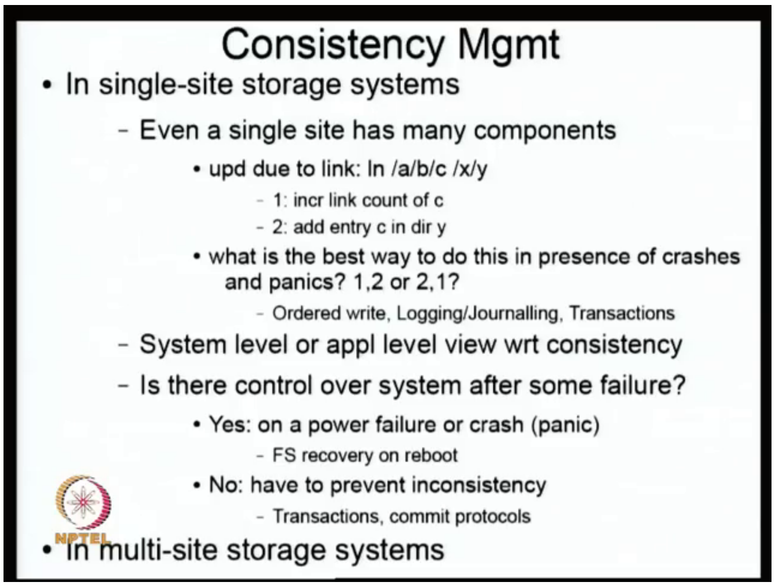
Storage Systems
Dr. K. Gopinath
Department of Computer Science and Engineering
Indian Institute of Science, Bangalore

Theoretical Foundations
Lecture – 32
Theoretical Foundations of Distributed Storage systems_Part 1: Consistency management problem, Commit problem, Commit Protocols

Welcome to the NPTEL course on storage systems. In the previous class, we were looking at some aspects related to consistency, and we mention that consistency is connected with the consensus problem or the commit problem.

(Refer Slide Time: 00:33)


Consistency Mgmt
- In single-site storage systems
  - Even a single site has many components
    - upd due to link: ln /a/b/c /x/y
      - 1: incr link count of c
      - 2: add entry c in dir y
    - what is the best way to do this in presence of crashes and panics? 1,2 or 2,1?
      - Ordered write, Logging/Journalling, Transactions
  - System level or appl level view wrt consistency
  - Is there control over system after some failure?
    - Yes: on a power failure or crash (panic)
      - FS recovery on reboot
    - No: have to prevent inconsistency
      - Transactions, commit protocols
- In multi-site storage systems

So, today we will take a look at the consistency management in storage systems. And now we will get started with some aspects of the commit protocols.

So, first of all consistency management is critical both in single site storage systems as well as in multi storage multisite storage systems. It is not just only the case that a multisite storage systems have to worry about it. Why is that, because if I at look even a single site storage system that site itself has many components. And therefore, the consistency problem could still be here also. To appreciate whether it is the case let us just take a an example of a file system operation.

All of you know that there is an operation call link in UNIX, and basically it create an additional link profile. What is this particular command? L n slash a b c to slash x y. What does it do? This is the source this is the target. This is typically a directory x slash y is a directory this is a file; that means, that what the semantics of this is that the file c will have an another name in the directory x slash y.

So, in addition to a file being in a b c, there will be another file called slash x y c. And basically, what happens is that, the there is a link count that increases by 1. So, it is essentially you might call it as the way of referring to the same object through 2 different pointers. It is basically what is happening. Now what is that if I want to do some operation like this which is registrant to failures, and it is not that trivial. Why is that the case? Let us think about what kind of things I have to do to do this operation. What is involved here? First of all, the file c, it has got a metadata right the metadata corresponding c, right.

And it keeps track of how many copies are there right. So, you have to increment it is called the link counter c. So, basically says that instead of a doing present in 1 place it is now present in 2 places, and the inode there is only a single copy of the object. There is only single copy except there are 2 pointers one from the director slash a b, there is the thing that is pointing to that c and the directory x y that also is pointing to c.

So, first of all you have to where you need the link count. So, that in case either parts is deleted. Sorry, sorry it is deleted from either part either directory. You have to keep track of one touchable read the file. For example, for delete after having them this link if I delete slash a b c this still should be slash x y c should be it should be still be there.

Similarly, if I delete in x slash x slash y c, then this copy should still be there. So, link count keeps track of it. Then the next aspect of this is that I need to actually put the entry into the directory y. Because now c also is going to be there in the directory y. Now we have to think about it this 2 operations 1 and 2, they are 2 updates to the source to the system when incrementing counter c, you are incrementing some field in the inode this is a different place on the disk than when you add the entry c to the directory y. So, this 2 are 2 different place on the disk.

Now, there is no magic by which you can update this 2 at the same time. Whatever method we think of it require some kind of what you say sequentially done even if you think of doing it simultaneously, it is a very complicated procedure because you want to do it simultaneously we need to do some time synchronization and time synchronization is not possible within a few milli second accuracy ill catch that into it. So, turns out if you think harder about the problem, it turns out it is impossible to do it in in a safe manner simultaneously it is just not possible.

So, in the sense one has to proceed the other, there is some kind of ordering that happens. Now there are 2 possible ways. Do I do one followed by 2 or 2 followed by 1. Now depending on which way, we do it with if there is a failure there are different consequences. In one case we have what is called production of garbage, in other one that is called dangling point. So, depending on which one you do we have one of these 2 possibilities. That is 20 why that is the case. Now critically it is better to have garbage than dangling pointers. Dangling pointer means we are referring to something it does not exists it is a more catastrophic error. Whereas having some garbage which nobody points to is the more secure thing.

