


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

**Storage Reliability, Performance Security**  
**Lecture – 27**  
**Errors, Reliability of data, Data Protection mechanisms in Storage systems**

Welcome to the NPTEL course on storage systems.

(Refer Slide Time: 00:23)

	'12	'15	'18	'20
Max Nodes (M)	0.1	0.1	1	1
Max Concurrency (M)	1	10	100	1000
Memory (DRAM) (PB)	4	10	30	60
MTTA Interrupt (days)	1	1	0.5	0.25
Storage Application Visible interrupt (days)	20	18	16	14



Today we will start discussing a bit about reliability, especially for future systems. I have given some information a evictor from multiple sources, and the for example, it is believed that currently, about we have about 100 1000 nodes in large systems. And this is straightly already straightly old information, because for example, at (Refer Time: 01.02) recently somebody used 1.5 million nodes to come, do some computation. We have rocket which is going up, it makes a lot of noise right.

So, there is a highly non-linear systems that operation of the time, when the gasses are going out, how to reduce a noise and some of the use some 1.1 million M nodes, to do some computation figure out how to reduce the noise that is generated. I can see the issues that are there when you think about storage systems. If there is some minor errors here and there, can the corrupt the whole let us say result that you finally, depend upon

right. How do you know that particular, some particular nodes, they are not doing the wrong things. It could be for multiple reasons, it could be because of the code itself is wrong it is, or it could be because the hardware is generating errors which is not predictable, and basically because I mentioned to you we can have memory errors, disc errors, HBA errors, network errors all those kind of things. So, we talking about 1.5 million of this guys, how do you trust what is coming out that is the big issue.

So, again the concurrency levels also are going to be heavy, this in terms of millions that talking about thousand fold. Like you can see that already this numbers seen a bit old, because I told you already in 2012, they have done 1 million. So, I am not clear. This is just some information I picked up from some presentation. And if we look at the d ram, we are talking about 4 peta bytes now 10 30 60 peta bytes, this is amount of data. Now what is interesting is for us is following here, what is there.

What is the mean time to abort; that is, if you start some computation how long does it take before a system dies; that is you have to restart a system, because since that day, it is one day suppose at now and it is going to become one fourth of a day; that means, it daily works of 6 hours then it will die.

Now you can see why these things are not slightly difficult things to work out; why, if is taking one fourth of a day and dying, which has, the first place in that have to store we want to check point what is happened, as you are going through right. And if you have 60 petabytes, this 60 petabytes have to be written somewhere, where will be stored. There has to be stored most likely in the storage system. Now ask yourself how long will it take to store 60 petabytes ok.

Now, if we assume discs then hopeless tasks; 60 petabytes on disc will be this generally hopeless, but if we have things like; except these, this may be. Basically just can think about it usually you find that a disc will give you about, nowadays we can assume one gigabit per second approximately or 2 gigabit per second. So, if you talk about 60 petabytes, we are talking about 10 to the power of 6 approximately more, 10 to the power of 6. we Are talking about 10 to the power of 6 seconds approximately, or somewhere in that region 10 to the power of 5 to 10 to the power of 7 seconds; that is already a good part of the year, from few months, almost a month or something like that. So, you really have to change a storage system systematically.

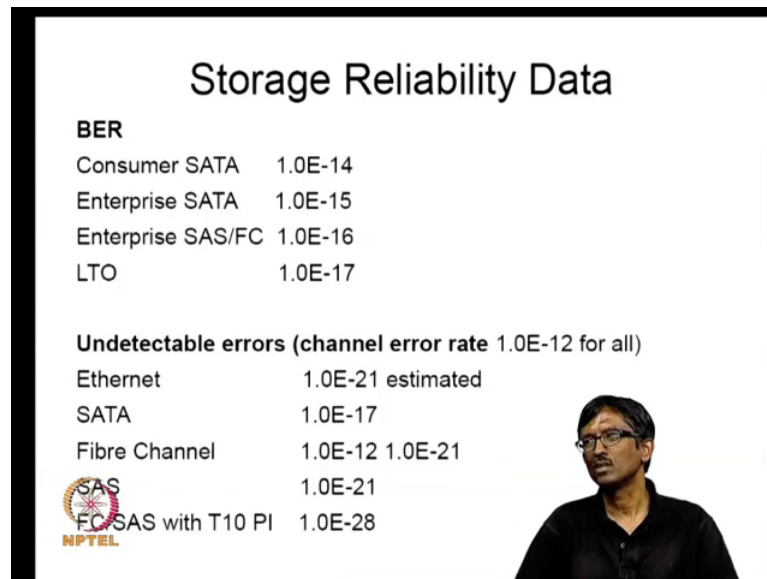
We have to ensure that we need to accessories which will have at least 2 or 3 orders of magnitude higher bandwidth, then you can start thinking about check pointing it. What is, I think all of you and hope you are all familiar check pointing, what is check pointing it is. Basically way in which we do some computation, you know that something bad is going to happen. So, at some regular intervals you say this state so that if that back strap happens, you can restart from there; that is basically set point ok.

Now, our issue now is that check pointing is going to be highly storage intensive. Every time you are going to checkpoint, you have to write it some persistent storage in basically storage system, and you have to start thinking about that. So, this is going to be a big issue. Other issue is, there is a similar thing with respect to storage system alone, and they say that this is luckily 14 days etcetera, storage application this will interrupt. Basically something that is going back in the storage system, and you can detect it. This happening, because of, some other errors typically occur in memory CPU communication wherever, because here so many of them.

