

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

Storage Interfaces and Device Drivers

Lecture - 20


Device Drivers _ Part 4: Network Block Device driver design, Design of FAT, TFAT, F2FS, LFS filesystems and interaction with higher-level filesystems like FTL

So, welcome to the NPTEL course on storage systems again. So, in the previous class we looked at some aspect leading to block devices, I will just mention one more type of block device, it is called the network block device ok.

(Refer Slide Time: 00:39)

Other Block Devices

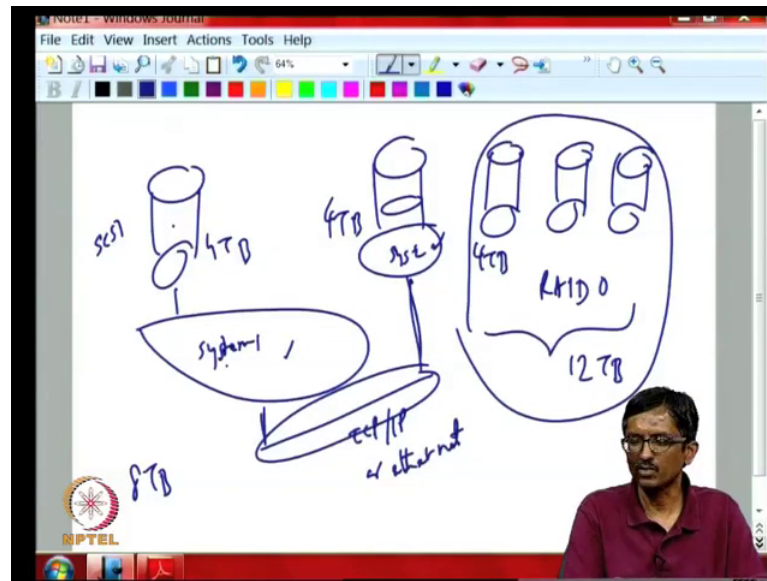
- Network Block Devices
 - Instead of bus/SAN protocols, use IP networks
 - Higher rates of failure (non-availability or data loss)
 - Can use redundancy in software
 - Slower but more flexible (eg. use heterogenous devices)
 - Allows for much higher scaling
 - Bus/SAN networks cannot span continents
 - Parallelism an imp issue



Now, instead of having your devices on a bus or what is called a storage san storage area network protocols right you can use standard TCP IP; that means, that your devices are not connected through electrical or some special storage protocols they are connected through regular TCP IP the internet protocol or Ethernet networks it could be Ethernet networks it could be standard TCP IP whatever right, then the basic idea is that you want to see if it is possible to collect various devices and make it look like a very big device.

So, similar to what we talked about a volume manager last time except, now it is going to be through network.

(Refer Slide Time: 01:38)



So, what is the previous situation we had various disks disk one disk 2 etcetera and if you use raid 0 basically you want to provide. If this is 4 terabytes, 4 terabytes, 4 terabytes around provided twelve terabyte disk. If this is 4 terabyte each, right and that we said is the job of volume manager and typically in most of these cases they are all sitting on a bus and its all within a system you are not using any you are using some electrical protocols typically to connect each of these guys to talk to each of these guys to pick up things put it write, we then write.

Suppose, I want to do it across the network basically it still has the same thing, but we now we go through some system and there is another disk that is going to another system another sub system these guys are talking TCP IP or Ethernet and basically idea is to still give this is 4 terabytes this is 4 terabytes now we can provide a 8 terabyte disk raid one raid 0, but it is now through TCP IP; that means, you need to 2 systems here this systems talks to this disk system one system 2 talks to this disk and so, these 2 systems are also in picture.

Previously, it was being done in the controller itself because of its done in the controller it could be possibly more efficient because it is being done at a very lower level. Here the system has to come into picture operating system is going into the picture right all kinds of things are going to be just that delays and latencies can be substantially higher, but a good thing about this is that this systems can be desperate this could be raid hat system

this could be a Solaris system, right. So, a lot of things can change this can be one type of disk this can be another type of disk for example, this can be skuzzy disk fiber channel disk this can be a ide drive or whatever.

This could be solid state disk or whatever right. They do not have to be same because finally, these systems here are basically making them. So, that they all look same to TCP IP because finally, only bytes are coming out right, it does not matter which form what it is stored in what form. So, the good thing about this is that you have more flexibility, but it can be slightly slower and the other thing is it surely typically since it is a electrical protocols here or san protocols you are limited to smaller distances whereas, here they can be on internet scale it can be one thing can be sitting in japan another can be sitting in some place in India or USA or whatever; you do not care. So, this kind of things are also possible ok

So, you can use IP networks or Ethernet networks. Nowadays Ethernet has become very popular because now you have what is called metro Ethernet; that means, it is Ethernet across city. So, you take a city about thirty miles or fifty kilometers or. So, whatever right in radius you can use Ethernet as long it is fiber and you can design systems based only with Ethernet protocol not even TCP IP that is you are working strictly with Ethernet kind of protocols typically you have TCP IP layered onto Ethernet; that means, you have a higher level layers 1 3 1 4 1 5 etcetera on top of 1 2, they are linked layer right. Now you can have designs also which only use 1 2 and this metro Ethernet and those kind of things have come recently in the last few many years which has made this completely practical ok.

So, basically now you can get gigabyte Ethernet across the city once a gigabyte across the city you can connect with them up and now there are various vendors who will provide you with example you can say; I want 300 megabytes per second between 4-5 nodes in the city and if you have those things, then we can build the network block device we can, but the problem with these kind of devices is that it is as higher rates of failure now the failure can be for 2 types the data is intact, but your connection went off all the data itself goes off there are 2 possibilities that networks typically you is still there, but the connection is gone you assist to as a non-availability situation ok.

So, the typically networks are much more fragile than if you use bus based systems or san based systems on a pc for example, once you put your disks typically unless you do a lot of shaking the pc or there is a lot of dust or such or you pour water things of that kind usually nothing happens to it, it continuously runs without you are doing anything with it right. So, the electrical connections are made properly and once for all and usually the once properly usually, they will not give any trouble right the disk might fail, but the connection may not give you trouble typically, but. So, in the case of the networks all of us know the networks that are bit more chancy right and even more chancy if we use wireless protocols they have amazing amounts of irresponsible ok.