So, here the idea here is I am trying to figure out whether to if I want to decide between 1 and 2 I will try to choose one of them which produces garbage rather than produces ga for example, suppose I go away with 2 first. If I add entry c in directory why? That means, I am already pointing to some object, but I am not ready to link it yet, but still not done with our work actual linking it. And if there is a power failure or panic whatever then the entry will point into garbage basically all of them is put an entry into the directory. But it is not really because I let us say that I flush that f to disk that is x y c, right. What I want is that directory entry has been flushed to disk. So, that because of power failure I am not been able to increment this part of it.

So that means, that that there is a record in the system persistent a system parallel system. That there is an entry in directory y, but actually the link count has not been incremented. Whereas, about other possibility is if you increment a link count that goes let us say flush to disk. And then you fail to update the entry in y, suppose there is a power failure in between that is you increment this this has gone to disk, but as this is not yet gone to disk it is still sitting in the memory. And it died once that power failure happened it could not be put into it could not be flushed to disk let us say; that means, that when I delete c, right? What happens is that now what is that because I did not add this one. There is actually exactly still only 1 place c exists, but the link count is not 2.

So, if I delete it the link count becomes 1 and since it does not exists anywhere else nobody is gonna delete another time c; that means, forever it can sit as the file can be existing without anybody noticing it this is an example of garbage. Which is sitting there which the blocks are allocated it is not pointed to the anybody and it is still sitting there this is because I made a mistake in this is a way it happened the link count is increased, but there is no corresponding entry in the directory y.

So, in a since that garbage is produced and this is the safer situation because whole was a right consumer of c got it and then disappeared. And there is no new consumer of c again and it just stays as some blocks that are not cannot be returned to the pool again. That is all that happens. So now so, this is less of the problem, and that is what that is the reason why if you talk about if you look at older file systems. They will go through step one followed by step 2. And they do not go step 2 followed by step one. This is what is called ordered write, basically ensure that whatever happens you first make sure that this part of the update to the storage system that is the metadata corresponding to the inode corresponding to c that is flushed to disk first, then later this is done.

So, this is what is called ordered write you order, the writes in which the things have to go to disks. But the other methods what is called logging journaling and transactions which are also used. So, basically, I just want to motivate the issue that if you are looking for the consistency of the file system information itself. You may have to do some management here, and that is done by many of these methods ordered write logging journaling and transactions.

So, logging is a slightly more elaborate method to do the same. Basically, you make a note about the kind of changes that you are going to make in presence of store before actually, I make the changes it is are called the journaling. And there is also something called transactions which is basically it has been popularized in the database community as basically equivalent what you mentioned called acid that is while you do certain things some multiple activities together as one single transaction, you want to make it sure that it satisfies acid property. For example, in this case, you can say that this both this 2 things can be taken as one single transaction, and in acid property is present then both of them either happen or not either happen.

So, that also is possible. In the case of logging and journaling what you do is you essentially can log this step that I am going do this followed by this. And then as you do it you keep on essentially saying I finished part 1; I finished part 2, etcetera. If I later I discovered that I say part one is finished and not part 2, then I know that I already said I am going to do this I going to do what is called redo operation. I know that 2 has not complete and go and do 2. And this happens in the case of typically file systems also usually do this kind of model. And the reason why is that is it turns out if you have a power failure crash for example, or a panic the system has to reboot a reboot time in the file system can do some corrective action. For example, there is something called fsck etcetera that can be used as a way to get controlled over the system with respect to it consistency.

So, that is what some models basically we do we do the thing; that is, you already have make a note of what has to be done is know that some parts have been completed, but some things do not have a records in the log and has been completed. Therefore, when you get back and you get control over the system you basically redo it.

So, in some situations you can do it in typical file systems follow the redo model of login, there are some other situations where it turns out you really can not afford to have any inconsistency. And typically, this is the case with concurrent transactions which have some real-world implications like example money multi aspects transactions database kind of things. So, here the basically we have to prevent inconsistency therefore, you do certain things and then you might undo things because it didn't satisfy some consistency properties. You do not wait for system to reboot before you do it you have to do it, if there is inconsistency you have to immediately correct it.

So, transactions commit protocols are on this category. So, again just to reemphasize what I said previously. There are different ways you can look at consistency. You can do it at the level of system level like file system do or you can do it at the application level, because application level has given you some information about what kind of things it expect expects to invariant. And therefore, databases or this kind of systems they can actually do application level model consistency. So, why am I motivated why this consistency management also is important in single storage single sited storage systems basically because they are essentially multiple components.

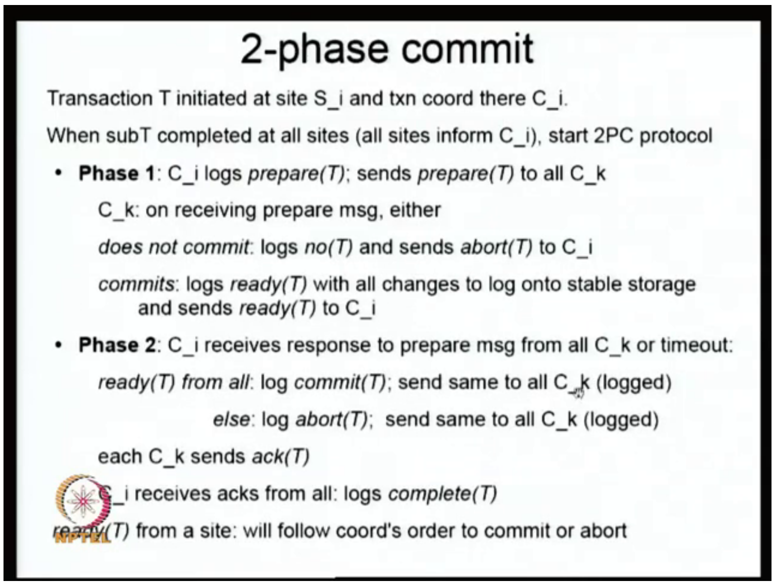
And so, they are essentially similar to in principle to what happens in multi sited storage systems. So, we will look at the consistency management in the multi sited storage systems. We will first start with some simple models, then we will go to slightly more models. So, what will do is we will look at commit protocols. First as I mentioned earlier commit protocols are used in those systems in which persistency is an important aspect. For example, file systems in databases, because when they fail when some or other components fail they usually have accumulated some state about what they were doing.

