Reliability and Security in the Distributed Databases

Source : S.Ceri-G. Pelagatti: Distributed Databases

How can the quality and security of a DDBMS ensured?

Quality:
- functionality
- reliability
- flexibility
- services

Security:
- access control
- audit

Reliability:
Measure of the success with which the system conforms to some authoritative specification of its behavior. When the behavior deviates from that which specified is, this is called failure .

Levels of reliability:
- application dependent
- application independent

The application dependent part refers mainly to the integrity constraints.
The application independent part refers mainly to the ensuring of the ACID concepts.

Reliability

Main aspects of reliability:

- *correctness* : the system works in accordance with the corresponding specifications
  Some kind of software testing methods ( ) can be used.
  There is no appropriate tool to proof the correctness of a complex software system.
- *availability* : fraction of the time that the system meets its specification

Usually the requirements of the correctness and of the availability can not be met at the same time.

Increasing correctness decreasing availability
Increasing availability $\Rightarrow$ decreasing correctness

Correctness oriented applications (banking, simulations)
Availability oriented applications (tourist information system)

Trade-off problem : which of the two aspects should get higher priority?

2-Phase Commit example:
A distributed transaction is performed. Some of the participating nodes sent a 'ready to commit' message to the coordinator. The network connection is getting now down to a site which did not give an answer yet. What should the coordinator do?
- waiting for answer : high correctness, low availability
- take a decision : high availability, low correctness

Main problem areas of reliability

To provide an appropriate protocol for transaction commitment
A such method is need which could provide a good trade-off in the case of failures too.

To provide an appropriate protocol for data replication
A data element may be replicated at different sites. How can the different instances managed in the case of failures?

Determining the state of the network
How could we ensure that every participant has the correct information regarding the operational states of the other partners?

Detection and resolution of inconsistencies
How can we detect and resolute the inconsistencies caused by a not fully correct decision of the coordinator?

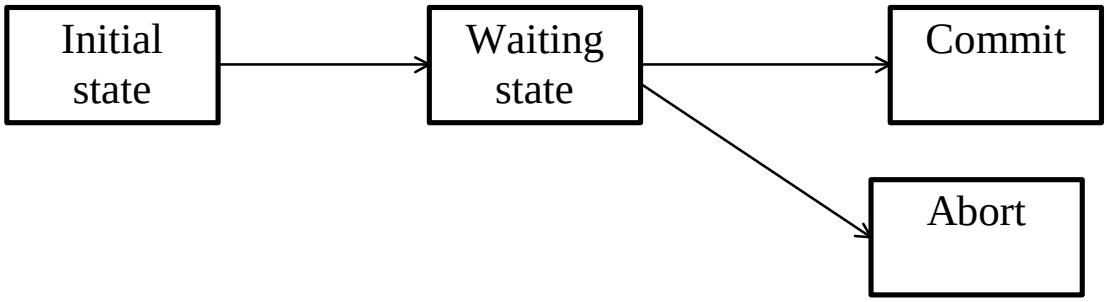
Checkpoints and cold restarts
How can the global consistency of the distributed database ensured in the case of a cold restart?

Commission errors
How can we detect and correct actions that seem formally well but semantically are incorrect?

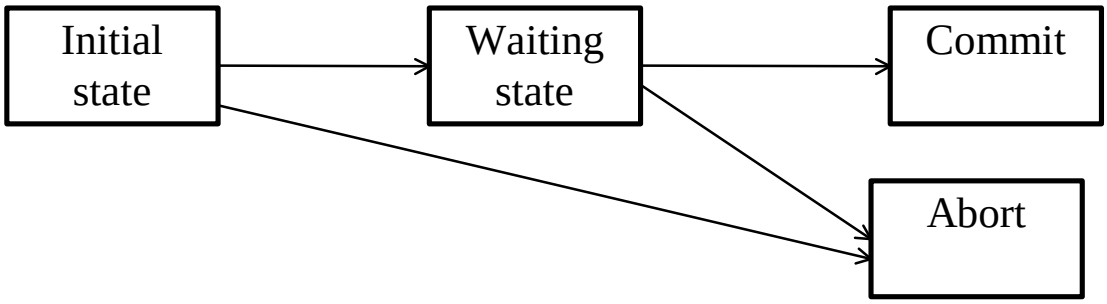
Nonblocking Commitment Protocols

Blocking commitment protocol:
in the case of failure, the participants should wait until the failure is repaired.
Pending state : the site waits for decision from the coordinator

State diagram of the 2PC
coordinator site:



participant site:



Every state is associated with input/output messages
input message → state transition → output message

If the failure occurs after input message, the site remains in the previous state, the input message will be lost. No output message will be generated.
If the failure occurs after the state transition, the site has a new state, but no output is generated.

