

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

Communication Protocols for Networked Storage Systems

Lecture - 10

NFSv2 Problems continued, NFSv3 Enhancements, NFSv4-Overview/Changes/Data structures/Leases/Security, CIFS, Unix-Windows integration in NFS

In the previous class, we started with the NFS as an example of a network file system.

(Refer Slide Time: 00:33)

NFSv2 problems

- Maintaining UNIX semantics
 - Open file access permissions
 - Posix checks on 1st access; NFSv2 on every access
 - Atomic I/O operations ®
 - Deletion of open files: what if server deletes file?
- Cache consistency guarantees
 - NFS 2 checks if mod time of client cached data diff from server mod time. Works if server only making changes
- Security
 - Access control: User credentials
 - Securing data traffic
- Performance: UDP storms, Synch writes: ad-hoc opts.
Needs Portmapper, mountd, lockd, statd
- Functionality: 4GB file size limit

And again just to recapitulate with storage area networks we assume that there is a single application which is managing the storage. So, there is no essentially you cannot you can say that there are no consistency problems. Somehow, somewhere in the application they are managing issues relating to consistency, issues relating to quality of service security etcetera. So, it is not part of the storage system to take care of it. It is in the part of the some other upper layer typically the application to handle it, whereas, with once you go to NFS. The idea here is to provide some level of support in some other series. It could be in the case of consistency management or it could be in terms of security etcetera. Now, NFS version 2 when it came up it was very early on 1980s.

So, it is particularly simple in its ideas about how to handle these issues. So, we will see; what are some other problems. I have mentioned some of this in the previous class, but I think that this reiterate one more time. For example, maintaining UNIX semantics is not

easy for NFSv2, Why is that? When we open file in UNIX right, what happens is that regularly UNIX checks on first access. Once you have access to the first time, is not checked here. Once you have a file descriptor that's it, you no longer have subject to be checks. Whereas, NFSv2 being a stateless protocol. Essentially, what happens is that if the permissions have changed, then it mainly it essentially it gives different behaviour than Posix gives you. Same thing atomic I/O operations, it is possible that while we are doing some long I/O. Let us say four megabytes or whatever. Since, it takes multiple times and multiple network packets to go back and forth some things can change on the server.

So; that means, you cannot give atomicity guarantees. Similarly, there is a issue of what is called deletion open files. This is a standard way in which temporary files are created in that to make sure that do not forget to delete them, what happens is you open the file and delete them. And the idea here is that in most it is where it happens is that the open file is essentially used to available for reading and writing. Whereas, was the protocol temporary file is closed, then the file blocks were returned back completely. Now, this semantics and Posix also has some problems in doing supported in NFSv2. For example, if the server itself deletes the file. Then basically as we discussed previously, every new file name is reused the generation number increases because of that it turns out that if the client now goes and asks for the same file.

There open file is has to be its going to get a different let us say generation number and therefore, we come back and say that it is a stale file. So, you can not really have essentially the colour medium that is there in Posix of deleting an open file. So, for that you need to do some other things. For example, in the case of some of these things there is some checks possible some sorry solution is possible, but there is still a case of deletion of open files. Suppose, this is a client and one of its processes has opened the file and some other process on that same client deletes it. Then, since the client actually all the requests have to go through the client. The client can keep track of such things coming from various processes in that particular client.

And in case somebody has the deleted a file, open file. It can actually know that this what is happened. I mean therefore, it can essentially still provides Posix kind of guarantees, by rename that file to some name which cannot be given by anybody else and excessively making that particular file still available to the party it has got to open

first. Whereas, but this solution does not work in case of server deletes of file. So, there are some holes in the NFSv2 can handle some of these Posix kind of semantics. Now, similarly there are some problems with cache consistency guarantees and basically NFSv2 checks in the modified term of a client or the cached data different from server modification time. If by this way; it can figure out if something has changed.

But if this works only the server is making changes, if some of the client is making changes this does not work. Essentially, the client has some modified time and there is a different time which if it is informed, then it can figure out if something has changed. But if multiple clients are modifying it they might not necessarily inform the server. Therefore, across multiple clients this may not work. So, in a sense your cache consistency guarantees are quite weak. Again with respect to security there are various models again an NFSv2 came on the scene first, there was no serious issue of cryptography. For example, public key and persistence came out only in about approximately 1977 or so, by the time various methods worked out it was rather let us say only the next one decade. So, security that NFSv2 provides this closer to the standard UNIX model.

Where it's assumed that across all the clients there is uniform UNIX like UID and GID, user Id's and GID's and therefore, that is the model that has been initially proposed and this turns out to be somewhat problematic and larger installations and especially when administration is handled by different multiple parties and similarly the data traffic itself goes on unsecured networks. There is no notion of encrypting the net the traffic that is being sent out in form of requests or the data that comes in. Again there is user credentials is happen is NFSv2 came much earlier before security become important and local area networks are internet and therefore, better models attempt to handle this all of it. You can see the kind of problems about again this area. For example, route on one machine could be let us say should not be allowed to act like look on some other machine.

If that is allowed then any user who whose route on a particular machine is essentially it has complete control of the whole of the network. So, which is a problem and therefore, there are some elementary methods available in NFSv2 to take care of this kind of problems, what they do is if somebody says your route they are mapped to and nobody. That means, that even if you came to the route the minute the request comes with the

route as this one it is mapped to nobody on the server side. So, there are some elementary methods, but there is certainly not very satisfactory.

Similarly, performance problems are also serious in NFv2. For example, you can have UDP storms. Again NFv2 manage first thing about they initially proposed that UDP is be the choice of protocol. That in on being one can early 1980s when these protocols were defined TCP was considered to heavy because taking too much passive in the CPU.

Therefore, UDP which was less of issue with respect to processing actually has that any processing. Therefore, it was considered to be a better way to get effective through put in the system. Now, the UDP you do not have any flow control of any kind. Therefore, it can turn out that you can have UDP storms. So, with that the NFS let us say server can get overloaded. Once it gets overloaded it looks as that has crashed and all kinds of problems were start coughing up.

Similarly, because of one critical issue of the NFS being that this the server being stateless any writes it have to be sent to the server or synchronous. Now, once we have synchronize right it terms out to be make the server very slow in terms of request. For example, again if you have discuss your device for synchronous writes.

