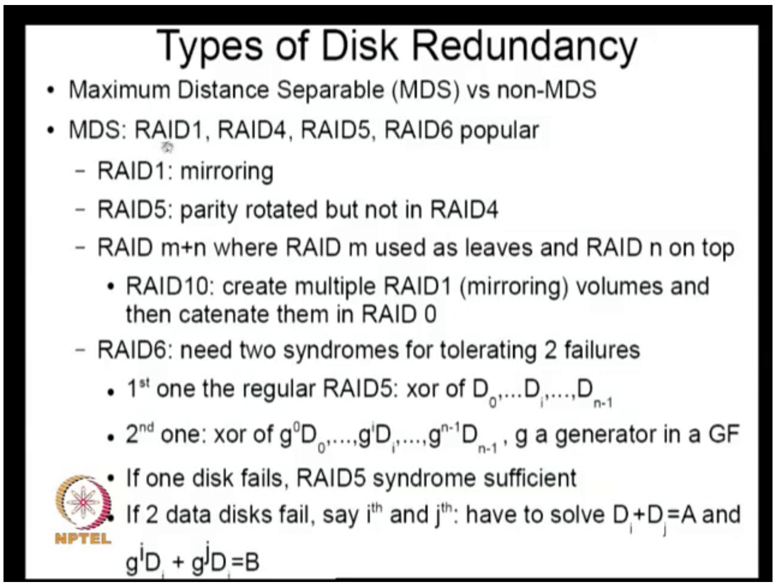
Storage Systems

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

Storage Filesystem Design / Types of Storage Devices and Systems, Long-term Storage / Storage Reliability, Performance, Security
Lecture - 26
Filesystem Design_Part 6: Implementing RAID in Filesystems continued (RAID-Z and NetApp RAID4), Flash Storage Systems Technology & Flash filesystems

Welcome again to the NPTEL course on Storage Systems. In the last class, we were looking at some aspects of redundancy.

(Refer Slide Time: 00:38)

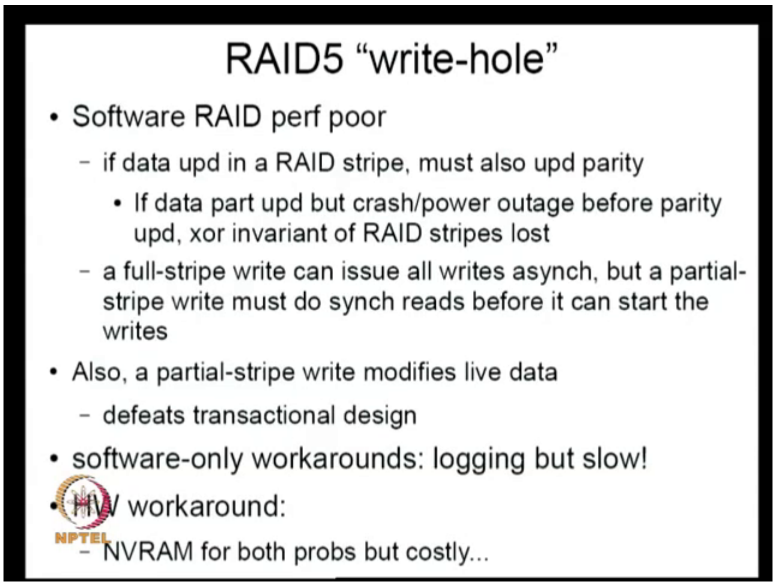


Types of Disk Redundancy

- 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 $g^iD_i + g^jD_j=B$

And then I briefly we looked at what RAID1 means what RAID4 means RAID5, RAID6. And I briefly went through some of these things in some highlighting some aspects.

(Refer Slide Time: 00:54)



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...

Now, we will get back to the ZFS. Now, as I mentioned earlier RAID5 has a problem, which has been called a write-hole by this ZFS designers. And basically I will give some details about exactly what this is. First let us should be clear that if you do RAID and software the performance is poor why is that we first have to bring the disk blocks to memory, compute the parity and write it back. Again the CPU utilization that solves the fact that you have to traverse the network from the disk all the way to the memory there could be substantial (Refer Time: 01:49) example to the storage area network, it might had go through network controllers, cache controllers as well.

So, but as I had mentioned earlier software raid is very flexible we are not limited to exactly one type of disk or whatever, it can be done on slightly or can be done on arbitrary types of storage devices. So, software RAID is RAID5 a bit of a problem; it should be software RAID5, it is not just RAID. If data updated in a RAID stripe we have to update the parity also, because there is an invariant network keeping. What is the invariant, the invariant is that like I said in last class basically the parity is basically the is the XOR of D 0, D 1 0, D m minus 1 this invariant only we have to keep the parity will always be the invariant of the XOR of all these cases.

So, if you update the disk, then you have to update the parity also, then only the invariant is maintained. But our problem is that the data part updated, but there is a crash or power outage before parity updated then invariant is lost.

So, we somehow have to make sure that we can avoid this situation. You can certainly not avoid a power outage and this is not something we have any control over, now we have any control over what is called a crash a panic kernel panic, there is no control over this can come in any time so that means, that somehow we have to recover from this situation. So, there are various ways to handling a problem; one way is to say that I will always write full stripes, whatever I do I write full stripes. If I do full stripes, that means, at even if something happens, I just have to redo it, I just redo it everything will be fine again.

Even if I did something halfway, something bad happened of course, the invariant is no longer true, but if I redo it again one more time then everything will be still be ok, as long as I can has a ability to that. So, that means that somehow you need to do some way you have to remember what is happening. So, you might have to do some login etcetera, but once you have login that means, it has to be somewhat persistent.

There are multiple ways of doing this. One is you have a disk called NVRAM that is battery backed memory and essentially you keep track of this information log it in NVRAM. And then because of battery backed, in spite of whatever crash or power outage, we may able to recover from it, so that is why there is a (Refer Time: 04:46) NVRAM is that it is fast and it is a different technology RAM disk. So, we will find that (Refer Time: 04:56) might be less and just by seeing, but it is a good solution, that is why NetApp for example, this is NVRAM and many other companies use it, so that is one part.

Other thing is if you are doing full the reason why it is good to always to do this full stripe write, is that you can do all the raids in synchronously. And as long as you are able to recover from any problems that occur you are ok. And the problem with the partial raid stripe is that you have to do a read modifier for RAID cycle; that means, you are going to have to read it first that means, the read latency is very much in the path linear path. So, it is going to be very slow. So, the performance is poor.

Now, because of the performance is poor, people like to use something like a NVRAM or some logging device to get to write to essentially do the probably the write to these devices, and then later those writes will trickle down to the actual RAID device. Previously, we talked about logging and a file system, but now this is a logging and a volume manager or at the device level. So, basically there are software only workarounds, but this is slow basically logging.