We have something like more than one million nodes, and we have a one million level of concurrency. This are all stupendous numbers and. So, we are concerned about a correctness, what we are doing, and question is there is a correctness that the storage can give you, but other systems may not be able to do it at the same way. So, that is the reason why there is a difference between one fourth of the day and 14 days here, because these systems are much more well engineered with respect to error management.

The good example is that when you do anything with the disc for example, we have a, for example, a CPU, we have a host bus adapter, and you have the disc. Every single jump from one to the other there is somebody checking things. Nobody can. You cannot write from CPU directly to disc, the HBA sitting there and checking things. So, there are multiple levels at which any error is caught, but this has to be done more systematically, and that is where the issue big issues.

(Refer Slide Time: 06:51)



BER	
Consumer SATA	1.0E-14
Enterprise SATA	1.0E-15
Enterprise SAS/FC	1.0E-16
LTO	1.0E-17
Undetectable errors (channel error rate 1.0E-12 for all)	
Ethernet	1.0E-21 estimated
SATA	1.0E-17
Fibre Channel	1.0E-12 1.0E-21
SAS	1.0E-21
FC SAS with T10 PI	1.0E-28

Again let us look a bit about some of the reliability data. if you look at what is called bit error rate BER, consumer SATA is E to the power of minus 14, 10 to the power of minus 14, and enterprise SATA is one out of the magnitude better, if you go to what is called scasi serial attached storage and fiber channel. Let us beyond one more level better. There is something called. This is a tape, is a one particular technology of tape L T O. we can see this is really 3 out as the magnitude better than SATA.

It is the reason why this are, let us say these are so good, is equals for the long time, you want you want really save something for the next 10 years 15 years, tape was the only medium and. So, there has been lot of stringent, restrictions on the; let us say the specifications for this tape, because they have the long term storage in some sense. And you notice that even movies and all those things are used typically storage and tapes only. So, this L T O is basically very good quality system, and it is typically guaranteed also, that if we store rate I guarantee that 10 years, tape will not go back, it will be available.

Whereas, if you look at CDS and DVDS, you know that one times the good bad to them one year, because I think I must have misunderstood sometimes back. Typically there is some dice used, from the dice if they degrade then you lose the bit. So, again what is more interesting is that these are errors that you can catch right. There is some error correcting codes, something somewhere sitting there which is actually catching it for

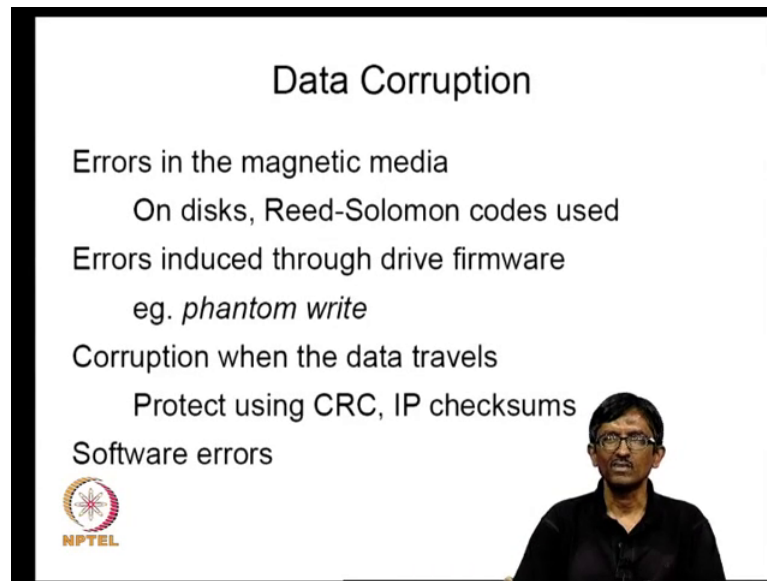
you. I think we already discussed something called latent sector error, where you try read a sector and the system tells you something is bad there, and that is basically this can we detect. There is even worst things called undetectable errors, and you will see that in SATA it is  $10$  to the power of minus  $17$ ; that means, that if you have a system with about 100 petabytes ok.

You are going to have at least one undetectable error. We do not know where it is going to hit, depending on where it hits, it may be calamities or it could be something minor, nothing to worry about right, where it hits. it is like for example, if your injury in the brain or at in the toe, probably the injury on the toe we can easily handle, but for injury on the brain you cannot handle, something that kind. So, you can see that there are various technologies, it turns out Ethernet, it is really gigabits Ethernet, it is very well designed, because they have additional levels of protection in headers, it is the Ethernet headers and.

So, the essentially you can see that channel error rate is this much for everything, because they use the same fiber optic cable essentially, the reason is equivalent of that all over this. This is same medium, but different coding mechanisms have been used that is how, you will find different kinds of, let us say rates of undetectable errors. And I made a mistake which should not be here. So, you will see that, if you take fiber channel and this thing, and use some additional protection, you can go all the way  $10$  to the power of minus  $28$ . We will discuss this briefly today, this part of it. So, this is almost locks on it. We are talking about. We need to have about one trillion petabytes or more, before you start worrying about this problem. So, this is a one stupendous scale.



So, this I think is good enough absolutely good enough, we can really depend on it, but this is going to be expensive, because it talking about fiber channel or SCSI attached storage. So, let us, given this background I think you should be clear about, there are certain types of errors that can be detected by your error coding. These are this; that is a bit error rate and there are some just cannot be trapped caught, because your error coding technique is very good, but still it has certain points in the coding space, which do not result in an error being signaled, they are basically problem.