And when they recover they have to use the information to somehow make the system again consistent. And we mention also that there are consensus kind of protocols, where you do not really try to recover from if something has failed we proceed we basically proceed because there are some real time other reasons, why you cannot wait for some sites to come up as we discussed last time.

So now, we will talk about storage systems which each component have some component, which each component has some persistent state that has been accumulated. Later we will discuss what is called group committing systems, which are used also in storage systems so that a consistent view of the system is maintained when you send messages and the messages can be received or lost that one is closer to the consistent kind of model we will look at that also.

So now we will start with the commit protocols.

(Refer Slide Time: 15:23)



2-phase commit

Transaction T initiated at site S_i and txn coord there C_i.

When subT completed at all sites (all sites inform C_i), start 2PC protocol

- **Phase 1**: C_i logs *prepare(T)*; sends *prepare(T)* to all C_k

  C_k: on receiving prepare msg, either

  does not commit: logs *no(T)* and sends *abort(T)* to C_i

  commits: logs *ready(T)* with all changes to log onto stable storage and sends *ready(T)* to C_i

- **Phase 2**: C_i receives response to prepare msg from all C_k or timeout:

  ready(T) from all: log *commit(T)*; send same to all C_k (logged)

  else: log *abort(T)*; send same to all C_k (logged)

  each C_k sends *ack(T)*

  C_i receives acks from all: logs *complete(T)*

ready(T) from a site: will follow coord's order to commit or abort

There are many types of commit protocols. We will start with the 2-phase commit protocol. There is also a 3-phase commit, and there are also other kinds of commit protocols. For example, (Refer Time: 15:31) is another one we will look at some other ones later.

So, what is involved in a 2-phase commit. We have other called we have transactions which are let us say executed in distributed fashion; that means, that it is executed not only the coordinator we will call it cfi. Cfi is the coordinator, and it is also executed at multiple other places. The idea is that if there is any let us say is a transactions to be completed it has to be completed under grid to 1 by everybody else, or nobody should proceed with the transaction any subcomponent with transaction.

So, we will assume that transaction T is initiated at site si and the coordinator there is cfi. Now what happens is that the transactions coordinator sends information to all the other parties, saying that you have this small sub part to be done. Please do it, once you are done, tell me when you are done.

So, that is what happens when all the other sites have completed their part as the work they inform the coordinator saying that they finished that supplier section that they were expected to do. Now once the all the sites have informed the coordinator, we must start the 2-phase protocol, 2 phase kind of protocol. So, our idea now is that we asked each site to do some piece of this transaction either all of the effects of this should be persistently, let us say made taken into account or none of them should be, because if we are talking about an acid property. It should be atomic either all of them do it or none of them do it.

And so, basically this protocol is all about that part. So, what is there in the first phase? The coordinator has got all the information, from all the sites saying that they have done, their work that was assigned to them. But they have not committed it to stable storage, they have not committed it they have basically kept. They have finished their work they have kept in volatile storage. And once the commit happens they will actually update it to stable storage, that is the basically. So, what is the issue here basically the coordinator now says prepare T he is trying to tell everybody, now they prepare to commit and going to and got all the information all of you to you to be ready for the commit part of the protocol.

So, I will ask you to prepare and C i basically what it does is it logs prepare T and flushes it to disks at the stable storage. And then it sends prepare T to all cfk. What is the way it is ordering it first it logs it flushes to disk, and then only it sends a message again it is not done in the reverse order you do not send a message followed by this again if you do that you will get into some even essentially that all inconsistence process.

So, basically, we have to prepare T flush it to disk make it stable. Then you send a message cfk on receiving a message prepare message, now it has done it is sub transaction sometime in the past. Now it decides that either it can commit it or not. Commit for example, you tried to get a hotel booking the hotel booking did not work out therefore, the whole transaction should be flushed it should be not committed. So, here is basically what it does is it says that I am not going to commit.

So, basically says logs not again it will sends about it to cfi. If you if other hand this sub transaction is really successful. It is ready to commit. So, it says to signal that is ready to commit it says logs ready T. Says I am ready to commit in case the coordinator is agrees to that it has to be committed.

So, conditional thing it is basically saying, but I am ready that is all it says with all changes to log onto stable storage again because that for example, the hotel reservation or whatever, you are going to log all of it to stable storage, because now the minute has a ready T; that means, I am willing to carry out my portion of it. I am going to make it stick to make sure it is sticking I have to whatever changes I have to make to make the transaction the sub transaction go through I have to commit to stable storage. And then it sends ready T to cfi. Again, I commit everything to stable storage then only I send a message.

I should not be doing it in opposite direction. Again, what is the critical aspect of stable storage? And if your storage systems person, we notice that all of this depends on the fact that there is some magic called stable storage. How is this stable storage itself created is a big question. So, for the time being in this coming protocol which is assume there is some body has given you stable storage. Because what is that disk it is by definition it is not a stable storage. I can write to a disk and then it can fail, and it will next time I read it I cannot I may not get the result back it is possible.

So, disk by itself is not a stable storage. So, if you want to make stable storage out of unreliable components like disks, we need to do something else. That itself requires surprisingly a recursive aspect of commit protocols. So, we will talk about it later. So, again before I just to re capitulate in phase one what happens is that once all the sub transaction completes the work the coordinator sends the prepare message to all of them, and then each site it decides whether to commit or not commit depending on the success or failure of it is own sub transaction.