Essentially you have worked in those days about 25 milliseconds per byte. That means, you could at the most have 40 I/O per second you could not go beyond 40 I/O per second or something equal that some other of 40 to 50. So, to handle this problem there are various ad hock optimization that had to be devised. One of them was to have let us say non-volatile memory of some form and usually that involve the non volatile essentially memory with battery backed up which is battery backup and there is other kinds of models. Other issue about NFv2 is that it requests lot of other infrastructure something called Portmapper, mount daemon, lock daemons, stat daemon etcetera. Let us go in the reverse order.

Stat daemon basically is summing of notification and some machines have seemed. Sometimes you have to do some elementary, you need to know which machines are up and down and stat d is something that is required this is not part of the NFS protocol. So, this also have to be taken care of, but your NFS system itself needed to access this information. That means, it had to go through some other protocols outside of it. Similarly, lock daemon; now we will have multiple users updating a particular file

system, you need to have some notion of let us say who amongst the file for the time being while it will be updated. If you have concurrent writes in the same file it is possible that you do not get any consistent version of your view of the file. So, you need a lock daemon.

And this lock daemon by definition had to be stateful because that to be stateful it could not be affected into the NFSv2 model because NFSv2 essentially says that the server is going to be stateless. It is not going to keep track of blocks it has been given to various see various clients. So, this lock daemon again sits uncomfortable with restore infrastructure, there is also another protocol called mountd, which basically helps you figure out which all systems are available out there and which can be mounted and again this is outside of the NFS protocol and typically is done through some user level daemons and somewhat different from the way the file system model is there in NFSv2. Which is typically kernel solutions, kernel based models. Similarly, this Portmapper also was another issue that has cropped that came up.

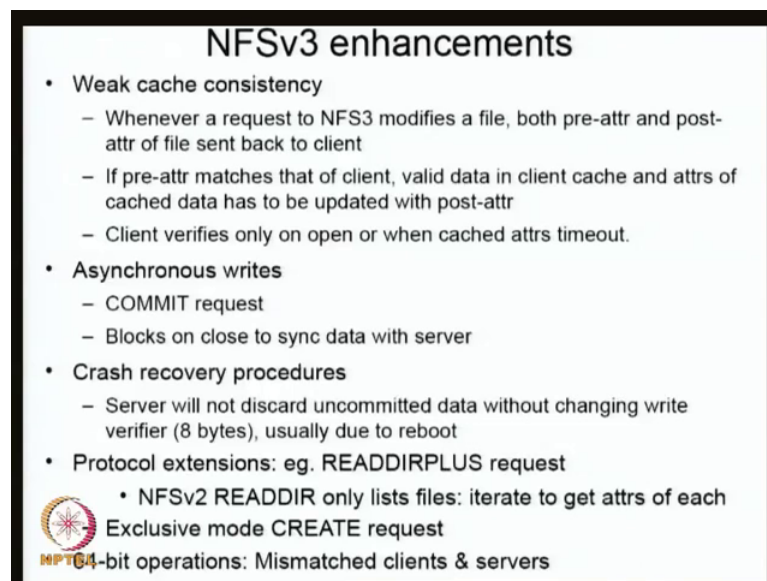
Basically, there were low standard ways in which RPC calls could be mapped to services. So, there have to be some way of mapping any particular service we needed to some particular port numbers and these port numbers were not specified in the NFSv2 protocol. Essentially, when NFSv2 came up networking was still in its infancy. So, not a many not many things were defined. So, it is expected by that some other models will be used to figure out all these things. For example, if you had to have this mount daemon or lock daemon they will be listening on some ports. The question is; how does the NFSv2 protocol move where these guys listen is involved. So, this Portmapper had to come to the picture. So, this also in the beginning was sort of semi ok, but later it became complicated.

Because it turned out that, these ports mapped were mountd lockd etcetera. They were can get into problems with once you have firewall. Again the security issue of local area networks was not very clear in the very beginning. So, once you cross local area networks then it turns out that there could be firewalls involved and they might block these ports and the ports that were initially used were not one of those system ports. They were the ports that could be defined by users and therefore, there were some problems with proper operation of NFS protocols assembly. So, there are various issues of this kind including some smaller slightly more simpler issues like the file size limit being 4

GB etcetera. So, that NFv2 solve one important problem that of giving access to files in a local area network quite well.

And so, it probably became very a popular in departmental setting where you have to use data from multiple big servers. And so, it became very soon, but there was certainly need scope for improving the way it performed or it the kind of services it provided. So, after NFSv2 the v3 came into picture.

(Refer Slide Time: 14:25)



NFSv3 enhancements

- Weak cache consistency
 - Whenever a request to NFS3 modifies a file, both pre-attr and post-attr of file sent back to client
 - If pre-attr matches that of client, valid data in client cache and attrs of cached data has to be updated with post-attr
 - Client verifies only on open or when cached attrs timeout.
- Asynchronous writes
 - COMMIT request
 - Blocks on close to sync data with server
- Crash recovery procedures
 - Server will not discard uncommitted data without changing write verifier (8 bytes), usually due to reboot
- Protocol extensions: eg. REaddirPLUS request
 - NFSv2 REaddir only lists files: iterate to get attrs of each

Exclusive mode CREATE request

64-bit operations: Mismatched clients & servers

About, almost I think almost a decade later and so, basically here you try to be slightly better with respect to cache consistency let us look at the kind of scene that we can do here. So, what is that in the case of NFv2 you have a, if you have if you crash some attributes, you do not look at its anything has changed because of other access used by other client or server you only look at it if the attributes timeout . So, the timeout essentially defines your consistency vendor whereas, in the case of NFSv3 what you can do is; whenever request to NFS3 modifies of file, you basically the server sends both the attributes that existed before and the ones that you are asking the 2 change it to.

Now, if the pre attribute matches that other client where the client had already cached. Then we know that the data is valid in the client cache and then you can update the attributes on the cached data that you can do. Basically, because you know how the matrix check now you can check it out. Now, the only issues that the client verifies only on open or when cached attributes time out; so we still have the same as NFSv2 that is if

somebody has changed in between I cannot figure out. But it is little better because I if it transfers that the pre attribute does not match the; that of the post attribute.

I know somebody has changed it or I can program on some of the resolution protocol on top of this. I can not part of NFSv3, but I can still do it when it is possible. Now, other issue with the NFSv2 was the fact that your writes have to be synchronous. Now, to take care of this they introduced something called the asynchronous writes in to NFSv3. Basically, the idea here is that when somebody writes to a server. The server can cache it and keep it in its cache it does not have to send it to disk in its local units storage system and only the it is only required to do it. If the client says not please commit it then it is the question will commit it then the server has to committed to disk and then we will be latency at that point in time.

