Storage Systems

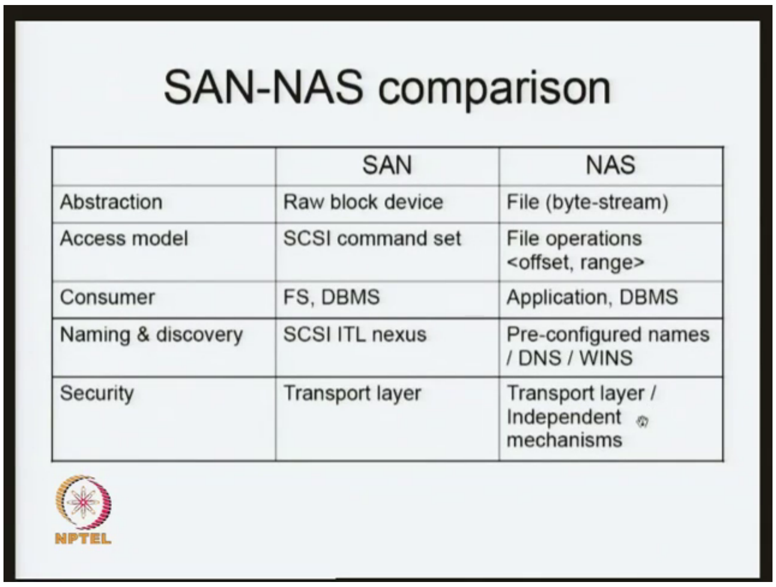
Dr. K. Gopinath
Department of Computer Science and Engineering
Indian Institute of Science, Bangalore

Communication Protocols for Networked Storage Systems
Lecture - 09
Overview of SAN/NAS Networked Storage System Models, SAN/NAC/ClusterFS, NAS implementation, NFS Architecture and Protocol, RPC/XDR services in NFS, NFSv2 implementation, NFSv2 Problems

Welcome again to the NPTEL course on Storage Systems. And in the previous classes what we are doing is to start from let us say the lower device level and we were trying to move upwards. I mentioned earlier that we can become a very high level extremely crude characterization as devices protocols and applications. So, we had a look at the devices in bit then we introduced this SCSI protocol and then if you have multiple devices to be accessible. We try to understand how to use SCSI to do it.

So, we will discussed a bit about the fibre channel protocol and how it can be used to designer to use it inside. Today what like what I would like to do is to slightly go up the chain; we start looking at how files can be served across let us say multiple devices across the multiple systems. So, today we will talk about some aspects of network file systems.

(Refer Slide Time: 01:39)



SAN-NAS comparison

	SAN	NAS
Abstraction	Raw block device	File (byte-stream)
Access model	SCSI command set	File operations <offset, range>
Consumer	FS, DBMS	Application, DBMS
Naming & discovery	SCSI ITL nexus	Pre-configured names / DNS / WINS
Security	Transport layer	Transport layer / Independent mechanisms

Again let us make a SAN and comparison between the two ways of thinking about the problem. The one is the approach what is called the storage area network approach and the other one is the network attached storage approach. In the SAN approach basically the abstraction is a raw block device. So, because it is operating only at the raw block device it means that we can only refer to block with numbers; that means, that there is no user let us say understandable name for this storage device storage.

Whereas, in a network attached storage we are now is stop talking about raw blocks. We are starting to talk about in terms of files. And the file abstraction is well known, because it is well known is that you more convenient. In a sense NAS is easier to for the user SAN is a bit more, let us say a bit too difficult to use as of now. Now, the access model is for the SAN is SCSI command set and we looked at some of this sometime back, whereas for NAS it is going to be file operations.

The only issue is this as we discussed early in the very beginning classes this file operations are now going across the network and the cause they are going across a network a kind of semantics that was possible in a single system may not be possible here. It is going to go more complicated. Essentially a lot of consistency to show up and so it is a more tricky thing to provide this kind of capability.

A consumer in a storage area network is typically either a file system or a database and, whereas in the case of a network attached storage usually most applications are written assuming file abstraction. Therefore, applications for them it is quite appropriate. Whereas, like other possibility is database also might use a NAS because database also might find easier to manage things if it goes through various file abstractions or tools available for managing file systems. That is why NAS is somewhat more popular because it is easier to manage. Whereas, for SAN you need to have specific tools which are highly specific with respect to the type of sign we have.

Again when it comes to naming the flows for example, or you need to have in SCSI as I mentioned it is going to be a initiator, target, and the logical unit. So, these are the thing that is going to be used for connecting up two entities that are involved in storage transfers. Whereas, in the case of NAS we are operating at the level of let us say some file systems are getting exported and they are being let us say imported into some file system on the client side and so they need to have some way in which you can refer to those file systems are having exported to the server.

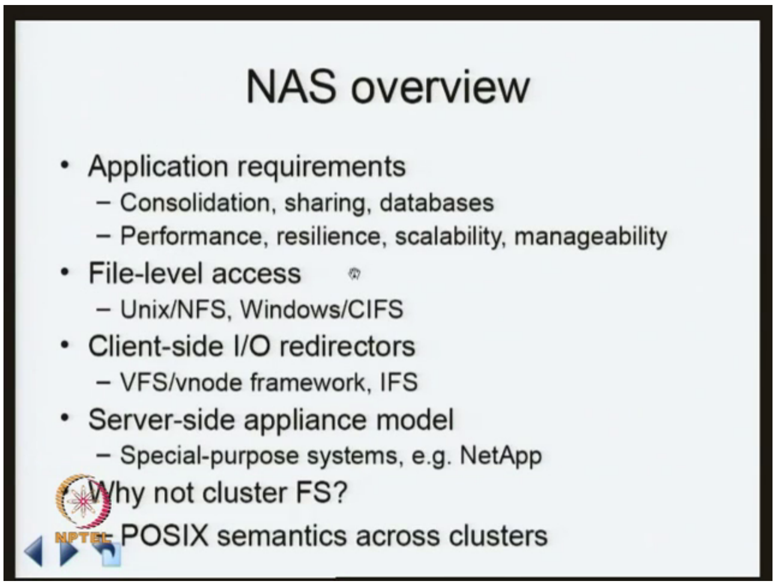
And normally we will use IP based methods; for example, DNS to figure out how to talk about a server etcetera. In the windows world they use a slightly different model with in smaller for name lookup etcetera. In for scientifically as we are mentioned earlier SAN devices were designed assuming that its operating in a data centre and therefore, the security has been relegated to the transport layer. There is not serious security at any of the lower levels; it is all of the higher levels.