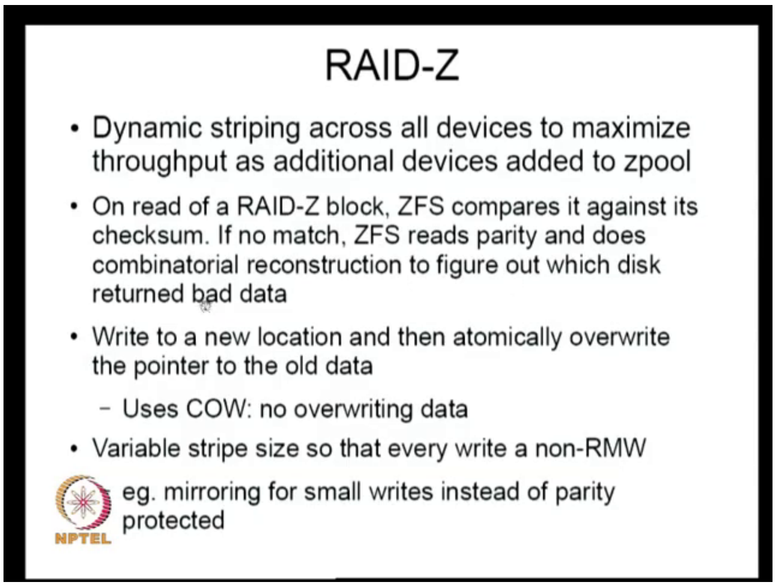
If you use NVRAM it will not be slow it can be fast, but if you use another disk for logging then it is going to be slow and similarly the lot of people are now suggesting instead of using NVRAM, which is costly, why not use flash because it is persistent probably that might also. Only problem flash is that its write stripes are not that very good compared to disk (Refer Time: 06:53) it does not have the synch time slash rotational latencies that is where the big win is. Big wins for flash in this game is that you do not have a read and write latency, the rotational and the synch latencies, which can be as much 15 milli seconds, 10 to 15 milli seconds, so that will essentially limit the logging speed to not more than above 50 to 100 per second or so.

Whereas, if you use something like a flash, your number of loggings you can do can be factor of 10 or 100 times more. So, as I mentioned the hardware workaround is for use something like a NVRAM and this can be so both the problems one for avoiding the RAID performance, and the another one for the write-hole. What is the write-hole basically you have a mentioned yesterday, you have there is a failure of one disk; that means, that disk is not responding at all. Your timing out you ask for a; you are trying to write on the guys is coming back and not giving any response (Refer Time: 08:030 writing it. So, you know that because there is no response back, you know that something has gone back. So, you want to fix it.

So, you want to fix it, you have to first reconstruct that particular stripe or the reconstruct that disk that means, you have to read all the other disks, so that we have we can copy to onto a stripe disk. But while you are reading it, you will have to do whole reads of the other disks. And there is always a phenomenon in disks that if you read there is no guarantee that you will get let us say correct results for this. There is always some residual what is called uncorrectable errors, example there is something called in disks you have an uncorrectable error of one bit per ten to the power of ten bits that memory also has got similar array you want (Refer Time: 08:57) you have some similar numbers.

So, because of this that disk dies and you want to reconstruct it, reconstruct it so on a different stripe disk, during the reconstruction it can also fail that is the disk write-hole. So, to avoid the write-hole, basically we have to again to login. So, basically you can combine both these two things, avoiding write-hole it means you use a log and improving the raid performance also can be a log. As long as the logging device is fast it will work out a way. So, the one issue has been that the best logging device for this kind of issues has been NVRAM which has been expensive it requires some battery backed memory which is also expensive. So, what ZFS wanted to do was to see if there is a way to avoid this problem.

(Refer Slide Time: 09:49)



RAID-Z

- Dynamic striping across all devices to maximize throughput as additional devices added to zpool
- On read of a RAID-Z block, ZFS compares it against its checksum. If no match, ZFS reads parity and does combinatorial reconstruction to figure out which disk returned bad data
- Write to a new location and then atomically overwrite the pointer to the old data
  - Uses COW: no overwriting data
- Variable stripe size so that every write a non-RMW
  - eg. mirroring for small writes instead of parity protected

So, they came with some model called RAID-Z. And what these guys do is they do what is called dynamic striping, because this design is an integrated design that means, it has got both the file system and the volume manager. The design has been done together, it is not by done by two different parties. For example in Linux the file system is designed by one guys the MD or whatever volume manager is there is done by somebody else. Same thing in Veritas file system slash Veritas volume manager though for a same company that done by two different teams. So, it is not an integrated design.

The ZFS on the other hand has been designed to be an integrated design. So, basically what we do is what is the problem with writes, the thing is you might want to do partial writes that means that instead of suppose you have a RAID device it has got five disks of which four are data devices, and one is the partial that is a one disk is used for parity. Of course, the parity is going to be split across all the disks, but for the time being let us assume that the stripes for example. Four disks will have the stripes and one disk will have parity.

Now, if you are going to do any partial write that means we are only going to write that is a 64 kilobyte, where the stripes adjust on 32 kilobytes that means, it cannot be a full stripe write. A full stripe write will be 32 into 4 that will be 128 kilobytes, we are not doing 128 kilobytes we are writing only 64 kilobytes. They since you are writing 64 kilobytes, you have to update the parity also. The only way to update the parities to read everything and then change whatever has been changed and writes it back, so that is a read modify.

So, again we as we discussed before there read modified RAID cycle essentially can clear the situation where the invariant may not be upheld in case there is a crash or whatever. So, what these guys did, ZFS did is why not make every write somehow a full stripe write, everything somehow. And how can it be done, you are having you have 128 actually four disks are there and principally it has a 128 kilobytes stripe and which provides a parity. So, what this guys decided whatever it is since where the file system information if somebody writes 64 kilobyte will call it mirroring; if it is 128 kilobyte it will be let us say a RAID with 3 disks plus 1 parity that means, that the whatever size that you write you make it a corresponding RAID. So, the writing only one stripe, it will be RAID1.

If you write two stripes, for example, if you write 32 kilobytes you basically make it into a full stripe write with a parity of another 32 kilobyte that is two 64 kilobyte write we may get into a 64 kilobyte write plus one parity. So, whatever writes you write it is always a full write you keep on varying the geometry of the RAID device that you are thinking about. And this information is known to the file system the file system knows at this point in time, because at integrated design, I am writing 64 kilobytes. Therefore, I know that it is not a really the full stripe write as if we done by regular treat because it knows it is not a real full 128 kilobyte write it will say that for this particular write it will be a two stripe raid plus one parity again this has become a full write now.

