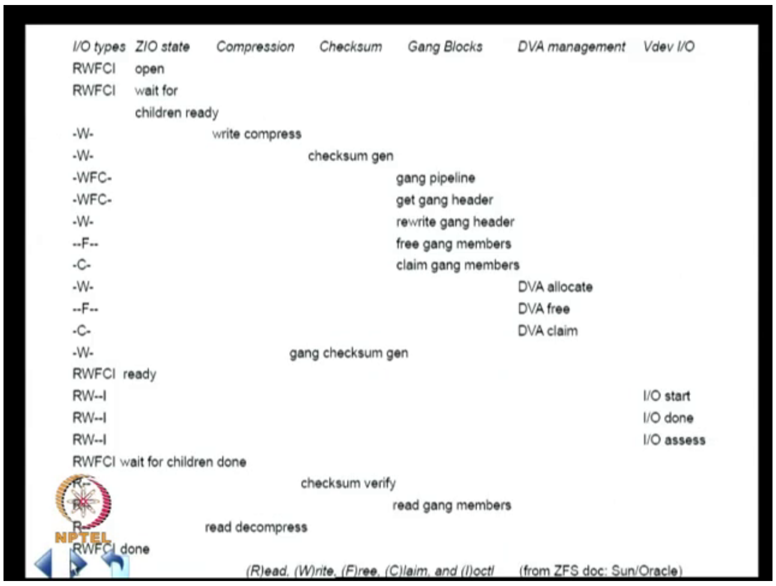


- SPA: glues ZIO and vdev layers into a consistent pool object
- ARC: Adaptive Replacement Cache
- ZIO (ZFS I/O Pipeline): all data must pass thru this multi-stage pipeline when going to or from the disk
  - translates DVAs (Device Virtual Addresses) into logical locations on a vdev
  - checksum and compression if necessary
  - splits large block into "gang of blocks"
- VDEV: Virtual devices form a tree, with a single root vdev and multiple interior (mirror and RAID-Z) and leaf (disk and file) vdevs
- LDI (Layered Driver Interface): interact with physical devices and files (VFS interfaces)

I think I already mentioned about this is basically the storage pool allocator which is basically the third part if you look at a thing this is a this is the holes is all this stuff is basically; this is the cache I already mentioned this pipeline basically it turns out there are lots of phases that an IO operation must go through for example, you might have encryption or you might have compression or you also might want to checksum because you want to check whether the data is actually valid, right.

So, what they have done here is to provide a pipeline by which all the operations go through just like a you know you know you know machine you have pipelining the machine; you also have a pipelining of operations out here ok.

(Refer Slide Time: 26:40)



| *I/O types* | *ZIO state* | *Compression* | *Checksum* | *Gang Blocks* | *DVA management* | *Vdev I/O* |
|---|---|---|---|---|---|---|
| RWFCI | open | | | | | |
| RWFCI | wait for children ready | | | | | |
| -W- | | write compress | | | | |
| -W- | | | checksum gen | | | |
| -WFC- | | | | gang pipeline | | |
| -WFC- | | | | get gang header | | |
| -W- | | | | rewrite gang header | | |
| --F-- | | | | free gang members | | |
| -C- | | | | claim gang members | | |
| -W- | | | | | DVA allocate | |
| --F-- | | | | | DVA free | |
| -C- | | | | | DVA claim | |
| -W- | | | gang checksum gen | | | |
| RWFCI | ready | | | | | |
| RW--I | | | | | | I/O start |
| RW--I | | | | | | I/O done |
| RW--I | | | | | | I/O assess |
| RWFCI | wait for children done | | | | | |
| R-- | | | checksum verify | | | |
| R- | | | | read gang members | | |
| R- | | read decompress | | | | |
| RWFCI | done | | | | | |

*(R)ead, (W)rite, (F)ree, (C)laim, and (I)octl* (from ZFS doc: Sun/Oracle)

So, I think; I am not sure, it is very clear, but basically you might find; for example, compression is one phase checksum is another phase actual IO is one phase; for example, you have a write here you might want to open the device it may be that the IO is multiple children because one IO might be composed of multiple sub IOs.