In the NAS case we are going to have at both that in the transport level as well as at the file system level. Now, whatever file system provides you that also is possible and it also is possible at slightly different levels. For example, typically when you are making storage requests they might go through some infrastructure called remote procedure called RPC. It might have its own methods and that might also be an issue. So, we have to look at the security could be given by that particular aspect and I have an important issue I think which is not listed here specifically because we will going to detail much more in some depth the consistency issues.

As I discussed it in the previous class storage area network at you know the simplest form or its most basic form assumes that there is only one party which is accessing it, whereas since it with an NAS you are at the file system level. Usually file systems especially UNIX variety or the more including the windows variety also. They have a notion of multiple users. And therefore, there is already some infrastructure for making sure that there is some security of one user from another with respect to let us say readability or write ability. So, in a sense the NAS basically excels with respect to providing some model about sharing. SAN actually it has to be retrofitted on top of it or some applications out agree between themselves how they going to do it so.

So, there are other differences, but I think I listed some of the more important ones.

(Refer Slide Time: 07:43)



NAS overview

- Application requirements
  - Consolidation, sharing, databases
  - Performance, resilience, scalability, manageability
- File-level access
  - Unix/NFS, Windows/CIFS
- Client-side I/O redirectors
  - VFS/vnode framework, IFS
- Server-side appliance model
  - Special-purpose systems, e.g. NetApp
- Why not cluster FS?
  - POSIX semantics across clusters

So, if I look at NAS why are they required I think basically because of multiple reasons. As I mentioned more than reason is for sharing purposes because the file systems in the picture therefore, sharing is much more easy. It is for consolidation basically because you can have multiple file systems on a server and you can put all of them together in one place and I mentioned already that databases might find using the file system as a way to manage its sources much more easier than otherwise and the other requirements like performance, resilience, scalability, manageability.

All of them maybe slightly easier and sometimes and slightly more difficult because of the kind of file model that we going to use because if you take the file system model. It might have its notions of how to provide a performance or resilience which may not be the right model if you integrate multiple things. For example, if you take a single file system the performance is improved by caching etcetera whereas when you put multiple file systems together, multiple nodes together. Then it may be that the caching entails some aspects of consistency which may be become problematic. So, there are issues that we have to resolve. So, fundamentally as we discussed before a NAS we have file level access, its not block level access. In UNIX it is NFS, in window it is something called CIFS.

So, basically what happens is on the client side we have what are called I/O redirectors. What are these I/O redirectors? They basically take the typical model in UNIX is that they can have multiple file systems on the client and each client each file system is abstracted by the kernel so that the kernel only needs to talk about the virtual file system operations. So, the kernel interacts with all the file systems through what is called a file system independent layer which is called the VFS virtual file system and similarly each metadata that a file system keeps for a file what is called Inode. The kernel keeps something called the Vnode as its representative. So, that it can deal with all the different files in different file systems in uniform manner.

So, UNIX has this notion of VFS slash Vnode framework. So, basically when a request comes in from a application on a client it will do something like a reader or write call. It first navigates through the file system independent layer. The VFS Vnode layer and that is in the client and that has to be now transmitted to the other side the server side where it actually meets the real file system. So, basically we have the VFS Vnode framework that basically used as a way to interact with the remote file system. So, that is what I am calling redirector because it is from the client side the request actually has to be redirected to the actual file system on the server side. They are also in the windows world the symbolic installable file systems there are similar to VFS Vnode framework to make its like independent of the file systems they also have similar framework.

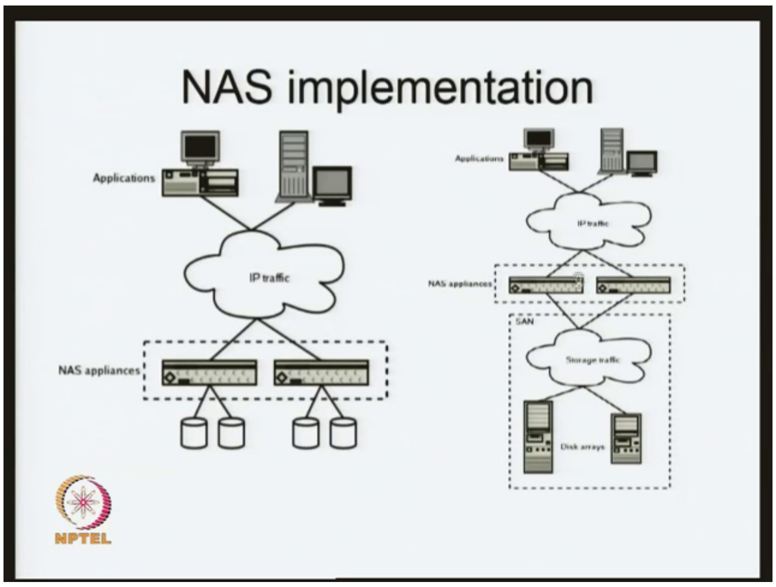
Now, on the server side it turns out that since you need high performance. It is often the case that is using general generic process generic machines for serving files you might also have specific appliances and which actually do a very good or which give very high throughput. So, that is companies are specialized in this NetApp is a good example of company which serves, which makes lots of high end network attached storage devices.