So, with networks your higher rates of failure so; that means, we have to really be do something more serious about it you cannot take it easy, usually you should not take it easy, even on a bus based system, but here you need to be a little more careful. So, you need to have some redundancy in software, I think if you look at it as I mentioned, it has to go through some systems, right or network based block devices now you should ensure that those systems are doing some redundancy in software if they do it then this would be improved. So, already mentioned it is these things are slower, but probably more flexible the good thing about it is address for much higher scale we can now have worldwide systems because IP can go everywhere san protocols we will discuss it later usually are limited to about wherever fiber can go and there is there is some limitations on the round trip type ok.

So, because of the round trip times for examples thirty miles long trip time is once speed of light if you take into account you are just about able to manage it. It takes up if you calculate it you will find that going beyond thirty miles or forty miles creates problems for protocols like san protocols. So, you cannot really use some of these things, but IP it has a; it has been designed for the slight different perspective; it is able to handle larger amounts of latency and lot of distances. So, that is why you will find that if you use IP kind of networks, then we can do a lot more scaling even Ethernet networks may have problem here maybe Ethernet also you need to do some timeouts, etcetera, they usually set to be small values whereas, TCP IP timeouts are slightly bigger to handle larger networks and worldwide networks ok.

So, I am not going to discuss this, but there is also an issue of parallelism how to ensure that multiple activities multiple actions multiple access can be going on a; at a same time

in this kind network block devices everything else let discuss it right now. So, I just. So, what we have done. So, far is we looked at a simple character device driver, but counting the interrupts we also looked at some aspects of the block device I just briefly mentioned something about network block device we will start looking at network based storage system. So, I just wanted to say that it is an important issue and networks are critical for scaling the only problem is with networks is that the failures are very high. So, your error management or failure management has to be really very good it is a absolute critical thing that you have to do ok without that this whole thing just falls apart.

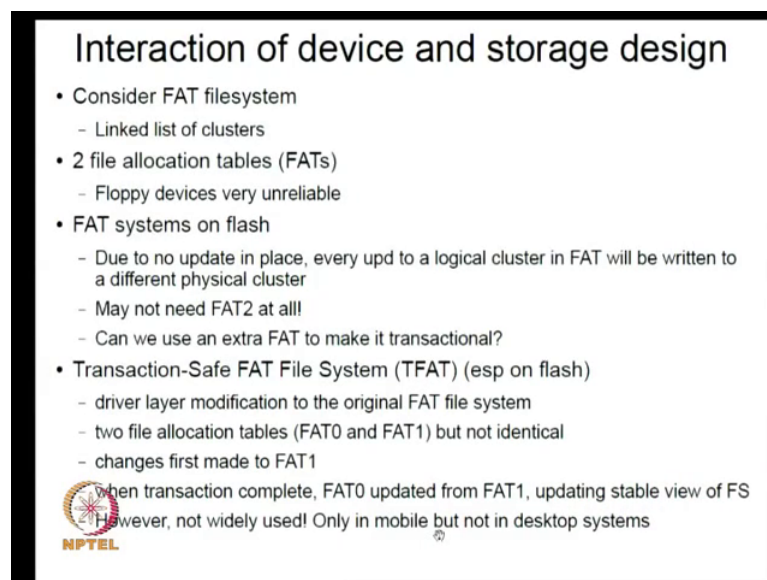
So, if you look at all these google file systems etcetera they are basically based on network based models their file system is based on having a network across many nodes and together they provide the file system; that means, that for them error management it has to be Centro the only reason why google file system works is because of error management they are able to handle it and they have a system which is I do not know how many nodes they say some people say one millions are whatsoever there or whatever some people say 5 millions are what is whatever and they are actually have a lot of storage and the storage is available anywhere any place and all its happening through networks.

Again the issue is what level is you can do it you can do it through a block device or you can do it with a file system block device means you are exposing at the level of at much lower levels whereas, the google file system has designed it to a at a file system way the reason why they have done it that way is because they use the native file system ext 3 on the nodes and on top of it they constructed it. So, that way we now have a simple file system ext 3 working reliably for them they not need to any worry about lot of things ext 3 handles. They will only handle the upper layers things for example, running c across geographically dispersed areas whatever in the sense what they have done is it is the division of the labor ext 3 is doing what its good at handling files locally then you have the google file system a layer on top of ext 3 which is good at doing things like working with large block sizes 64 megabytes working at replication working on things like error management failure management

What happens in that particular have 3 copies one copy is gone should I keep one more copy how do I sync it with whatever 2 other copies are there all those issues are handled at the gms server ext 3 is a some kind of a lower level service which is used to construct


up higher level service that is what it is going out. So, all I want to just I just want to end this particular discussion, but basically saying that there is higher rates of failure and that is the reason why if you now multiple types of let us say failure management will be happening ext 3 will handle certain types of failures and google file system will handle. So, other types of failures same thing with all these newer things called how do filesystem glusterfs files all these file systems also have 2 levels structure a lower level traditional let me call it node based file systems they will do certain things and you put some additional things on top of it. So, that is across multiple nodes you basically; so that you can scale on a much larger at a much larger layer.

(Refer Slide Time: 13:36)



Interaction of device and storage design

- Consider FAT filesystem
 - Linked list of clusters
- 2 file allocation tables (FATs)
 - Floppy devices very unreliable
- FAT systems on flash
 - Due to no update in place, every upd to a logical cluster in FAT will be written to a different physical cluster
 - May not need FAT2 at all!
 - Can we use an extra FAT to make it transactional?
- Transaction-Safe FAT File System (TFAT) (esp on flash)
 - driver layer modification to the original FAT file system
 - two file allocation tables (FAT0 and FAT1) but not identical
 - changes first made to FAT1
 - when transaction complete, FAT0 updated from FAT1, updating stable view of FS
 - however, not widely used! Only in mobile but not in desktop systems

 NPTEL

Now, let us also try to look at some issues regarding devices and storage design. So, we will look at for example, the fats file system I think many of you are familiar with the fat file system and its fixed in this simple file system it was designed in the late seventies a nearly eighties and its widely used now because it is the way you are memory sticks your camera all these things actually use fat its widely used, it is an extremely simple system for example, a file is nothing more than a link list of clusters. It is a very simple design; that means that if you want to get to one particular cluster you have to walk over the list its order of n depending on how much you have to walk. It is a very bad design you work on it. It is a too simpler design, I should not call it a bad design, it is too simpler a design ok.