(Refer Slide Time: 11:19)



**Data Corruption**

- Errors in the magnetic media
  - On disks, Reed-Solomon codes used
- Errors induced through drive firmware
  - eg. *phantom write*
- Corruption when the data travels
  - Protect using CRC, IP checksums
- Software errors

So, what are the kinds of problems, we have essentially we, since we are in the storage business the data corruption is a big problem. So, what are the problems, it could be magnetic media. Typically use something called Reed-Solomon codes. Typical thing is about some 40 bytes or so, will be used to handle 5 12 bytes of sector. It also could be, there could be errors, because of driving for drive firmware.

I think all of you know that firmware is also some pieces of software right; result that it is being done, let say at a level much lower than we normally see, and this is also a some piece of code. So, in this nowhere to write perfect code, and therefore, you can also make mistake here. So, it could be that, you asked it to write a particular place, and somebody made a mistake in the firmware, and some peculiar condition right, where instead of  $n$  it became  $n + 1$  or  $n - 1$  whatever. The standard kind of half by one errors, it could be writing in the wrong place.

For example somebody could be saying write to track number 77, and it the drive firmware makes it 78; that is possible. Also everything is on the data when it travels for one place to another place, there could be problems. So, typically when you send out data right, you are not within, we have to amplify the signal, and you send it out, but there is also lot of noise outside. So, if the noise is, it is not shielded, suppose it is very well shielded, then you get into problems where the bit can flip.


So, then we have to make sure that, when you send it from one place to another place, somebody has to have certain types of, let us say summary information by which we can figure out like what you are getting something is wrong with it, and this is often called CRCs cyclic redundancy checks, or for example, in IP networking, we have something IP checksums.

So, the some, basically in some sense what is doing is, you have some checksum, you make the checksum travel along with the data, on the recipient side you look at the data compute the checksum again, see the checksum is same as what has been sent to you. If they both are same then you know that at least there is some chance that it is actually correct, it is a good chance it is correct, because there could be some methodological case, where the checksum and the transmitted checksum, the computed checksum on the transmitted checksum, pathologically do some interesting things that they are come out to be right, even that is an error, that is also possible, but that happens so rarely that they do not have to worry about that. So, because that is very rare case of course, software errors, that also is possible. So, there are various errors, and we will look at some of these parts of it.

(Refer Slide Time: 14:21)

### Handling errors

- Enterprise servers use RAID to protect against drive failure
- System busses have protective measures such as parity
- Modern file systems such as *btrfs*, *zfs* have checksum on data as well as metadata
- But no standardized way to ensure “end-to-end” integrity till recently

 Data Integrity Field (DIF): for storage stack  
Data Integrity eXtension (DIX): above storage stack

So, what do we how do we handle these things? you use if you are talking about servers, the used rates, I think we discussed it briefly, basically you compute parity across multiple disks, and use that as a way to see if one of the drives fails, then you can recover

the data from the parity for example, same thing with system buses, they are also have parity. And again this is something which has to be done across the whole system. If you are having ECC in the memory, you should also have ECC in the cache also. We have ECC in the cache, then you have to ask you should have ECC on the registers also. If you only have it up to particular point, then it may not have sufficient.

So, it has to be all through. Sometimes people will decide that, as long as it safe in the cache and it may not be in the certain level ECC on the registered contents. I think that is how it is done, I do not think that ECC in the registers, that we have to design very carefully thinking about it. suppose I do not have a, can I get into a problem, and engineers or people design this, thinks carefully and say that the chance of happen the solo, that I do not have to worry about it, that is how it is done. Basically it is some errors depends on the volume also, how did the data is.

So, busses typically have protective measures. For example, you have an ALU bus. You may want to ensure that the ALU bus actually has a parity. In addition if you take advanced file systems. Like example there is one in Linux called B T R F s, or there is one with some had comes to the z f s. They have checksum and data as well as metadata, that the big problem has been that everybody is doing around little thing for error management, but there is no overall scheme, which actually handles all of it; that is called end to end integrity. From one side to another side for example, pick it up some from the storage side, it is travelling from the storage to HBA to network to various things right, and it ensure that the whole path is protected.

Not just only what here and there; that is basically what is called end to end integrity. Now there are some standard circum of, that something called data integrity field DIF for the storage stack. There is extension, this is not actual standard of parts I know, this is for above the stack. If you both DIF and DIX, then you are mostly covered, but this is has been standardized this part, there is a, what you call standard body which has actually go on head and standardize this one, this I is still in the, let us say people are using it, but it is not part of standards body as possible ok.



(Refer Slide Time: 17:16)

### Standards-based Protection

*End-to-End Data Protection std*

- approved by T10 Technical Committee of INCITS/ANSI

An extension to SCSI block device protocol known as *Data Integrity Field (DIF)*

Standard Data Block ( 512 bytes )                      DIF ( 8 bytes )

*DIF: an 8 byte header for the standard 512 byte sector*

Reference Tag ( 4 bytes)              Meta tag ( 2 bytes )              Guard ( 2 bytes)

*Guard tag: a 16-bit CRC of the sector data.*

*Meta tag (Application tag): a 16-bit value that can be used by OS*

*Reference tag: a 32-bit number used to ensure individual sectors written into right physical sector (depending upon type the drive supports)*

How does this work. So, there is a end to end protection standard; that is, there is a technical committee, some American National Standard Institution has got some committee, and the T 10 is basically the one which handle storage, and what they have done is, for if they have a they attach for every data block 5 12 bytes, they attach a data integrity field of 8 bytes. Just like your ECC also right, for every 32 bits you had 7 bits of ECC right, something like that. And if you look at the DIF itself, it is got, because it is 8 bytes right. So, basically 512 bytes plus 520 bytes.

So, the new sector is 520 bytes, and of this 8 bytes, there is a reference tag meta tag and guard. So, basically, the guard is a basically the one 16 bits CRC of the sector data. So, basically if you read that data, and then you compute a C R C, the data has some CRC value and what you read CRC is different, they know something is wrong, you read sector wrong. Now that is the part of the there is 2 bytes. There is one meta tag which is left unpopulated.

