Hệ thống phân tán

Khả năng chịu lỗi

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Khả năng chịu lỗi - Fault Tolerance

- Recovery: bringing back the failed node in step with other nodes in the system.
- Fault Tolerance: Increase the availability of a service or the system in the event of failures. Two ways of achieving it:
  - *Masking failures*: Continue to perform its specified function in the event of failure.
  - *Well defined failure behavior*: System may or may not function in the event of failure, but can facilitate actions suitable for recovery.
    - (e.g.,) effect of database transactions visible only if committed to by all sites. Otherwise, transaction is undone without any effect to other transactions.
- Key approach to fault tolerance: *redundancy*. e.g., multiple copies of data, multiple processes providing same service.
- Topics to be discussed: commit protocols, voting protocols.

Hành động nguyên tử

- Example: Processes P1 & P2 share a data named X.
  - P1: ... lock(X); X:= X + Z; unlock(X); ...
  - P2: ... lock(X); X := X + Y; unlock(X); ...
- Updating of X by P1 or P2 should be done *atomically* i.e., without any interruption.
- Atomic operation if:
  - the process performing it is not aware of existence of any others.
  - the process doing it does not communicate with others during the operation time.
  - No other state change in the process except the operation.
  - Effects on the system gives an impression of indivisible and perhaps instantaneous operation.

Committing

- A group of actions is grouped as a transaction and the group is treated as an atomic action.
- The transaction, during the course of its execution, decides to commit or abort.
- *Commit*: guarantee that the transaction will be completed.
- *Abort*: guarantee *not* to do the transaction and erase any part of the transaction done so far.
- *Global atomicity*: (e.g.,) A distributed database transaction that must be processed at every or none of the sites.
- *Commit protocols*: are ones that enforce global atomicity.

2-phase Commit Protocol

- Distributed transaction carried out by a coordinator + a set of cohorts executing at different sites.
- Phase 1:
  - At the coordinator:
    - Coordinator sends a COMMIT-REQUEST message to every cohort requesting them to commit.
    - Coordinator waits for reply from all others.
  - At the cohorts:
    - On receiving the request: if the transaction execution is successful, the cohort writes UNDO and REDO log on stable storage. Sends AGREED message to coordinator.
    - Otherwise, sends an ABORT message.
- Phase 2:
  - At the coordinator:
    - All cohorts agreed? : write a COMMIT record on log, send COMMIT request to all cohorts.

2-phase Commit Protocol ...

- Phase 2...:
  - At the coordinator...:
    - Otherwise, send an ABORT message
    - Coordinator waits for acknowledgement from each cohort.
    - No acknowledgement within a timeout period? : resend the commit/abort message to that cohort.
    - All acknowledgements received? : write a COMPLETE record to the log.
  - At the cohorts:
    - On COMMIT message: resources & locks for the transaction released. Send Acknowledgement to the coordinator.
    - On ABORT message: undo the transaction using UNDO log, release resources & locks held by the transaction, send Acknowledgement.