Now, before I conclude this particular slide let me just mention why we are not talking correctly of cluster file systems. Now what is the cluster file system? Typically what we refer to the cluster file system is that you are used to the POSIX semantics on a single desktop on a single node. So, the POSIX semantics these are able to provide across multiple nodes then that will be called a cluster file system.

So, now this turns out this ability to provide POSIX semantics across cluster because the POSIX semantics is slightly complicated especially with respect to failures, how to handle failures in the right because if I tried to provide POSIX semantics in the context of failures. There is going to be a lot of state that is going to be kept around and they had to be recovered when failures occur and that is not easy right. So, we will actually start looking into it more systematically. I will today introduce the NFS part; I will just briefly show some other kind of problems that are there with respect to consistency in this particular session.

But in the next classes will start really looking at what is this consistent problem. Why is it so hard and what are the things that we can do to handle problems. So, today we will just I will just indicate some of the problems, but there are deeper issues which crop up which has to be resolved. So, there are cluster file systems available, but they tend to be more difficult to engineer, more difficult to use also because having getting POSIX semantics across clusters is not easy.

(Refer Slide Time: 14:07)



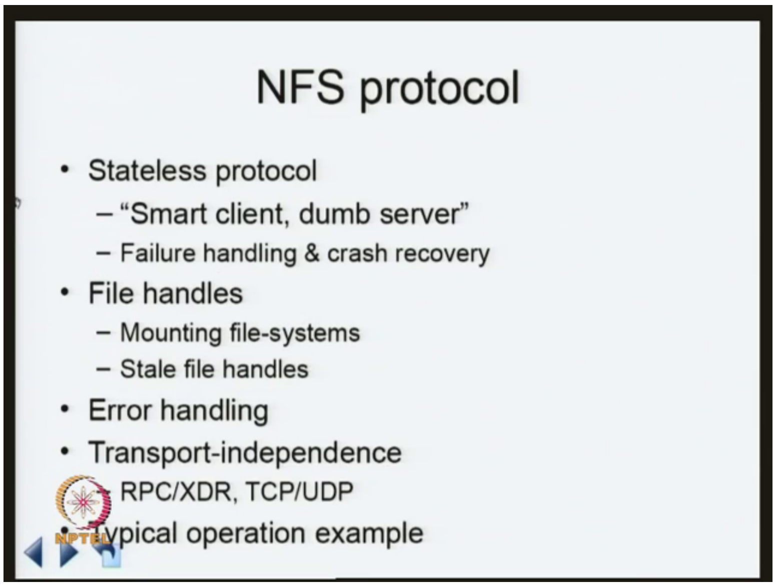
NAS implementation

So, let us look quickly at the kinds of implementation that are possible. So, normally what a simple NAS implementation will be that applications are out here and they client in a NAS clients on this both sides, NFS clients on both sides and they send the request through the IP traffic and there are NAS appliances or devices or servers which actually directly access the file systems which are store in the in this disks.

So, this was simple model the slightly more involve model high end model will be that you start talking to this that we talks to a storage network. So, basically here what we are doing is we are sending IP traffic here. And here you might send from this NAS appliances or servers they are sending storage protocol traffic here. Now, this can be further channel, internet band or it could even be let us say Ethernet it is up to us to decide what we want to be here. So, for the point of applications they have no idea about how this is being done of that they are talking the NFS protocol or the system call this again.

So, in a sense SAN and NAS are not really they can quite complimentary, because this product is handling the file protocol part of it and this part of is handling is SCSI protocol part of it or the block access part of it. So, very high end systems typically are in the situation. They have both the SAN and the NAS. This is somewhat different from if you look at even larger systems. Now Google file system or it is a Facebook storage file system. They do not find this kind of models that particularly useful they want to have their own application semantics with respect to search or whatever that you know. And so they will not go through these kernel protocols then must still have something equivalent to some is similar to this, but they might not also again use the storage protocol per se. They might want they might be comfortable for cost reasons to use IP network everywhere.

(Refer Slide Time: 16:26)



NFS protocol

- Stateless protocol
  - "Smart client, dumb server"
  - Failure handling & crash recovery
- File handles
  - Mounting file-systems
  - Stale file handles
- Error handling
- Transport-independence
  - RPC/XDR, TCP/UDP
- Typical operation example

So, what is the basic premise of a NFS protocol? Basically our problem with the networks is that a networks are may be unreliable.

Because networks may be unreliable, there are some problems that show up. I send a message it may be that the message is received by the server and it also responds, but the response may be lost or it may be possible that I send a request the request itself is lost. So, and especially if you are think about a fairly complex network in between a flank and the server we cannot be sure where the how it can be engineered that you have the case that when I send a message it actually reaches there and the response actually reaches back to me. So, it is very difficult to engineer this and so it is somewhat easier to assume that I do not know anything about the intermediate network and then I put my all my smartness into the end devices.

So, what I will try to do is go with this particular model "Smart client and dumb server". So, basically because I cannot distinguish whether a message sent and lost is because the server is not working or the network as lost it or the response has not come back to me. I mean there are multiple possibilities and from the point of a view of the client is all the same does not matter how it happened and it has to still do recovery and in the right kind of recovery is that if it has information about all these things. Where exactly it happened? How it happened and you can do something about it if you do not know anything about it. The best thing is to say that it is too difficult to work with stateless protocols, best is to assume stateless protocols. Assume that the server cannot remember what was happening previously and somehow see if you can work around that model and that is how NFS actually started.

Because, if you remember NFS started about early 1980s and people tried to that time there was a 3 megabyte per second Ethernet and as extremely unreliable compared to our Ethernet right now. So, there were always having problems with file systems being consistent with respect to accessing some protocol is try to give POSIX semantics. Finally, people realize that providing POSIX semantics across multiple nodes is very difficult it is not for the trivial thing. So, some let us say careful engineers decided that you do not need to provide that kind of semantics we will provide something simpler and that is basically what is semantics. So, basically it have a stateless protocol failure handling and cache recovery will comeback simple. Why is it that simple? Because if failure handling happens I do not have to recover the state on this on the server and.