And then it is if it is not successful it is going to say no, and sends abort message. If it succeeds any sub transaction it says it is ready to proceed as long as the coordinator agrees to as long the coordinator says everything is fine with respect to everybody else, and it sends a ready message. Now in phase 2 cfi receive response system to do a message from all ck or timeout. Basically, it waits for some time again there is another timeout, because things can also fall apart as things are going on. So, C i receives response to prepare message from all ck, and if in case it is ready T from all everybody says that there is a sub transaction succeeded.

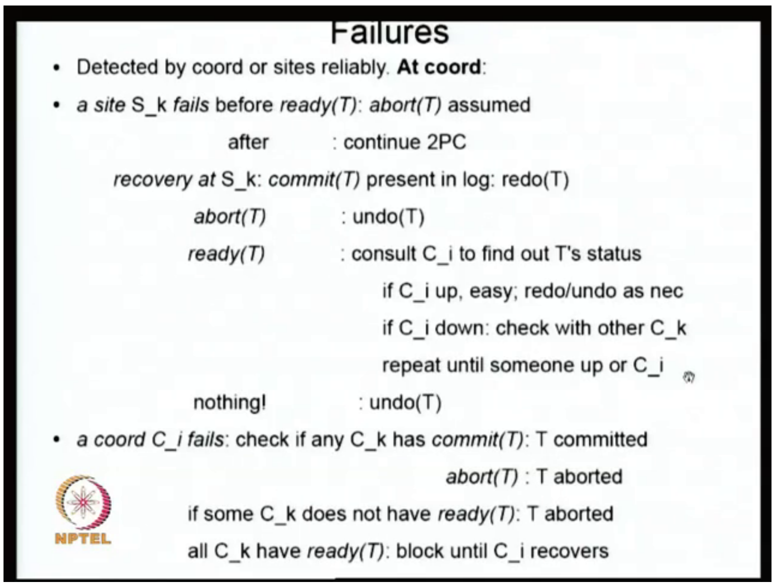
Then a log commit T basically what is the coordinator going coordinator got a message from a ways that everybody is with proceeding the main transaction. So, you log com so, the coordinator now logs commit T. And it sends it is message to everybody else. And each party who gets also a log that message that it is committed. Otherwise if it is not ready T from all, then there is some problem in the transaction. At least one or 2 people at least more than one person at least one person had a problem. So, you have to abort.

And again, you the coordinator logs this along flushes to stable storage, and then it sends the file to abort to everybody else. Who in turn once receives the message? They will also flush it to storage. Now you can see that there is so much flushing going to disk crossly because we want to make it persistent. That is why you notice that the throughput is going to be determined by the storage system how fast it can persistent to write something. And then once it is done, once each of these cks get this they again send an ack back when C i gets acks back from everybody it basically says that the transaction is complete.

So, you can see that commits is got a lot of parts to it can fail any point. It can fail just of the sub transaction has been given before a prepare T has been received after prepare T has been received before this commit has been some particular site or commit or it can fail at any multiple places. The question is how they recover from each of these possibilities. Because you are logging most of these things we always have some record or you can clear the log on both the coordinator as well as this all the sites you can figure out where exactly things break down and try to see it can fix it.

So, again just one thing to notice that when a ready T is sent from a site, where they basically promise by a site they will follow whatever the coordinator decides; that means, it is with thing it just that if the coordinator says I will proceed, otherwise I do not want to proceed. That is basically the idea. That is the ready T ci.

(Refer Slide Time: 25:46)



Failures

- Detected by coord or sites reliably. **At coord**:
- *a site* S_k *fails* before *ready(T)*: *abort(T)* assumed

  after : continue 2PC

  recovery at S_k: *commit(T)* present in log: redo(T)

  abort(T) : undo(T)

  ready(T) : consult C_i to find out T's status

  if C_i up, easy; redo/undo as nec

  if C_i down: check with other C_k

  repeat until someone up or C_i

  nothing! : undo(T)
- *a coord C_i fails*: check if any C_k has *commit(T)*: T committed

  abort(T) : T aborted

  if some C_k does not have *ready(T)*: T aborted

  all C_k have *ready(T)*: block until C_i recovers

So, what are the various issues? Again, we are assuming that the failures are detected by coordinator or sites reliability. Again, as I mentioned earlier, detecting of a particular node is did are etcetera is not a reliable thing in the first place (Refer Time: 26:06) professor this protocol we assume it. So now, there are multiple possibilities either the site can fail or the coordinator fails. Now let us say, they detect at the coordinator there are sites S k fails.

Now, before it sends ready T with that site fails, it is equivalent to abort T. Basically, nothing has been done he has not told anything what happened. I have no idea what he has done whether the hotel transaction whether succeeded or not. I have seen that since the guy is not responding to me. I say that whole transaction has to be dropped. That is what this is. If it is after; that means, site S k you will notice that it has got a ready T in it is log.

So, what it means is that it is; it has done it is sub transaction, and it has it is ready to commit as long as the coordinator says. So, that means, that it is possible for me to continue the 2-phase commit protocol, and basically wait for some response. So, the we will figure out that happens in this recovery procedure. So, that part of it is here, when I said after ready T this part is here. So, we will come to that is all.

So, at the site S k, what is the basic recovery procedure? Now the simplest ease of commit T is already present in the S k if your commit T is present; that means, that it has already received a message from the coordinator to commit. And therefore, it can all you have to do is to redo the transaction. Because it was waiting for the commit T and, but it fail before it actually completed the transaction. Because what is that; there is a ack here right C i receives acks the thing is not received died before that that is why we are talking about this situation.