But then it is guaranteed by it is there is a guarantee for the client that the data will not be lost and basically the; this particular model now you can be the follower. A client can write to the server, but it will keep a cached copy of the data and then it can keep writing and the time when it wants to be show that is actually on stable storage it comes at commit. Once it gets the response a successful response from the server it knows that has been committed to stables storage and server side, then it can discard its own cache copy. So, till that point of time when it gets the successful response it has to keep the; whatever it has returned there is local client cache. So, by this place what can happen is that you can ensure that you can write at much you can support higher I/O operations then as possibility NFSv2.

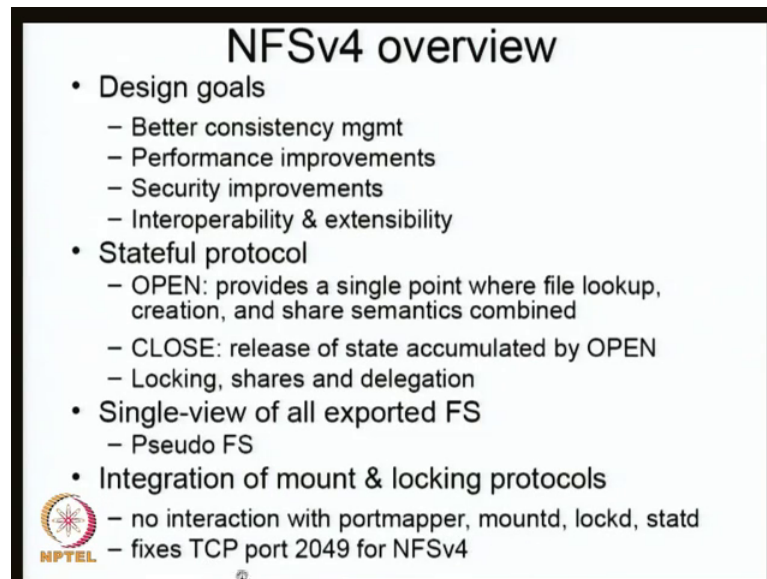
Other thing what we can do is then any process does when you do a close. Basically, there is a; you can do what is called you can do synchronize the data of the server. So, that there is a; some got a blocking operation that takes place. So, that whatever data that you have on your system actually it can be synced to the server. Now, the other kinds of things that are added was something called a write verifier. What is the write verifier? It is some kind of idea that that machine has, every time it reboots it increments it id. So, anytime; that means, that it keeps track of the number of times that machine has rebooted in some sense. So, what the NFSv3 guaranties that server will not discard uncommitted data without rebooting. Either, it has rebooted or it has kept the data until it has got a commit ok.

Until this told commit you can keep it and only way it cannot do that if it is rebooted and the client can figure out that was rebooted because the write verifier has changed. So, in a sense the client now has some additional information that is I asked you to write something and if it so, happens that is rebooted the client can figure out that is rebooted. So, there is some kind of elementary guarantee that is available. So, there are additional enhancements, but basically NFSv3 is did not change too much. It has basically like to make things a bit more efficient. So, for example, one example in that category is REaddirPLUS. In NFSv2 REaddir only lists the files. So, example if I want to a less minus file. It has to first be a re directory it will get the list of files and for each file it has to go on say get me the attributes of each file.

So, this can introduce lots of back and forth requests and responses again it can search in the network. So, the idea here was to avoid this lot of chatter between server and the client, but it was to produce on this part three directory plus which actually in addition to providing the files it also provides you some of the attributes of each of the files. So, that common operations like LS minus file can be handled more effectively. So, there is also other extensions like exclusive mode create request. Again as I mentioned before you cannot provide atomic operations in the case of NFSv2 because each request can span multiple net network requests and therefore, it is possible that the kind of atomicity you are looking for generally possible. For example, in Posix there are certain exclusive mode create models of available.


And the idea is to see if you can do it also in the case of in the network environment with NFSv3. So, there is an additional models are created. So, there are various additional enhancement that for made, but fundamental to the model is still the same as NFSv2. So, this was the situation for quite some time and once windows systems became very dominant. There was some need to support the models of the Microsoft windows systems and basically at this time there was a substantial modifications that was take undertaken with NFS and this NFS before essentially is a rework model of NFS taking into account 2 main issues.

(Refer Slide Time: 21:40)



NFSv4 overview

- Design goals
 - Better consistency mgmt
 - Performance improvements
 - Security improvements
 - Interoperability & extensibility
- Stateful protocol
 - OPEN: provides a single point where file lookup, creation, and share semantics combined
 - CLOSE: release of state accumulated by OPEN
 - Locking, shares and delegation
- Single-view of all exported FS
 - Pseudo FS
- Integration of mount & locking protocols
 - no interaction with portmapper, mountd, lockd, statd
 - fixes TCP port 2049 for NFSv4

 NPTEL

One is that of making sure that there is better consistency managed make one making it will closer in the kind of knowledge that is done with window sold. So, that both UNIX and windows systems can be clients and also provide better security models. So, there are various aspects that have been made and fundamental one major thing that has taken place in NFSv4 is to go for what you might call a stateful protocol. This is quite different from what is there in an NFSv2 and v3 and again if you look at the windows system they also had a most stateful protocol and so once NFSv4 going to a most stateful protocol. It is possible to support the windows kind of semantics and for the some reason it is also much more complex. The only plus point is that this reengineering was done in the late 90s, but this time a lot of experience with networking had already been accommodated. So, it was ease it was easier to design a much better system then was available previously.

So, sort a summation before the most important changes stateful protocol. For example, it has got something called a open call. What is that in the case of NFSv2, there is no open system call? There is no there is no open protocol, there is no open protocol message is meant this standard thing is to do a lookup and then we have to get what you do with there is you do a look up on a on a string of string that represent a file name and you will get a file handle and using the file handle you proceed and there is no such thing as an open call. Here, what it is you have open and this is where everything is taken care

of with respect to lookup, creation and what is called share semantics? I will come to that soon.

Basically in windows you have the notion of a share. The share is basically a way in which you can say that. I am accessing that file and I don't want anybody else to be able to look at it till I am done. So, I can essentially have a work process lock and till that particular system is alive. So, till that particular process is alive it has got exclusive control over what happens to that particular file because there is something which is there in the windows word and windows word called the share semantics and that is what there was incorporated here. So, let me close you can if you have accumulated a lot of state you can ensure that you release the state through various flushing mechanisms. So, that the state that is across the servers and clients in the crowd they are somehow made more let us say harmonized. So, again what are the various models we can have locking at a high level?