So, that any OS can decide what to do with it, it is a extension. So, you can left for future use, or any clever guy who knows how to use it well. Then there is a reference tag. Now basically is that is a 32 bit number is to ensure individual sectors written into right physical sector. Basically what you do is a following if I am trying to write a sector.

Suppose I am suppose right to sector 47. So, you put 47 in the reference tag and write it. Fact that I said right to 47 does not mean that somebody down below, they actually look a 40 set of 48 right, but luckily for me it goes to 40 and clobbers something on 48 fine, it is fine, but luckily for me the database still has the 47 as the reference tag, when somebody reads it, that guy is reading to, he is going to sector 48 he reads it, but the reference tag is 47 that I know the, something is wrong, it is just clear.

Basically the idea is the following, you always try to describe where the data should be, that errors supposed go to sector 47, but by some horrible mistakes somewhere down the, below the stack where I am upgrading, it actually went to sector 48. so, but luckily for me it is one sector 48, but the reference tag is what I said it, it went it has, because I am sending everything on one shot the data plus this part, and that one stress 47. Since it is got 47 then I read next time, I went to the sector 48, but the data says, but it actually 47 then I know that something is wrong in this particular sector. So, that is how we can catch some errors, and these are considered to be quiet, not exactly rare people have discovered, there are 40 devices in the system, which has a tendency to do some of these things at various points in time ok.


So, again these kind of errors you might think are steady strange, but you will start seeing in, when a devices which you commonly use for example, some of you might have noticed that if you go to an high end cameras right, they also firmware, and the firmware nowadays. You may download firmware into our camera, because it has a bugs in it, because it is being something wrong, it being some color equalization, whatever is doing various things.

They are all algorithms, and it could be that some of it is wrong, minor in wrong in mostly correct, but there are minor canvases, where it is gets wrong. So, again the firmware has to be downloaded to fix the problem, even the lenses for example, turn out to be managed by certain firmware, and you might get into some difficulties, if the firmware is wrong and. So, basically what we are talking about is, there is a lot of firmware out of the place, and the firmware can do mistakes, because some computation is going on, and it is not going at right place.

(Refer Slide Time: 21:47)

## DIF in operation

- HBA
  - calculates a checksum value for the data block on writes
  - stores it in the DIF
- Storage device
  - confirms checksum on receipt
  - stores the data plus checksum
- On a read, checksum
  - checked by the storage device
  - also by the receiving HBA



So, basically what is the, how do you use this DIF, I mentioned that host bus adapters. This is basically a proxy for the CPU right. it calculated checksum value for the data block on writes stores it in the D I F. So, basically the data is there, and store it in D I F, and this whole thing is sent out, and then the storage receives, gets it. It has got the DIF that has computed by HBA, and is got the data. Now it sees if the data, that it is got, it will compute the checksum again on that one, one in the data see if it is, what the DIF, it got from the HBA, if they both match, mostly its, and then it stores the data plus checksum. So, one a read for example, because it has stored checksum also, it will checksum we get the data onto out of it, and then you check the receiving HBA also, checks it. Now we are talking about DIF, which is basically only to the storage stack. So, we are not talking about CPU side; that is the reason why it is only between HBA, and the device; that is the only between that part. So, these are how it is works.

(Refer Slide Time: 23:06)


## DIF enough?

T10 standardizes 520 bytes/sector for SCSI drives with DIF format

- prevents content corruption & misplacement errors
- protects path between HBA & storage Device

But does not provide integrity for software stack (such as filesystem), where errors like bad buffer pointers, misdirected writes occur

Hence need also to provide integrity in filesystem over DIF.




So, the real issue is the DIF enough, I think I already mentioned that above the storage stack, we need to be the something also. So, that is why it is not important. So, what basically the DIF does is, it prevents content corruption, and misplacement errors. Already mentioned misplacement instead of going to 47 sector, it is going to 48, and it protects path between HBA and storage devices, but does not provide integrity for software stack; such as file system.

Now file systems itself is the fairly complex system. it is really can be extremely complicated, because it is typical highly concurrent, and it is also not only talk into applications, it is also talking to the virtual memory subsystem, and if it is and it also, if file system also is quiet complex (Refer Time: 24:00), because it has to bridge the gap between memory speeds and disk speeds. So, it has to be lot of caching, and there is cancels in the outcome. So, because of that it turns out a file system also can there, could be some minor, some very certain bugs. And for example, if you have that buffer pointers misdirected writes can also occur. So, we need to provide in addition to DIFs, some additional stuffs that DIFs is basically that attempt, but I am, as I told you that has been standardized.

(Refer Slide Time: 24:29)

### ZFS error correction

- ZFS uses end-to-end checksums
- Checksum not stored with the data block, but in the pointer to the block
- All checksums done in server memory, so catches
  - Phantom writes, where the write is "null"
  - Misdirected reads or writes, where disk accesses the wrong block
  - DMA parity errors between a storage array and server memory or from the driver, since checksum validates data inside array
  - Driver errors, where data stored in the wrong kernel buffer
  - Accidental overwrites, such as swapping to a live file system
- Cost: approx 1 GHz of CPU time to process 500 MB/sec I/O
  - 1 cycle of CPU for 4 bits of I/O!
- Not a standard!



Let us just take one example of various collection above this dif layer. So, z f s uses what is called end to end checksum, again it is a file system. So, it has no control about what is above it. So, it can only do end to end or from the point of view of starting from where it enters. The data enters into it system till it goes to disk and back, till up, till point it goes back up again, that is all it can handle whereas if you go for DIFs, it can go all the way to the application also.