The 3-Phase-Commitment Protocol

The new protocol is needed to reduce the blocking effect of the 2PC.

Failure with blocking:
We assume both the coordinator and a participant has a failure. If only the coordinator would be in a failure state, one of the participants could replace it to resume the transaction. But in this case it is not possible. We assume, that all of the working participants are in ready to commit state.

The global transaction may
- aborted if the failed site aborted yet (but the others no received message yet)
- commited if the failed site made a commit yet (as he received the message from the coordinator, but the others no yet)

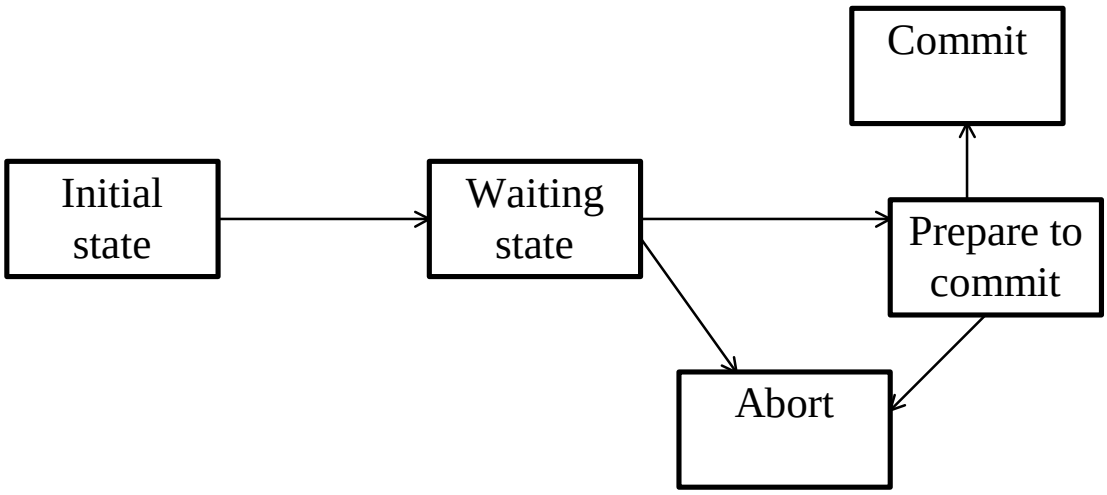
The others have to wait until the failed site can be repaired.

To resolve this type of problem, a new commitment protocol was introduced, the *3-phase commitment protocol*.

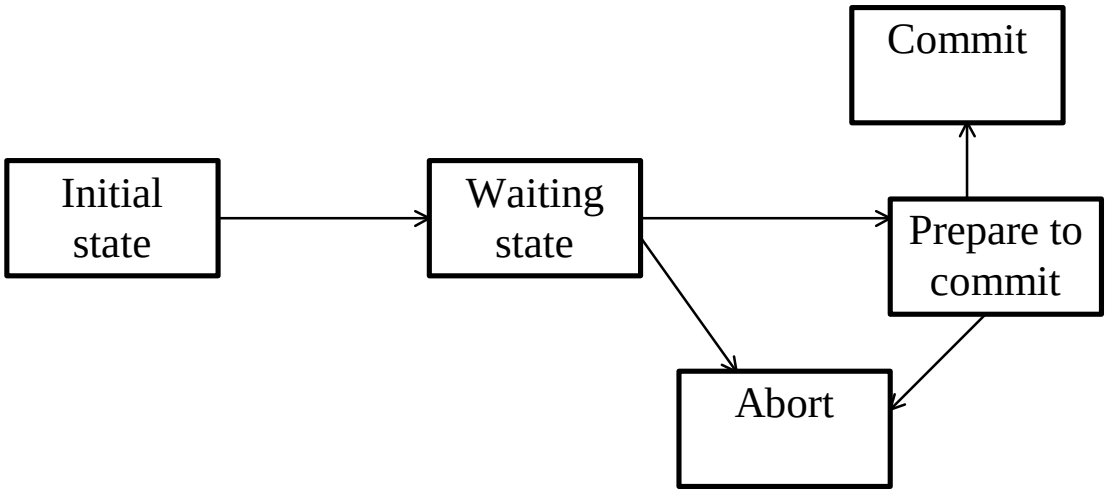
The 3PC contains and additional state, phase between the ready to commit state and the commited state. This state is called *prepared to commit state*.