So, you want to wait for all the IOs to be completely processed for issuing it out that is what this is you might want to compress all of them you want to because you doing compression; for example, or encryption and then you want to generate that because writing it you might want to checksum because you want to write the checksum as I mention in the ZFS. There is a checksum that is kept as part of metadata away from the where the data is right. So, that you can check whether metadata the checksum there is kept to the metadata it is the same as what is there in the data and since that 2 separate places on the disk hopefully one error cannot actually curve both of them.

So, this also has to be done and this is required in case an IO is you do not find enough space in a single disk for example, you have to get some spaces from multiple areas. So, there is a big IO I want to do not everything is available in one single place. So, I have multiple sub IOs; I have to split it. So, that I get some space from here some space from here and some space from there. So, that is what the gangs are basically one IO becomes multiple gangs and this is basically the part which does things like finally, I have to do the allocation because when I am writing to it because this remember that most of system is basically does not do any write in place ok.

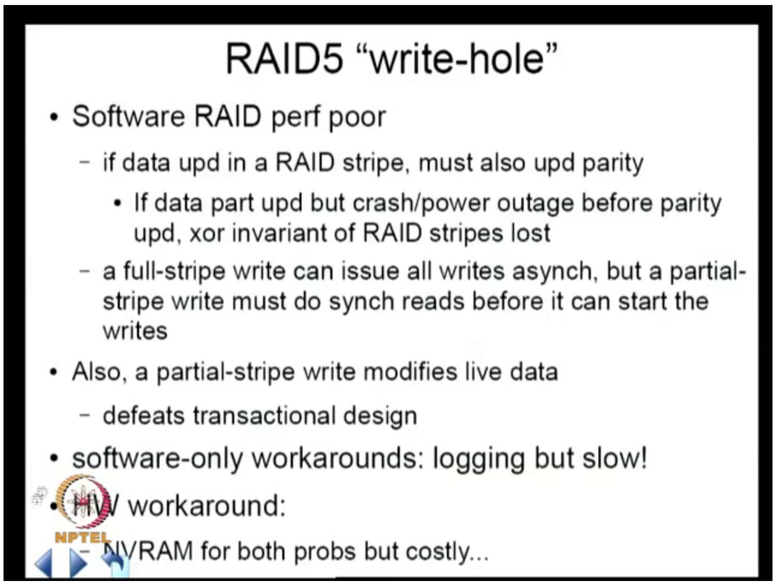
Since there is no write in place going on here you have to essentially allocate new places where it is going to go that is why all this stuff about allocation is there whenever it doing a write and then after that you have to do the actual IOs and you wait for all the children together of course, I am simplifying it a bit too much this is for write you can see any time; there is a write right all these things are going on this is not part of the write. So, that forget about it this is part of a write this is part of the write ok.

But as you want to go read only you start from here it is open the read also might be composed or multiples my multiple sub IOs, then there is no r here. So, you wait for all the children to get ready they become ready now you can start reading all the IOs, then wait for all the children to be finished all the sub IOs to be finished you verify the checksum and then in case it may be that may need to decompress it because read also could be compress I have to decompress it.

So, the read logic will be slightly different than for example, read logic will ways basically go from open the device wait for children to be ready do the decompression verify the checksum and then you can increase; there are multiple sub reads you have to do you have to pick each other things and then check the checksum for each one of those things. So, there are multiple phases and each of this read writes actually go through those phases, but basically the way they done it.

So, again one important way it has to do is to take this virtual block addresses and convert it into actual physical locations that translation has to be done here and in case there is a the Vdev; finally, is a thing that handles whether its mirroring or RAID etcetera. So, this is the part which we will go into the bit in detail soon. Now this the thing which actually does the physical devices or it may be that your file system actually is built on top of some other files, then it has to go through the VFS interface; for example, you might have things like loop devices, then you will do it.

(Refer Slide Time: 31:31)