So, I can just when the server comes back up again they does not have to remember anything which was being before you just can comes up and the it does not have to do anything . So, that is one thing which is so NFS has become popular for that reason. But because it is not provide POSIX semantics the user has to be careful, you cannot use

NFS and assume that somehow the kind of applications that run on this single machine will actually run on. If you distribute you know that file systems on to multiple machines using NFS protocol that is not going to happen. So, you had a very careful about how you use NFS protocol

So, again because NFS protocol is based on this concept of dumb server. The idea here is that you have a nodes number is called a file handle. What is the file handle? It is the way in which the client can request a file. Normally what happens in UNIX systems is that I have a path name, I provide the path name and then I get the file and the thing about path name is it has to be passed multiple times. Basically, suppose I have a file name like ABCD how to pass a first go to the directory corresponding to the director slash get from it. Then again I have to go to the directory I have to go and look up b unit then I have to look up c unit. So, I have to do multiple directory traversal.

And so, it becomes a bit expensive to keep on doing it all the time. So, the idea here is that somehow the server will actually after the first access somehow it gives my file handle and the file until is used again to you access it the next times when I want to access a file.

So, that is what is file handle. So, its somewhere in some sense it is similar to a file descriptor on a single node system, but this is going to be across two nodes the flank and server nodes and one of the guarantees NFS makes is that once you are given a file handle. Even if the server crashes and comes up again the same file handle I can give you the same file. In a sense what the server has to do is to encode information in the file handle which is opaque to the client. But it is makes sense to the server. So, that it can look at the file handle and decide what file we are talking about. So, that is one the other thing is you have to also a mount file systems. So, essentially in some sense you have to be able to say that I need to incorporate some server file system as part of my tree structure on the client side.

So, what is the mount what is mounting a file system I have some tree of files on my client system and when I mount some remote file system. It has to decide at some mount point on my client system and this is particular mount semantics is there given for local file systems whether sustaining it across the network. So, once you have mounting of file system it is going to be at a particular place in the file system hierarchy and I can traverse they just like what going before now because there are multiple actors when we talk about NFS the client server also and multiple process is.

You also have some interesting problems what is called the stale file handle. What is this? Basically it may be it may be the case that I ask for a file I will get a file handle. Before I can use it surprisingly its get deleted by some other party and then that particular Inode gets reused by some other party. So, if it happens then if the server has encoded that information into a file handle then the client might get some file he did not intend to see. It is somebody else's file or somebody who deleted who happened to pick up the same equivalent Inode in some sense. So, you have to guard against this particular problem. So, the way it is done is that every Inode on the server side then it is reused again there is something called an incarnation number, which is incremented sorry it is called generation number sorry.

Is called a generation number is incremented; so any Inode that is used on the server side on the server side file system. Every, Inode has a generation number every time it is reused, the generation number is increased. So, as I mentioned earlier the servers has to encode some information in the file handle. So, that any time the client requires the file handle you should give this the file that the client expected. Now if you want to prevent some other file to be provided as the file in the client because of the file getting deleted and being used by somebody else you can put in to the generation number into the file handle. So, once you put in the file the generation number into the file handle the next time the client which innocently got a file handle and was planning to use it sometime later.

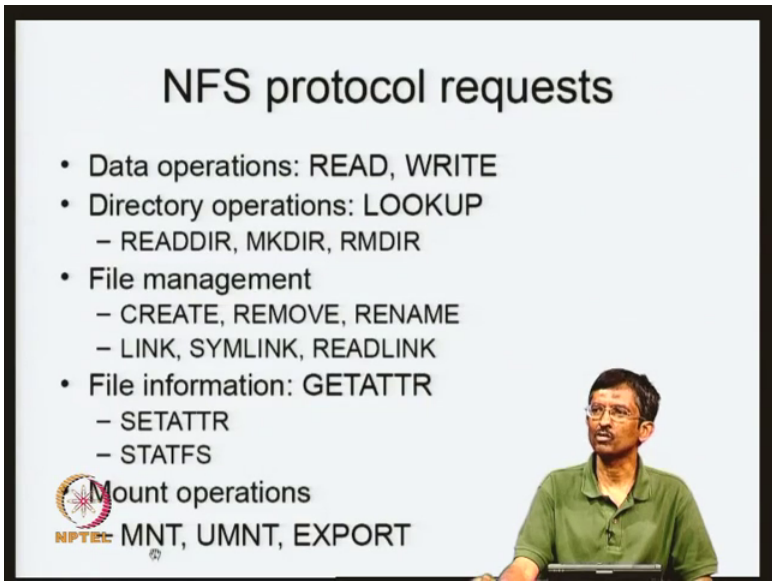
It will provide that thing basically what will happen is that the server looks of it sees that the file handle that has been provided to it has in generation number which is a different from the current generation number. Therefore, it knows that something has happened and the file system can tell the server file system can through the NFS tell the client inform the client. That you are referring to a file that is no longer available, its a stale file handle. So, that is the way it has to be done similarly there are some issues with respect error handling because the errors are the happening are happening on other side not on the client side. For example, there could be something like ENO space. What is ENO space? It is the way in which the file system tells the application that there is no available space ok.

Now, this is something which is happening on the other side its not happening on the local system. So, the minute we have divided the say separate to the client and server the kinds of things that you are assuming in the case of a single system no longer are valid and therefore, all this issues keep. Now NFS also provides what is called transport independence. That means, that it does not matter whether I am using TCP or UDP or some other protocol. And therefore, it has to also do some encoding etcetera and uses it uses something called remote procedure call which will come to which we will discuss. So, basically you have the way in which each of this NFS protocol requests can be mapped to certain procedures on the server side. These are what I call program numbers which the RPC in a sense every request is mapped to a particular program.