So, the commit is present therefore, we just have to make sure that whole transaction went through that is what is given here. In other hand, if abort is present; that means, the whole transaction did not go through at all. Therefore, we just have to undo whatever we had something in your if you had made some temporary, what you say markings about your hotel transactions, you just can do all these things. If in case they were actually sitting in persistent memory. For example, temporary files and what not

Often times when we do some transactions those kind of things, you just get rid of all those things. If it is ready T the case I am talking about here what we are talking about here. The reality what happens? The thing is I was ready to do it, but I do not know what really happened. So, I have to go back to the coordinator to find out really what happened. If C i up, it is easy to find out what the coordinator says if the coordinator says that please go out and complete it or just drop it you do appropriately. If unfortunately, at the time when this site S k comes up C i is also down, let us check with other nodes to see; what really happened. So, you keep checking till somebody or C i itself comes up saying what the situation is.

So, there is a issue of getting for somebody coming up, and that is where some aspect of blocking is going on we will come to that again when we talk about this coordinator. So, you have to keep waiting till some when situations becomes clear exactly whether somebody has already committed it or so that you can make consistent. Basically, we are again worried about consistency that is why depending on what other people say you have to follow the orders.

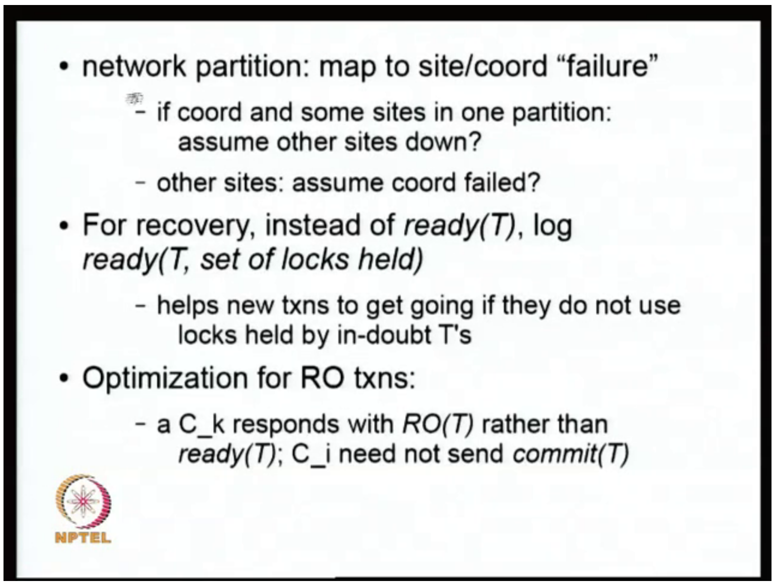
If in case there is nothing at all, then it is equivalent to not having to worry about not doing anything we just undo whatever temporary things that you have done. In your site in case they happen to you again persistent, that is what it is. So, we took care of the situation with respect to the site failing what about if the coordinator fails. The coordinator fails to check if any ck has commit T. If any ck if any site has got commit T; that means, that the coordinator already has instructed that the transaction is going to go through. Therefore, it makes sense to commit T everywhere.

If in case there is not a single guy has got commit T, then it makes sense to abort it. Because; that means, that the coordinator actually have not made up his mind about what exactly have to be done and aborting this is a safe procedure. So, basically if any ck has abort we just abort it because and if there is not a single ck which has ready T then also aborting is make sense. The only issue which is problem is all the other guys have ready T, but the problem is that we do not no what was ca thinking. Because ca also would have been doing some part of sub transaction. It may have decided to abort itself, or it might have decided to commit if everybody else agrees. So, because of that situation you really have to block till C i recovers.

So, this is basically the blocking situation that we will talk about, when they say 2 phase commit protocol is blocking. The reason is that everybody else is ready to commit, that the party is actually is trying to make the transaction commit as a whole. That partys state itself is not known. Because that party that state is not known, you have to wait till cfi recovers if it recovers then you can proceed otherwise you can not proceed. So, normally we generally believe that this situations are a bit rare; that is, what most system do still use to face commit. This is often there is a very bad situation. For example, suppose there is a some kind of database system, it is got multiple sites, and the coordinator phase, and the system cannot come up for 2 ways at a time for example, or undo. When it turns out that not only this transaction other transactions which are connected are may be have to hang till the particular site comes up.

So, this is considered a problematic thing those kinds of situations, but often it is not that bad therefore, most people are used to face commit.

(Refer Slide Time: 33:39)



- network partition: map to site/coord "failure"
  - if coord and some sites in one partition: assume other sites down?
  - other sites: assume coord failed?
- For recovery, instead of *ready(T)*, log *ready(T, set of locks held)*
  - helps new txns to get going if they do not use locks held by in-doubt T's
- Optimization for RO txns:
  - a C_k responds with *RO(T)* rather than *ready(T)*; C_i need not send *commit(T)*

When it comes to network failures, you can map some of the failures to site or coordination failure. For example, if coordinator and some sites in one partition, we can say that other sites are essentially, down we can just map it that way and proceed with as failure. Other sites which are not do not have the coordination, in there partition they can assume coordination has failed coordinator has failed. And so, they can do the recovery or whatever they want.