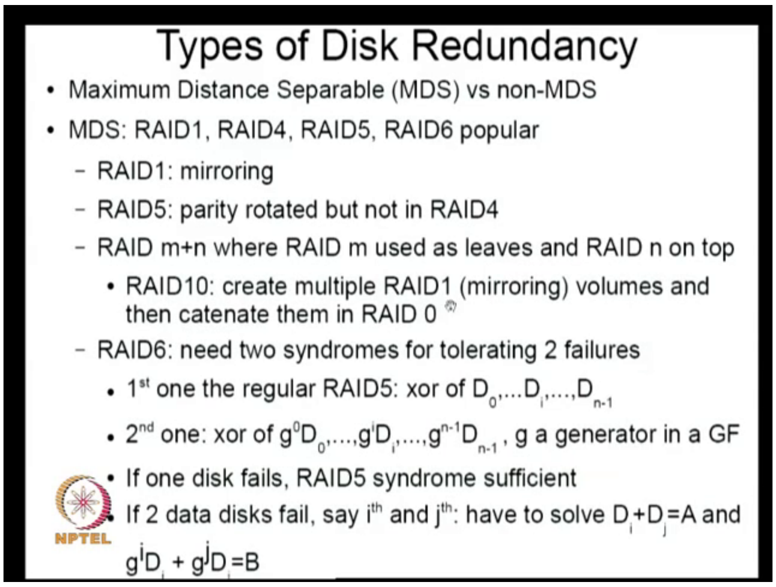


## RAID5 "write-hole"

- Software RAID perf poor
  - if data upd in a RAID stripe, must also upd parity
    - If data part upd but crash/power outage before parity upd, xor invariant of RAID stripes lost
  - a full-stripe write can issue all writes asynch, but a partial-stripe write must do synch reads before it can start the writes
- Also, a partial-stripe write modifies live data
  - defeats transactional design
- software-only workarounds: logging but slow!
- HW workaround:
  - NVRAM for both probs but costly...

So, now, let us look at one interesting thing that ZFS does. So, for this you have to understand something about RAID 5.

(Refer Slide Time: 31:49)



## Types of Disk Redundancy

- Maximum Distance Separable (MDS) vs non-MDS
- MDS: RAID1, RAID4, RAID5, RAID6 popular
  - RAID1: mirroring
  - RAID5: parity rotated but not in RAID4
  - RAID m+n where RAID m used as leaves and RAID n on top
    - RAID10: create multiple RAID1 (mirroring) volumes and then catenate them in RAID 0
  - RAID6: need two syndromes for tolerating 2 failures
    - 1st one the regular RAID5: xor of $D_0,\ldots D_i,\ldots,D_{n-1}$
    - 2nd one: xor of $g^0D_0,\ldots,g^iD_i,\ldots,g^{n-1}D_{n-1}$, g a generator in a GF
    - If one disk fails, RAID5 syndrome sufficient
    - If 2 data disks fail, say ith and jth: have to solve $D_i+D_j=A$ and $g^iD_i + g^jD_j=B$

So, I think before I go it let me just go slightly further ahead and then try to look at types of disk redundancy.

Now, there are 2 types of models one is called the MDS model and then non MDS model the MDS model, it is basically you can handle a certain number of failures, it does not matter which stub failure which disk fail, etcetera, it is uniform any 2 disk fail, I can handle it or any 3 disk fail I can handle; it thinks; this mostly that kind of model here basically there is some specificity with respect to if to disk D 1 and D 3 fail, then I can handle it possibly, but sometimes I am not able to handle some other 2 disk fail.

Basically because the way the parity is computed is done through a certain that done by using certain combinations and in some combinations; you can recover from it in some combination you cannot recover from it of the type of the failures. So, this is the one which is most popular because you are guaranteed that any 2 errors can be handled whereas, here it may be not possible in some cases; for example, 3 errors are there not all 3 errors can handle some 3 errors can handle, but some 3 errors may not be able to handle is it.

So, of the MDS these things are quite popular RAID one RAID 4 RAID 5 and RAID 6 RAID 6 is just about being used, but this things are use quite a bit what is the RAID one RAID one is basically mirroring; that means, you have a piece of data you have another copy of it on a different disk ok.

Now, you can ask yourself how much is it being copied is it at a bit level is it at a byte level word; level is it at a block level normally when I talk about disks we try to RAID the at a block level typically it is 4 kilobyte or something 8 kilo byte etcetera. So, when I doing mirroring; that means, that I am doing at the block level. So, basically I have 4 kilobytes, I have written onto one disk I make sure that the same 4 kilobyte information is also written to another disk. So, in case one fails I can figure out from another one.

Now, one thing one should remember in in the case of memory versus disk is the difference that in memory then it fails you cannot figure out there is its not self-describing in terms of where the failure is because when a memory fails, it is a very simple device; it cannot tell you that I failed in some sense whereas, a disk can will tell you that there is some additional information basically because you go through a certain sophisticated protocol if you are not able to read it I will tell you that I cannot redo; I cannot give you, it gives some information back, there is some level of additional information saying that I failed.