Program for the read or write etcetera is not a particular program number and in the server actually looks at that RTC request has come in and execute the corresponding request. that is how the NFS protocol request get mapped to RPC and there is a external data representation XDR. So, that whatever request you are making now can go across the network and not bother about whether which operating system, which there is a type of network that is being used, whether it is based on single endian machines or sorry big endian or little endian machines.

So, all those issues are taken care of by XDR. So, once you have this that means NFS now can operate between two different nodes and the nodes might have not much in common. For example, people have tried NFS between a UNIX machine and the PC, the PC client and NFS on a server; the server being on a UNIX machine. So, the lot of intricate things that has to be taken care of and that is why you need lot of infrastructure and that is it RPC, XDR and the transfer protocol etcetera.

(Refer Slide Time: 28:42)



NFS protocol requests

- Data operations: READ, WRITE
- Directory operations: LOOKUP
  - READDIR, MKDIR, RMDIR
- File management
  - CREATE, REMOVE, RENAME
  - LINK, SYMLINK, READLINK
- File information: GETATTR
  - SETATTR
  - STATFS
- Mount operations
  - MNT, UMNT, EXPORT

So, let just look at some other typical things that we do. So, what are the various protocol requests? First thing I have to mention is that NFS if it is stateless it does not have an open call. In standard PROSIX semantics before you can do something with a file you need to open the file get a file descriptor. Here what we do is we ask for a handle once you get a handle then we can do read and write operations the handle is got by using a lookup. So, you do a lookup on a path name you get a file handle.

Once you get a file handle then you can say read that particular file. As we are also discussed in a very beginning class I think read and write should have and offset it cannot go without offset, because in the case of a POSIX kind of system. The offset is maintained by the kernel it remembers implicit pointer. And therefore, you do not have to specify where you are reading it from at the offset by reading it from, but in the case of NFS this offset has this is basically mentioned. So, you have this lookup operation you have also re-directory operations which essentially gives you the list of files that come in. Now this actually creates certain complications basically because once you get a list of files I have to now iterate through all the files to get the metadata corresponding file. For example, suppose I want to do LS minus L.

On NFS base system first I had read directory there will give me the list of file names and then iterate over each of the file names asking for attributes like length of the file etcetera or the redirect permissions; so read directory is especially problematic thing in

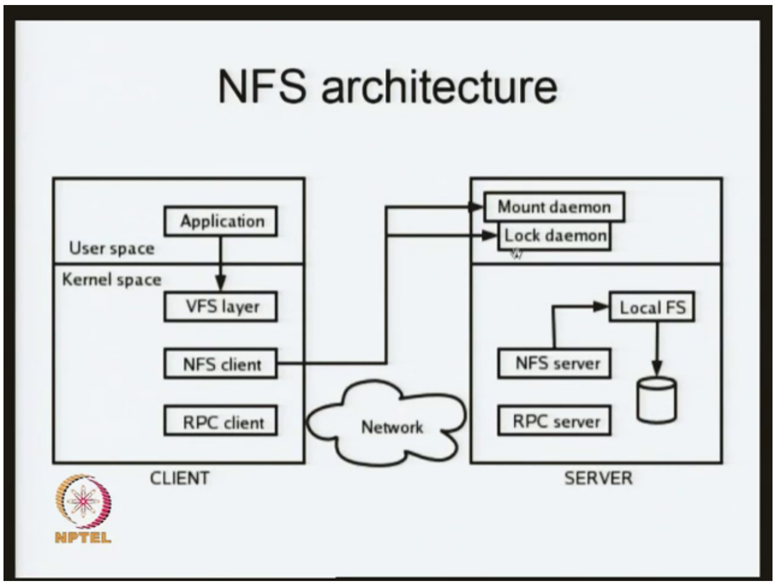
NFS when it was later changed in NFSv3: for example, to read director of plus for example. So, this is the primitive that was their NFSv2 version two. Similarly have make DIR remove DIR. So, there is file management create in a rename some of those things are essentially similar to what is there on similar to POSIX. We also have there are issues regarding some interesting issues with regarding some of these issues.

But some of this particular calls, but I will discuss it later you can also get information about the file getter tributes. This is very important system NFS call basically protocol request basically because if there are multiple parties who can be updating files. Then you may want to find out if my particular copy that I have kept from us anything caching. I want to know if this particular value is update is up to date the only way I can do it is to by sending get attribute.

So, the plant keeps sending get attribute calls to the server saying please tell me what is the current status of this particular file, because other parties could also be updating it. So, this get attribute tends to be heavily used and that is why if you look at any NFS protocol trace. You will see lot of get attributes the reason being that if there are multiple parties updating it every 30 seconds or 15 seconds depending on the system they will flush the tributes that have been cached. So, every so often you have to keep on getting requests the getting sending request to the server asking what is the current stage of this particular file.

So, similarly there is something called STATFS. So, that you can get information about the file system itself and there is this way get attribute you also have a corresponding set attribute. In case you want set some information on the file. Now you also have what are called mount operations and basically it turned out that the NFS. This separated the protocol in two parts the NFS protocol and the mount protocol that is what they decided design in the beginning. Later version of NFS they integrate all these things. So, basically in mount operations you have a you can specify what particular exported file system by the server you go to mounted into the client file system let us mount we mount and there is also a way in which you can say what file systems are exported.

(Refer Slide Time: 33:31)


NFS architecture

So, this is the high level NFS architecture. So, you will see that the application the user space is here and I mentioned that there is a file system independent layer. This is the NFS, the VFS layer. Now for a point of view of this application it is a there is NFS file system out layer. So, this VFS layer essentially we will talk to the NFS client and this NFS client in turn takes a request of the application and makes it into some messages to be sent with RPC client which goes across the network to the RPC server. And this RPC server in turn delivers the request to the NFS server which talks to the local file system talks to the local disk or whatever it is here and then provides a response back. So, you can see that there are multiple paths there are numb the numb the places where let us this can happen.