Now, checksum not stored with the data block, but in the pointer to the block; that is one minor difference than what I said before. Again all checksums are done in server memory. So, it can catch various kinds of errors. It can catch phantom writes. Sometimes as I mentioned the write actually present, actually alternative, what does it happen, because you are trying write, and suddenly what happened was there, is big dust particle was sitting there. So, your disk head, because of Bernouli effect, it actually goes over the dust particle. And essentially the magnetic field that is used, that is so weak, that actually, nothing happens below, it becomes phantom write.

So, we already discussed misdirected reads or writes, where disk accesses the wrong block, because this already. There is also parity error is between storage array and server memory, or storage array or driver. There are those kind errors also. Sometimes you have the device driver errors, you are writing device drivers. Now you know how we can make mistakes right. So, where data can be stored in a wrong kernel buffer itself. Of

course, usually most of these systems we use, have been tested extensively. Especially file systems are tested extremely vigorously, because everything depends on a file system being correct; otherwise nothing, no work is possible ok.

But in spite of that there are bugs, and once in a while some corner case, which nobody ever expected shows up, and who knows the data might be stored in the wrong kernel buffer itself. Especially concurrency can get into these problems. There have been very unusual errors reported for example, I remember in 96 years. So, this Microsoft had a file system on the servers, and if you access web page rapidly enough, it will corrupt the data completely.

Only if you access the web page very quickly; that is you are asking for information at a very high rate, then that case it turned out, that it will corrupt the data; that is fairly good explanation why it could happen that, it because of the corner case, and now turns out that this error correction is not exactly cheap. For example, you need to process 500 megabytes per second can a throughput. So, much data is streaming through right. You need certain amount of CPU processing power, and especially if you look at it. It is something like one cycle of CPU right for 4 bits IO, because somebody has to touch something right.

Because basically you are doing checksum kind of things he can do caching, he can do all kinds of things, work. In spite of that there has to be some cost associated with that, and essentially one cycle of CPU for 4 bits of IO, and it is a costlier thing, because in one cycle you could have done add. You can do 32 or the 64 bit add. So, what you are doing is, instead of doing one at a 64 bits, you are touching the IO bits 4 bits at a time, equivalent of that. And we essentially extended the amount of CPU processing power to do it; that is the reason why most of you, when you are doing some of those things often times, we need hardware acceleration, and the hardware acceleration is understood in the context of encryption and decryption, but within this kind of stuff I think it is still not much there ok.

Everything about ZFS error correction; that is, it is not based on any standards. It is done by some Microsystems there, in their file system and. So, all this stuff what you are talking about, may not be available on the systems. For example, it EXT3 did not have it, EXT4 has some basic checksum going right now. So, EXT4 is going to be

somewhat more reliable than ext 3, but it may not do some of these things. For example, EXT 4. I do not think it handles misdirected reads or writes, if it uses a dif or dif for example, then possibly Linux also will do it, that if you want to make Linux EXT 4 handle misdirected reads and writes you need to have DIF DIF, but that also means that you need support in the various layers in the system.

For example, if you have buffer cache, even the buffer cache now should actually start having this checksums, and what is more difficult is that any time anybody does any mapping. For example, you have something called memory maps right, m maps right. Now m maps basically are mapped to memory and anybody can be changing any bytes; that means, that any bytes somebody, any can, any bytes takes place, any change in the bytes takes place, somebody has to recomputed the checksum again, as long as that is not possible, then the whole system is not trustworthy enough; that means, that the only way you can guarantee this, is if you use what is called direct I O.

What is direct I O. Those things that goes through virtual memory subsystem, we are having some I O which does not go through the caching mechanism of the virtual memory system, then it turns out the. It turns out that you cannot update those areas of memory through other applications or whatever right, other threads or whatever activity they cannot touch it, because you are guaranteed that you are going only through only one interface which can control, and therefore, you can do all these things ok.

So, you can see that even if you are interested, the current models of memory map (Refer Time:30:56) will not allow you to do a good job of all those things, this is how. So, you have to think carefully before you decide to do this. We have to fix all your interfaces before you can start putting, trusting all these things. It will catch some errors, but the qualifiers it will not be caching and again it depends. Suppose you have, you talked about 30 60 petabytes in memory right.

Now you look at the kind of ECC that error correcting code; that it is got. If it has got let us say much weaker error correcting code capability than what the disk is providing. Then probably all this stuff what we talking about is not is. So, some might has to be equal approximately about the equal kind of capability. Your disk correction capability, error correction capability and the memory error, error correction capability should be somehow matched. They know it make sense out, take all this trouble. As I mentioned

you have to change, buffer change, you have changed. So, many layers before I can get it what. Now, in top of this DIX as I mentioned already we also need something.

(Refer Slide Time: 32:00)


*Data Integrity Extension (DIX)*

DIX defines a set of interfaces that host adapters must provide in order to exchange integrity metadata with the host operating system

- application calculates checksum and passes it to the HBA, to be appended to the 512 byte data block.
- provides a full end-to-end data integrity check

To provide End-to-End protection

- controller has to be DIX capable
- storage device DIF capable



So, what application also can get some additional guarantees? So, DIX basically, as far as it was defined some people at oracle. It defines a set of interfaces that host adapters must provide in order to exchange integrity metadata with host operating system. So, basically the application calculation checksum, and passes it to HBA, and this is appended to the 512 byte data block. So, in the sense now application and HBA, they now understand each other. Just like previously HBA and the storage device to understand.