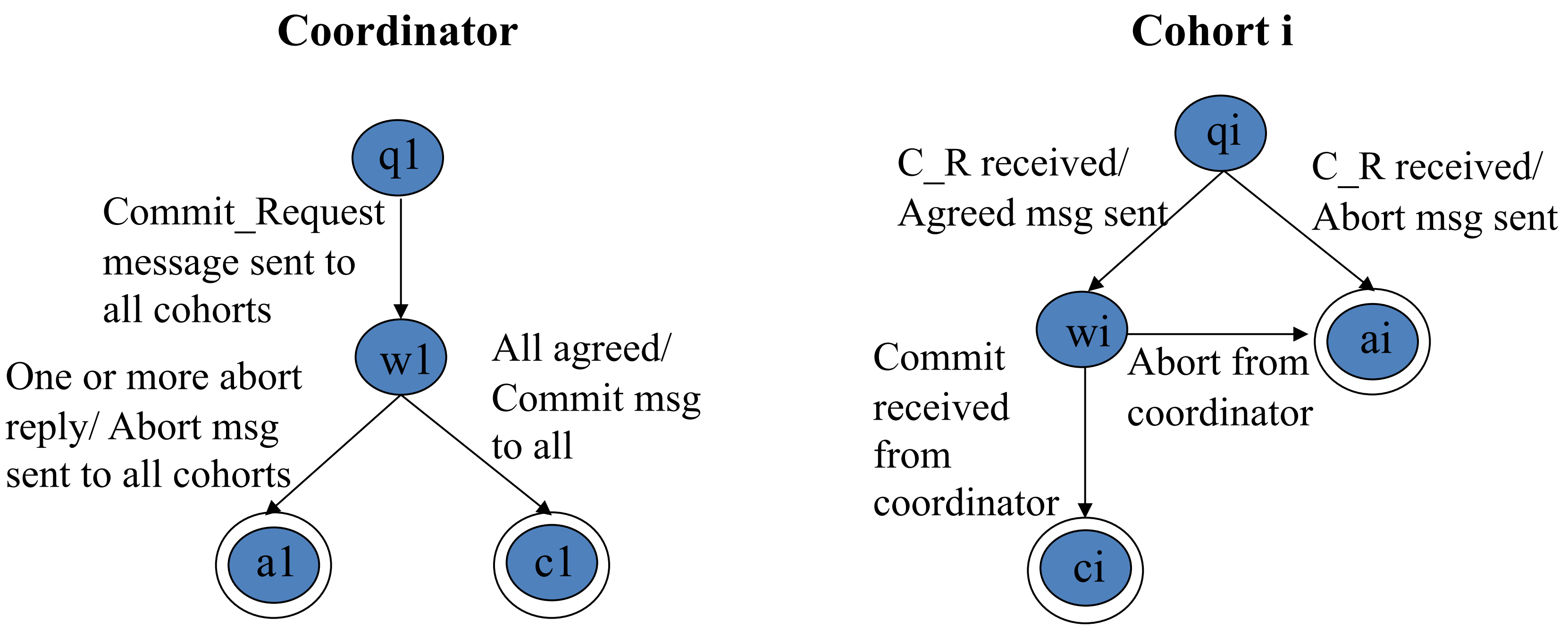
Xử lý lỗi

- 2-phase commit protocol handles failures as below:
  - If coordinator crashes before writing the COMMIT record:
    - on recovery, it will send ABORT message to all others.
    - Cohorts who agreed to commit, will simply undo the transaction using the UNDO log and abort.
    - Other cohorts will simply abort.
    - All cohorts are blocked till coordinator recovers.
  - Coordinator crashes after COMMIT before writing COMPLETE
    - On recovery, broadcast a COMMIT and wait for ack
  - Cohort crashes in phase 1? : coordinator aborts the transaction.
  - Cohort crashes in phase 2? : on recovery, it will check with the coordinator whether to abort or commit.
- Drawback: blocking protocol. Cohorts blocked if coordinator fails.
  - Resources and locks held unnecessarily.

2-phase commit: State Machine

- Synchronous protocol: all sites proceed in rounds, i.e., a site never leads another by more than 1 state transition.
- A state transition occurs in a process participating in the 2-phase commit protocol whenever it receives/sends messages.
- States: q (idle or querying state), w (wait), a (abort), c (commit).
- When coordinator is in state q, cohorts are in q,w,a.
- Coordinator in w -> cohort can be in q, w, or a.
- Coordinator in a/c -> cohort is in w or a/c.
- A cohort in a/c: other cohorts may be in a/c or w.
- A site is never in c when another site is in q as the protocol is synchronous.

2-phase commit: State Machine...


Coordinator
q1
Commit_Request message sent to all cohorts
w1
One or more abort reply/ Abort msg sent to all cohorts
All agreed/ Commit msg to all
a1
c1
Cohort i
qi
C_R received/ Agreed msg sent
C_R received/ Abort msg sent
wi
ai
Abort from coordinator
Commit received from coordinator
ci

Hạn chế

- Drawback: blocking protocol. Cohorts blocked if coordinator fails.
  - Resources and locks held unnecessarily.
- Conditions that cause blocking:
  - Assume that only one site is operational. This site cannot decide to abort a transaction as some other site may be in commit state.
  - It cannot commit as some other site can be in abort state.
  - Hence, the site is blocked until all failed sites recover.