We can do it at the process level or we can do it at the client level. So, there are three models that are available NFSv4. So, locking is the regular locking that UNIX also provides which is basically at the file level. That means, that I can have a file open for read and nobody else with if I lock it in the read mode nobody else will be able to write it. I can at the same time I can, I will have multiple parties to read it. But if this file lock it in the right mode I am only the part the party which clocked it in the right mode only has exclusive access and nobody else can write it. Similarly, in the case of I already mentioned about shapes about delegation basically it turns out that oftentimes a client can take responsibility for a particular file and completely and that means, that it manages all aspects relating to a particular file.

So, the idea basically is that a server can be supporting multiple different activities and these activities could be most likely disjoint. If they are disjoint then it does not matter if multiple clients are accessing this particular file server. All that we have to ensure is that somehow each client accessing a portion on the particular files exported by the server and if that is done then it is sort of delegated it makes it each of those particular files each of those clients which need that. Then it is possible that the client does not have to send any network messages to with respect to consistency. So, certain groups of files can managed by one the writing or reading whatever and that party is going on want to look at it. So, if I have a delegation for that part that portion of the file system they essentially

do not have to actually be they do not have to that client does not has does not have to send any messages to the server to check if number has modified or not that basically delegation.

So, so this things is also have been provided and for providing these things blocking shares and delegation you need to have a stateful protocol. Server has to keep track of the state there is some state that is kept by the client as in the previous cases NFSv2 and v3. But there is now state that this NFS were also has to keep track of especially what all delegations have been done what shares are the existing in addition to what locks has been taken what have been taken by various tracks.

So, all these three things have to be done and there has been the because of the state there has been accumulated. It turns out that if there is a failure they can failure some multiple reasons. It can be failures because of client crashing or the client or the client crashing server crashing or that could be network partitions because of all three possibilities the stateful protocol is going to be complicated. It essentially I figure out how to manage the recovery process in when some of the scene happens. So, so there is lot of complexity and if you look at the specification of NFSv4 it is quite long and it discusses all these issues what happens when the server dies, when the client dies what not. Other thing that has to be done, so that has been done in the case NFSv4 is that it gives a single view of all exported file systems.

Essentially, it provides to each client a single of all the things that it can access. In addition many times what happens is that you have on a server you wanted you want certain parts of the file system to be exported. For example, you may be interested in looking like slash a, slash b and you also might be interested in slash a, slash b, slash c and slash d. Now, I want only access to slash a, slash b or slash a, slash b, slash c, slash d. Now, in previous cases in the previous NFS notions you need to have 2 mounts and both these mounts had to be handled separately. Here it comes out that because of this single view they close out had a uniform view of what the server is exporting and it also when it even when you are doing the path traversal it comes out it figures out that you are going through.

For example, some little maybe at nodes which are not reduced supposed to be seen by the client. So, all those things are done by NFSv4, the other aspect that NFSv4 does is integration of mountain locking protocols. As I discussed before it turns out that NFSv2

has to interact with other infrastructure like Portmapper, mount d, lock d, stat d and they were complicated for multiple reasons. There could be security holes in each of these programs that is one possibility. Suppose that there could be different model security programs because there are different programs. So, there could be firewall issues. And so, various issues on this kind kept cropping up again and again. So, finally, NFSv4 decided anywhere you going for stateful model and you have to make sure that they are interoperate between multiple systems which might have different mode of the security, etcetera.

We may need to have some uniform way of handing this and therefore, they eliminated all these things and for example, in the NFSv4 there is a port number 2049 which is fixed for NFSv4. It is only TCP this may be and because there is no UDP some other issues with respect to UDP is transfers things or have gone away. So, essentially TCP's as we have discussed before has some condition mechanisms condition algorithm mechanisms and therefore, it is a slightly more stable protocol and because it using TCP. Now, NFSv4 can now be used across intermediate in case it is feasible.

And the issue there in why of course, is become feasible is because they are much more efficient computation of TCP nowadays compared to all was there in the 80s. And so, TCP is no longer considered to heat protocol it has been optimized fairly well. So, its performance is not any inferior to unity protocols nowadays even though it does quite a bit more like condition avoidance.

(Refer Slide Time: 31:59)

NFSv4 changes

- COMPOUND procedure
 - Non-atomic operations
 - Error handling
- LOOKUP semantics
 - Multicomponent; traversal across multiple fs
- REaddir semantics
 - Can specify how much data can be sent per transaction
- Named attributes
 - OPENATTR procedure
- FS migration & replication
 - *fs_locations* file attribute
- Better cache consistency models but complex

So, let us look at some other major changes that are taking place in NFSv4. First of all it has got something called a compound procedure. If you look at a NFSv2, it there could RPC look at the remote procedure calls and for I try to be one RPC is at a time. So, if you wanted to make some complex atomic change on the file system you have to send multiple RPC's and because there are multiple RPC's one could not be sure that actually it could be done atomically because it is sent across multiple interactions.

So, this was changed in NFSv4. So, that you have what you call composure. So, that you can now provide a all the operations in one single package in some sense and send it across and the business of the server to ensure that all these things are done if possible atomically. Now, it is essentially the ball is now in the service code, it is not a lot of interaction between client and server or a specifying multiple operations. Of course, this introduces some complexities. The question is that of for your handle previous in the case of RPC if you send some requests you will get another you will get some response thing without that part of it settled or not and you have to figure out what to do with it.

Here now what is happening is that you have multiple operations in being sent out. So, you need to get some idea about vary field. For example, if you have ten operations and it fed on a third one you need to go tell very clearly that I did first 2 of them was somewhat, somewhere in, somewhere in the midway third somewhere and dead here whatever. I had to give some additional information. So, that if in case this one client had to be the picture had to get exact information. So, that some clerk and I handle it could be attempted. So; that means, at this part of has become much more complex. The other aspect which had changed also, here is look at look of semantics. One problem with the NFSv2 was that it when you traverse the file system, once you get a file handle you have to do it one component data. For example, if I wanted slash a, slash b, slash c. slash d. I first get a file handle first slash.

Then I look up a then I look up b then I look and it is if it is then basically it was that I have four or five round trips and that means, you have a lot of little scene here . So, this one in NFSv4 they made it multi component. So, that you can send a lookup or slash ABCD and it will give you the file handle correspondent at in one short. So, there are some other mode that changes that have been made. As I mentioned before you can also traverse across multiple file systems. Basically, the client has a notion about various file

systems it has mounted it can actually mount multiple different file systems and they could all be on the same server.