So, if that is why what we have done is the systematically made that every write is a full stripe write a under RAID is definition of a full stripe. So, this essentially avoids the write volume because even if something that happens right we just have to redo it that is all, you do not have to do anything more. We do not have to worry about losing the invariant, so that is why it is called dynamic striping. It is not fixed stripe across RAID to maximize throughput (Refer Time: 14:21).

Other thing is that you also as devices are added to the system, you can also take care of example you had a five disk system, one new disk is added, now you do not have to worry about or reconverting your RAID5 and five disk system to a six device six disk system. We can dynamically how new stripes of with five disks someone parity. So, when one thing also goes away, we can also try to readjust it within the future. Of course, you may have to work around how to recover from what has been lost, but in the new inserted, you do not have to let say you can start using the reduce size of the set of disks that you need to for a full stripe that also can be, because the information is kept to the file system.

The file system knows all about these things. So, when you have to traverse the things you go for that file system and traverse it. Whereas, if it was not at another file system then we would not know what is there, there is no way to figure out where that information is has to be kept or basically about the fact that it is this particular two piece of data actually are read two sorry read five with two stripe units for the parity. All that information will be scattered all of the place nobody has that information and it is only known to the file system. So, that is not known and so when we have to recovery from that situation it is not possible; whereas, the file system keeping which you can do it.

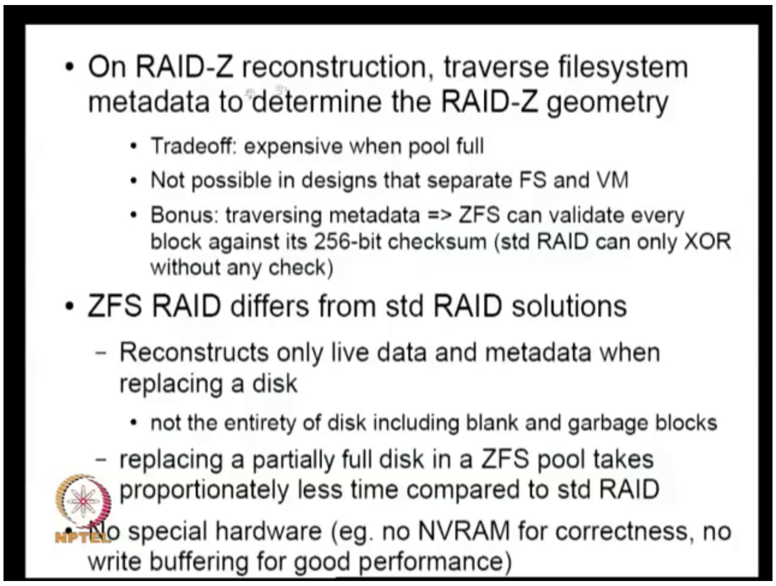
So, when we read from a RAID-Z block for example, also ZFS compares it against its checksum. If there is no match, ZFS reads parity and does combinatorial reconstruction to figure out which disk returned bad data. How does it happen again it turns out that there is somewhat what is called a (Refer Time: 16:15). What is (Refer Time: 16:17) basically I have a checksum of some leafs of some structure, and then the checksum of the second level tree is completed again the checksum of the next higher level. So, basically tree power tree of checksums.

So, if any time there is a problem, we have to see we have to search in the trees see which checksum is actually bad. We might have a top level thing wrong then you have to going down and then try and both sides of the trees which checksum actually is not correct, and then keep going down that path where it is not right till we get to the place where actually the checksum is found. So, essentially you go down the path which has got try to locate the place where the checksum has gone bad, so that is for read.

For write, what you do is as I mentioned this one never really does no overwriting of data, you always write to any location and then atomically overwrite the pointer to the old data that is why it is not fetching the things are revealed here. So, as I mentioned already variable stripe size, so that every write is a non RMW, you do not have to do re-modify so; that means that you are raid that performance problem is gone now. So, that way the only problem of course, is that you are getting variable raid geometries for a same application.

So, the only good thing about is that you might have asked for RAID5 type of reliability, but sometimes you get an RAID1 type of reliability, just a little fine and it is not a serious problem except that you have disc utilization would be not as good if you were to have done systematical raid for everywhere right. So, there are some minor problems, but they say that this avoids lots of the headaches that are there in a other systems, just a minute.

(Refer Slide Time: 18:36)


- On RAID-Z reconstruction, traverse filesystem metadata to determine the RAID-Z geometry
  - Tradeoff: expensive when pool full
  - Not possible in designs that separate FS and VM
  - Bonus: traversing metadata => ZFS can validate every block against its 256-bit checksum (std RAID can only XOR without any check)
- ZFS RAID differs from std RAID solutions
  - Reconstructs only live data and metadata when replacing a disk
    - not the entirety of disk including blank and garbage blocks
  - replacing a partially full disk in a ZFS pool takes proportionately less time compared to std RAID
- No special hardware (eg. no NVRAM for correctness, no write buffering for good performance)

So, as I mentioned already when you have to do a reconstruction traverse file system metadata to determine the RAID-Z geometry. So, you are going to start from the like FSCK here also you start from the power of file system hierarchy. And every single thing you know the file system knows what type of greats geometry test use it for it, every single block or whatever right or extent knows what it is. So, using that will actually goes and traverses those structures in term and at every level it knows what the thing is right.

Again when the file system is not full it will only have to traverse a small amount; when it is really full, it traverse the whole thing, it is just like a (Refer Time: 19:21) not very different from it. But one interesting thing is that these form a different from a strict volume and the design; in a strict volume and the design what happens we have no idea about what is useful information, what is not useful information, because there is no metadata corresponding to data. So, we actually have to do a block by block comparison; and it turns out that the way it is done on size by using something called (Refer Time: 19:50) logging in a traditional volume manager like for example, the way transfers regardless of the volume manager and other kinds of designs similar designs.

You need to keep something called a dirty region logging that means, when you are writing to a for example, a mirror for example, it is possible that if you have written on one and then before you finished it, you could not complete other one. And then you have to resynchronize in mirrors. Now, if you do not have any information, then you need to compare both of them right and some of you got which is a more recent version. And then use that as the new copy and overwrite the old copy, but that means, that you have to go through the whole disk, the full virtual disk (Refer Time: 20:35) have to go through.