So, people want the more advance file systems do not use linked list of clusters they use for example, beatrees or it could be some tree like structure like Unix does. So, with multiple levels of indirections various such things are possible. So, that you do not have to walk through that order of n you want to do log in typically and that this being an extremely simple system it uses layer and this may be called appropriate for an example you know the camera right most of the time you are writing sequentially right and typically things are coming in large chunks because you photos for some twenty 4 photos or of hundred photos and then you probably delete all of them.

So, the sense you have a huge sequential whole which again you can use later. So, camera is an example where linear linked list of cluster is possibly fine it maybe also fine for memory sticks you basically what you are doing how are you using memory sticks you are basically using it for transferring files from here and there right and if you keep writing sequentially probably its partly fine, but for a regular high access file system shared with a lot of people this will be not a good system. Now let us look at what this particular fat file system what does it for originally it is designed for floppy devices the fact file system came on floppy devices first and it migrated to a disk later. Now it is migrated to SSD also SS that is your memory sticks camera all those things right.

So, it has gone through multiple kinds of devices now the thing what a floppy devices is it is highly unreliable compared to a disk of those days also a floppy was highly unreliable it could bend somebody can keep it in the sun you know a number of things right. So, one way they decided to take care of error management was to have what is called 2 file allocation tables 2 copies.

So, that if one of them one part of it became bad cannot read forever after the bad thing about this is that anytime you update it you have update it 2 places. So, it comes slow, but floppy is by definition are not fast are no problem, we do not care because if human some human beings are writing and reading once and while of course, you can use floppy based systems even for doing computation, but that was in the very early days most often only on a input of the mechanism ok.

So, now these fats have also gone into flash as we have mentioned by a go now we can see that previously we had 2 fat tables the reason why fat tables was there for what; what reason it was there. So, that if one part of it became bad other part could be used

basically its keeping track of the fat table is keeping track of each file which order it is the various clusters in which sequence or order it is right and also it keeps track of which clusters are bad basically a cluster is a sequential set of bloc sectors. So, it is also the fat is also keeping track of which areas are bad and which are unallocated all those things are drawn and if there is a allocated files the file will have a pointer or it will you can go through the link list of pointers to get the complete file that is basically what is a fat is doing. So, the fats are important.

So, that is why there was a duplication in the early floppy devices, but. So, much are put on the on a flash what is the issue about a what about a flash you have basically if you really want to use flash properly as we discussed earlier, you do not want to do update in place if you do update in place essentially we have to erase the whole thing if you want to write a small thing here if you do update in place you have to erase it completely a much bigger area that is a pretty awful area of using flash. So, because of that you typically have no update in place you write one place if you have to re you again want to rewrite on that same thing we actually get a new erase block you write there and you continuously try to fill up that new erased block that you got right.

That means that by definition you are moving around your fat physically where there are going to meet at a time previously in the in the floppy case your fat is always in one single place alright and if that is got bad you are in trouble what about here now you are actually smearing your fat table across good parts of your device every time you write a fat that guy is going to moving somewhere else right; that means, you are not.

So, concerned about around one particular place you have to worry about errors in general about the whole place rather than a one particular place see what is the reason why you worry about disks and floppies if you have a single dust particle somewhere you might completely that whole error is gone right and that might be a few blocks whereas, in this system even if there is some problems somewhere you can always many time you write it you move to some other place. So, you are always doing it all the time ok.

So, every update to a logical cluster in fat will be written to a different physical cluster therefore, having another fat may not be that critical. So, some people have decided that for flash; you do not need a second copy of the fat throw it off not necessary. So, that is

one type of design some people have started using that way the other process now given that are freed up one fat can use it in more interesting ways one way to do it is what is called make it transactional what is this transactional I think as we discussed earlier when you do a file system operation you might update multiple things now if you conclude that operation all the changes should make it to disk or none of them should make it to disk what are not happening is part of it make it part of it do not make it.

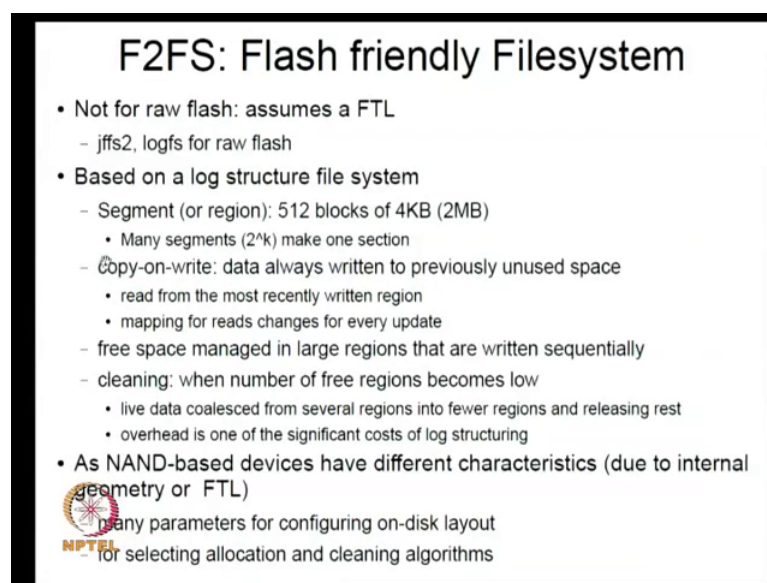
A transactional file system will guarantee that any logically connected operations updates they will all either go to disk or not now in the case of non-transactional fat kind of file systems we have this problem that if you take the floppy out while its writing etcetera you are in trouble alright you are right if some power fails as you are writing there can be problems you are not updated. If suppose, it is not transactional. So, the question is how do you make it transactional one way is this some a new type of fat file system was designed it called transaction safe fat file system for flash because flash anyway you have one extra typically you have instead of 2 fat tables you know we have only one other can be used. So, it makes sense for flash ok.