Using the 3CP protocol, the failed site is in this PtCS state. This state can be revoked unlike the Commited state in 2PC.
The 3-Phase-Commitment Protocol

State diagram of the 3PC[ ]
coordinator site:



participant site:



Main steps of execution:

- Every site receives the task from the coordinator
- Coordinator and participants are in uncertain state
- Every participant performs the received tasks with success or with failure

The 3-Phase-Commitment Protocol

- Every participant gets in ready or abort state, and notifies the coordinator
- If the coordinator receives an abort message, it gets into abort state
- If every message is ready message, it gets into before commitment state, and send messages to participants
- Participants get into prepare to commit state, and send message to the coordinator
- If coordinator receives OK message from every participants, it gets into the Commit state, and sends Commit message every participants.
- The participants after receiving the message perform the Commit statement

Avoiding blocking

If none of the operating participants received the prepare to commit message, the participants can abort because there can not be a participant (that may be at this time unavailable) which executed a commit. Thus every participants can abort the local transaction.

The failed participant will abort the interrupted transaction after the system restart.
During the third phase, when the coordinator fails during the prepare to commit state. Every participants are waiting in the prepare to commit state. Solution: a new coordinator is elected.
Termination protocol for 3PC

Rules of termination:
- If at least one operational participant has not entered the prepare to commit state then the transaction can be safely terminated
- If at least one operational participant has entered the prepare to commit state then the transaction can be safely committed

The preconditions of the two rules overlap each others, thus there allows different alternatives

Protocol types:
- progressive : it tries always to commit when it is possible
- nonprogressive : it tries to abort when it is possible
- centralized : there is one coordinator
- decentralized : it allows local decisions too

Reenterable protocols:
In the case of coordinator failure the process can be continued after the election of a new coordinator. The new coordinator can replace the old one without any loss of functionality.

Strategy of the election:
There may be several variants
Example : priority order
Every participant has a priority value, the sites are totally ordered by this priority value.
After failure of the coordinator, the participant with highest available priority will replace it.
Termination protocol for 3PC

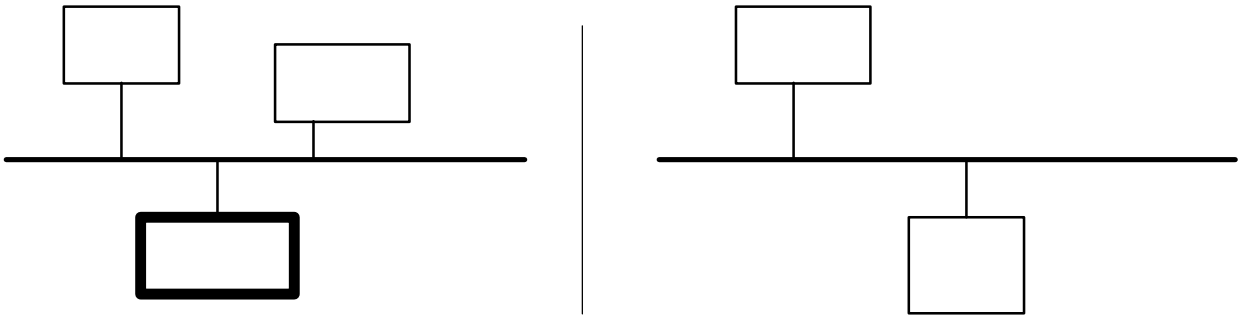
A centralized, nonprogressive protocol:
After failure of the coordinator, the activity depends on the current state of the new elected coordinator:
- If it is in the prepare to commit state, it sends messages to every participants to enter this state
- If it is in the commit state, it sends messages to every participants to commit the local transactions
- If it is in the abort state, it sends messages to every participants to abort the local transactions
- If it is in the commit state, it sends messages to every participants to commit the local transactions
- otherwise it sends messages to every participants to abort the local transactions

Modifications to make the protocol progressive
In the otherwise case, the new coordinator will ask the participants to send their state descriptors. Depending on the descriptor value, the coordinator sends an abort or commit message to the participants.

Restart protocol
Performed during the site recovery.
The site must collect information about the global state of the transaction to perform the correct activity (abort or continue).

2PC in case of Network Partition

Network Partition:
Due to a network failure the participants are separated into smaller distinct groups. There is no communication between these groups.



2PC and network partition:
If the coordinator can not get answer from some of the participants due to the network partition, the coordinator will wait until a timeout occurs and after then sends an abort message → blocking for timeout period
If the network partition happens after when the coordinator sent a message, the participants in the isolated partitions should wait until the network is repaired → blocking for unlimited time

Protocol to avoid blocking:
If one of the participants in the isolated partition received the message from the coordinator, then the new local coordinator elected by the isolated group will be able to coordinate the participants in the partition.
The new local coordinator asks every participants if they received a message from the original coordinator. If yes, it will continue the coordination based on this message.

Independent Recovery Protocols

In the previous protocols a failed participant should usually get information about the result of transaction to terminate the local transaction correctly during the restart process.