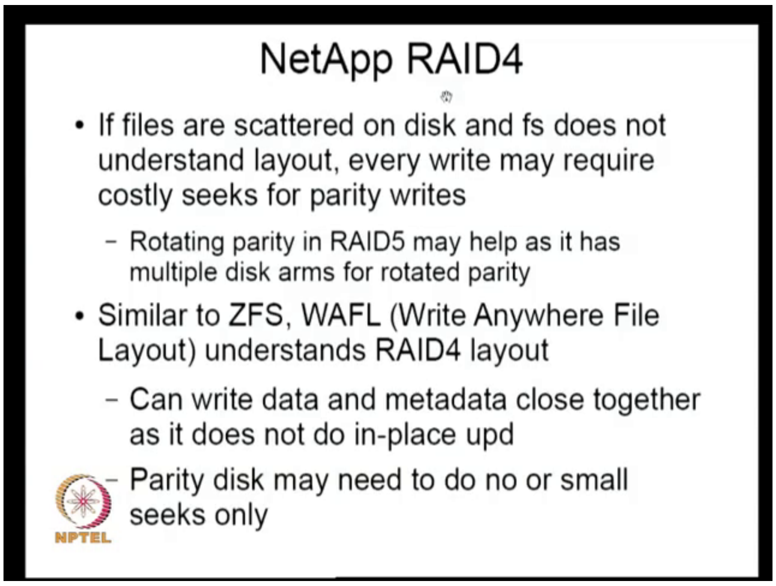
Whereas, in ZFS what is called dirty region logging, you keep a small little another small area on the disk, where you note down disk like logging, you note down which sets of logical block addresses are getting written, and they are not been written on other side. You have to keep noting all these things. Just like logging, you also have to note somewhere. So, in some power crash whatever happens we can look at this dirty region logging and we are able to figure out that only this part has changed not everything. So, if you do not have a design that so basically unless we use dirty region logging we have a serious problem with regular value managers, you have to really essentially look at a whole disk it can be very slow.

Whereas, in this kind of situation, it is only expensive when the pool is full; if it is the file system is not highly populated then you just do whatever is required that is one good thing about this kind of designs. So, also while you are traversing the metadata, you can keep on essentially you can use likes the checksum you can keep on checking this checksum as you go along. So, again standard RAID all you can do is to check the XOR invariant that is all we can do, nothing more can be done. Whereas here, you can also do the checksum, because the checksum is kept separately from the actual data, so that is available with file system level, so you can keep on checking as you go along. Whereas, volume manager is typically they do not have this kind of simplicity of doing it, if they have to do it.

So, basically in some sense you just formulize ZFS raid differs from standard raid solutions, how reconstructs only live data, (Refer Time: 22:40) you do not have to look at the whole disk. Because this new information at the volume manager level which is a blank or garbage block, because it is raw device, this no information about the disk block is being used also no nobody knows it that is the difference of block device. The block device in sense that you get is get raw blocks and we have no idea whether it is being used or not, only the upper layer is known. So, when something bad happens you have to copy the whole thing, even if there are absolute garbage which if you had some higher level information, you know that is a garbage there is no need to copy anything.

So, another thing about this kind of model is that you do not need any special hardware no NVRAM for correctness, no write buffering (Refer Time: 23:24) you do not need any of these things. So, I think we have already discussed there are plus sides in this RAID-Z there are negative small minor negative things. So, because there is checksuming also going on, so it is a reasonably well integrated system for getting good performance as well as avoiding some correctness problems.

(Refer Slide Time: 24:21)



NetApp RAID4

- If files are scattered on disk and fs does not understand layout, every write may require costly seeks for parity writes
  - Rotating parity in RAID5 may help as it has multiple disk arms for rotated parity
- Similar to ZFS, WAFL (Write Anywhere File Layout) understands RAID4 layout
  - Can write data and metadata close together as it does not do in-place upd
  - Parity disk may need to do no or small seeks only

Now, let us stripe to look at another design. NetApp has got some one of the fews or people use RAID-4 is NetApp; there are few other people use it but they are very special applications, but these are more systematic use. Now, one of the problems about RAID5 is that if the parity is on different disks, because it is rotated because rotated what happens is that when you are doing data reads you have to skip over parity disks also parity blocks also; that means, you have to seek past it. So, there is going to be some loss of performance because of that.

So, basically what happens in RAID5, the lot of data stripes one by one, but suddenly somewhere in between after we will get a parity disk parity block. You have to skip over it where other guys has device. So, basically in some sense what will happen is that they will keep on getting non synchronized as you are reading through the things somewhere this guy will face a parity block and sometime let us this guy will face a parity block. So, you have to keep on going stepping across, so that is one problem.

So, also what happens is that in standard RAID5, if you are using a regular file system and your files are scattered all over the place on the disk that means when you write anything for example, you are doing you are updating multiple files, since they are scattered a disk. If may be that the blocks are in different place in the disk on the corresponding parity disk also parity areas also different places that means, at every write the parity disk for example, has to keep on seeking from one place to another place it has to just keep on seeking, so that is the problem with RAID4.

RAID5 it may be because essentially we have equivalent of five disk counts because parity is distributed over all the things, but RAID5 that is why RAID4 has not been used seriously. But NetApp has discovered a very good use for this. So, again just to summarize RAID4, the problem is that that multiple writes go there the files are in different places and the parities are also could be at different places that mean, that the parity disk always has to keep on seeking. So, you are limited by the how fast seeks can happen because the write speed of the RAID4. RAID5 you have an advantage that your five disk comes for the parity, so it might be helpful, but only problem is that when you are doing reads for example, you have to skip over the RAID parity blocks that is the negative part of RAID5 also.

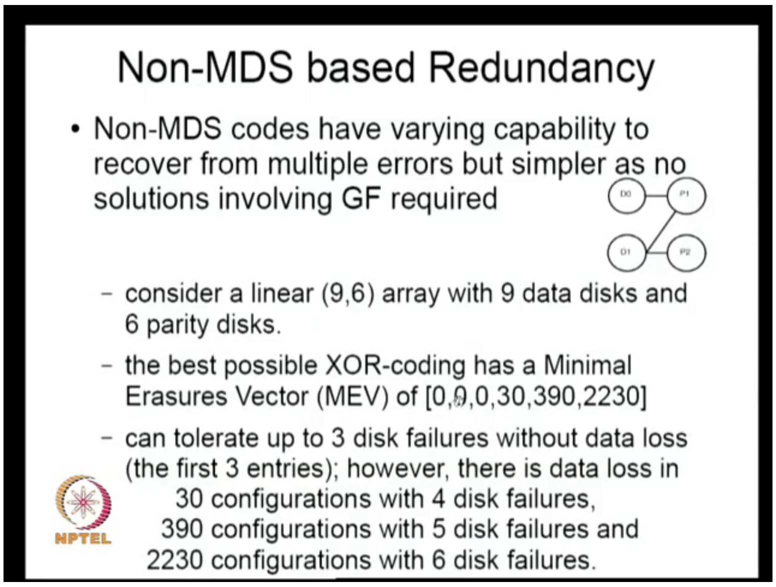
So, what RAID NetApp what has done is because they have this notion of write anywhere. So, basically what it means is that whenever they are doing any bunch of writes temporarily together, then they were all be hitting the same areas close by very close by areas. And the parity disk also the areas will be at close enough. So, basically what happens is that there because the write anywhere aspect it turns out that the seeks on the RAID4 what we have sort of a problem with the RAID4 is not there in the case of RAID4 and that is how they are able to use RAID4.