Once it may be that your failure is at VFS layer, NFS client layer, RPC layer, network layer, RPC server layer, NFS server anywhere they can be failures. So, if you are talking about the SCSI system with appropriate consistency mechanisms. The errors can be in so many different places that a stateless design is actually very useful if it is possible. It turns out that a completely stateless design is not advisable sometimes it is difficult, but it is worth doing it that is why to notice that NFSv2 was stateless, NFSv3 add some state, NFSv4 is fairly state full. So, it makes sense to keep certain types of state, because it helps you to actually loop things more efficient efficiently and then the various applications expected to be expected. In addition to this you see there are two different user processes on the server side in mount daemon and loch daemon.

So, the NFS client can request a particular file system to be mounted that energy goes to the mounting now and NFS client also if there are multiple parties, but updating this particular file program in that are not only the local file systems here, but there is 3 or 4 clients which are updating this particular local file system, you might require some locking mechanisms. And that is why that is where that is why we has something called a lock daemon and oftentimes this lock daemon is not while is used.

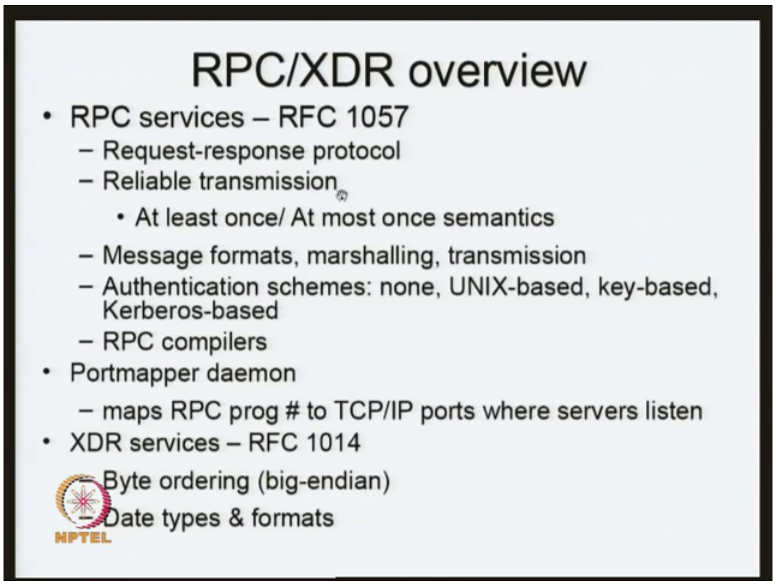
That means, that somebody else is taking care of it that no unusual. Let us say concurrent request state place. That is the responsibility at some other way now we will look at lock daemon etcetera and the issues with respect to consistency, issues with respect to failures. They are quite critical for the lock daemon and we will try to and understand this at a deeper level in the next class and further on there is lot of complexity is this point of it ok.

The lock daemon you can you can easily see what can happen if the client takes a lock kind of particular file and it dies then other clients may not able to access it. So, there has to be some recovery procedure and us what we discussed earlier was that that the client is state full, but the server is not state full. That means, that seems that client had the state is nobody is able to see the state of the client.

Then, nobody can do a reasonable job of lock recovery over here it is going to be slightly missy over. So, what say then why we will find that lot of NFS are used and typically there is no lock daemon because the semantics are too difficult to or generally believed to be difficult. And so people just say I do not know what to make of it I will just not use it that is all from the way most systems are deployed and that also leads the NFS we had to be careful.

In terms of what considered the model you have trigger we should have seen in original methods.

(Refer Slide Time: 37:52)



RPC/XDR overview

- RPC services – RFC 1057
  - Request-response protocol
  - Reliable transmission
    - At least once/ At most once semantics
  - Message formats, marshalling, transmission
  - Authentication schemes: none, UNIX-based, key-based, Kerberos-based
  - RPC compilers
- Portmapper daemon
  - maps RPC prog # to TCP/IP ports where servers listen
- XDR services – RFC 1014
  - Byte ordering (big-endian)
  - Date types & formats

So, let us before I look at some aspects of the NFS protocol. One know one of them is RPC. RPC stands for remote procedure call is a request response protocol and ideally it should have a reliable transmission. But that is not really feasible with networks and so, that is why RPC can have two types of models what is call at least ones at most one semantics. What is at least once? That means, that somehow if I send a request it will be retried another party gets it its like your email then I send my mail to you I would like that in case some there is the problem somehow somebody keeps retrying and since it thing finally, it gets there.

But the problem of that model is that it is possible that I might I decided too early that my mail has not reached you. Now, somebody has decided that the mail I sent has not reached you will try retry before actually waiting for the actual mail to end up in the right place I was too eager I could a waited for sometime more probably it could I made it because I could not figure it out I have some fixed time outs and then I retired from water. So, it is possible that multiple emails landed up at the same time not in same time multiple emails give me. So, oftentimes if this multiple emails coming if not an issue, but if the multiple emails refer to withdrawing some money or something or some such operation that can be slightly difficult because this is to be what we are talking about is we are talking about what is call idem potency of requests.

So, what is when is a idem potent if I repeated K number of times it is same as doing it once. A good example of idempotent request is reading them reading a value. I read a value once I read that value twice threes it is still the same value as long as nobody expressed it nobody else expressed it. So, if you have idempotent operations then at least one semantics is quite meaningful. Whereas, if it is not idempotent then at least one say at least once semantics can be problematic.

So, since file systems do have both idempotent and one idempotent request you have to someone handle the non idempotent rectus there has that is to be somewhat handled otherwise things are going to be problematic. So, some of other RPCs have also at most once semantics, but the problem with this is that it means that you will not send more than one email, but your email may not also may also not reached that also is possible at most once. So, its zero or one; so these also a problematic thing.