Independent recovery protocol:
The failed site can terminate the local transaction correctly during the startup without any information from the other participants.
Ideal case for failure of arbitrary site (maybe the coordinator):

- There exists an another site that can perform the coordination
- The failed site can perform the startup-recovery without any external information

In the case of network partition for every participants seem the participants in the isolated partitions to be in failure.
But the these two types of errors have different consequences.
In the case of network partition every partition should terminate in the same way.

3PC and network partition

- The network is divided into isolated groups
- The participants may be in different states (ready or prepare to commit)
- Every group elects a new local coordinator
- The termination of the group level transaction depends on the state of the new coordinator elected by the group

- Thus some of groups may commit, others may be abort the group level transaction

Proposition of the Independent Recovery Protocols

There was a lot of effort invested into research of ideal independent recovery protocol. The results are mainly negative.

Prop. 1
Independent recovery protocols exist only for single-site failures.
There exists no independent recovery protocol which is resilient to multiple-site failures.

Prop 2.
There exists no nonblocking protocol that is resilient to a network partition if messages are lost when the partition occurs.

Prop 3.
There exist nonblocking protocols which are resilient to a single network partition if all undeliverable messages are sent back to the sender.

Prop. 4
There exists no nonblocking protocol which is resilient to a multiple partition.

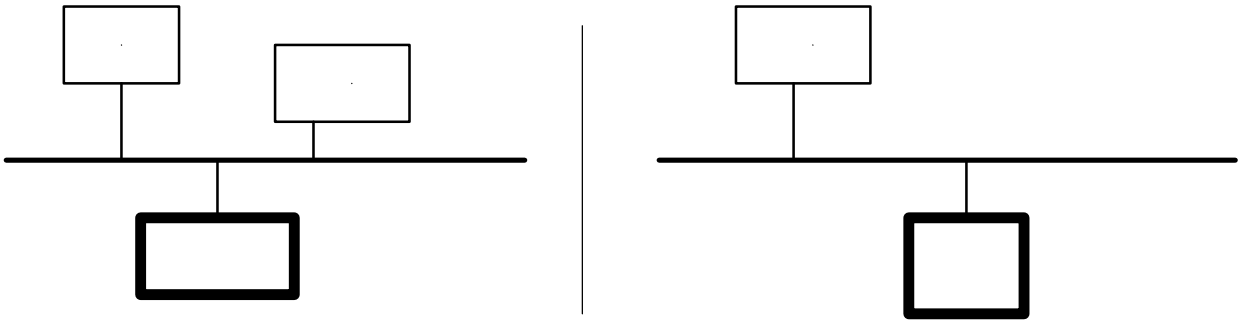
Thus it exists no a general solution of this problem.

Practical solutions: the largest partition terminates the transaction to be not blocked.
Problem: which partition is the largest?

Primary site approach and the majority approach

There are different methods to decide which partition is largest to terminate the group level transaction.

Primary site approach:
A site is designated as primary site, and the partition containing this primary site is allowed to terminate the transaction.
It is usual to denote the role of primary site to the coordinator. In this case all transactions within this partition are terminated correctly.



If the primary site differs from the coordinator site, then a 3PC termination protocol should be used to terminate all transaction of the group with the primary site.

Majority approach
Only the group containing the majority of sites can terminate the transaction. The sites in the groups may vote for aborting or for committing. The majority of sites must agree on the abort or commit before the transaction terminates.
It may happen that no group can achieve this majority.
Quorum-Based Protocols

The *quorum based protocol* is an extension of the majority protocol. A weight value is assigned to every participant, site. A weight unit is called vote. A given number of votes is required to terminate the transaction.

Rules of the quorum based protocol
- Each site i has associated with a positive vote value
- A transaction must collect Vc votes before committing
- A transaction must collect Va votes before aborting
- Va + Vc > V where V is the total sum of votes

Application of quorum-based protocol in 3PC.
The transition from the Prepare to Commit state into the Commit state can be only performed the necessary Vc votes are collected.