The memory can for example, can give arbitrary wrong information it will not tell you to go it something wrong that is why it turns out if you look at these kind of systems if there is a single failure I can figure it out that which disk actually failed and then I can use parity for example, to fix the problem in memory based systems if there is a I have a parity a single parity if there is a; if the memory is gone bad; all I can know is that the memory is gone bad I cannot fix it ok.

Whereas in disks because the disk tells you that I went bad; therefore, I know how to fix it memory does not have that capability that is why if you look you might have heard about ECC; ECC is basically error correcting and error; error ECC what;

Student: Code.

Error correcting code; error correcting code; ECC, right. So, you notice that unless you have with a single parity; it cannot do it you can you need at least 2 bits. So, without 2 bits that is no related to correction whereas, with the disk we just do parity on the concurrentives the difference being of course, the as I mentioned already because in some sense disks because of complicated gadgets they are there are some self-description capability with respect to error reporting they will tell you later I have failed, but memories do not have that capability that is why you need ECC for correcting code parity is not enough here parity is enough.

So, the RAID one is basically mirroring and there is also other varieties RAID 2 use a something called hamming code the regular hamming code which also is used in memories basically you take disjoint sets of bits usually 2 to the power of pi. For example, where I equal to 0, 1, 2, 3, etcetera and then have parity for all those things and then you can essentially figure out where the error because depending on which parity is not working out you can figure out where it actually failed that is the basis of ECC essentially.

Now, RAID 2 also can be similar things you can use hamming code here and typically it is done at the bit level. So, RAID 3 also is at the level of word the word level and these things require whatever called synchronized disk windows means suppose you are want very high speed extremely high speed access to disks, then you can take multiple disks and then synchronize the speed set with the right the synchronize completely synchronized.

So, if you read one bit from here we can read a corresponding bit from the other disk at the same time. So, then you will essentially get if you have ten disks or let us 16 disks;

they are essentially getting 16 bits every let us say bit time 16 bits we are getting it. So, you can do it, but the problem typically is that getting the 16 disk or 8 disk or 4 disk properly; synchronize is a very tricky business that has been done in the past was done in the early 1980s, it was done because disk was slow and for doing some critical things you needed that speed and you take the trouble of synchronizing the spindles this spindles and you would get 4 x improvement in throughput etcetera.

But nowadays nobody almost nobody does it, there are still possibly some very demanding applications where synchronized spindles are worth the trouble, but generally nobody uses it. So, that is why RAID 2 and RAID 3 are not used RAID 4 is used not very commonly, but there are specific designs like netapps design where RAID 4 make sense for them, I will come to that next they use RAID 4.

Raid 5 is the one which is widely used if RAID 1 is not being used typically RAID 1; RAID 5 are the typical choices and many your mother boards or RAID disk controllers; they support RAID 1; RAID 5 typically there is also something called RAID 0 which is not redundancy model it is basically a concatenating model it basically takes 2 disk and gives a bigger combined virtual disk.

So, RAID 0 is not really a redundancy model. So, you have; so, RAID 4 and RAID 5 basically RAID 5 what happens is that you take a few number of disks and then compute the parity and store it on other side, but if you do it for all the blocks then the parity is going to be used all the time the parity disk because for every write you have to keep on accessing parity disk right. So, it turns out that the parity disk could be the bottleneck because of the parity say any access anywhere you have to; for example, write access for example, you have to write the parity of the move data right; that means, anytime you do any write you have to actually access the parity disk and therefore, depending on where the data is there could be lot of 6 going on all the time.

So, to avoid the excess activity on the parity disk a RAID 5 what it does is it rotates the parity what it means is that you take the first 4 blocks and have the parity on fifth disk then you take the for the next block what you do is you take the first 3 blocks and the last disk block and put it in the fourth disk it basically store the parity on a diagonal form.

So, that is what they call it rotated RAID 4 does not have rotation basically there is a recreated parity disk and that comes into picture for every parity write every parity write it comes into picture. So, of course, one good thing of RAID 4 is that if you have a different technology medium like for example, flash and if that is being heavily used for example, then it may turn out that putting that parity disk and a new made technology might help out also some people have proposed those kind models also whereas, RAID 5 may not be able to use it because they all have to be the homogeneous disks all of them, they cannot have a separate technology for a parity alone.