So, this was designed for mobile kind of systems not desktop systems mobile kind of systems and what do we do here you basically change the design of this file system driver layer modification to the original fat file system. So, you have you still have 2 fat fats fat 0 and fat one, but these are not going to be always synchronized there are always, but we will make changes to fat one and only then you know that all the changes corresponding to an operation file system operation are complete then you try to update fat 0 from fat 1. So, it is always a case that one place we always have a consistent version ok.

See it nots one; one is not saying that when you are updating from fat 0 that also can stop, but the thing is fat one is not intact. So, one place at least it is intact either fat 0 is intact because of updating fat one something has happened or I finish fat one I am trying to copy it to fat 0 and something bad happened in while you are updating fat 0, but I have fat one in consistent. So, you always have some kind of a consistent view of the file system that is possible. So, this has been designed it turns out as not been widely used because they are not available in desktop systems. So, if you want normally why do we use we always go back and fold between mobile device source and desktop systems right because all the series what happens are not on mobile devices typically even now.

So, the thing is there is still a problem if it was available in desktop system for it would be more badly used, but it is not available these are some commercial things is Microsoft to our business why they have done it on only for mobile not for desktop systems if it was on Linux probably, they would made available everywhere, but this is this t fat is a proprietary file system. So, it is available only on mobiles windows mobile systems. So, since I wanted to give some indication that your design starts very much dependent on your storage and depends on very much on the devices and various ways you can expect device.

(Refer Slide Time: 24:34)

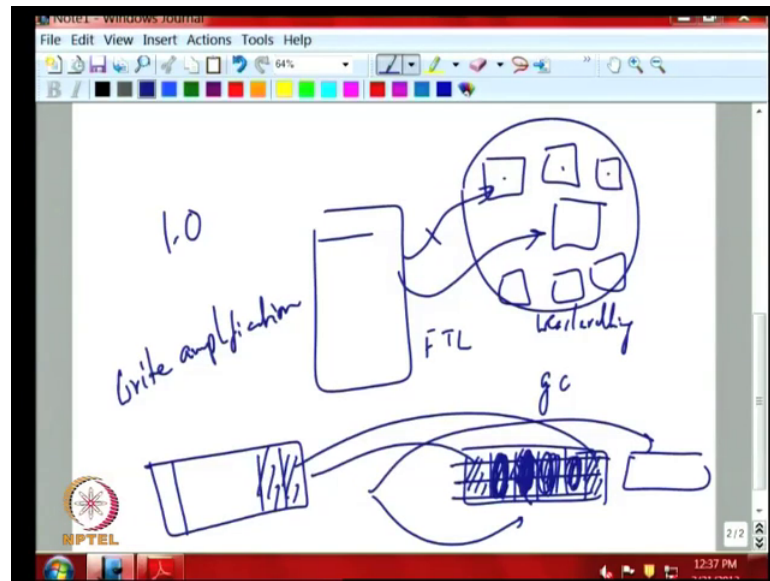


F2FS: Flash friendly Filesystem

- Not for raw flash: assumes a FTL
 - jffs2, logfs for raw flash
- Based on a log structure file system
 - Segment (or region): 512 blocks of 4KB (2MB)
 - Many segments (2^k) make one section
 - Copy-on-write: data always written to previously unused space
 - read from the most recently written region
 - mapping for reads changes for every update
 - free space managed in large regions that are written sequentially
 - cleaning: when number of free regions becomes low
 - live data coalesced from several regions into fewer regions and releasing rest
 - overhead is one of the significant costs of log structuring
- As NAND-based devices have different characteristics (due to internal geometry or FTL)
 - many parameters for configuring on-disk layout
 - for selecting allocation and cleaning algorithms

We will just look at it in one more different term of this there is something called flash friendly file system which recently has become part of Linux this is done by Samsung because Samsung is one of the biggest manufacturers of flash nowadays and they designed something called flash friendly file system. Let us briefly look into what these are all about first of all, if you know that this particular file system is not raw flash what does that mean it assumes a file translation there we have discussed this before what is a file translation layer basically what happens is that since; if is no update in place is what you are looking for.

(Refer Slide Time: 25:32)



So, if there is some block here something some pointer is pointed out of this and then you update this, then this pointer should be broken and should connect to this.

So, there is some table out here and this is needed for every update and somebody has to keep track of it and that is done by trans translational via FTL. So, every single updates you have to keep doing it. Now this FTL in addition to doing this it also does things like what is called wear leveling and also what is called garbage collection both the things are done. So, what is wear leveling it tries to ensure that if this is the device all of them are erased in equal number of times possible equally if possible. So, at any point in time the wear level number of times erase has happened here or the number of times erase has happened here or here should always be same approximately same that is wear level.

What is garbage collection as we mentioned before we have we are writing in we have various blocks, but writes are much smaller than that see writes could be this much each write could be this much, but erase block is this much. So, it may be possible that I wrote this I wrote this I wrote this, but I overwrote this one because I am doing overwrite actually this means that no longer the pointer is pointed to this it actually pointing to some other place. So, this is value is dead now this area will be dead similarly this could be dead we can also imagine that someday this also becomes dead ok.

Now, if this also becomes dead you can see that most of it is dead material only few are live; let us say these 2 are live. So, if I want to erase it if I need some space ideal thing

would be to copy this into some place into a new erased block again copy it here copy this 2 places this guy goes here, here and now this can be completely erased once you erase it completely. Now it is available again for allocation this is garbage collection. So, what is the problem with garbage collection you have to first do the copies which is can be significant costs second thing is you are doing another extra write you are reading it, but you are writing also to a new erased block.

So, you are introducing extra writes and there is a measure called write amplification write amplification that happens due to garbage collection if you do not do any garbage collection that extra 2 writes will not happen right because in garbage collection 2 extra writes also happen. So, in a sense we can average our given the. So, many writes you are doing application writes how many extra write you are doing because of garbage collection you take that ratio.