So, basically you can write data and metadata close together as it does not do in place update. So, standard file system for example, the data and metadata can be at different places. Because it is write this basically will write anywhere that means, the data can go and write metadata also can go, they can go very close together just like in a lock structured file system, so that is why it RAID4 is not a problem for NetApp. And parity disk may or may need to do no or only small seeks do not have to do much.

So, we can see that there is interaction between the upper and lower layers here, you should be very clear. If you have a certain upper level design, then there are some interesting things that depend on the lower level or vice versa. So, in here for example, a NetApp, it has a write anywhere part of it; that means, that you are at a particular place the disk arms at a particular place and idea is that because everything is just now updating place, you can select a new block nearby. And there is some mechanism by which you are making these blocks available. There is some I do not want to go in details, but there is some a garbage correctness that is going on in the case of NetApp file system by which there is always some amount of three blocks available wherever the disk is arm is present there is some mechanism that is why you are able to all this things.

So, we talk about there are two types of redundancy as I have mentioned the one RAID1 RAID4, RAID5, RAID6 they are systematic codes. That means, that they are able to give guarantees saying that any two errors we can handle any three errors we can handle any one errors we can handle etcetera. Then there is other codes the non-integers.

(Refer Slide Time: 30:18)



Non-MDS based Redundancy

- Non-MDS codes have varying capability to recover from multiple errors but simpler as no solutions involving GF required
  - consider a linear (9,6) array with 9 data disks and 6 parity disks.
  - the best possible XOR-coding has a Minimal Erasures Vector (MEV) of [0,0,0,30,390,2230]
  - can tolerate up to 3 disk failures without data loss (the first 3 entries); however, there is data loss in
    30 configurations with 4 disk failures,
    390 configurations with 5 disk failures and
    2230 configurations with 6 disk failures.

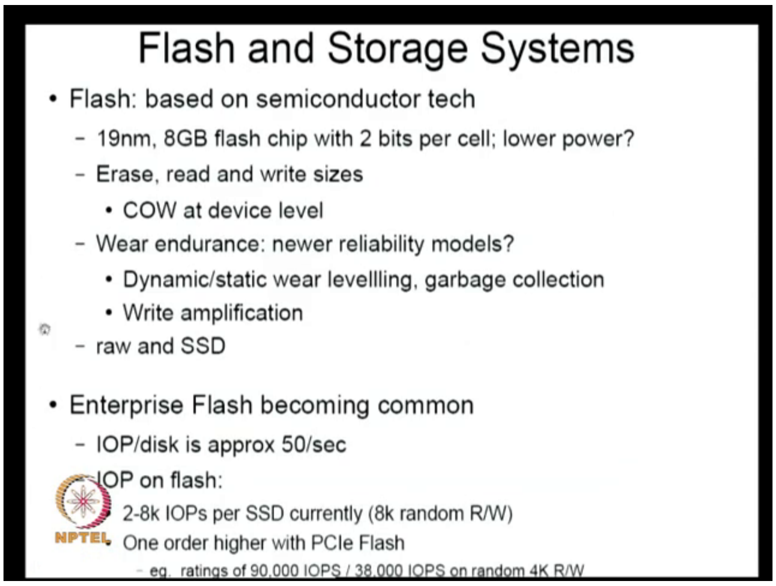
Their varying capacity capable to recover from multiple errors, but good thing about this is that they are simpler as no solutions involving Galois fields, this is as simpler. One example is here for example, there is two data blocks D 0 and D 1 what you do is you take the parity of D 0 and D 1 that is this, but here instead of you are mirroring D 1 in P 2. So, this one is using essentially two disks to get the parity here you are just taking directly this as mirror. So, there are multiple ways in which you can do these things. For example I have one more four five disks I can have parity of some of them P 1 and then P 2 will be parity of some others sets of disks and I can use whatever since it is reasonable to me ok that is possible and there are some designs by which you can do this.

Now, depending on how these parities are let us say computed and which data disks are used for computing a parity, we will have various design possibilities. And for example, if you have a nine disk a six parity system, so it turns out there is a particular way in which you can join various disks to produce some parities that is the and people are computed disks, so that one has a certain reliability measure. For example, the best one has this reliability measure. What does it mean it means that any three disk failures it can handle without any problem because that one is for example, this gives you how many ways you can fail when so many errors are there. For example, you fail in 0 number of ways and there is only one error; you fail the whole system fails in zero number of ways if there is two errors. This is the way in which zero ways in which you can fail in case of three errors; that means, that any three errors it can handle, but there are 30 configuration which it can fail over four errors four disk.

Similarly, there are three 90 ways of doing it in there are, these things have a different types of capabilities here. The good thing about this is that you do not need if you remember this part of it, basically if you want higher tolerance to more than multiple disk failure tolerance. In the case of RAID6, you have to solve this equation this equation is people have done a lot of work on how to make it fast, but still it requires some computational resources.

So, if we use this non-MDS, you do not have to do any serious computation just of the XOR everything is XOR, but the problem with that is that you do not get the guarantee that you can handle k number of failures in all situations that is not possible. You have variable amount of ability to tolerate failure that is basically the problem of non integers, but this looks promising and some people are doing some workarounds in it.

(Refer Slide Time: 33:40)



Flash and Storage Systems

- Flash: based on semiconductor tech
  - 19nm, 8GB flash chip with 2 bits per cell; lower power?
  - Erase, read and write sizes
    - COW at device level
  - Wear endurance: newer reliability models?
    - Dynamic/static wear levellling, garbage collection
    - Write amplification
  - raw and SSD
- Enterprise Flash becoming common
  - IOP/disk is approx 50/sec
  - IOP on flash:
    - 2-8k IOPs per SSD currently (8k random R/W)
    - One order higher with PCIe Flash
      - eg. ratings of 90,000 IOPS / 38,000 IOPS on random 4K R/W

Now, let us move onto a slight different topic. Again ZFS also uses flash and same thing with even the current other solutions on NetApp etcetera, various important players in the game. They all have flash as a part of their system. So, let us try to understand; what is the issue with flash and what is happening here. So, say for we are doing based on a disk base technology; of course, previous to that tape disk technology, which is also still surviving or even flourishing. Because it can store really huge amounts of data and there are certain guarantees with respect to tape being available even after tenth multiple decades, so that part of technology is still going on.

So, whereas, flash is basically a semiconductor technology compared to the other two which are electromechanical, this one is semiconductor tech. That means, what is a good thing about this is that we can now integrate CPU storage and memory in the same technology. Similar tech you know of course, if you talk to the serious IC people, they will tell you that CPU technology is different from memory technology from flash, but in principle from semiconductor fabrication point of view that is the same.