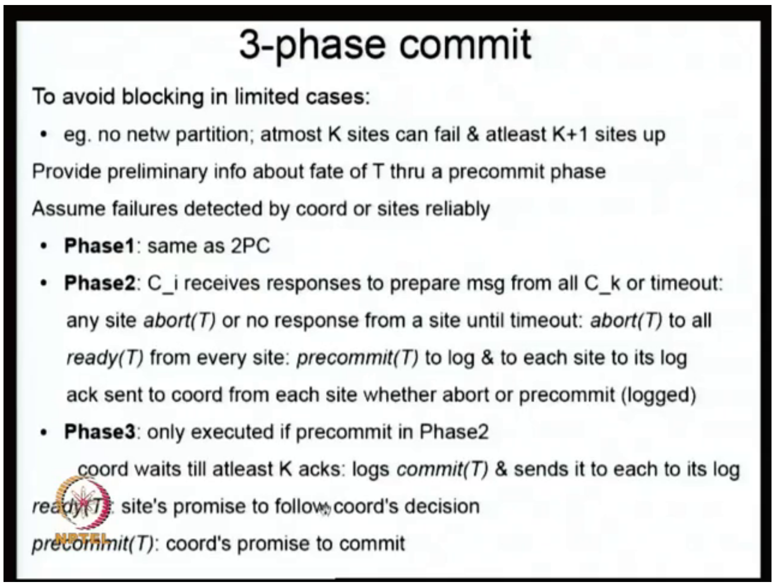
So, essentially this this particular protocol is correct, but not live it does not guarantee liveness, it guarantees correctness. And that is the case of almost all the protocols will talk about for some time because 3 phase commit plus back source all those things have the finish to it because, correctness is far more important than liveness in some of these cases. Whereas, when you talk about other systems on the web for example, we can do it some amount of inconsistency.

So, their availability is more important. Here the idea is to keep dating till the coordinator somebody comes up till the situation comes here. For optimization there are some optimization which people often do, sometimes we have to when we are doing some operations updating some things we need to take certain logs. And we actually log the set of logs also held so that other transactions may need the same thing same locks for same data, we are able to figure out by looking at the log.

So, for other transactions they can keep they can get going they do not use locks held by locks are hanging style. If I do not use locks then in principle that hanging transaction could be holding onto some locks and we have to wait till the coordinator comes up. So, there are most often people do this. So, this is the way in which the blocking aspect even if it is problematic it does not really bother it too much there is another optimization for read only transactions basically because you are not updating anything.

So, you can send the fact that the sub transactions relies on the evaluate basically it can send read only transactions rather than ready T. Therefore, C i does need not send commit T because nothing has to be updated. That is also is possible. So, there are some so, this is one way in which you can do it is consistency across multiple sites. And it is that it is a same thing that a clustered file system also has to do if you are talking about a clustered file system clustered database, we have to do all this exactly same thing because correct correctness is important. We have do exactly this steps.

(Refer Slide Time: 36:42)



3-phase commit

To avoid blocking in limited cases:

- eg. no netw partition; atmost K sites can fail & atleast K+1 sites up

Provide preliminary info about fate of T thru a precommit phase

Assume failures detected by coord or sites reliably

- **Phase1**: same as 2PC
- **Phase2**: C_i receives responses to prepare msg from all C_k or timeout:
  any site *abort(T)* or no response from a site until timeout: *abort(T)* to all
  ready(T) from every site: *precommit(T)* to log & to each site to its log
  ack sent to coord from each site whether abort or precommit (logged)
- **Phase3**: only executed if precommit in Phase2
  coord waits till atleast K acks: logs *commit(T)* & sends it to each to its log

ready(T): site's promise to follow coord's decision

precommit(T): coord's promise to commit

Now let us look at 3 phase commit. In 3 phase commit the idea is basically is we have this blocking situation, we want to see if you can avoid the blocking.

Now, even in 3 phase commit, because of the fundamental aspect of the theorems we looked at right for example, flp result etcetera. There is no way to actually have a new blocking. Essentially, either you if you look at the consistency, you will not be lie. There is basically issue. So, even this is so, this has also the same issue. Here what we are assume is that, sites can fail, that we will assume that there is no network partition. So, what we are assuming the utmost k sites can fail, and at least k plus 1 sites up. That is what we are making assumption; that means, we sums up we are talking about the majority sh we have majority of sites that up.

So, we basically here is; since we are blocking in the previous protocol, because we couldn't read the mind of the coordinator. The coordinator was did something, but did not diverge what he was planning to do to others, right. And therefore, everybody got stuck. Idea here is it possible that if T also the coordinator sends information about it is particular aspect to other parties then they can actually do something interesting that when that is called a pre-commit phase.

Since they have does it. Again, we assume failures detected by coordinator or sites reliably. In phase one the same as 2 phase commit. What is involved here? We basically requesting each site to respond and each site responds by saying whether they are going to they are ready or they went abort.

So, C i receives responses to prepare messages from all ck or timeout any site abort T. Or no response from site un till timeout and abort, it sends abort to everybody. Ready T from every site. Now this is the time when the coordinator sends a pre-commit to log and to each site to it is log. Basically, in the sense we are communicated to everybody saying that everything is clear am going to commit.

So, everybody knows about the fact that what coordinators intentions are ack sent to coordinate from each site whether abort or pre-commit. So, basically, we send it to everybody and then ack is received from every site whether they got this message whether abort or pre-commit. And these things are also logged on their systems.

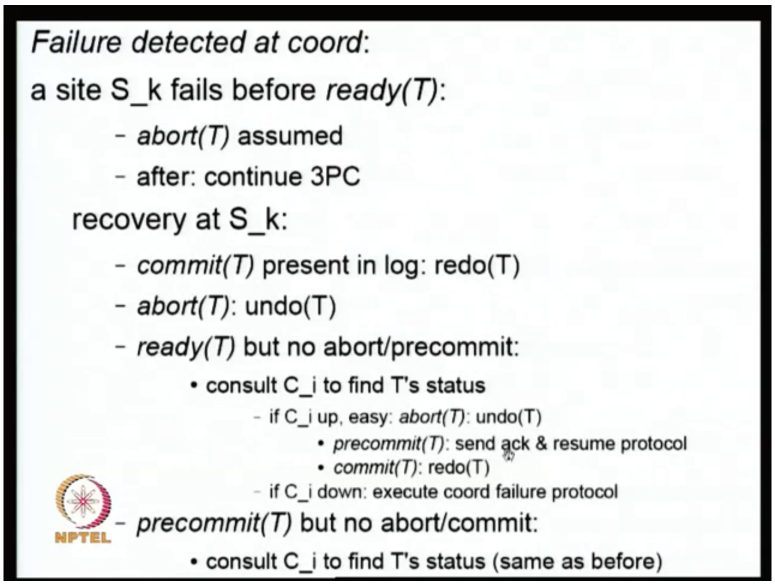
So, in case see if in here the coordinator may decide to go to pre-commit or may not decide to go to pre-commit. If it decide not to go to pre-commit; that means, that she is not going to go through the transactions. If he has said pre-commit; that means, that he is expecting further progress in the transaction, and that is what the phase 3 is support. Only executed if pre-commit in phase 2. Coordinator waits till at least k acknowledges.