The Prepare to Commit state of a site means voting for Commit

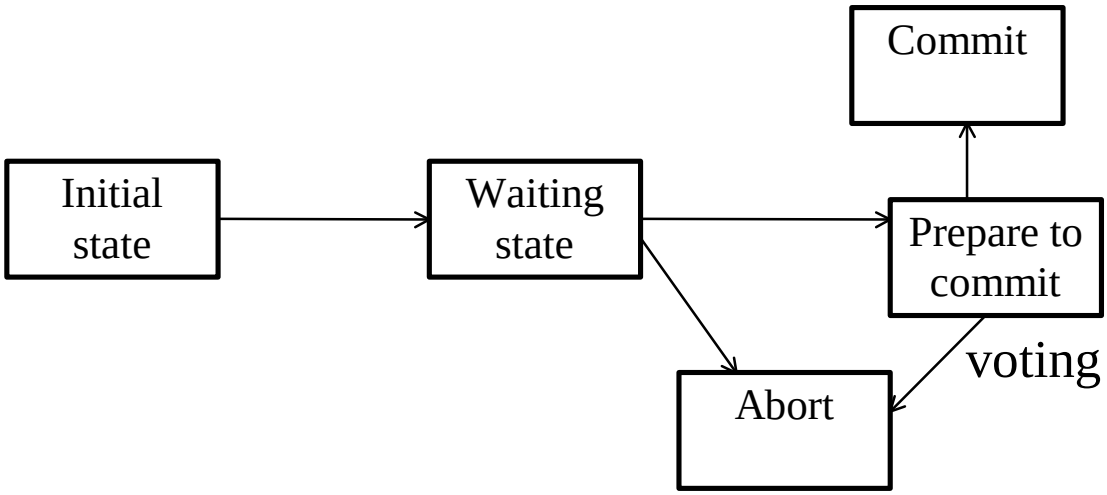
Which state should mean the voting for Abort?
The Ready state is not appropriate as it means an undecided case where the state does not know which state to choose.

The solution : a new state is introduced to vote for Abort

Ready to Abort state:
This state lies between the Ready and the Abort states. It indicates that the site agrees to abort the transaction.

Quorum-Based Protocols

The termination depends on the result of the voting. Thus from the Prepared states can be transited into both direction.



Rules of operation:
- if at least one site has committed send a Commit message to the participants
- if at least one site has aborted send an Abort message to the participants
- if the number of votes of sites in Prepare to Commit state (Nc) is greater then or equal to Vc, send a Commit message
- if the number of votes of sites in Prepare to Abort state (Na) is greater then or equal to Va, send an Abort message
- if the Nc plus the number of votes of sites in Ready state is greater then or equal to Vc, send a Prepare to Commit message
- if the Na plus the number of votes of sites in Ready state is greater then or equal to Va, send a Prepare to Abort message

Transaction management in the case of failure

What is the requirement to perform a transaction regarding site failures?

The site failures may restrict the data access.

Let's take a T transaction. The data accessed by the transaction may be subdivided into two groups : read-only and write data.
RS(T) - read set of T
WS(T) - write set of T
A failure may restrict the access to both types of data elements. But the two types of errors can be managed differently.

Access failure to RS(T) : fatal error, the transaction is aborted
Access failure to WS(T) : not fatal error as the data written out is known for the transaction if it would needed.

Deferred update : as the data element to be updated is not available at prompt, the new value is stored in an intermediate structure. If the data element gets available later again, the update will deferred executed.

Nonredundant database:
Every data element is stored in only one instance, no data replication.
In this case, the failure of a site will cause the aborting of all transactions requiring read data from this site.

Redundant databases

Although the user-level redundancy causes anomalies in the applications, the redundancy controlled by the DBMS has several benefits:
- better efficiency, better access time to the data elements, less network traffic
- increase the reliability and availability

There are several aspects to be considered during the planning of the replications. The replication of $x \in RS(T)$ on a site S has the following *benefits*:

- if T runs on S, there is no network messages needed for locking, terminating.
- if T runs on S, there is a shorter access time needed.
- if T runs in partition containing S, the T may be executed instead of aborted.

The *disadvantages* of this replication:

- a T' running on other site and accessing the x should send messages to this site
- every update of x should be performed here too

The actual replication schema should depend on the

- access schema
- possible partitioning schema
- network and data access costs

Weighted majority locking

Majority locking:
A transaction can read or write an object in only that case when it can lock the majority of the item replications.
Weighted majority locking:
A weight (vote) is assigned to every item replication. The transaction should gain a given amount of votes to access the item.

Operation rules:
- every x item has a total number of votes : $V(x)$
- every x_i replication has a vote $V(x_i)$, so that $\Sigma_i V(x_i) = V(x)$
- every x item has a read quorum $V_r(x)$ and a write quorum $V_w(x)$
- $V_r(x) + V_w(x) > V(x)$
- $V_w(x) > V(x) /2$
- a transaction can read x if it obtains lock on so many copies of x that the total votes are greater than or equal to $V_r(x)$
- a transaction can write x if it obtains lock on so many copies of x that the total votes are greater than or equal to $V_w(x)$

Only one transaction can gain the write access at the same time.

Several transaction can gain the read access at the same time.

The quorum protocol increases the network traffic as a transaction should lock the majority of the data copies resident at remote sites.