Let that will let to clarify it you can say from fabrication point of view they are all the same. Whereas, with respect to disk they are not the same, there is some totally different thing that happens may have a disk, it has certain you know things like motors or (Refer Time: 35:20) that have to move things around etcetera which is a different technology.

So, I think that the ability to use the same type of fabrication is a great advantage, I think it is gone dramatically start changing things in the future.

So, nowadays you can go to based really highly dense flash based chips. For example, you can get that what is called a 19 nanometer technology this is the feature size, this is the smallest reliably you can mark on the chip 19 nanometers. I guess it is 19 nanometer is basically is how much is it, it is 19 to 10 to the power of minus 9 meters. So, it is 19 into 10 to the power of minus 7 centimeters minus 7 that means it is 190 into 10 to the power of minus 8 centimeters that means 190 angstroms. So, this is 190 angstroms and you might know that one atom is 1 hydrogen atom is how many angstrom. So, we are talking about something like 400 atoms. So, we are talking about 400 hydrogen atoms sitting here that is the size of a lines you can mark on the chip and this is the best on that thing.

And they use something called, so basically we are now people are able to make 19 nanometer 8 GB flash chips. And there are various types of flash designs something called single level cell, and multiple level cells - MLC. And single level cells are of course, the more reliable because there is only two level 0 and 1 equivalent of that; whereas, with two bits per cell there are four levels - 0 0 0 1, 1 0 1 1 in terms of some analog quantity. You have to mark out the analog quantities from 0 0 to 0 1, 0 2 (Refer Time: 37:29). So, this MLC is cheaper, but it has more errors. So, you need a good amount of error coding for it.

So, (Refer Time: 37:47) coding or other types of coding also being used. For example, you might have some XORs, some parities horizontal and some parities vertical all kind of things are different design. You can have grade also on top of these things, so many designs are coming out. One thing about this also is that has a potential of doing lower power. It is still under case that it is a very low power. For example: I recently looked at one new design from Intel they say that for a 800 GB kind of device the amount of power they consume is about 20 or 30 watts which is again very close to what a disk also does. So, the only good thing about this is that it probably currently it is still not really low power of course, this is what is called an enterprise flash basically a PCIE flash the number I mentioned just now.

You might have slightly cheaper versions, which are really low power, but for these ones it is not really low power, so that is a still comparable to disk on in terms of low power. But a good thing about it is that if you are interested in designing system which have a certain transactional throughput then these kinds of systems will give you a higher transactional throughput compared to disk based systems. So, what you would have to do for a disk based system is to have larger number of disks and that will achieve you to the power. Whereas, say or you can work with smaller number of devices and so that way you are going to because that is each of them has got a very high transactional throughput rate. Therefore, your power could be lower than if you were to have a comparable system with disk with the same transactional throughput.

So, now, this flash has an interesting thing that it has got in addition to read and write sizes like a disk, it has got something called erase. Basically once written you have to erase it before you can write again. So, there is some specific sizes at which you can do this, for example, in many systems more recent systems read can be at 4 k level write can be at 16 or 32 k level, erase can be let us say 256 kilobytes (Refer Time: 40:24) somewhere that way. So, or it could be 256 or 512 kilobytes, these kind of ranges

Now, in the sense what is happening is that you are able to because you cannot write in place, because you have to erase it before you can reuse it. It is sort of makes us think that you can do copy and write in the device level. Basically what you have to do is if you want to rewrite, you do not write the same place you write some other place and you change your pointers; otherwise you are going to suffer a read modify write cycle. Suppose, I want to if I want to update a particular location, the only way I can do it in this particular technology is to read the whole thing, update that portion and write back again the same place, because I have to erase itself, I have to erase it first and then write it back etcetera and erase size is usually much bigger than the write size. So, what we have is essentially a read modify write cycle.

So, we want to avoid this read modify write cycle, you might want to think of using copy and write, but these are the happening at the device level. So, far we have talked about it happening at the file system level. So, the question will be can we use this kind of model efficiently for designing large case storage systems. Now, one of that aspect about this system is that in addition to having to erase before we can rewrite, it has got some wear endurance that means, that it can be written only so many number of times.

or example, a cell C kind of system single level cell single level cell – SLC, it turns out that you can only write about hundred thousand times typically. And then with MLC you can write about ten thousand times. So, the manufactures typically make it better than this by using some error coding, they try to add some redundancy at various places. So that finally, the device on hand what we get is they say that you can rewrite it about let us say not exact hundred thousand in case of SLC, but might be closer to one million times that is in additional level of let us say organization has to be introduced, so that you can take to that level.

So, that is why you need some newer reliability models. Again because certain parts of data can be heavily used and certain parts may not be heavily used. For a period of time different parts of the device will get written at different number of times that means, that the wear level across the device will be non uniform. So, you have to figure out in a way of making sure that if one of them gets over one too much then totally you may not be able to use that device. So, you have to figure out a way in which you can wear level the thing uniformly across all the parts of the disk all the parts of the disk of the device.

So, there are multiple methods of doing it there is dynamic models or static models. The names are peculiar, but basically dynamic is what I mentioned just now basically you do you keep on writing to a new place, and then you do something a garbage collection afterwards that is it is determined by bytes. As bytes come in you can keep on putting at some other place; whereas, static wear leveling as it is called.

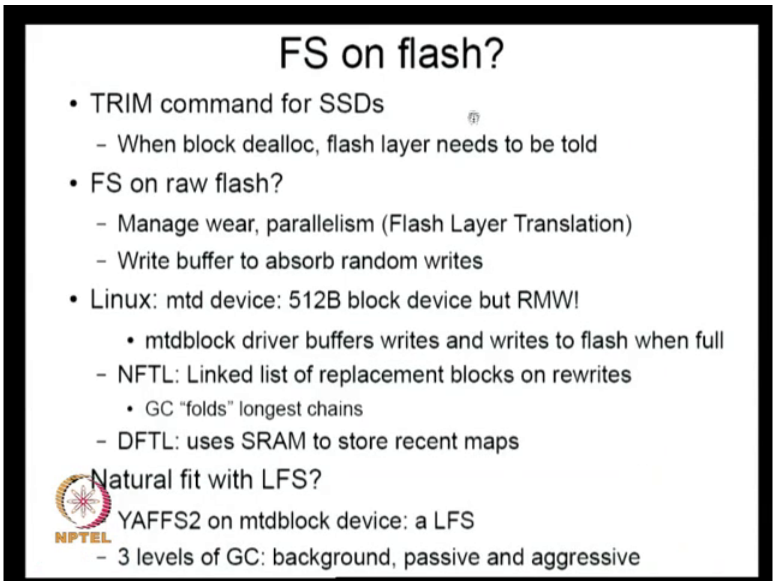
You basically look at the distribution of how many basically independent of writes; we will actually try to move things around. Basically you do a write live data in some other place and then release those things what has been already written out to some other places. So, the issue about this is that anytime you do any wear leveling, you are also introducing new writes. So, there is something what is called write amplification and goes on that is for every write possibly some other writes will come into picture later because you want to do the leveling. So, this write amplification is an issue you want to make sure that this does not become big.