So, the good thing about RAID 5 is that it essentially you have 5 disk arms if you have a 5 disk system 5 disk arms for parity not one disk arm and these and why this important is that you might be able to do partial writes for example, I write only 2; 2 disks D 1 and D 2, then I also have to still update D 5, the parity disk right later I can write D 3 and D 4, I still have to update D 5; in case it is RAID 4 whereas, I can do it on 2 different disks in case it is RAID 4. So, RAID 5 sorry, ok.

So, RAID 5 is common except that this RAID 5 can be done in hardware or it can done in software the good thing about doing it in software is that you are now independent of the disk technology you can mix this of different types different technologies it can be for example, who knows a sata disk with a SCSI disk or I can have a sata disk with a sas disk I can do all this if I am doing it in software.

Whereas for do it in hardware typically the hardware manufacturers they design it. So, that all the disks are of the uniform same type of disks because the way the hardware is done basically connectors and enclosures all these things typically or decided by various other factors mechanical other factors. So, it turns out hardware RAID 5 or RAID 1; if it is done in hardware it demands uniformity of the disk infrastructure disk, etcetera.

Whereas only good in software querying you are basically independent of that particular part, but doing it in software it is very slow. So, RAID 5 is not really popular in done in software then is too slow and also if you do RAID 5 in hardware the hardware secretary can do parity computations and parity let us say raids is a priority computations quite fast in hardware because its mostly xor you can already have do some simple xoring code xor logic which is very easy to do it in hardware ok.

So, normally you will find that RAID 5 is done in hardware, but it makes your you; you lose flexibility with respect to what kind of disk drives you can use it has to be of the same type usually of the same size also; for example, all should be one terabyte all should be etcetera these kind of things.

So, you also have things like RAID m plus n where you use the RAID m part on the leaf side and the RAID n on the top of it. So, basically you are doing 2 level of RAID now for example, if I do RAID 1 0. I basically create multiple RAID 1 volumes that is mirrored volumes on top of it I concatenate them I get that is why it basically RAID one 0 I can also have things like RAID 5 1 that is what I have is I multiple RAID 5 devices RAID one means I basically also mirror them. So, I can have all kinds of devices of this kind depending on your requirements. So, RAID 6 0; for example,; that means, I have multiple RAID 6; let us say the systems multiple RAID 6 systems and then I am joining them together in a concatenated fashion on the top all these things are possible ok.

So, essentially we can create arbitrary trees of structures. So, basically RAID one is simple basically just mirroring exact copy is kept RAID 5; what we are doing we are taking the data disk at a block level and we are computing the parity and storing it in the parity disk. So, parities being computed here in RAID 6 you want to be because as I mentioned in RAID 5 if there is an error in one disk the disk will tell you I had made an error and I am not able to supply the information because timeouts and also these kind of things there is some way to figure out that a disk is not working which disk I know which disk is not working because I know which disk is not working since I have the parity I can compute the value of the block on the disk which not working I just have to do xor of everything else, right.

So, that is what I am getting by RAID 5. So, it is simple if there is a single error I can handle it the problem with RAID 5 is that I have what is called if there is an error I have to read the rest of the disks what happens if while reading other disks I have I suffer a failure and that failure is often possible through what is called latent sector errors; that means, that I will try to read something at that time the disk comes back and says me I cannot read this term it is not a whole disk failure it is just that I am not able to intersector I think those of your familiar with floppy’s and other kinds of devices will notice that sometimes it reads something sometimes it does not read some things, right.

So, certain parts the disk may have gone back only certain parts; it is able to read other things quite well. So, not everything is known why it happens how it happens, but general is that it is possible for you to try to read something and you fail reading a particular sector or 2 other sectors in nearby rest of it is completely accessible ok.

So, my problem is that when I am fixing a RAID 5 problem that is one disk has completely failed its catastrophically failed; that means, there is no way to evaluate. So, what I am going to do is I am going to read other disks xor it and get the value of the block on the disk we just like completely and hopefully I have a spare disk I am which I am going to transfer it this is basically my model it recover from.

But my problem is that when I am reading other things the other guys can also if our luck is bad; you are unlucky.

Student: (Refer Time: 48:32).

That one of the guys will say I cannot read a particular sector.