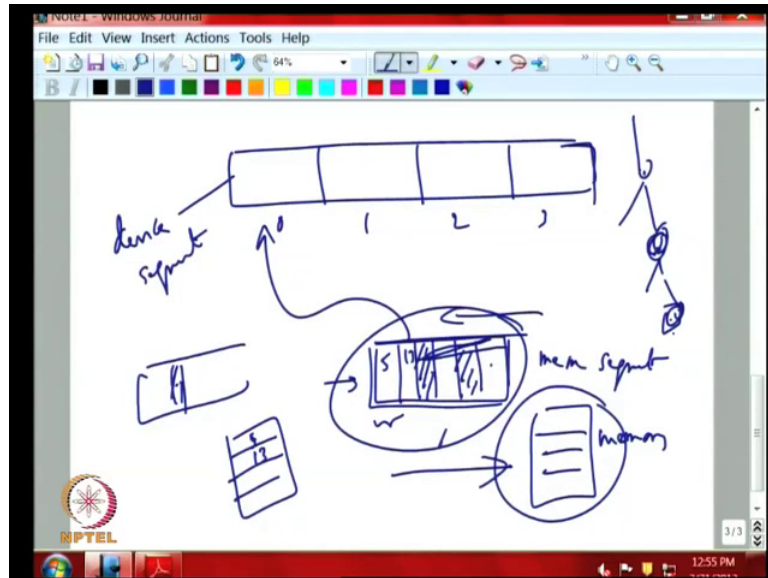
Such if you do not want a very high write amplification you should be as close to 1.0, if possible 1.0 is impossible; that means, you are not doing essentially you do not have any garbage collection going on you want it to be 1.1; 1.2 something in that if possible. So, this particular systems assumes it is an FTL a raw flash will not have any of these things the file system itself has to do all of these things it includes for example, remapping it might include garbage collection wear leveling all those things have to be done by the file system if the file system is running on top of raw flash.

Whereas if your file system is running on top of FTL then the wear leveling garbage collection all those are some of the remapping etcetera is done by the device itself the FTL enabled device that most of the things we carry around memory sticks they all have FTL on inside them. So, most widely available is FTL based. So, if you design something for raw flash most likely there is not enough market for it because what people want is somewhat simpler interface and that is what FTL based storage devices do now this particular system is based there are there are some other filesystems in Linux called jffs 2 logfs for raw flash. So, those things directly work on raw flash we will not be discussing it here right now.

Well this particular Samsung based Samsung designed file flash; flash friendly file system is based on a log structure file system. Now let us just quickly look at what log

structure file systems are. So, the basic idea about a log structured file system is that you think of your device in terms of some.

(Refer Slide Time: 31:00)



So, many segments some hundreds of segments I am I am putting only 4 of them here segment 0, 1, 2, 3 and basically in memory you keep a memory segment this at this segment or device segment this will be your memory segment. So, it will be let us call it device segment.

So, what you do is anytime any writes happen you keep on appending the writes in the memory keep on collecting that you keep collecting and basically what you do is you also keep a track of which block you wrote essentially this will be you will create a structure by which you know what block it is and the data corresponding to it in some sense it is some more tables are there saying that this particular thing is block number 5; this is block number thirty seven oh thirteen etcetera, but we are writing it continuously. So, this will be block number 5 thirteen etcetera, but they are logically they are not contiguous, but physically they are contiguous ok.

So, once you collect one complete segment worth of it you can erase it to this device, but while collecting it you may find this becomes dead that become dead you; you decide we will see that is it worth cleaning it if it is worth cleaning I will clean it and then once it is again semi full I will put it here. So, that is that you do it in memory first you keep on accumulating the stuff in the memory now only thing that you have to worry also is now

that writes can always be they will have the most recent value right for reads what you have to do is you have to when you want to read a write you have to read from the backward direction to get the most recent value ok.

Writes are always going in this direction, but suppose suddenly I want to for writes I do not have to figure where the things are I just have to keep on appending it, but for reads I need to know where they are right because previously it was somewhere here right in some of their segment because a overwrite happened I instead of updating in place I put it here it could be the same segment or the some other segment; that means, for I need to keep a indexing table which tells me; if I want to read something there is a most recent copy that its elves again will be memory that indexing table will also be memory right this way it is going to be updated every time any rewrite happens ok.

So, when you want to read you have to look up this guy as it which is a most recent version of that one and you have to it. So, all of this stuff is in memory and. So, you have to flash these things to disk every. So, often even this might have to be flashed to disk once again because this keeps the track of the read if I want to read something it has to be where the most recent copy is. So, roughly what we are doing is we are accumulating stuff writing in bit chunks and then we will also keep a indexing structure. So, that for any block we want to know which is the most recent copy, where is the most recent location all these stuff is typically in memory, but some of the structures for check pointing purposes you might have to put it onto disk also on into stable storage or into flash.

So, you have this is roughly the design of a log structure file system basically the idea here is the log itself because what is a log in the way you continuously, write you know that is a log the log itself is a file system that has everything in it whereas, in other file systems like in ext 3, ext 2, etcetera, right, you always do update in place. So, the log is actually not if you use a log in those kind of systems there is quite different from the actual data where is there here the log itself has everything. So, this is based on a log structured model what we do is a segment is here 512 blocks of 4 kilobytes of about 2 megabytes.

So, your segments are 2 megabytes; that means, you have to allocate 2 megabytes chunk of memory and everything is accumulated there and every. So, often approximately 2

megabytes will be flashed to disk if you want and then in addition they have to make it for even writing longer sections many segments can be made into what is called a section and typically is in the 2^k that is you either have one segment as one section or 2 segments as one section four segments as one section, etcetera; as I mentioned already its copy and write data always written to a previously unused space read from the most recently written region and mapping for reads changes for every update that is basically log structure file system and free spaces managed in large regions that are written sequentially ok.


So, basically you manage in terms of regions and these are written sequentially and it turns out even for flash just like for disk writing it sequentially is better than writing it randomly. So, what do we do again as we mentioned before you have to cleaning the number of free regions becomes low live data is coalesced from several regions into fewer regions and releasing the rest of it. So, overhead is of copying is one of the significant cost of log structure if you use log structure the copying cost becomes important in the flash case the write amplification is also important cost these 2 things are important.