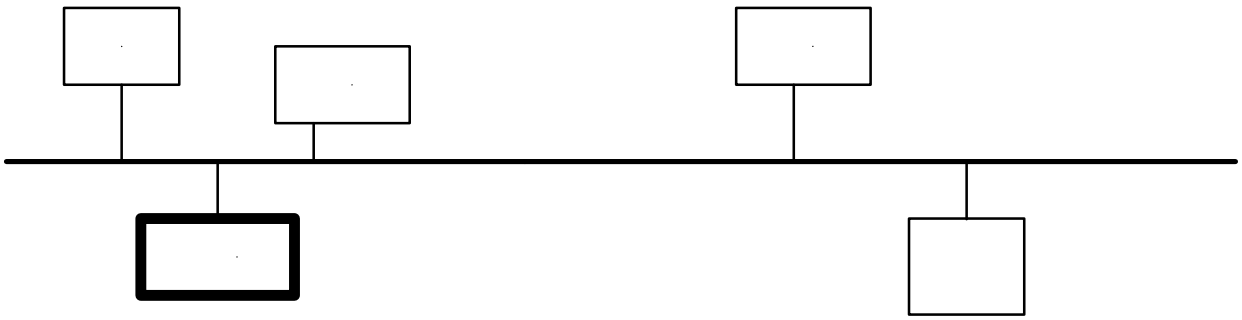
Network Partitioning and Quorum Protocol

The quorum protocol is applicable in the case of network partitioning too.

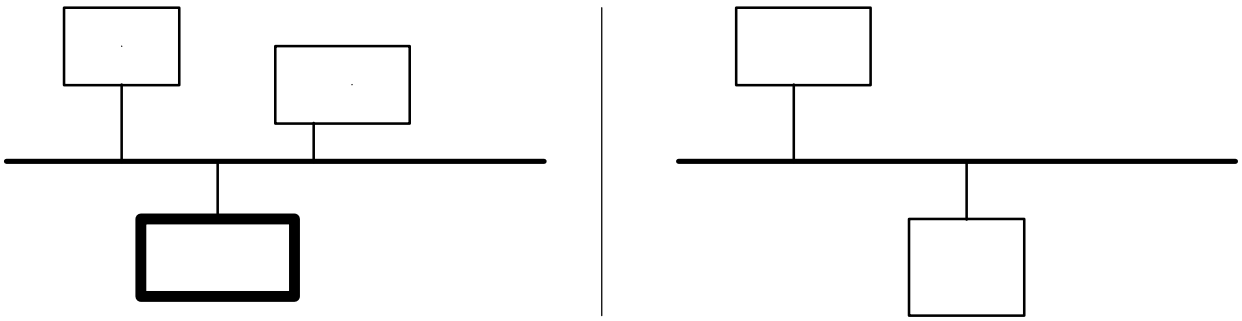
The transaction in the partition containing the majority of item copies can be processed.

Only one partition is available for a transaction.

before partitioning



after partitioning



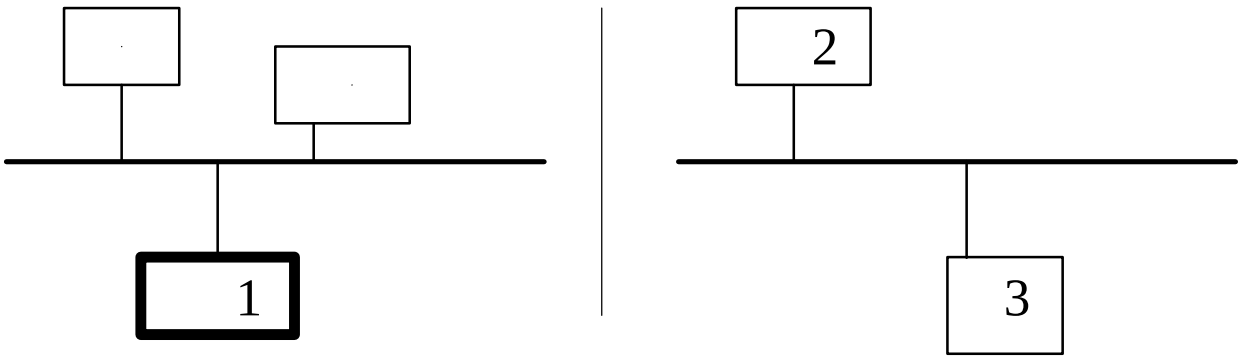
majority of votes
transaction can be processed

Consistent View of the Network

How can a site manage the actual state of the network, i.e. which other sites are available? How can it get information about the changes in the state of the network?

Usual solution of failure detection: *time-out method*
the requested answer does not arrive in time.
A site can not distinguish a failure from a network partitioning

Problem of the time-out method:
A site may delay with the answer although it is available
Consequence:
The information about the actual state of the network gets incorrect and inconsistent.



The message from site 2 to site 1 is delayed.
The site 1 registers site 2 as unavailable
The site 3 registers site 2 as available

There should be used such a state registering method which provides consistent and correct information.

State Table-Based Monitoring of the Network

State table:
Every site stores a table that contains the actual state of the network from the viewpoint of this site. The table registers for every site whether it is up (available) or down.

The different sites may have different state-views due to the network partitioning.