And then as you are traversing it if you are traversing between one file system to another file system because they incorporate something called a file system Id in some other data structure. It turns out you can figure out that we are traversing different file systems and NFSv4 can actually track all these changes and it can provide the right semantics with respect to look up. There are some other issues with respect to semantics in the case of re-directory semantics as I mentioned earlier NFSv2 you have to get the each file and ask for the attributes of each file monitor time. In NFSv3 made re-directory plus, the NFSv4 essentially it is the same as NFSv3 except that because of UDP operation where you have to send a lot of information example if you take a very large directory.

If you do LS minus L on it. It can have huge amounts of data that is coming already to you. So, the UDP sending such chose amounts of data it can create some difficult situations. So, NFSv4 is essentially has modified re-directories, re-direct semantics. So, that we can say how much did I actually you can send. Essentially some kind of flow control at the protocol level so that something else. So, you can see there are various changes in the NFSv4. So, it is a fairly substantial change from the previous mounts and it is v3. We essentially has something called named attributes that is you can because you are in cooperating with multiple different types of systems UNIX system, window systems and other systems.

You can you can discover what attributes are present and so, there is some the idea here is to be able to at one time figure out, the kind of objects that are there and use that to decide how to interact with the NFS server. So, for that they have a procedure called open attribute procedure by which you can get attributes and file at directory you can get directory and then use that information to figure out what to do with it certain things may be available some with may not be available. Similarly, because now the scale of the systems is becoming larger and larger they wanted to support things like file system migration replication. So, sometimes a file system is available in particular place and the system administrator can move it to some other place either for temporarily or if for permanently for reasons of efficiency or management etcetera.

Now, if you are written the application and it has certain model of where the data is present it is not a very good idea, but suppose it has that such a model. Then there is a

subsidiary protocol NFSv4. Now which you can figure out where are locations there is a there is something called FS locations and this file attribute will tell you very exactly the location of the file system is and then use that to actually make the right kind of accesses. Same thing with replication it is possible for you to come up with some way in which you can distribute the load also, again based on this kind of model. This kind of information as I mentioned earlier one of the major changes in NFSv4 has been, better cache consisting models that it is more complex. For example, we still have the same model that was there in NFSv3. For example, you can as I mention before both pre attribute and post attribute file can be are send to the client and you can check whether did has we considered.

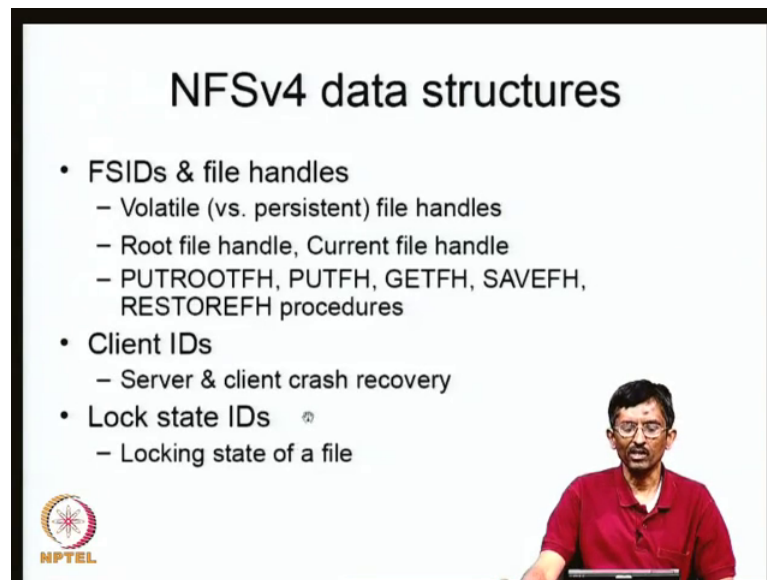
So, this is still there in NFSv4. But because you have models like delegation you are guaranteed that certain things cannot be change without your knowledge, you either you basically get exclusive access to something the only way you can fall apart is in case you have crashes on the system especially when you have network partitions. So, these are big issue that we are going to look at from soon. Basically our issue is that for cache consistency if network partitions take place then not much can be guaranteed. So, one thing that NFSv4 does to make sure that things are more reasonable is that it tries to be right through as far as possible. So, in the NFSv2 essentially is write back the caching is always write back; that means, that you can keep some information about what changed not reveal it to anybody till some other major till something interesting happens.

Whereas, as in the case of NFSv4 some parts of it of the system are made right through. So, that you can as things are changed you can get some information for example, if have a delegation; that means, that of a particular file and you are writing it. So, here what you do is you keep on sending some information like attribute information to server. For example, you could be sending that changing the size of the file. So, some parts of it can be made available in server. So, that is change with the other parties can actually find out that some which is taken these. So, in some places write trough is used. So, if you think about the carefully you will see that there are too many issues not being trying to be handled NFSv4 there are some things are like NFSv3 then it comes to pre attribute and post attribute.

And some models where there is write trough is being attempted there are issues with respect to partitions which are also expected to be handled. So, there are many things that

are coming in and it turns out that integrating all this thing non trivial. So, the model is very complex and so; that means, that some of you need to figure out a better systematic way for understanding while cash consistency or consistency models are not. So, easy to engineer in a system with networks which is what we look at.

(Refer Slide Time: 41:27)



The slide is titled "NFSv4 data structures" and contains the following bulleted list:

- FSIDs & file handles
 - Volatile (vs. persistent) file handles
 - Root file handle, Current file handle
 - PUTROOTFH, PUTFH, GETFH, SAVEFH, RESTOREFH procedures
- Client IDs
 - Server & client crash recovery
- Lock state IDs
 - Locking state of a file

In the bottom right corner of the slide, there is a small inset image of a man in a red shirt, and in the bottom left corner, there is the NPTEL logo.

So, again how does NFSv4 data structures look like basically we have file system identifiers? It basically identifies the kinds of file systems that are being exported and you also have a file handle just like an NFSv2 and v3. I think as far as we discussed earlier in a Posix we have file descriptors to NFSv4 file handles. Now in NFSv2 and v3 file handles were persistent; that means, that once if somebody gives a file handle.

Then server is bound to respect the file handle any time anybody provides it one million years later they are supposed to produce the data, which is a bit of a let us say too much to expect. So, finally, NFSv4 I decided that it is no point in the having the fiction that the file handle is you know it has to be permanent and they has to support permanently. So, they have come up with something that volatile file handles. So, it helps servers which do not want to provide permanent file handle is could also be a security issue. For example, I have some I give some access to a particular file for some time and if I make it volatile it is guaranteed that I can drop it without because something has changed I can, I do not have to honour a particular file handling that has been produced to me that is given to again.