So, typically it is at least once is being is typical assumed and somehow somebody else has to take care of it. For example, in the previous diagram it could be that NFS server or RPC server somebody takes care of what happens if it is sent more than once. I this typically in the case of NFS file systems NFS server takes to a bit, but we are doing something called lock daemon etcetera. Now, other thing that the RPC talks about is message formats basically when a request has to be sent across the network you have to both the parties have to understand what the message all about right. So, you need to essentially encode it in some format which both the parties can understand and.

This comes at very fairly tricky because there could be differences with respect to architecture compilers because whereas, compilers can do guiding in different ways they might also use different types of lengths of objects. For example, one machine can use assume that integer is thirty two bits whether can assume its sixty four bits. So, there are a lot of issues that crop up when it comes to probably saying what may of set is or how much length of file has to be read etcetera. So, there are various methods to work around it example there is something called ASN dot one obstruction has syntax notation one, which essentially guarantees across multiple different machines of different varieties whether its compiler differences or voice differences architecture differences network byte order differences instead of all of it, the sender and receiver agree on what value to be used.

So, similarly there has to be something about what was called marshalling that means that when you have lot of parameters they are going to send it as a single message. So, you have to somehow figure out how to place all the objects or the parameters in some linearized form how is that to be done in the context of compilers having their own notions are how to be providing etcetera.

So, this also have to be standardized you can also have because now if across in network you need to have some authentication schemes either there are none because you believed that your systems are reasonably secure or it can be UNIX based and this UNIX based. Basically you have this notion of UID and GID user ID and group ID and the assumption made is that across some departmental setting there is administrator which is who is sort of managing this UIDs and GID.

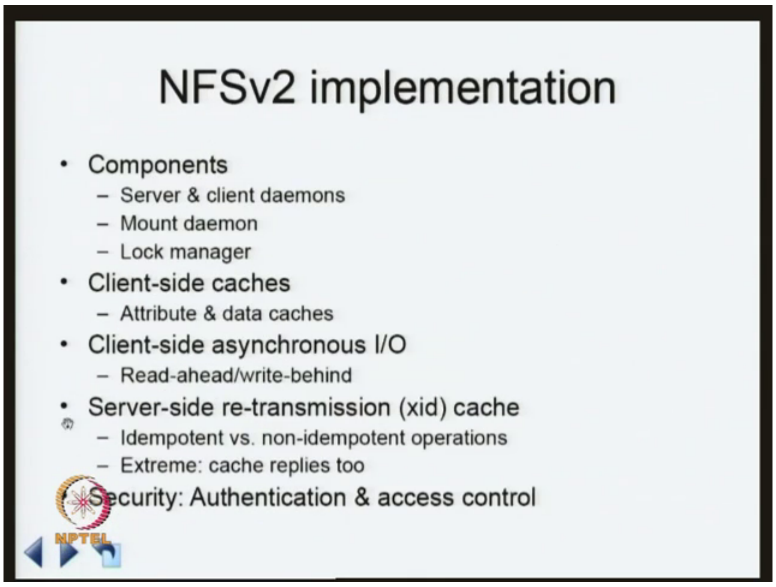
So, there is a consistent motion of UID and GID across all these machines and you can essentially use that as a way to do issue. So, again you notice that this can be somewhat of a problem because the route on one machine should not probably be the route on the somehow machine if it is going to be this UNIX base. So, that person very often times what happens is there is some kind of mapping that is done if you claim to your route on one client machine. It most securely is most much is its try to be slightly security conscious there will mach route to the null user on the server side. So, that the request does not have this route permissions on the on the server side.

So, you can also have some key based approaches or Kerberos based approaches. So, Kerberos is a cryptographic method wherein there is key granting servicer and you get keys from the Kerberos based servers. So, this RPC also has got some little RPC compilers. So, that your application with that whether return in c language or java or whatever it is right they can essentially assume that their message requests they will get compiled to get colour language that are using so that the request can be made from the language structures that will have the way that it has to be done in a local form. So, that it can go across the network ok.

That is there are various RPC compilers are there. Also you in the case of NFS in it and the version two which came first they also have some motion off at port map port map reading the reason being that initially there were no when the standard was defined by the NFSv2. There the servers listen these ports or numbers were not standardized. So, there are some way in which you had to some infrastructure had to be provided that essentially a load that kinds to figure out there is the server to listen. So, for example, this the NFS server would when it butted up it would resistor itself with something called port mapped demand saying that I am listening on this particular port. And so the request that coming right they will have the RPC program number and that they have to be mapped to the port numbers where they are going to listen to ok.

So, there is some other auxiliary servers that are also needed and this particular thing has been essentially eliminated as you go to NFSv4 for example. RPC server node now again with respect to the encoding you have various things like byte ordering and big endian, little endian etcetera. All those things are handled through the XDR. The more involved version ASN dot one, XDR is a spec ignored less ambitious version of how to make sure that your data back listening is able to be accessed on the other side without any change in the semantics of the data backs you are sending.

(Refer Slide Time: 47:11)



NFSv2 implementation

- Components
  - Server & client daemons
  - Mount daemon
  - Lock manager
- Client-side caches
  - Attribute & data caches
- Client-side asynchronous I/O
  - Read-ahead/write-behind
- Server-side re-transmission (xid) cache
  - Idempotent vs. non-idempotent operations
  - Extreme: cache replies too
- Security: Authentication & access control

So, let is quickly look at whatever things in a NFSv2 implementation. I should mention that NFv1 was never standardized. So, NFSv2 is the first standardized version.

It has got an RFC. So, what are the various components NFSv2 it is got server and client daemons it is got a mount daemon as I mentioned that is where you is a mount protocol and the client has to talk to the mount daemon to get this particular file system to be mounted on the client server. It also needs a lock manager and is a very critical component if there is some sharing going on across multiple clients for accessing the same NFS server.