So, flash has got some interesting new issues that has to resolved. We have to do wear leveling and also garbage collection. Basically these two things are part of this system somewhere and that is the difference between raw and SSD. Raw systems do not do any of these things for you, you have to do it yourself. These solid state devices SSDs solids solid state devices, they basically have some policy how to do it, and it is internally done in the device and you may or you may not have complete access to how they do it, but it is done for you. And this SSD essentially gives you the same interface like a disk basically a disk has got a for example, a SCSI kind of interface and then will give a SCSI interface.

Now, enterprise flash is also becoming quite common, basically enterprise flash is what basically the same similar to this flash, but used to an type of (Refer Time: 46:08) or critical information technology systems. So, let us say the point of comparison the number of IO operations you can do for a disk is approximately at the most 50 per second. Whereas, with I with flash we can get over from 2000 to 8000 per SSD; and this is on 8 k random read and write. And if you instead we also have a variety called PCIe flash where it is directly sitting on the PCIe bus.

For example, you can have the much higher ones. For example, instead of 2000 IOPS we can go to probably 38,000 per writes this 8 k is this for reads that is the corresponding here is 90,000. So, you can go a order of ten times or more if you go for PCIe flash. So, of course, more expensive, but this is more of the high performance kind of systems.

(Refer Slide Time: 47:13)



FS on flash?

- TRIM command for SSDs
  - When block dealloc, flash layer needs to be told
- FS on raw flash?
  - Manage wear, parallelism (Flash Layer Translation)
  - Write buffer to absorb random writes
- Linux: mtd device: 512B block device but RMW!
    - mtdblock driver buffers writes and writes to flash when full
  - NFTL: Linked list of replacement blocks on rewrites
    - GC "folds" longest chains
  - DFTL: uses SRAM to store recent maps
- Natural fit with LFS?
  - YAFFS2 on mtdblock device: a LFS
  - 3 levels of GC: background, passive and aggressive

Now, just let us just look at a few more things corresponded to this. Now, that a flash as come in enterprise flash available, do you want to build a file system on top of it. Now, either I am using the flash and through SSDs, now let us just look at because SSD is basically giving a discontinuous. So, essentially there is no major thing here, but it turns out there are still some issues here also. What is one issue is that if the file system reallocates the block, it is just useless right, but this information not known to the lower levels.

For the lower levels point of view it is still a block, so it will not be garbage collected it will not be it will restore the lower level still thinks that it is something useful because nobody told him anything. Because it is just that from the file system point of view it moved from the link list of with respect of file you just moved it to the field list that particular block right which was released then it was reallocated. But how does the lower level know that this is something called a field list and something called a RAID5 list of blocks, so just now information about these things, so it can figure it out.

So, there is a new command that has come in called a trim command. So, if you are writing a file system that is going to do mostly on flash then it may be good for your file system to issue trim commands to such SCSI layer. So, this some trim is basically a command for a new command for SCSI. So, if a SCSI controller gets trim command, it checks whether it is a flash device. If it is a flash device, it actually tells the thing that this particular block is also carrying garbage, you do not have to do keep it intact and the worry about copying it, because you are doing your own your level end or you are moving around things using static wear leveling for example. You do not have to try to copy it out it is basically jump you just can without copying you can move it, you can reassign it wherever you want to do. So, trim command is now part of SCSI you know sort of commands.

So, this is in case you use a CD disk. What about if you use a file system on let us say raw flash; that means, somebody has to manage where and also the parallelism. I will come to parallelism soon. So, basically this software layer which does management is called flash tran their translation FTR, why made as it should of a flash translation I am sorry I got it slightly messed up it, flash translation rate FTR. And also typically you will find that most of these systems have a write buffer to absorb random writes. If you are doing it on raw flash, typically there is some write buffer, so that you do not have to you correct all of those random writes enter pool and then try to send it to the flash.

So, there is some support I will just briefly mention the kind of things that are there in Linux There is in Linux there is a typically you have what is called the block device and a character devices Linux has the special type of device called mtd device. And basically it is a the way it is being designed in Linux, it is a 512 byte block device, but the problem is that anytime you write to a because it is a 512 block device anytime you write to a 512 sector or whatever you might call it, it does not essentially read modify write. So, it is pretty often it is only useful for typically for read only file systems.

So, there is also one type of a FTL - NFTL basically they have what is called primary blocks and replacement blocks and anytime you do a rewrite after particular block, you basically keep it in this you write it into replacement blocks. So, in a sense you have primary block and its updates and there is some kind of changes kept. So, the primary block is a original value and then you have its own updates. So, it will be for the chain of these updated values. And once it crosses a certain number of the size of that that chain, it goes beyond of some amount some garbage collection comes in and then cleans out all those things again. So, something simple like that has been done in NFT here.

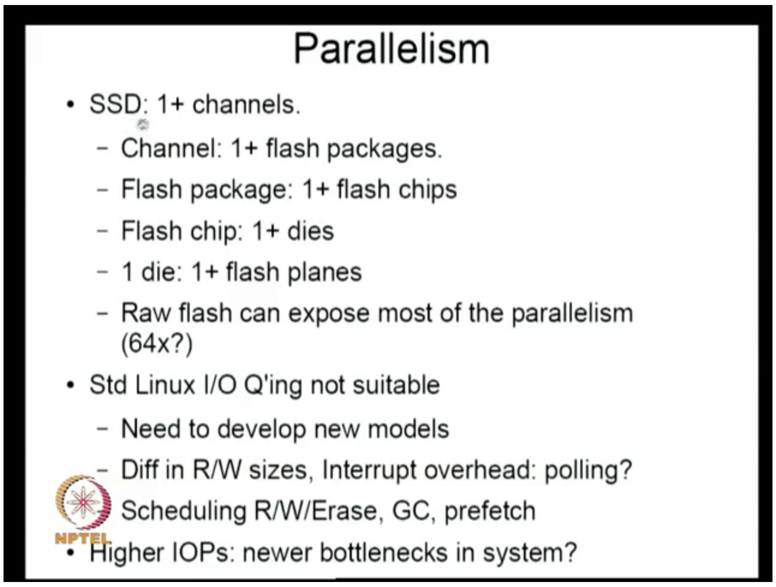
In DFTL, basically, we notice that for doing this copy and write or essentially non-updating place, we have to keep changing the pointers. So, if you have a large flash device, if you assume 512 block devices it turns out the memory required for keeping this point is substantial. It can be in close to gigabytes, so that is why most people keep it in the flash itself that is why you have to go through the linked list of replacement blocks to able to do garbage collection. Eventually, you have to go through all those things. Where DFTL what it does is it has decided that I would not put 1 gigabyte, I will put only smaller amount 64 megabytes or whatever, I will store the recent maps in SRAM. And rest of it will be as usual in the case of I will keep the actual other updates in the full table in the flash itself, so that is what they have tried.