So, now, we have one application all the way to the storage device and back. Again this is as I mentioned before, if there are caching is going on and there are threats which do not understand part on the updated things, and do not update a checksum you are lost nothing is useful. So, you have to control that part also. So, if you taken those kind of, if you handle those kind of issues, essentially provides a full end to end data integrity check. So, basically to provide this kind of protection, we have to have, the control has to be DIX capable in the storage device also DIF capable, both the same.



(Refer Slide Time: 33:21)

### General Method for Checksumming Above HBA

- use SCSI DIF/DIX to ensure integrity of data in filesystem using the *application tag*
- use algebraic signatures to ensure integrity of data to ensure integrity of both data and meta-data
- useful property: if can calculate algebraic signature of the whole block given algebraic signatures of sub-blocks



So, basically if you want to handle things with respect to integrity. There is some general mechanism, and as I mentioned there are certain things which are done at the storage level itself, this guard for example, and the reference tag. These two things are typical use at the storage level itself. This is for misdirected writes; this is for just errors, to catch errors CRC type. This meta data, meta tag is a left to the O S, or any layer above the storage, and the question is what can we do with this. So, that is what we are talking about here.

So, we can do. There is a method called algebraic signatures that can be used, to ensure integrity of data. Some small mistake here to ensure, I think I have repeated twice is algebraic signatures to integrity of both data and meta data. What is the usual property that is needed? If there is a big block, let us say it has got so many sub blocks, the property I am interested in is. If you give me the signatures of the sub blocks, I can easily compute the signatures of the whole block, and in the sense it is compositional. If you have that property, then I can do what I did before also. See remember what is happening here, if I look at this right, what is happening when I do this guard bit. Basically it is saying if I have so many bits, as sum them right I do a C R C. then the CRC that I get is that basically based on each of the bits of the data, it is basically depend on the data, it is compositional, and because of that when I send it across other party also can compute using the same algorithm, is the same CRC check. So, that is the reason why this whole thing work right. So, the question is that can we do sub way similar at this level also.

So, basic idea is that, come up with some notion of signature, which captures something about the data; like the C R C. So, that it is compositional, I give you sub parts, I should be able to compose the signature of the higher level entity. I take the higher level entities multiple of those guys across. Again I can compute the signature of, even higher level entity. I have basically some kind of a recursive mechanism, if have the property everything works out. It turns out you need some special theory for this. We are not going too much, I will just mention few things about it and.

(Refer Slide Time: 36:32)

### Galois Field

- Galois field is a field with finite number of elements represented  $GF(n)$
- Number of elements in  $GF$  is  $p^n$ , where  $p$  is prime
  - Here, only  $GF(2^n)$ , specifically  $GF(2^{16})$ .
- Elements in form of polynomials, binary strings etc
  - $x^4 + x^2 + 1 \Leftrightarrow 10101$
- Addition, Subtraction equiv to XOR, hence commutative
  - Similar defs for Multiplication, Division
- An element  $\alpha$  in the field is irreducible if it cannot be expressed as product of non trivial factors
- $\text{ord}(\alpha)$  of a non-zero element  $\alpha$  is the smallest non zero exponent  $i$ , such that  $\alpha^i = 1$
- An element  $\alpha$  is a *primitive element* if  $\text{ord}(\alpha) = s - 1$ , where  $s$  is the size of  $GF$

This is basically something called the field I think you heard of rings, groups and fields right. So, there is something called Galois field which has got this, which has got this property. So, I am not going to great detail of about this thing, I am just going to mention roughly what it is all, how it is done. So, usually in this Galois field, we have number of elements is  $p$  to the power of  $n$ , where  $p$  is a prime,  $n$  is a natural number. It turns out that in this case, because you use binary system, and  $p$  is prime.

Therefore, 2 also is a prime right, you know all that 2 is prime therefore, 2 to the power of  $n$ . So, typically we use  $GF(2)$  to the power of 16. And if you look at it, the various elements in this system are basically binary strings, which can also be seen as polynomials. And it turns out that addition subtraction equivalent xor; Therefore, they are commutative. So, I can say  $a + b$  is same as  $b + a$ , it does not matter which order I do it. So, I can do it in any order right. And there are similar things that you can do for

multiplication and the division. It turns out that to define this algebraic signature, I need something called reducible element. basically we are talking about in a field it turns out that, you can have multiple elements in this field, and some other fields are can be written as products of above elements. Just like you take our regular field integers right. We can write 6 as product of 2 into 3 right. Whereas, is irreducible is similar to our prime numbers, it is not a product of non trivial factors. So, it turns out that if you can get one of those irreducible element. Then if you have that irreducible element you can, it turns out that, if we keep multiplying by itself we finally, can get 2 one in this Galois field. This is not, may not be really valid for our integer systems, because we have, let us say we have unbounded integers, then it will not happen in our case; that in a field for example, if you have only 2 to the power of 16 elements in this Galois field.

Therefore, it turns out that once you start multiplying by itself, and it will come back to, it became part of itself in part of 2 to the power of 16. It keep multiplying what finally, come back. So, there is some I for which it becomes alpha to the power of I becomes 1. This is what is called order it turns out, if we choose if the there is something called a primitive element for which order is basically s minus 1 for example, if 2 to the power of 16 I want an order of 2 to the power of 16 minus 1, use that as a primitive element. If we have a primitive element of that kind, it will always be there for this kind of this systems. Then it turns out we can define a signature like this ok.