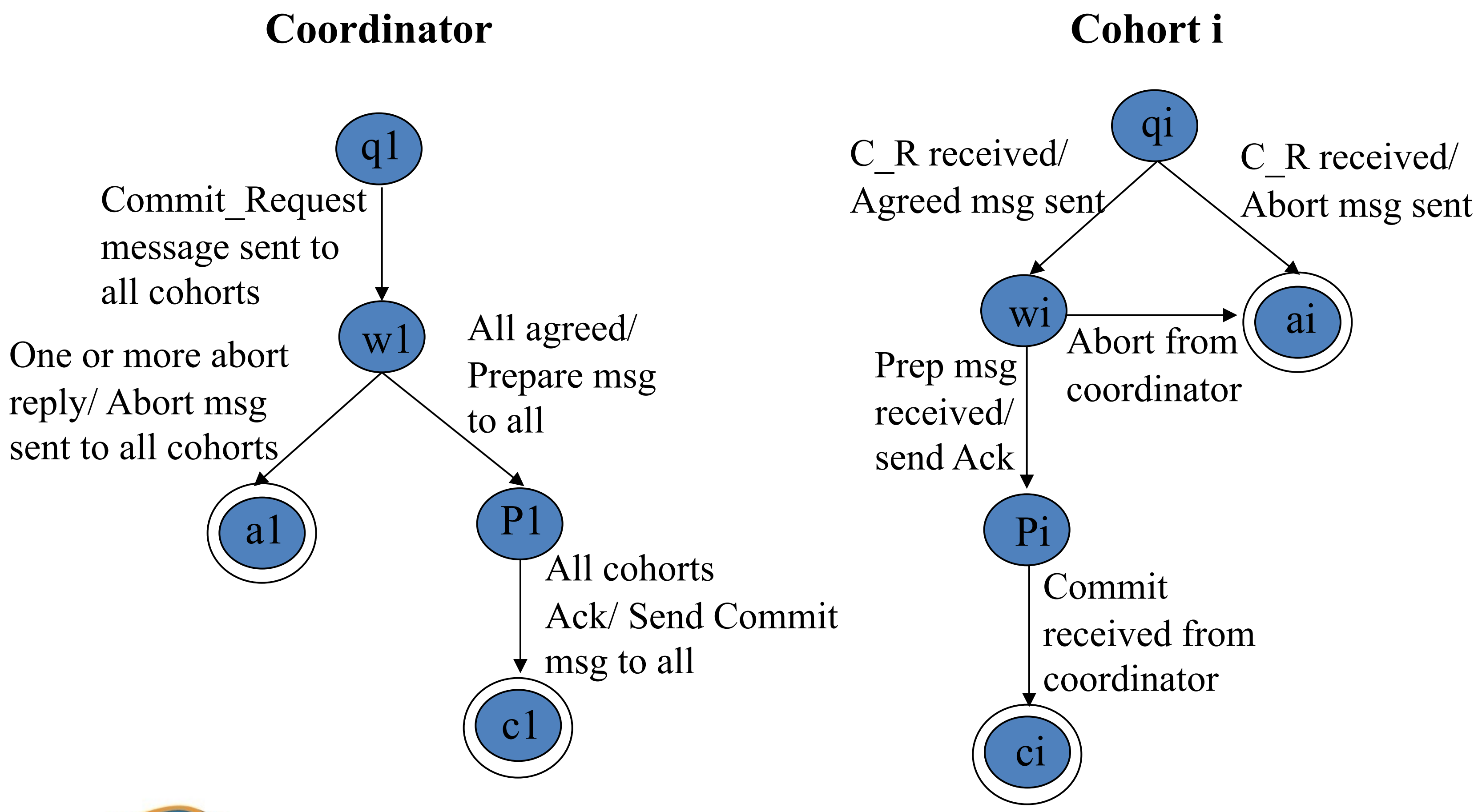
Nonblocking Commit

- Nonblocking commit? :
  - Sites should agree on the outcome by examining their local states.
  - A failed site, upon recovery, should reach the same conclusion regarding the outcome. Consistent with other working sites.
  - *Independent recovery*: if a recovering site can decide on the final outcome based solely on its local state.
  - A nonblocking commit protocol can support independent recovery.
- Notations:
  - *Concurrency set*: Let *Si* denote the state of the site i. The set of all the states that may be concurrent with it is concurrency set (C(si)).
    - (e.g.,) Consider a system having 2 sites.If site 2's state is w2, then *C(w2) = {c1, a1, w1}. C(q2) = {q1, w1}. a1, c1* not in *C(q2)* as 2-phase commit protocol is synchronous within 1 state transaction.
  - *Sender set*: Let *s* be any state, *M* be the set of all messages received in *s*. Sender set, *S(s)* = {*i* | site *i* sends *m* and *m* in *M*}

3-phase Commit

- *Lemma:* If a protocol contains a local state of a site with both abort and commit states in its concurrency set, then under independent recovery conditions it is not resilient to an arbitrary single failure.
- In previous figure, C(w2) can have both abort and commit states in the concurrency set.
- To make it a non-blocking protocol: introduce a buffer state at both coordinator and cohorts.
- Now, C(w1) = {q2, w2, a2} and C(w2) = {a1, p1, w1}.

3-phase commit: State Machine


Coordinator
q1
Commit_Request message sent to all cohorts
w1
One or more abort reply/ Abort msg sent to all cohorts
All agreed/ Prepare msg to all
a1
P1
All cohorts Ack/ Send Commit msg to all
c1
Cohort i
qi
C_R received/ Agreed msg sent
C_R received/ Abort msg sent
wi
ai
Abort from coordinator
Prep msg received/ send Ack
Pi
Commit received from coordinator
ci