Student: (Refer Time: 48:35).

Disk is fine mostly, but one (Refer Time: 48:37). So, I cannot recover that portion that part of the block on the spare disk I cannot recover to it. So, this is what is called the one and half failure this is the term that is used one and half failure

So, this is sufficiently common that many people are not comfortable with that level of protection that is why I have to do what is called scrubbing I think we talked about it previously in scrubbing what to do is we access every single disk every twenty hours or forty hours something that you just read it in case there is a problem when I am when the latent sector error occurs then hopefully at the same time there is no catastrophic failure of a disk at the same time whenever I am doing scrubbing hopefully.

So, therefore, I can actually recover from it I will rewrite it again or I will retry it into a different place right I try reading it I am not able to read that. So, I can the disks have some mechanisms by which they can do what is called revectoring that is you are trying to read from block number x you could not read it then you are able to recover the value from using RAID 5 the xoring part then you write it to another block x prime which is basically there somewhere as a set of blocks available for revectoring.

So, whenever x is mentioned actually goes to x prime this is there is some space in the disk for its what is called what is it called? It is called spare; spare block area something is called; it is a providing a name. So, there is a small set of blocks are available. So, that anytime you have this problem you can write it to that place and anytime you ask for x actually it automatically also it talking about the part which is gone bad is go on give you from x prime settle it over.

So, because of this. So, people who are unhappy about the possibilities like this these errors likes latent sector error they go for scrubbing. So, when you read the when you are enable to read from a particular place use RAID 5 logic to get at the block in a revector it different spare area on the description.

So, now you are same, but of course, that area is sort of limited hopefully you can only you can at the most probably handle some hundreds or so. So, blocks as you go back if that happens more than that you might you might have thrown with the disk and get it, ok.

Now, those parties who are not comfortable with this even at this level of this thing then it go to RAID 6. So, you want to be able to take care of situation where 2 catastrophic realistic this not one in the RAID 5 you handle one catastrophe if you remember complete disk completely died there is no response with matter.

So, here what you do is you in RAID 5 you are doing xor right of the disk blocks and storing it in the parity right in the second one what you do is you use some summary stone mathematics here. So, what you do is in this case you say that there is a generator in a field and you basically what you do is g to the power of 0 is a generator multiplied with the first block D 0 and you xor all these guys together ok.

Basically you take that here basically start with z D 0, D 1, D I, D n minus 1 say what you do is multiply this with g 0 g to the power 0 this g to the power I g to the power n minus one and then do the xor the same thing you do. So, as usual if one disk fails you can still use this parity and survive just like a RAID 5 if 2 disk fail suppose its ith and jth. So, what will happen I have the xor of all these guys right that is equal to some value right and I have value for everybody else right.

So, the only thing I do not know is di and dj. So, di xor D j plus everything else is equal to the known value therefore, I can get di plus dj equal to some value. So, that equation I will get one whereas, here also this also is again xor. So, I had of all these guys now all this values are known except I and j. So, I basically have the sum of gi to the power of di gi D to the power jth for dj this will be there rest of its all known. So, I finally, get this it turns out if you choose appropriate field and whatnot this is always solvable you are always get di and dj always do all you have to do is substitute for example, di into this you can always get dj from it always that of course, the trick is basically is the g; you generate it.

So, you need to study something about what is called Galois fields to figure out how why this works, but I do not know go in to much with here, but there is a way to if you choose your Galois fields, correct, but you will always be able to do it always being do it the only problem with this solution is that this requires let us say solution the solving this thing and also you have to produce this syndrome every time any times and data is modified you have to come up with a new syndrome ok.

So, it is expensive right people have figured out how to compute this efficiently using for example, in turn has got some better instructions for example. So, there are ways to do all this things efficiently, but it still takes time. So, most of the times if anybody has done RAID 6 it has been done in typical with hardware support, but people also have used using software for doing these things.

For example, if you look at netapp; netapp has got something similar to RAID 6 and so quite a bit I think quite a bit of it is done in software (Refer Time: 55:28) there. So, I just want to briefly mention various types of redundancy here. So, normally RAID one and RAID 5 are what is widely used RAID 6 for more demanding applications where you cannot handling errors. So, I think I will continue with some other issues with RAID 5 which ZFS status all and also lets look at and also look at netapp the RAID 4; why that particular thing actually makes sense we look at in next class.