(Refer Slide Time: 39:39)

## Algebraic Signatures

An algebraic signature of string of symbols  $x_1, x_2, \dots, x_N$  is defined by:

$$\text{sig}_\alpha(x_1, x_2, x_3 \dots x_N) = \sum_{v=1..N} x_v \cdot \alpha^v$$

Important properties of algebraic signatures

- If block length  $l < \text{ord}(\alpha)$ , where  $\alpha$  is a primitive element and every possible block content is equally likely, then the signatures  $\text{sig}_\alpha$  of two different sectors collide with a probability of  $2^{-f}$ , where  $f$  is the length of each symbol
- $\text{sig}_\alpha(x_1 + x_2) = \text{sig}_\alpha(x_1) + \text{sig}_\alpha(x_2)$


Can use algebraic signatures on sub-blocks and blocks in a FS to ensure integrity

- signatures summarizes protection information over longer sequences of information units such as blocks
- eg. for a block  $B_n$  the application tag split into 2 parts

IntegritySubBlk<sub>n</sub> stores the integrity metadata of block  $B_n$

- Stores integrity data of each subblock

IntegrityBlk<sub>n+1</sub> holds the integrity metadata of block  $B_{n+1}$  (summary)



So, basically you can define a signature as a following. What is signature that is defined by. Suppose I have symbols  $x_1 \times x_2 \times \dots \times x_n$ . what will I do, if you want to get the signature for this particular alpha, what I do is, I do a sigma of each of this since  $x_1$  into alpha to the power of 1 plus  $x_2$  into alpha to the power of 2, all the way to  $x_n$  alpha to the power of n.

Now this is basically a signature. it turns out signatures have an very interesting property of this variety, signature of alpha  $x_1$  plus  $x_2$  is signature of alpha  $x_1$  plus signature alpha  $x_2$ . What it means that if I am able to compute the signature of, let us say  $x_1$  and  $x_2$ , they are two blocks I have compute a signature on block 1. I compute the signature on block 2, and then I do XOR, because addition what is, what did I say addition was, is basically equal to XOR in this game. So, when you do that, you can essentially. So, I get the sub signatures, I can do XOR, and I get the signature of the upper level, like higher level  $x_n$ , the  $x_1$  is there two blocks, I contented two examples. So, that is one, one interesting property of this particular system ok.

Now, other thing that is there, because you have a limited number right. This 2 to the power of 16, elements are 2 to the power of 16; that means that you could be dealing with very last things. You could be dealings with 4 0 9 6 byte blocks. What is the space of possibilities there 4 0 9 6 bytes is 4 0 9 6 into 8; how much is that.

It is about 32000 right. So, what is the number of possibilities in how many different blocks are there, how many different blocks are there for type what, say with data it will be 2 to the power of 32000 approximately; that is a number of possible ways you can fill up in 4 k block. So, it is a tremendously huge number. That finally, what we are doing, we are only handling it with 2 to the power 16 right, we only have 2 to the power 16 codes possible; that is all, that is there, because maximum we have, we are talking in our, signature is basically 2 to the power of 16 what say an element in 2 to the power 16 G F 2 to the power 16.

So, when you go from 2 to the power of 32000, and you map it to something which is only 2 to the power of 16 right; that means, that there is a chance that you might get something backside it that is, what we are talking about here. So, then the signatures of two different sectors collide with a probability 2 to the power of minus F, where F is the length of the each symbol. So, basically it is 2 to the power of minus 16. So, basically

you are talking about, the fact that since you have a much limited file signature, only 16 bits. The chances there were two things will be shared access by just simple accident (Refer Time: 43:09) taken is 2 to the power of minus 16.

Of course, if you fill the 2 to the power of minus 16 is too small, then you can go to 2 to the power of the 32 bits you get that higher number. So, what is the, why is that this. Basic thing is that you, you already have various methods of error correction etcetera, system various participants. I think as I mentioned all this, guys are doing various things right. Your RAID is taking care of some problems, system busses are taking some problems, various guys are doing various things. Now you are putting on top of this particular thing. Therefore, even if you think that 2 to the power of minus 16, is not good enough, it can create problems, but once we have all these things, you are straightly in good territory that is a reason why I notice this one.

This is the T 10, what you talking about protection, your 10 to the power of minus 21, without this thing, without this end to end correction. Once you put the 10 it is gone to 10 to the power of minus 28. You have got now seven additional levels or orders of magnitude that are, let us say protection with this kind of model. So that can we mostly good enough for many proposals. So, again in engineering you do not go for, other is the very best possible thing, because that usually crossing too much, you do not have the resources should do it. So, what you do is increment and increases it so that you are in safe zone essentially that is basically what you are doing.

So, what do we do out here. So, the basic idea would be that you have signatures, and they have this property which is interesting property. Now it is same property like what we are doing checksum also, and this is excitedly also assumed in the case of IP checksum for example, in networking also, basically we have packet. We keep on doing the I P. We keep on every 16 bits, we do a checksum and then you compute the checksum of the next 16 bits, and because of the additive property you are in good chief ok.

So, basically what we are doing is, we will use this algebraic signatures on sub blocks and blocks, because what is a block in a storage system typically; that is it is a 4 kilobytes, but the hardware itself the disk itself, it has to be with 512 byte sectors. So, in the sense you are one block equivalent to 8 sectors. And if you notice our design or the DIF why, it is doing at the level of 512 bytes right, that basically what we are doing here;

5 12 bytes. So, thing is we have to figure out what is the corresponding signature for the block, and that is one part of it. The other thing is that, we also want to see, whether if you have, even longer things for example, file is there. A file is, it can be one byte, it can be 100 bytes, it can be 10 gigabytes, one single file right; that means, that you have to find the way in which you can take each of these block by block, composed the signatures one by one, till you get the signature of the whole thing ok.