The consistency can be provided only among the sites belonging to the same network partition: a site and all of the sites registered as up in its table should have the same state table value.

How is the consistency provided?
If we would assume that the state table would filled in separately, it would led to inconsistency again.

Controlled state observation:
Every site has a controller site which will check whether the site is up or not. The other sites should accept this view.

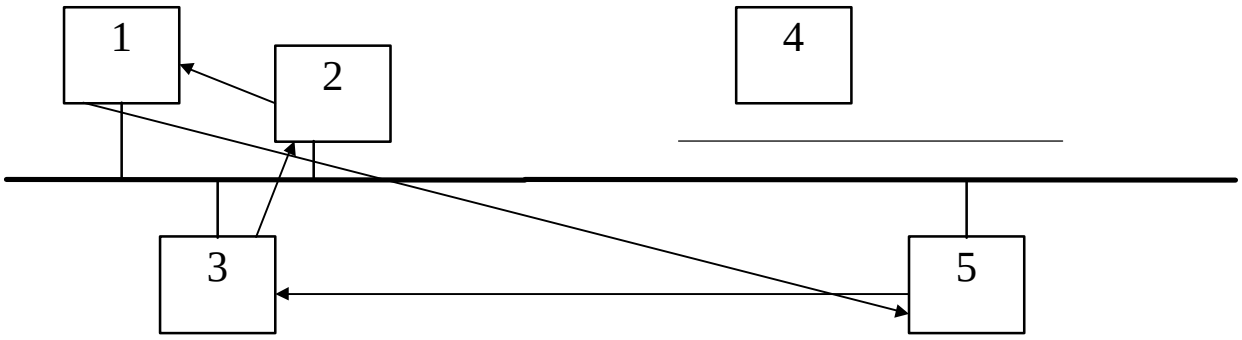
The controller uses the time-out method to detect a site failure or partitioning. But instead of sending request to the controlled site, this site should send regularly an I-am-up message to the controller. If this message is delayed, the site is recorded as down.

State Table-Based Monitoring of the Network

After the recovery of a site, the site has to send again the 'I-am-up' message to its controller.
The controller will update the state table after receiving the message.

How is the controller assigned to a site?

Circular ordering method:
The sites are totally ordered by any arbitrary method. So we can assign different integer numbers to the sites. Let us denote then with S_1,S_2 and so on. The controller of the site S_i is the site S_{i-1}. The controller of the first site is the last site.



If a site has failure then its controller function will also be down. To continue the work, the failed controller should be replaced by an another controller.
The new controller will be the next available ancestor in the ordered circular list. This new controller is responsible for every unavailable direct predecessor sites too.

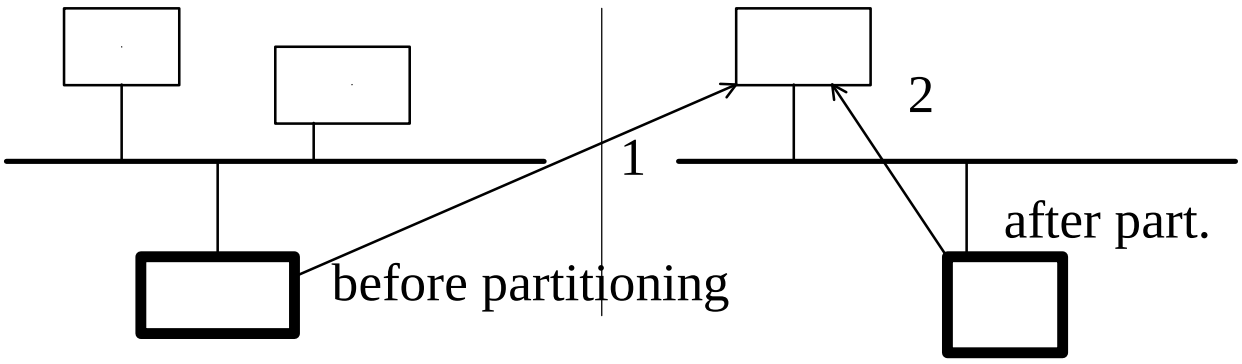
State Table-Based Monitoring of the Network

How can the state-tables of the controllers synchronized. Every controller is responsible for a subset of sites. The state values for this subsets of sites should be propagated to the other sites too.

Broadcasting method:
If the state table is updated at a controller site, this site sends messages to the other sites about the modifications.
As every sites should be informed the message is sent in a form of broadcast.
The sites receiving this broadcast message will update their own state table based on the new information.

Using this method every sites within a network partition will have the same state-table.

Problem of broadcasting:
In the case of partitioning, a site may receive inconsistent messages about the same site (from different partitions before or after the partitioning occurs).
To solve this kind of concurrency, every broadcast is assigned an unique time-stamp. The messages are processed in this order.