So, this particular system there was. So, tweaked a quite bit. So, the most devices come from you know various flash devices you have enterprise flash you have consumer grade flash you have camera type flash you have emmcs for camera you do some different things there are so many types of flash. So, because of the different characteristics they have various base to configure it also for selecting which type of allocation and cleaning algorithms to use.

(Refer Slide Time: 38:04)

FTL and LFS

- FTL typically uses a log-structured design to provide wear-leveling and write-gathering
 - two log structures active on the device
 - f2fs uses FTL: f2fs makes no effort to distribute writes evenly to provide wear-leveling, as provided by FTL
 - f2fs provides large-scale write gathering: when many blocks need to be written at the same time, they are collected into large sequential writes that FTL can handle easily
 - But instead of a single large write, f2fs actually creates up to six in parallel.
 - Each set of blocks grouped with similar life expectancies
 - Makes garbage collection process required by the LFS less expensive
- However, f2fs doesn't always gather writes into large regions

 Some metadata, and occasionally even some regular data, is written via random single-block writes: FTL takes over here

NPTEL Simplifies design

Now, let us look at the interaction between FTL and LFS it turns out FTL itself uses a log structured design. So, now, we have a flash friendly file system using LFS and we have the FTL typically if we see the design of the FTL itself what is FTL this is the translational layer it turns out the flash translational layer itself uses a log structured design because it its used to provide wear leveling and write gathering because that way also it has to write in larger chunks at the lower level.

Even if there is no file system, so that way we are also using a log structured file system. So, essentially this is a interesting design where there are 2 log structures active on the device one at the FTL level one at the file system level, but the way the; have done the design the top level f 2 fs that is the file system it tries to exploit the characteristics of FTL how does it do it; it knows that the FTL is doing the business of wear leveling it is trying to distribute at FTL. So, it does not try to bother about it, it says he is going to take care of it I am not going to worry about it. So, this business of wear leveling is going to be best for the FTL, I am not going to touch it, but what I will do is I will ensure that large scale write gathering happens in large sections ok.

So, and the FTL is much better at handling that we mean san large sections, but actually even more specifically instead of just doing it as a single large write it keeps essentially control about six sections are gathering the blocks at the same time six independent ones what is the idea it turns out that in any realistic system various blocks will have different

flash expectancies some files are mostly stable nothing happens to them. For example, flash, it is a password its changes only every time when a new user comes in and goes whereas, your cache; for example, if you are a Firefox there is a cache directory and lots of files are written and lots of files are deleted that is creation and deletion and there are some other situations where a lot updates take place.

For example somebody is keeping a track of how many users came to your website there is some small file is keeping that and it keeps on changing every. So, often right which is a very heavily busily visited website it will be changing every second possibly right. So, there are different files and correspondingly different blocks there is different life expectancies we want to do some intelligent garbage collection what is the right thing I have to ensure that those blocks which have single life expectancies are as similar to are in grouped appropriately. So, for example, those things that do not change if they are in one place I can clean them once every let us say hour because they do not change much and just once in a while they change and if I do it as slow as infrequently as one hour its ok.

Whereas if something is happening very fast all the time right; that means, that many many things are written and they get overwritten; that means, that in a non-updating place kind of system that things always dead the physically that place was is already dead. So, if none of the activity is going on and I keep I group all the highly updated things in one place even if you wait for a small amount of time I just let for one second probably everything almost everything is dead and once in a while there will be a few things much will be alive I just copy exactly one or the 2 things alright. So, what the LFS does is to do this intelligent business of grouping those blocks that are changing according to their life expectancies that is what is to do.

So, that is one part of the thing in addition as I mentioned LFS basically this flash friendly file system uses log structured file system, but does not use it everywhere; why does not it use everywhere because sometimes it has to update metadata; metadata is typically small its information about a file, they may not be as big as a file, etcetera because we have metal for example, you have a music file which is sitting on your memory stick the metadata will burnly by project at the most at the most 512 bytes some digit or else it just helps you the author the what say information about how big it is

when was it created all those kind of things that hugely fits in 128 bytes or 256 bytes or 512 bytes ok.

So, you are not talking in terms of 2 megabytes of segments or even longer sections or 4 megabytes section or 64 megabytes section we are not talking about that we are talking about much smaller things, but the thing is for correctness you might have to update this metadata quite quickly. So, what they have decided is the lower level is typically doing log structured file system that is the FTL upper layer does it mostly most of the times it does log structure, but when it comes to small updates it says I do not need to do this keeping all these things around and then putting it into some big structure that will take a long time because if you take a 2 megabytes section right if you are only putting in small I node information it might take a long time before we are done with it right. So, it creates some problems.

So, basically what they do is they say any FTL is doing this business of collecting things together physically right at the physical level I will do small transactions in in in in place. So, it turns out that by doing it, it simplifies the design. So, they do that one. So, in a sense it is slightly optimized for metadata that part of it the ones which do not go through the log structured file system part.

(Refer Slide Time: 44:30)

Reducing Cleaning Overhead

- f2fs has six sections "open" for writing at any time
 - different types of data written to each section
 - different sections allows for file content (data) to be kept separate from indexing information (nodes).
 - Also to divide data into "hot", "warm", and "cold" sections (thru heuristics)
- Directory data treated as hot and kept separate from file data
 - have different life expectancies
- Section full of Cold Data likely to not require any cleaning
- Hot Nodes expected to be updated soon
 - if we wait a small amount of time, a section full of hot nodes will have very few live blocks: cheap to clean
- Problem: whenever a block written, its phys address changed, so its parent in the indexing tree must change and be relocated, and so on up to root of tree
 - Uses a special table for indirecting to actual blocks

Tree stores offset into table only; metadata changes do not need mod of tree

NPTEL

- Table uses a special 2-location journaling to reduce overhead

So, as I mentioned earlier this f 2 fs has six sections open for writing at any time different types of data written to each section different section allows for file content to

be kept separate for indexing information I node kind of things and basically it divides the data into hot warm and cold sections and this is done through some heuristics

Again this is the part of the filesystem heuristics again it is not done based by FTL because FTL does not know exactly the application level information it is being designed by some vendor somewhere and it has no serious understanding about what high upper level case are doing.