Now because you are across the network it might help draw some caching. And so there are two types of cache as possible attribute caches and data caches attribute caches are basically the metadata of the file cache of the blocks will file. So, it turns out that in the if you cache it if this particular if a particular file is being shared across multiple clients and some of them have write permissions then it can be the case that the tributes that you have in your cache is not the current one.

So, that is why you need to you can cache it for some time small periods of time hopefully hoping that nobody else is modifying it and then after a small amount of time you flush your cached information with a particular file and then the next time around again you try to get attributes again. So, that is the typical model and NFS v two in the beginning started assuming that we can cache attributes for about ten seconds the that is have a more now once you have attribute caches then what you can do is if you have data blocks you can cache it locally. And then before you actually assume that it is valid information you can use the getter tribute information to check whether the data cache actually is valid. So, what can I do basically I have data blocks which have been cached I want to make it they are have still valid.

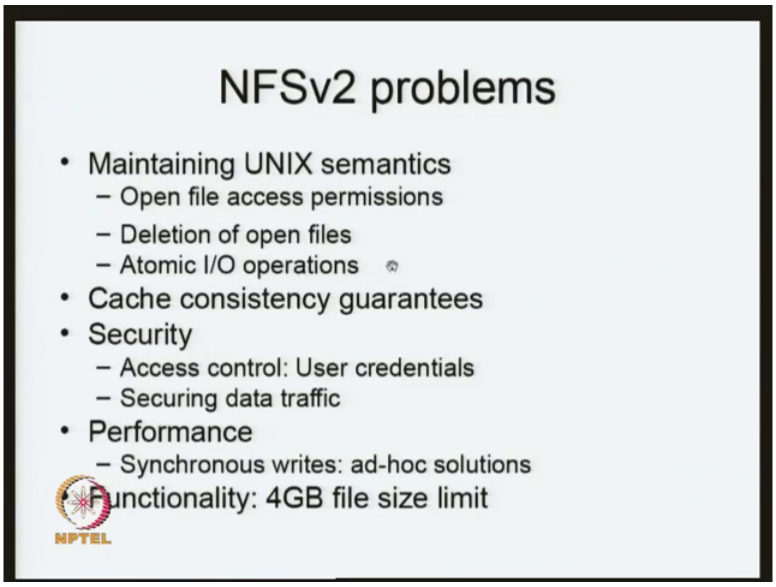
So, basically I use attribute information like go and if I am not sure that that particular cache could be holding valid information I can send the request to a server say please send me your attribute or the particular five file cache blocks and then I will check it whether it is still the same as what I thought it was it show have been and what the deduct cache that I had with me and that is the case then I will use it, otherwise I will have to get back data blocks again from the server.

Similarly the client actually can do because again because you have this network the client can initiate asynchronous. I hope it can get started and then not really wait till it is completely. So, you can do both read ahead and write behind read and write. In the sense that I ask for a particular file and I asked it to give me the read ahead read ahead versions also back on the flank side or I write some things.

And then I can show some time because I might again rewrite portions of it and then at some point later I want to send it out develop this console now as I mentioned earlier you have this problem about I do not put in this is known I did not put in operations. that is why you need to have what is called server side retransmission cache. So, what happens in this case is that let us say I get a request like remover file then I write each request with a id number if I see that the request comes to the same id number then I know that it is a repeat thing I can drop it. So, that is why I need to keep record. So, sever actually keep track of every single request is tagged with a number. So, the client is when it retries it has to send the request again the same id that is all the server can figure it out so.

So, there are things like rename delete remove file all the things have to go through this particular model. So, security for NFSv2 which has a, you can do through as I mentioned earlier multiple models you can use a UNIX model or the Kerberos kind of model.

(Refer Slide Time: 52:19)



NFSv2 problems

- Maintaining UNIX semantics
  - Open file access permissions
  - Deletion of open files
  - Atomic I/O operations
- Cache consistency guarantees
- Security
  - Access control: User credentials
  - Securing data traffic
- Performance
  - Synchronous writes: ad-hoc solutions
- Functionality: 4GB file size limit

Now, let us look at for today we will just look at some of the problems with v2, and I will continue from tomorrow next class on v3, v4. Then the issues that crop up for consistency then locking problems when you are sharing when generally we start talking about all these things. And we will go into commit protocols followed by including certain things like pack (Refer Time: 52:45) this kind of models.

So, what are the problems with NFSv2? It turns out that it is difficult to keep UNIX semantics across multiple modes; one good example is open file access permissions. Basically I can look up a file with a particular set of permissions and since I am not the only person who could be modifying things. When I actually go and access it next time around it may be that I might and denied the permission which I thought I had and that is a possibility. I can also have deletion of open files. In POSIX I can open a file and then if I delete it. I still can use the file handle the sorry not the file descriptor and only when the process exists that particular blocks that are corresponding to that file descriptor. Then that is time only when it is going to be released back. So, in a case of NFS because of the statelessness it turns out that I might have a I cannot give this particular semantics.

Basically because I there is no notion of an open file and opening a file in the first place because I do not have; let us see the open command available to me. that is one second is even if I get a file handle right and it may be that somebody has if I delete the file right then the file system in other side will actually remove all the blocks corresponding to it will not little keep it with you which is someone different from what process can do.

So, this particular aspect is also not possible. So, again for these kinds of things some work also possible, but they are not complete if the access and deletion of files happen through the same client then. The server can actually keep track of it and do something about it basically it can rename that deleted file to some file name which is unlikely to be guessed by anybody and then you give that particular file as the place where it keeps the blocks on the server and then the user can continue using it.

Similarly, is another problem here which is about atomic I O operations basically here what happens is that you notice that when do read and write there happening over the network and your read and write can happen over multiple network transactions. That means that during the network transactions you may not be able to provide atomic guarantees with respect to read or write because our network protocols cannot give you one gigabyte if you want the cost one gigabyte. So, might have to split it up once the split it up then you will find that it is possible that you may not be able to provide accommodicity that we are looking one.

I think I will stop here today. I will continue from this the next time.