So, basically as I mentioned at least k plus 1 it is basically waiting for the majority of acks. I think it should be k plus 1 here it should be k plus 1 at least k plus 1 acks, sorry. So, waits till the majority number of acks, and then it logs commit T and sends it to each log. What is that the parties already have some idea about, what the coordinator was thinking about. That is why there are some failure situations where they can actually once the cpu pre-commit they know that they can proceed. And we will come to exactly this (Refer Time: 41:07) situations.

So, what is the meaning of ready T? Here I can just like the 2-phase commit. It is sites promise to follow coordinators decision; saying that 2 phase commit was all one sided. It was only the sites promises to follow coordinator. There was not a reverse went from the coordinator to the sites. When you say pre-commit, there is a reverse kind of a handshake. The coordinator promises to commit. So, there is some additional state that has been generated which helps you to resolve the issue in some cases.

But again because of all the results we have seen theoretical results user also should block in some cases. There is no magic here. It just that it takes care of some situation. Basically, I have more extra information. That extra information can be used to resolve some cases, that basically what you can do.

(Refer Slide Time: 42:12)



Failure detected at coord:

a site S_k fails before *ready(T)*:

- *abort(T)* assumed
- after: continue 3PC

recovery at S_k:

- *commit(T)* present in log: redo(T)
- *abort(T)*: undo(T)
- *ready(T)* but no abort/precommit:
  - consult C_i to find T's status
    - if C_i up, easy: *abort(T)*: undo(T)
      - *precommit(T)*: send ack & resume protocol
      - *commit(T)*: redo(T)
    - if C_i down: execute coord failure protocol
- *precommit(T)* but no abort/commit:
  - consult C_i to find T's status (same as before)

Now, let us look at what this is; the site S k fails before ready T, and this is similar to previous situation. Nothing really seriously has happened therefore, you can assume abort. Now it is it can be after ready T to be continue 3 phase commit again we will come to the recovery part of it. Here again this part of it is this ready or commit and pre-commit this 3 situations this, this and this.

So, either we have abort or so, this abort is basically you have to clear transaction. If either one of this things we have to figure out how to handle it. So, what happens it is very clear if S k has commit T; that means, that there must have been a message from the coordinator saying commit. Therefore, it is quite safe to do redo T. It is abort as we mentioned before we basically undo if it is ready T. And there is no abort or pre-commit. Now we do not know really what happened. So, basically, we have to consult cfi to find T status. If C i is up it is easy if C i is above T then undo T. Basically, C i is up and turned out somebody else was not able to proceed with that part of sub transaction therefore, we decided to abort.

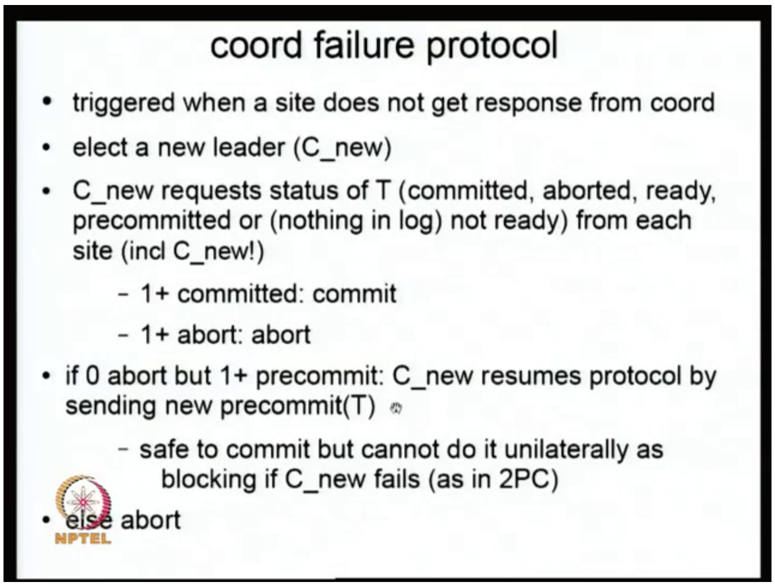
But our site failed before we got the message or whatever happened. That is why as long as somebody as long as C i is abort we can just go on undo T. If it is pre-commit T, basically C i says I have just done pre-commit and not proceeded for that I have not got response from everybody else. Then S k basically it is ready. So, it can actually say you are still stuck at the pre-commit stage and anywhere still ready so, I can proceed.

So, I will said I am going to send back saying I am ready to proceed. If you are ready to proceed, the coordinator ready to proceed. That is what it is another hand if c is says I have already said it is commit, right. Then all I have to do is to do ready T, right. Basically, what happens was that the what is that we have sent a pre-commit that is C i essentially sent a pre-commit to everybody, but this was not seen by this site sk, but since the protocol for reasons of liveness it waits only for it assumes that is k plus 1 guys are up. So, because of this our site this is say did not get those messages, but other guys got it on then the site the coordinator has actually gone hadn’t committed yet not waiting for this guy to come up S k.