So, that is one small change it also has got something called a root file handle. Basically, NFSv4 has a notion of a current file handle. So, whenever it is doing any file traverse. So, it always does it with respect to its current file handle and again this is possible because NFSv4 is stateful. So, it can keep track of something called current file handle. So, the way you start in NFSv4 in the previous case NFSv2 and v3 you have to use a mount protocol, mounting protocol you get a file handle. You basically send a string of a file name, send it to look up and then finally, you create the file handle. Here what you can do is you can start with the root file handle.

So, you can essentially say root file handle and then you it basically gives you the file handle server into a; some buffer that you have specified and that is essentially gives you the root file handle. Using that you can traverse the file system using lookup calls and then you can save the file handle, you can also once you have saved it someplace can get it back. Again all this is because it is a stateful model all this can be done on the server. Save file handle, restore file handle all these things were again possible right now. To ensure that you can do server and client crash recovery there is a notion of a set client Id. So, now what is this it means that different clients will have different client Ids, not only based on a machine it also it can be based on the specific real carnation of the particular client.

What it means is that the client also can live under. So, every time it comes up it keeps track of the; which version of the client which life of the client we are talking about. So, that it can describe between different lies of the client. So, if the client dies sand comes up it will have a different client id. So, that you can discriminate between various types information for client similarly for locking state you keep track of the client information. So, that if have to be any recovery it is possible for me to figure out who has the current information who does not have the current information updated appropriately. So, then all these things we can see it is because it is not removing a stateful model and so, it is going to be slightly more complex.

(Refer Slide Time: 45:42)



The slide is titled "NFSv4 leases" and contains a bulleted list of topics. A presenter in a red shirt is visible in the bottom right corner of the slide frame. The NPTEL logo is in the bottom left corner.

NFSv4 leases

- Lease semantics
 - Crash recovery operations
- Locking
 - CIFS-like share reservations
 - Lock request sequence IDs
- Delegation
 - Attribute & data caches
 - Open, close, data access, locking
 - Revocation operations via callbacks
 - Fallback to close-to-open consistency

One major thing that NFSv4 has done is using what is called leases. What is a lease? A lease is a way of saying that you have permission to use this particular file exclusively for some amount of time. It can be through a delegation lease or it could be through other kinds of models of sharing and the idea here is that.

If for example, the client dies then a then our lease the server is free to take back essentially takes back the lease and do it anybody else. So, so the recovery operations becomes very simple, but this also means that there has to be because it is not based on time there has to be lock synchronization with an important part of the system the without lock synchronization it does not work. Now, you know that lock synchronization is a complicated procedure and typically you do not get very high accuracy in terms of lock synchronization its usually few hundreds of milliseconds. Therefore, there could be still a big consistency window even if you use this semantics. As I have mentioned earlier that you have locking at various levels you have UNIX like locking where you do it per operation or it can do it at the level of per process.

Like what windows has CIFS like shares reservations or you can have it at the client level the delegations I mentioned before which we will talk about next. So, basically in delegations you are allowing a particular client to manage a file. So, for example, let us take the case of locking there could be multiple processes in the particular client and once the file is delegated to that client any process on that client it does not have to go through the server. So, in a sense it reduces the amount of network traffic all is there

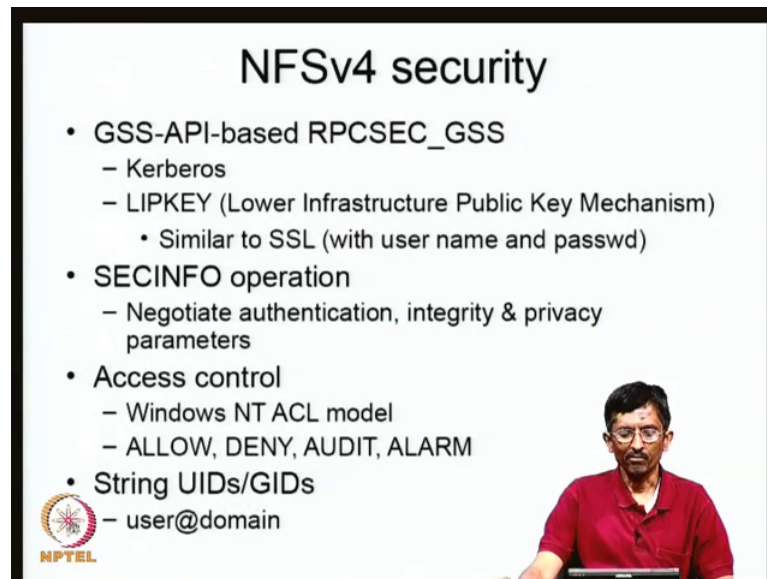
similarly when there is a question of closing let us say the file there is no need to talk to the server.

You actually talk locally to the client implementation NFSv4 and only when the delegation is taken of then you have to flush on the dirty data to the server. Now, there are some issues here basically if when I am trying to delegate etcetera you can also take back the delegation. For example, one client requires something to modify some files and some other client also can come and say it wants the file its possible for you could get back or call back the delegation. But it is very critical now that the network is not partitioned. So, there are various in all these operations anytime what happens is that the before delegation is given the first check whether you have firewalls and other things allow you to do a call back you need to check on these things.

Basically you can sometimes there are a symmetric access patterns you can send out that may not will receive it. So, because of the reasons delegations etcetera are carefully defined. So, that the very first thing that you do in a delegation is to check whether you can do a call back if you cannot do a call back then there delegation is not allowed to happened. So, a lot of interesting things that happen delegation mostly because partitions firewalls all those things also can create some complications and these things are defined in the in the standard in some detail . So, as I mentioned earlier you have this revocations possible via call backs and the most important thing of NFSv4 is that it has what is called close to upon consistency. That means, if you have a delegation and only when you close it the other people see the modifies you are made. This similar to what is there in windows world this is what is called close to open constituency and this is what is provided in NFSv4.


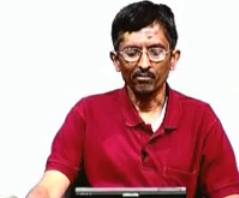
So, this is the only thing that you can assume. So, the model of consistency in NFSv4 is slightly complex this is got too many let us say ways of handling it and therefore, you had in very careful when you start the using NFSv4. It is better mostly, but you have to know the details.