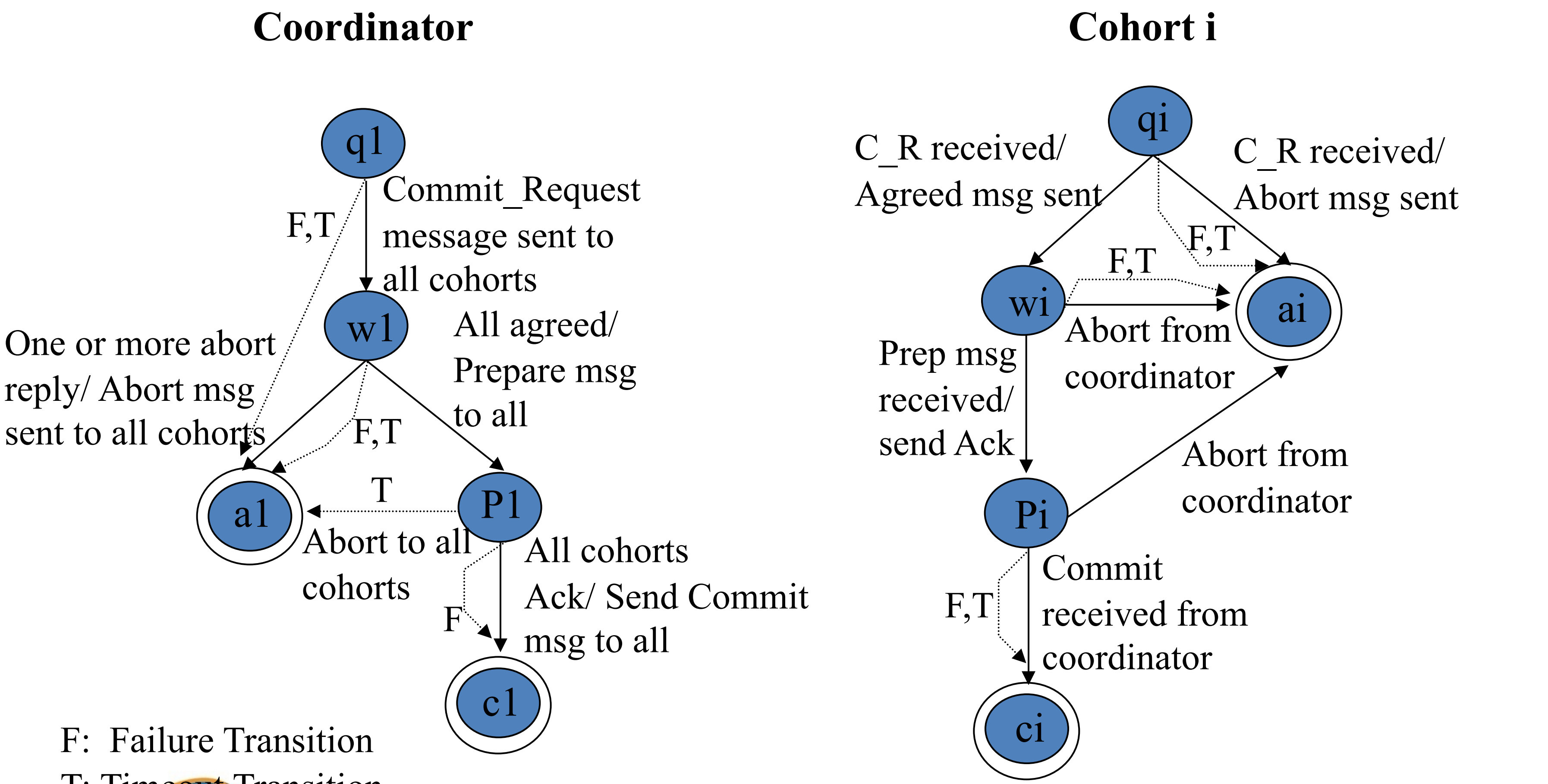
Bước chuyển Failure, Timeout

- A *failure transition* occurs at a failed site at the instant it fails or immediately after it recovers from the failure.
  - *Rule for failure transition*: For every non-final state *s* (i.e., *qi, wi*, *pi*) in the protocol, if *C(s)* contains a commit, then assign a failure transition from *s* to a commit state in its FSA. Otherwise, assign a failure transition from *s* to an abort state.
  - *Reason*: *pi* is the only state with a commit state in its concurrency set. If a site fails at *pi*, then it can commit on recovery. Any other state failure, safer to abort.
- If site *i* is waiting on a message from *j*, *i* can time out. *i* can determine the state of *j* based on the expected message.
- Based on *j*'s state, the final state of *j* can be determined using failure transition at *j*.

Bước chuyển Failure, Timeout

- This can be used for incorporating *Timeout transitions* at *i*.
  - *Rule for timeout transition*: For each nonfinal state *s*, if site *j* in *S(s),*and site *j* has a failure transition from *s* to a commit (abort) state, then assign a timeout transition from *s* to a commit (abort) state.
  - *Reason*:
    - Failed site makes a transition to a commit (abort) state using failure transition rule.