Now, of course, if you have a something like a 10 petabyte file, and there is a 15 another 10 petabyte file. The number of times you do it is so high, possibly that it may not be worthwhile, it may be that, it may be actually equal by some with the high probability that is also possible. Unlikely, but probably that is a good chance, but if you are talking not on next scale, what you can do is. You take the algebraic signature from sub blocks and blocks, and you compose in such a ways so that you can essentially again check it just like previously we did with C R C.

So, what we need is, we need as new signatures that summarizes the production information, over longer in sequences of units; such as blocks. for example, suppose I have a block  $B_n$ . So, now, this block is itself composed of multiple sectors. Each sector you get 2 bytes right; that is basically what our designs is, each sector is getting 2 bytes. So, you have so many sectors right for example, in the case of 4 kilobyte block, how many sectors are there. 8 sectors are there.

So, you take 8 sectors into 2, you have a 16 bytes of information available. So, what we can do with the 16 bytes. I am going to say that, I am going to take the 16 bytes, and put some. Certain things will store the, let say 8 sectors right. Let say that output, some of the sub blocks right I keep those things in the 16 bytes, but I also keep the integrity block of the next guy. So, what is the idea. The idea is that when I read some stuff, I read 2 blocks and then I will check the correctness of not only with respect to what I have here, but also with the next one. In the sense what I am doing is I am creating a system by which, every piece of data is not only help to a standard with respect to consistently itself, but also with respect to some information nearby.

So, it is something, somehow there is some mistake somewhere, you can catch it. So, those kind of things can also be done. I just giving you some high level idea, that there are lots of things I actually worked out, and as I mentioned to you. This kind of stuff will

require amazing amount of changes in your high system stack. I think you have some experience of looking at the file system stack. For example, this will essentially change your block layer right, to change your buffer cache, it will change your file system, it will change your i nodes structures it will change all those things ok.


Suppose somebody wants to design something of that kind, they can go and change every one of these things. So, it is not something which most it will want to think of think about; that is why you will notice that these kind of designs are very good, are very important, but it is not being at, see the screen, design into systems, that were there right now.

So, people are doing what is appropriate, that is they will do it only at certain extent which catches most of the problems, and then they will leave it less, leave it to the, let us say. So, that is why you will see this DIF DIX, DIF has been standardized, DIX has not been standardized, because DIX means that now we have to worry about all these things, all the structures all the way from the file system, all the way off, because it will notice it right. Where do we stopped here. We go and left to h b s right ok. We are only refused is one of the HBA and storage device. Whereas, if you want to go to DIX you have start thinking about everything from file system, and file system is again a very complex thing database, all these guys are database things everything has to change their. So, that is the reason why it is not common to see DIX etcetera are going on. DIF I think is beginning to become common standard.

(Refer Slide Time: 50:57)

## Conclusions

- Error Management will become central in future large designs
- Need support across various components of the storage and appl stack



So, basically error management is going to become extremely important in future large designs. I think I already mentioned the biggest problem that people see, is when you have so many threats so many what is say, concurrent activities going on. If you want to contain the error you want to actually find a way of validating the results are being generated, and especially when you are have something like one million nodes they will be traversing networks and various things buffers and this and that right. And every time they going through somebody has to be checking the correctness of all these things, and that is more makes it very difficult. I already mentioned, you need amazing amounts of support across various components of the storage and application stack. I think file system is complicated enough, once you go into database etcetera, there will be output for example, in the case of mentioned file system, but also you can have things like volume manager right.

There is typically will see that there is something called volume manager which takes all this disks and makes it pretend as you if it is a big one single disk, there also it gets changed, then on top of it file system has to change, then lib c has to change, then the application has to change. Everyone one of these guys should go take all this integrate information back and forth, and they have to update these things and then there is a chance that you will actually have reasonable guarantees, but that turns out to be extremely difficult to do in practice; that is why you notice that, currently we have only the disk that is being done because it is in a selected area, where the errors are slightly on



a larger scale, but if you talk to people in large scale system like C R M. they will actually be quiet serious about using this kind of things, that unfortunately there is nobody to provide this kind of solutions even as far as I know. There is the standard way to do it is by engineering the components better and better ok.

So, for example, disk bit error rate for example, sometime in the past was 10 to the power of minus 13. Now they gone to 10 to the power of minus 14, you go to 10 to the power of minus 15, and then is cyber channel disk are 10 to the power of minus 16. So, now, that is typically in which you handling it, basically you make each of subcomponents really very good. So, you do not think about it, but when you are talking about very large system, you have to think about it.

There is no way to, there is no escape. And that is the same reason why, if you look at Google kind of systems, they have decided to go, what is called spread out their systems, rather than having extremely powerful machines in one single place. Basically what happens is that once you do it, you tend to be able to scale the systems, again you can error management also has to scale appropriately there, and again Google kind of file systems they actually have a lot of checksums all over the place. So, that even if there is any error that you get from somewhere right, we are able to recover; that is why we have three copies right. So, you can see all kinds of things there; E X T 3 will be doing some error checking, the hardware will be doing it, the O S will be doing it, the network will be doing it, and G F s will also doing it on top of it. So, all this guys are working together, there is no way, there is some chance that you get reasonably good, let us say system that is operational; otherwise you will get the situation where every 6 hours it will not work, which is same as what is to be there in 1940s.

If you look at 1940s that extremely, the components were extremely error probe, extremely error probe. So, in the 1940s when the computers are designed for the first time, your typical let us say mean time to failure was few hours; that is if you want to get some computation we started, and hopefully if your computation is within 3 4 hours you get the results, otherwise it will die, and then everybody have to check the results. We have to check whether this result actually is meaningful.

So, there was all this complications those days. Finally, we have come to situation where things, components are reliable enough, that you can mostly forget about it like a laptops,

but if you go to high scale, you will still have to think about it. Ok I think I will stop here.