(Refer Slide Time: 50:32)



NFSv4 security

- GSS-API-based RPCSEC_GSS
 - Kerberos
 - LIPKEY (Lower Infrastructure Public Key Mechanism)
 - Similar to SSL (with user name and passwd)
- SECINFO operation
 - Negotiate authentication, integrity & privacy parameters
- Access control
 - Windows NT ACL model
 - ALLOW, DENY, AUDIT, ALARM
- String UIDs/GIDs
 - user@domain

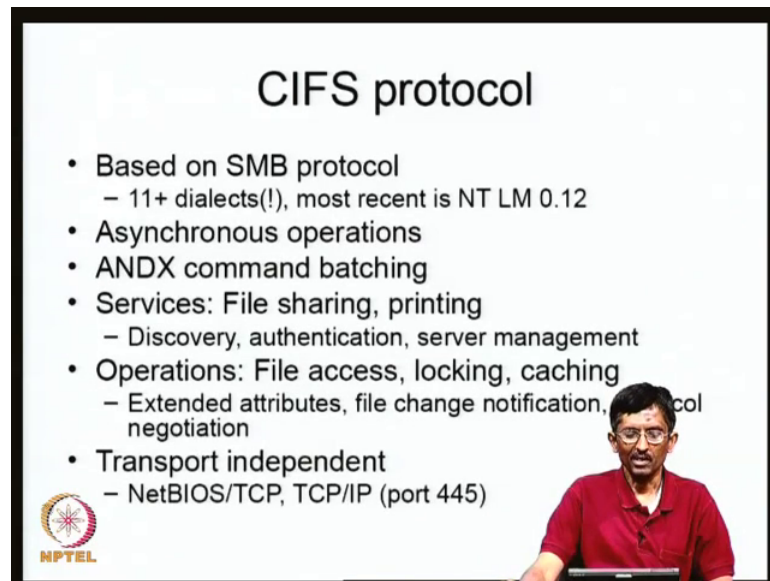
 

Again with respect to security they have thing made it much tighter than better. So, one is Kerberos you just also there in I think in NFSv3, but they provided another model which is called LIPKEY and this similar to your SSL model. I think SSL I think all of most of us are familiar with basically sure you have the server has the certificate, but the client does not have certificates and the clients authenticate themselves through username and password.

Because usually clients going to have certificates. So, NFSv4 also has provided something similar. So, it is based on a modern like SSL. So, that you can you won't access it you can give name and password and that allows you access into the system. There is also some other types of models by which you can negotiate authentication integrity and privacy parameters there is some particular aspects to the protocol. The access control is closer to the Windows NT model this is access controls model what is this somewhat different from UNIX. UNIX has this RWX model basically you have only categorizes that three parts user, group and others. This one is slightly more elaborate and so, this is model that has been evaporate.

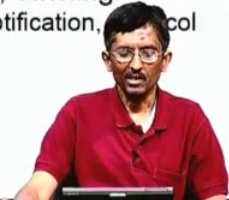

Again there is some minor additional things like string UIDs and GIDs.

(Refer Slide Time: 52:03)



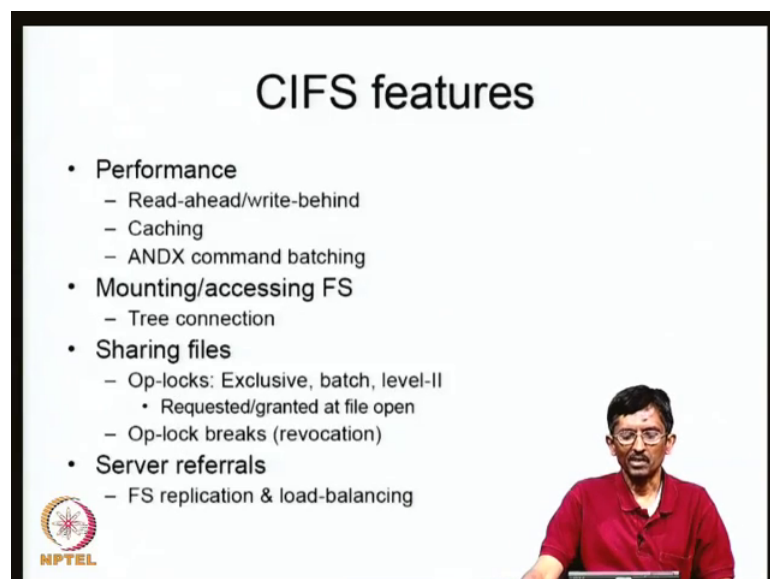
CIFS protocol

- Based on SMB protocol
 - 11+ dialects(!), most recent is NT LM 0.12
- Asynchronous operations
- ANDX command batching
- Services: File sharing, printing
 - Discovery, authentication, server management
- Operations: File access, locking, caching
 - Extended attributes, file change notification, protocol negotiation
- Transport independent
 - NetBIOS/TCP, TCP/IP (port 445)



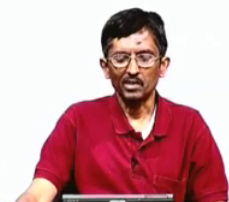

I will briefly mention this is protocol, I just highlight some of the important aspects is that asynchronous operations just similar to what is there in what we discussed now NFSv4. It has got command batching is just like the compound operations and it has got let us say some of the similar service that the NFSv4 and so, provides things like file access, locking and caching. Aspects, they are similar to what is now there in NFSv4 and I am not going to go in too much about these things.

(Refer Slide Time: 52:45)



CIFS features

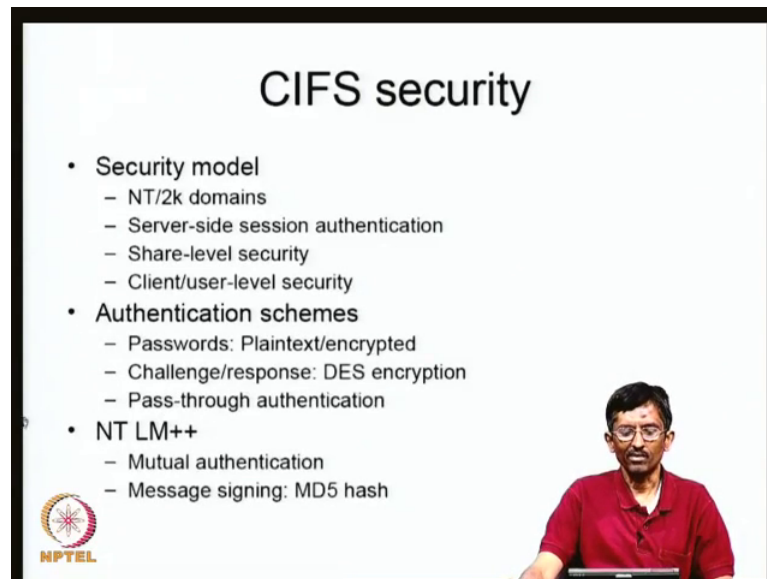
- Performance
 - Read-ahead/write-behind
 - Caching
 - ANDX command batching
- Mounting/accessing FS
 - Tree connection
- Sharing files
 - Op-locks: Exclusive, batch, level-II
 - Requested/granted at file open
 - Op-lock breaks (revocation)
- Server referrals
 - FS replication & load-balancing



And they also has similar ideas about performs enhancements read ahead and the write behind and when it comes to sharing files also. There are the locks are multiple it has are locks are there.