So, that is why we can do redo of T because it has been committed by the coordinator, and coordinator stepped saying. So, our problem here is if ca goes down; that means, that this site S k has failed, and then you tried to came up you try to do something with try to discover the state of cfi the coordinator, but then cf down cfi itself is down. So, this is a situation where there is no further information I have to really do some (Refer Time: 46:08) again this is an example where I cannot proceed till I do something else. So, in here part of it in execute coordinator failure protocol, what we do is we try to elect a new coordinator. Again the 3-phase commit protocol, it tries to keep the system moving forward.

So, if the coordinator is dead, right. Because coordinator already had sent information about the pre-committed etcetera. Now I have to what I have to proceed. So, I have to get a new guy and then we will see whether he can make it make the protocol move forward. That is what I am telling. If you find that at S k there is a pre-commit, but there is no abort or commit, again you have to consult cfi to find ts status we will come to this one we will discuss it in more detail basically. So, exactly same as here, you have to do the same thing up. So, you have a pre-commit; that means, that the coordinator promised to go ahead, but before that happened right, your machine crashed and you are trying to figure out if the what reverse happened to the T.

(Refer Slide Time: 47:37)



coord failure protocol

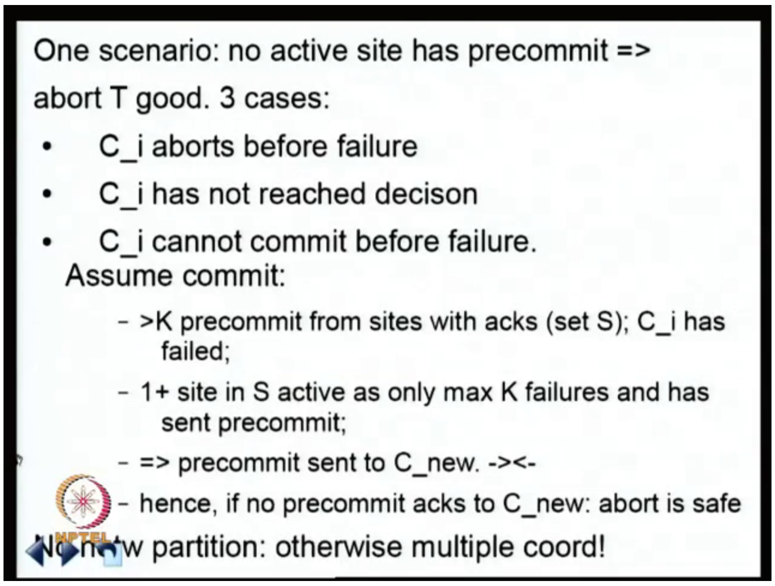
- triggered when a site does not get response from coord
- elect a new leader (C_new)
- C_new requests status of T (committed, aborted, ready, precommitted or (nothing in log) not ready) from each site (incl C_new!)
    - 1+ committed: commit
    - 1+ abort: abort
- if 0 abort but 1+ precommit: C_new resumes protocol by sending new precommit(T)
    - safe to commit but cannot do it unilaterally as blocking if C_new fails (as in 2PC)
- else abort

So, this coordinator failure protocol is triggered when a site does not get response from coordinator. So, what you do is you elect a new leader, and then this new leader requests status of T from each site whether it is committed aborted ready pre-committed or nothing is happened. So, if even one or more guys are committed; that means, that it actually was committed in the past. So, it can go ahead and commit it. If one or more busy says abort, then we have to abort it. If there are no aborts, but at least one pre-commit this is a case where it is interesting case 0 abort, but at least one pre-commit. Now c new has to resumed protocol by sending new pre-commit. Now you really cannot go ahead and say commit, because we have to see what c new itself has it really cannot do commit right away because.

If that happens then we essentially get same situation as 2 phase commit. So, what happens is that you know that the previous coordinator is planning to do forward, in term of transaction that is what that 1 plus pre-commit says. But before it could be done there is a failure, and the coordinator failed right. Therefore, the right thing to do is to try to move forward, and moving forward is by sending the pre-commit just as the previous coordinator did when he had decided almost to proceed right. Now the reason why you need to do this is because if you doing this pre-commit, then you can get into the blocking situation if c new fails. So, for this reason we actually restart from where the pre-commit phase are there make sure somehow make sure it tries to proceed.

So, this essentially gives you a way of looking for a progress. It is not exactly the same as a previous situations, but what happens is that we have to do this electing a new leader. Every leader goes through the same phase and this can keep on continuing. So, because it can keep on continuing, essentially, they might they are not guarantees about liveness here also.

(Refer Slide Time: 50:41)



One scenario: no active site has precommit => abort T good. 3 cases:

- C_i aborts before failure
- C_i has not reached decison
- C_i cannot commit before failure.

Assume commit:

  - >K precommit from sites with acks (set S); C_i has failed;
  - 1+ site in S active as only max K failures and has sent precommit;
  - => precommit sent to C_new. -><-
  - hence, if no precommit acks to C_new: abort is safe

No new partition: otherwise multiple coord!

So, actually there is one more scenario where no active site has pre-commit, and here it turns out abort the transaction is good. Actually, there are 3 cases. The coordinator aborts before failure it is very clear that you can abort it. C i has not reached decision there is very clear you have to abort it. It cannot commit before failure we can show that it is not possible because if you assume that this C i has committed then at least more than k pre-commit from site that has to commit, and then the new coordinator sees those things. And since the new corner c set pre-commit must have already been there, but no active site.

Therefore, abort is quite safe in this case also the one reason that I am assume that no new network partition is because, it turns out if I do not assume this it can be multiple coordinators also. There also has to be avoided. So, in the next case what we will do is we will look at taxos which also resolves the same issue just like different manner.