    - So, the operational site must make the same transition to ensure that the final outcome is the same at all sites.

3-phase commit + Failure Trans.


Coordinator
q1
Commit_Request message sent to all cohorts
F,T
w1
One or more abort reply/ Abort msg sent to all cohorts
All agreed/ Prepare msg to all
F,T
a1
T
P1
Abort to all cohorts
All cohorts Ack/ Send Commit msg to all
F
c1
Cohort i
qi
C_R received/ Agreed msg sent
C_R received/ Abort msg sent
F,T
F,T
wi
ai
Abort from coordinator
Prep msg received/ send Ack
Abort from coordinator
Pi
Commit received from coordinator
F,T
ci

F: Failure Transition
T: Timeout Transition
F,T: Failure/Timeout

Nonblocking Commit Protocol

- Phase 1:
  - First phase identical to that of 2-phase commit, except for failures.
  - Here, coordinator is in w1 and each cohort is in a or w or q, depending on whether it has received the commit_request message or not.
- Phase 2:
  - Coordinator sends a Prepare message to all the cohorts (if all of them sent Agreed message in phase 1).
  - Otherwise, it will send an Abort message to them.
  - On receiving a Prepare message, a cohort sends an acknowledgement to the coordinator.
    - If the coordinator fails before sending a Prepare message, it aborts the transaction on recovery.
    - Cohorts, on timing out on a Prepare message, also aborts the transaction.