Now, one thing that people often have noticed is that this seems to be a good fit of flash with lock structured file systems because. What is a locked structured file system, you keep on you have the log itself as the primary way of storing information. And if your RAID size is let us say 512 kilobytes or 1 megabyte then what will happen is that you can buffer all the writes in memory as a log and then every. So, often you push out these 1 megabyte segments or 2 megabytes segments or 512 kilobytes segments, and write it full in one shot to the flash in its erase size units. If you do that, then essentially you are exploiting the flash to write for its capabilities, so that is why there is a file system on Linux called YAFFS2.

And this is basically a locked structured file system. And but a locked structure of a file system has a need for garbage collection, because once you have you keep on accumulating writes in a segment file. But the thing is that you might be doing overwrites that means that you might have written a value v 1 and then just moments later you would have got a value v 2 for the same block. And the value v 1 that block because you are not doing it in place is now a garbage that has to be reclaimed, so that means, that you have to do some garbage collection. So, lots such of file systems do have to do garbage collection.

So, that if you do multiple rewrites you only want to keep the most recent write and the other previous writes have to be sometime be reclaimed. So, the garbage collection over write is very much needed for this kind of systems and, but they seems to be a good fit of a locked structure file system with flash. So, these kinds of file systems are also quite common.

(Refer Slide Time: 55:42)



Parallelism

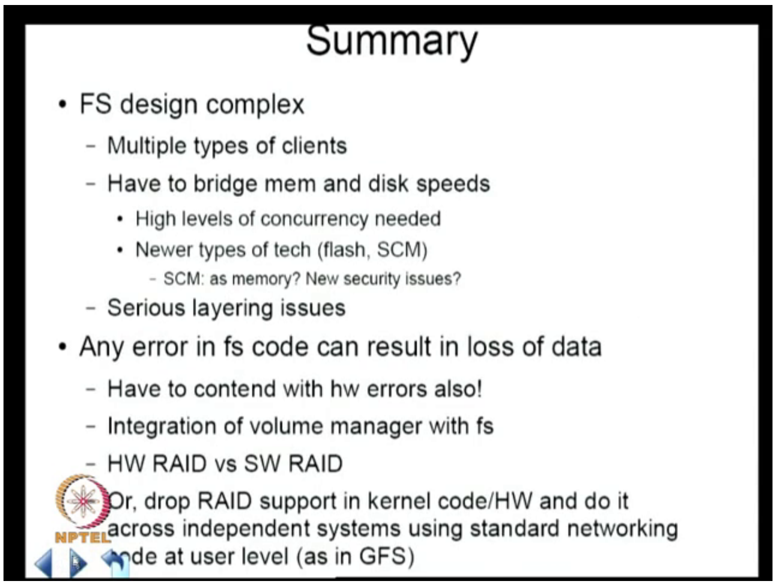
- SSD: 1+ channels.
  - Channel: 1+ flash packages.
  - Flash package: 1+ flash chips
  - Flash chip: 1+ dies
  - 1 die: 1+ flash planes
  - Raw flash can expose most of the parallelism (64x?)
- Std Linux I/O Q'ing not suitable
  - Need to develop new models
  - Diff in R/W sizes, Interrupt overhead: polling?
  - Scheduling R/W/Erase, GC, prefetch
- Higher IOPs: newer bottlenecks in system?

So, I think I will briefly touch upon parallelism and then I will conclude today. Basically, it turns out that there is a lot of parallelism in these kinds of devices. If you look at disk, there is only one single arm that limits the parallelism; whereas, here what happens is because it is semiconductor technology, you have lots of possibilities. If you are aware of that DDR ram for a memory for example, it is comes with multiple channels. So, here also this is multiple channels and so; that means, that each channel can be you can pipeline it to across these channels you can do lots of things on the state and. So, basically this is multiple channels each channel is multiple flash packages each flash package is multiple flash chips each flash chip is multiple dies one dies has got multiple planes, so many levels of parallelism is there.

So, we can get as much as 64 x or even higher levels of parallelism in this kind of systems. And the issue is how do we expose it. And there are some problems here and for example, standard Linux IO Q'ing is not suitable, because it turns out that for a disk the kind of time sitting in the IO queues is not a problem. But with flash you are talking about some tens of microseconds or right and same amount of queuing you are actually kill the performance of the device. So, you need to actually have a different system because in Linux what happens is that they do a lot of what is called accumulating all the writes in a particular way, something called plug in architecture is there, plug or plugging or something they call it. So, those kinds of things are not suitable.

So, you have to develop a new model altogether. And there is a difference in read and write sizes. And there is because you are doing it at much faster rate, you might think of using polling rather than interrupts, you also have to schedule read write and erase and also the garbage collection and prefetch, we have to think about how to do it systemically. There are lot of parallelism but you have to also schedule that. While another interesting thing about this is that now that you have so much high IOPs previously the storage system could have become a bottom neck, but now other party system also equal to bottom necks, so that you have to again reengineer a system, so that it is a reasonable design.

(Refer Slide Time: 57:54)



Summary

- FS design complex
  - Multiple types of clients
  - Have to bridge mem and disk speeds
    - High levels of concurrency needed
    - Newer types of tech (flash, SCM)
      - SCM: as memory? New security issues?
  - Serious layering issues
- Any error in fs code can result in loss of data
  - Have to contend with hw errors also!
  - Integration of volume manager with fs
  - HW RAID vs SW RAID
  - Or, drop RAID support in kernel code/HW and do it across independent systems using standard networking code at user level (as in GFS)

So, I just summarized now basically if you are thinking of what a file system design there are many issues to with taught about.

First of all we have multiple types of clients, right starting from user level programs then management applications about. One important thing we have to do is to bridge memory and disk speeds this will lot of we need to actually some exploit the concurrency levels, newer technologies are showing up and that might even be new security issues because now things are persistent. So, you may want to see how to prevent some information with permanently bigger while it should not be. So, there are (Refer Time: 58:36) issues, and how do you integrate a volume manager of the file system, those kind of issues or like Google file system has done it, drop all these ideas about RAID and then do error coding across by having copies, across multiple systems.

So, these kinds of design are also possible. So, this sort of design place is quite large and something that has to be systematically looked into.

So, I will stop here at this point.