Inconsistency

Reason of inconsistency:
In the protocols mentioned before there is no risk of inconsistency. The inconsistency may occur when the different copies of the same date item are updated by such values that would not be allowed according to the integrity rules.
The inconsistency may occur if we allow to process a transaction in more than one partitions in the case of network partitioning.
Some systems allow this risk in order to increase the availability

Management of inconsistency:
The system should perform the following steps after repairing the partitioning failure:
- *detection* phase: detecting the occurrences of inconsistency, which data items are affected by the inconsistency.
- *resolution* phase: eliminating the inconsistent values and assignment of a reasonable value to the affected data items.

Problems in inconsistency management:
It is impossible to develop a general method which could detect and resolute the inconsistencies exactly for all types of applications.

Detection of inconsistencies

The detection rule:
The consistency can be ensured only if every data item is updated by only one transaction copy after the occurrence of the network partitioning.

Why can not the rule base on the comparison of the values at the different item copies?
Because both the equality and the inequality of the values can mean inconsistency:
- salary assignment : equality is required
- airline reservation : inequality is required

Method of inconsistency detection:
We denote the item copies in a selected partition or main partition as *master copies*.
The copies contained in the other partitions are called *isolated copies*.
For each data item copy is a couple of version number registered to denote the sequence of updates. After every update the corresponding version number increases by one.
Original version number:
The version number used by the normal state, i.e. before the partitioning occurred.
Current version number:
The version number used by the partitioned state, i.e. after the partitioning occurred.

Detection of inconsistencies

We take an example with four copies

Initial version numbers:
V1(0,0), V2(0,0),V3(0,0), V4(0,0)
After an update in normal status:
V1(0,1), V2(0,1),V3(0,1), V4(0,1)
After partitioning the copies No 1,3 remain in the master partition, No 2 and No 4 get in different partitions.
V1(0,1), V2(1,1),V3(0,1), V4(1,1)

V1(0,1), V2(1,1),V3(0,1), V4(1,1) : no update occurred, consistent

V1(0,2), V2(1,1),V3(0,2), V4(1,1) : update occurred in one partition, consistent

V1(0,1), V2(1,2),V3(0,1), V4(1,1) : update occurred in one partition, consistent

V1(0,2), V2(1,2),V3(0,2), V4(1,1) : update occurred in more than one partition, inconsistent

V1(0,2), V2(1,2),V3(0,2), V4(1,2) : update occurred in more than one partition, inconsistent

Resolution of Inconsistencies

The resolution is always application depending. Sometimes the eliminating of the inconsistent values is not desirable.

There exists no general resolution protocol.

The application should be extended with the application specific resolution policy to avoid the inconsistency.

Avoiding inconsistency:
The application contains special methods to manage the different transaction copies.

Airline reservation system:
In the case of network partitioning, to every partition is an amount of available reservations assigned.
Every transaction copy can consume this amount of reservations.
As the total sum of amounts is less then the total available reservations, every local reservation can be kept valid after the repairing of the network too.

Cold Restart

Cold restart:
After a fatal failure of the system, the actual consistent state of the DBMS is lost. An archived older consistent state is to be recovered.

As a DDBMS has global consistent state, a site cannot change their state independently from the other sites.
If one site has to make cold restart then all of the other sites must make cold restart.

Local failure - global recovery

Consistent global state:
- Atomicity concept:
For each transaction T, the state contains the updates performed by all subtransactions of T at any site (T is contained in the global state), or it does not contain any of them.
- Serializability concept:
If the T transaction is contained in the global state, then all conflicting transactions which have preceded T in the serialization order, are also contained in the global state.

The duration in the ACID concept can not be guarantied as the effect of some transactions will be lost due to the fatal failure. But all of the transaction before a given point of time are recovered. The consistency requirements are met as recovery replies the earlier consistent transactions.
Checkpoint in recovery

Checkpoint:
A mechanism to provide global consistent state. The checkpoint denotes a point of time when a snapshot is taken about a consistent state of the database system.

A *global checkpoint* is a set of local checkpoints which are performed at all sites of the system and are synchronized by the following condition:
if a subtransaction of a transaction T is contained in the local checkpoint at some time, then all other subtransaction of T must be contained in the corresponding local checkpoints at other sites.

In the case of cold restart
- take the latest available local checkpoint
- take the corresponding global checkpoint and reestablish the local states at every sites according to this global checkpoint

Generation of global checkpoint:
The snapshot, the checkpoint must be performed when there is no active transaction. The exact solution needs a lot of idle time as site A should wait for site B.

Loosely synchronized checkpoints:
The sites can perform the checkpoint within a given interval. Every checkpoint is identified by an order number. The TM should guarantee that if a T finishes before Ci then all of its subtransactions should finishes before the local Ci.