Similarly for example, directory data is treated as hot and kept separate from file data and because they have different life expectancies usually it is that for example, often times you might keep some information about see you know particular directory there may be a lot of files and anytime any file is added or deleted you have to change the directory contents and directories are luckily not too many number. So, they are changing up every time there is a file activity and since there is smaller number you can essentially keep them cached that is why they is often times something called directly it name looked up cache for example, in Linux and typically these things are have a different life expectancies.

So, basically you keep one of these sections as directories. So, basically hot nodes expected to be updated soon if you wait a small amount of time a section of full hot nodes will have very few live blocks cheap to clean as already we discussed before, but one big problem with this kind of systems is that whenever a block is written suppose I have a tree like information whenever a block is written its physical address changes because of physical address has changed its parent in that indexing tree must also change because finally, you have a parent right.

See what is happening; let us say that I have some information like this if this I update this what will happen the physical because now we have no update in place; that means, address of this guy is no longer what this guy is previously stored; that means, this guys also has to change because this guy has changed this also will be the that not updating the place; that means, this guys will also change; that means, this guys also has to change it goes all the way to the root ok.

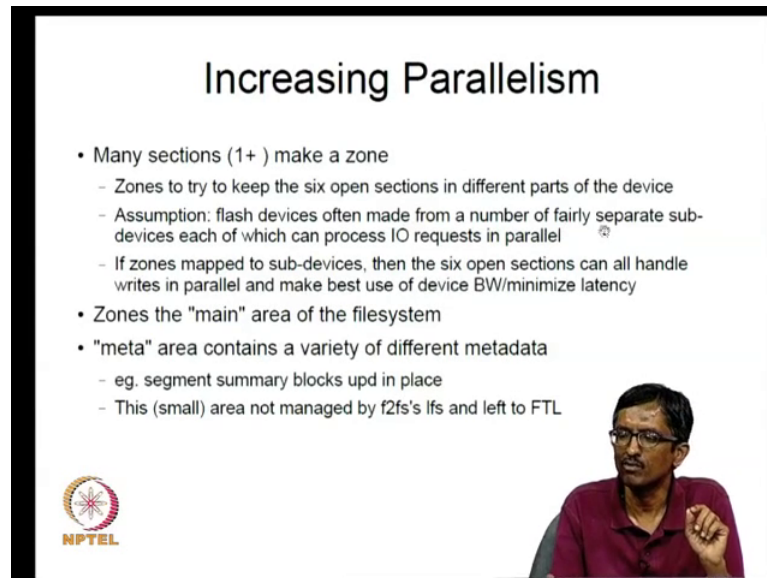
So, one change here means you have to change everybody up which is a big problem. So, basically, there is a big problem with respect to this. So, what people do is we have a special table for in directing to actual blocks. So, what you do is you keep another

special table and that will always point to the correct new location if originally it was here it goes to here, but the 3 itself will store the offset into this what you do is the offset is always fixed. So, the tree is always changing talking about offset into this table this , but essentially you are indirecting it to a fixed offset location and then that gets updated, but the tree itself is having the same offset all the time you use a special table for indirecting to actual blocks tree actually stores offset into table only and that is fixed irrespective of all the changes you are doing in spite of the because you have this non updating place it will be going to nearest places, but offset remains fixed and the offset location keep changing ok.

So, essentially the indirect at the same offset in the table, but our problem now is that the table itself is changing. Now it turns out that they use a special technique called 2 location journaling to reduce overhead its similar to what we; when we talked about 2 special location journaling its similar to what we decide what we discussed in the case of this this this fat 0 fat. Remember the fat 0; fat 1, you keep one thing consistent and right to work on then you go back and put right, it turns out that they do something similar here you have 2 locations for this node table this is what they call the special table and then they keep going back and forth. So, that reduces some overhead ok.



Say basic problem is that every time we update we have to start looking up a new another table. So, they avoid it by a some kind of recursion is there they want to avoid it by saying that I use these 2 location generally by fat 0 fat 1 kind of model that will essentially reduce the amount of additional tables you have to keep ok.

(Refer Slide Time: 49:41)



Increasing Parallelism

- Many sections (1+) make a zone
 - Zones to try to keep the six open sections in different parts of the device
 - Assumption: flash devices often made from a number of fairly separate sub-devices each of which can process IO requests in parallel
 - If zones mapped to sub-devices, then the six open sections can all handle writes in parallel and make best use of device BW/minimize latency
- Zones the "main" area of the filesystem
- "meta" area contains a variety of different metadata
 - eg. segment summary blocks up in place
 - This (small) area not managed by f2fs's ifs and left to FTL

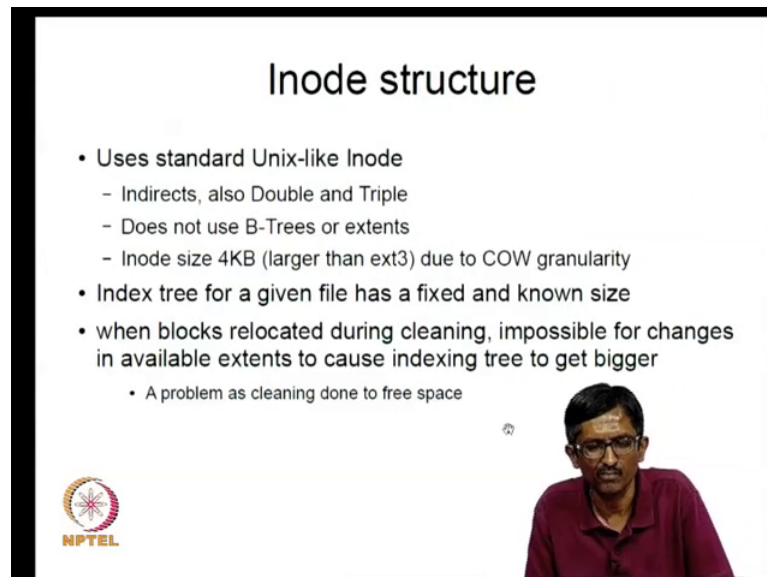
Now, let us look at one more thing as now it turns out that you keep many sections to what is do what is called as zone. So, basically now we have blocks segments sections and zones now zone concept has been come up basically the idea is that a flash device often made from a number of fairly separate sub devices basically as I mentioned if you look at a SSD it is made of many chips and many chips for example, nine of them it make into a bank it might have 64 banks some multiple interface etcetera right and basically what will happen is that each of those sub devices right if they can be accessed in parallel then you have parallelism ok.