And similar to what is there NFSv4 also you have you can handle file system replication load balancing.


(Refer Slide Time: 53:10)



CIFS security

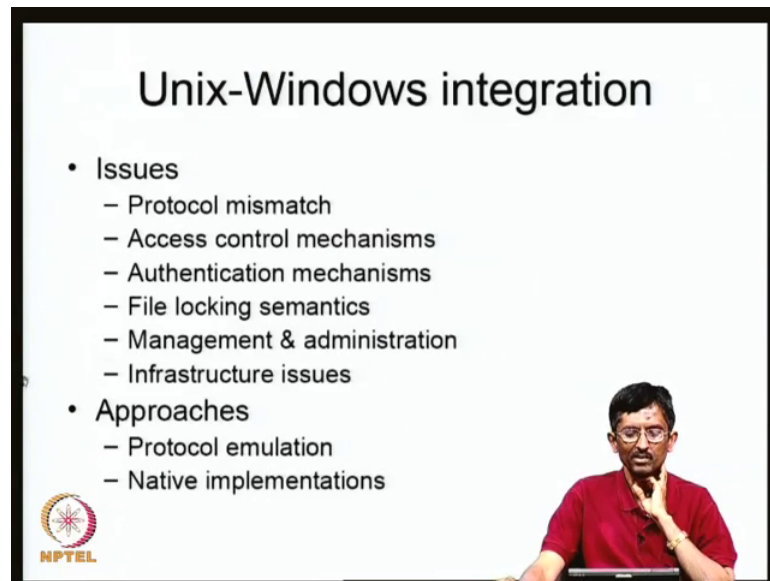
- **Security model**
 - NT/2k domains
 - Server-side session authentication
 - Share-level security
 - Client/user-level security
- **Authentication schemes**
 - Passwords: Plaintext/encrypted
 - Challenge/response: DES encryption
 - Pass-through authentication
- **NT LM++**
 - Mutual authentication
 - Message signing: MD5 hash

 NPTEL



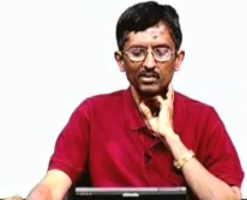

The security model is based something very what is there windows NT something called domains and it also has some authentication schemes like for example, change the sponsor models and again I am not going to go too much into it here.

(Refer Slide Time: 53:33)



Unix-Windows integration

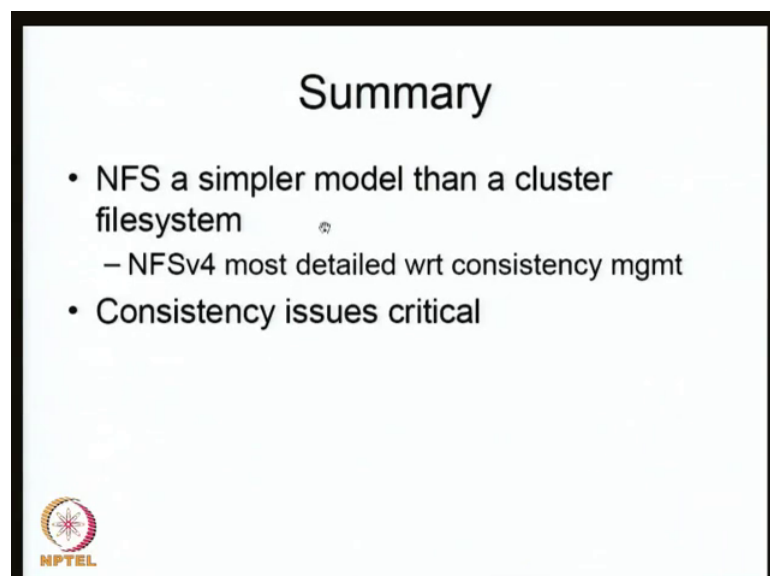
- Issues
 - Protocol mismatch
 - Access control mechanisms
 - Authentication mechanisms
 - File locking semantics
 - Management & administration
 - Infrastructure issues
- Approaches
 - Protocol emulation
 - Native implementations



So, when you integrate windows and UNIX there are a lot of issues there are lot of issues with respect to access control mechanisms and consistency management, file locking semantics etcetera. But if you look at NFSv4 most of the things have been harmonized to quite a bit extend. So, now the integration between UNIX and windows is much better, then it was with NFSv3 and CIFS it was very difficult. Before has actually done quite a bit of managing this discrepancy between the 2 systems.


And so, now it is easy now if you support NFSv4 essentially support both CIFS clients as well as UNIX client is the too much difficult.

(Refer Slide Time: 54:16)



Summary

- NFS a simpler model than a cluster filesystem
 - NFSv4 most detailed wrt consistency mgmt
- Consistency issues critical



So, I think I would like to conclude today by saying that NFS is a much simpler model than if you want to if you were to go for a cluster file system which is basically saying you mean Posix semantics across all the nodes. This comes out to a very hard problem and NFS started as the; a good approximation to this Posix model and it has become like a more over better and better. But there is an inherent complexity and difficulty of providing a cluster file system model over unreliable systems either networks or machines.

These are things which is a big issue that we want to study later. We would from next class onwards. We will start looking at why this issue of giving a let us say a proper semantics and distribution system is. So, difficult we will look at that part and. So, again the many point is I want to discuss again is that if you are using NFS be clear about the consistency model, that you are using. This can be a surprise and oftentimes many people are surprised by getting what is called a stale file handle because they don't know exactly how systems are behaving with respect to deletions or modifications by other parties other than the user. So, again we will in the next class; we will start looking at issues relating to distributed systems and consistency in models.

Thank you.