So, basically flash devices made up from simply fairly separate sub devices each of which can process IO requests in parallel, I think you are familiar with this part. So, if the zones if we have some kind of a zones in a higher level structure basically multiple sections making it one zone if zones are not the sub devices then essentially the six open sections can all handle writes in parallel and make best use of device bandwidth or minimize latency. So, essentially you are pumping the same device 6 operations can be gone at the same time because they are sitting on 6 separate groups of chips and each group can be considered in some kind of interface ok.

So, each group of them is its interface. So, I can essentially keep doing six or 4 whatever it is that is the unit up to 6 depends on the device. So, it turns out the zone are basically the main area of the file system there is also a meta area which contains metadata as I

mentioned earlier this ones use much smaller sized units of information that Inode kind of things and again instead of doing the doing it in the flash file system to manage this thing its left to the FTL to handle it.

(Refer Slide Time: 52:02)



The slide is titled "Inode structure" and contains the following text:

- Uses standard Unix-like Inode
 - Indirects, also Double and Triple
 - Does not use B-Trees or extents
 - Inode size 4KB (larger than ext3) due to COW granularity
- Index tree for a given file has a fixed and known size
- when blocks relocated during cleaning, impossible for changes in available extents to cause indexing tree to get bigger
 - A problem as cleaning done to free space

In the bottom right corner of the slide, there is a small video inset showing a man with glasses and a maroon shirt. In the bottom left corner, there is the NPTEL logo.

Now, if you look at the Inode structure also there is some interesting things here also normally most of people will say why do not we use b trees etcetera or extents I think you are familiar with b trees the good thing about b tree is that you give a typically a log n kind of access path to get to in particular location

Or you can use vertical extents; extents are basically instead of always storing in blocks of 4 kilobyte you might store in terms of contiguous chunks it can be 512 kilobyte chunk it could be 4 megabyte it could be 288 kilobyte whatever its depending on the contiguous thing you say that my file is composed of so many extents of contiguous nature. So, you might have a file with 288 kilobyte chunk followed with another 512 kilobyte chunk followed by 4 megabyte contiguous chunk 3 extents 1 of 288 kilobyte 1 of 512; 1 of 2 megabyte things like that right.

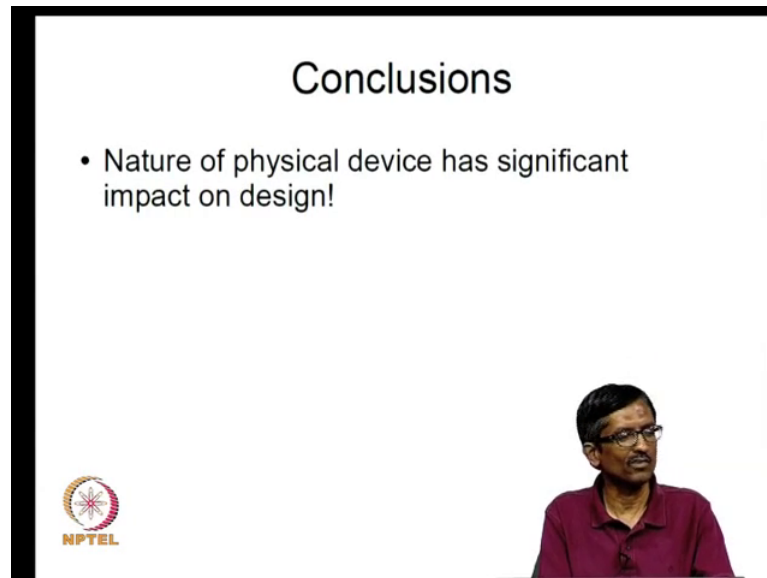
So, surprisingly this particular design uses only this standard Unix like Inode it has got this direct blocks indirect blocks and indirects also double and triple Inodes just like I think we discussed earlier right it does not use b trees or extents and there is a reason for it we will come to that soon it turns out the Inode size is 4 kilobytes its much bigger than ext 3; these ext 3 typically the Inodes are 256 bytes or 512 bytes this is much bigger

because it turns out it uses vertical copy and write for copy and write it turns out using 4 kilobyte chunks is more appropriate it gets closer to the peak size and turn out that that is better ok.

So, but the good thing about this is that if you use this kind of a model index tree for a given file has a fixed and known size why is that because I am trying to clean stuff when I am cleaning stuff it may turn out that I might get different. For example, let us say I have a ext extent based file system extent based model when I am just about say I am the reason I am doing cleaning because I want to get more space it is using certain types of extents, but while cleaning what can happen is that you might get different types of extents because it is a concurrent system while you are doing it somebody else can runaway with what is available and it might be turning out that you are forced to use extents which are not as bigger the previous one because you are copying things around.

So, what is contagious, what are the contagious sets of blocks, we get at the time you are doing the cleaning, what is available for you to put your live data that may be quite different from what you have? So, basically what will happen is that your size of your indirection can keep changing depending upon the availability of extents at the time you are doing cleaning now that is slightly a non-intuitive thing because you are doing cleaning to think that you are getting extra space, but as you are doing it; it turn out you actually start to getting bigger it can happen. So, this is not a nice thing therefore, if you use something like a the standard Unix kind of Inodes it is a slightly more non surprising in terms of operation does not give you shocks that kind of thing ok.

(Refer Slide Time: 55:41)



So, I think I just want to conclude here I just want to say that you think about the device you will see there are a lot of things had to be taken to account the device characteristics has a lot of implications on the way the design of the higher level sub systems have to be I just took fat and the flash friendly file system as the examples about how the non-update in place right how they have affected the design of the systems.

So, I think I will conclude here at this point.