Nonblocking Commit Protocol

- Phase 3:
  - On receiving acknowledgements to Prepare messages, the coordinator sends a Commit message to all cohorts.
  - Cohort commits on receiving this message.
    - Coordinator fails before sending commit? : commits upon recovery.
    - So cohorts on Commit message timeout, commit to the transaction.
    - Cohort failed before sending an acknowledgement? : coordinator times out and sends an abort message to all others.
    - Failed cohort aborts the transaction upon recovery.
- Use of buffer state:
  - (e.g.,) Suppose state pi (in cohort) is not present. Let coordinator wait in state p1 waiting for ack. Let cohort 2 (in w2) acknowledge and commit.
  - Suppose cohort 3 fails in w3. Coordinator will time out and abort. Cohort 3 will abort on recovery. Inconsistent with cohort 2.

Hạn chế: Commit Protocols

- No protocol using the above independent recovery technique for simultaneous failure of more than 1 site.
- The above protocol is also not resilient to network partitioning.
- Alternative: Use voting protocols.
- Basic idea of voting protocol:
  - Each replica assigned some number of votes
  - A majority of votes need to be collected before accessing a replica.
  - Voting mechanism: more fault tolerant to site failures, network partitions, and message losses.
- Types of voting schemes:
  - Static
  - Dynamic

Mô hình bầu cử tĩnh

- System Model:
  - File replicas at different sites. File lock rule: either one writer + no reader or multiple readers + no writer.
  - Every file is associated with a version number that gives the number of times a file has been updated.
  - Version numbers are stored on stable storage. Every successful write updates version number.
- Basic Idea:
  - Every replica assigned a certain number of votes. This number stored on stable storage.
  - A read or write operation permitted if a certain number of votes, called *read quorum* or *write quorum*, are collected by the requesting process.
- Voting Algorithm:
  - Let a site i issue a read or write request for a file.
  - Site i issues a Lock_Request to its local lock manager.

Bầu cử tĩnh...

- Voting Algorithm...:
  - When lock request is granted, i sends a *Vote_Request* message to all the sites.
  - When a site *j* receives a Vote_Request message, it issues a Lock_Request to its lock manager. If the lock request is granted, then it returns the version number of the replica (*VNj*) and the number of votes assigned to the replica (*Vj*) at site *i*.
  - Site *i* decides whether it has the quorum or not, based on replies received within a timeout period as follows.
    - For read requests, *Vr* = Sum of *Vk, k* in *P*, where *P* is the set of sites from which replies were received.
    - For write requests, *Vw* = Sum of *Vk, k* in *Q* such that:
      - *M* = max{*VN j*: *j* is in *P*}
      - *Q* = {*j* in *P* : *VNj* = *M*}
      - Only the votes of the current (version) replicas are counted in deciding the write quorum.

Bầu cử tĩnh...

- Voting Algorithm...:
  - If i is not successful in getting the quorum, it issues a Release _Lock to the lock manager & to all sites that gave their votes.
  - If i is successful in collecting the quorum, it checks whether its copy of file is current (*VNi* = *M*). If not, it obtains the current copy.
  - If the request is read, *i* reads the local copy. If write, *i* updates the local copy and *VN*.
  - i sends all updates and *VNi* to all sites in Q, i.e., update only current replicas. *i* sends a *Release_Lock* request to its lock manager as well as those in *P*.
  - All sites on receiving updates, perform updates. On receiving Release_Lock, releases lock.
- Vote Assignment:
  - Let v be the total number of votes assigned to all copies. Read & write quorum, r & w, are selected such that: r + w > v; w > v/2.

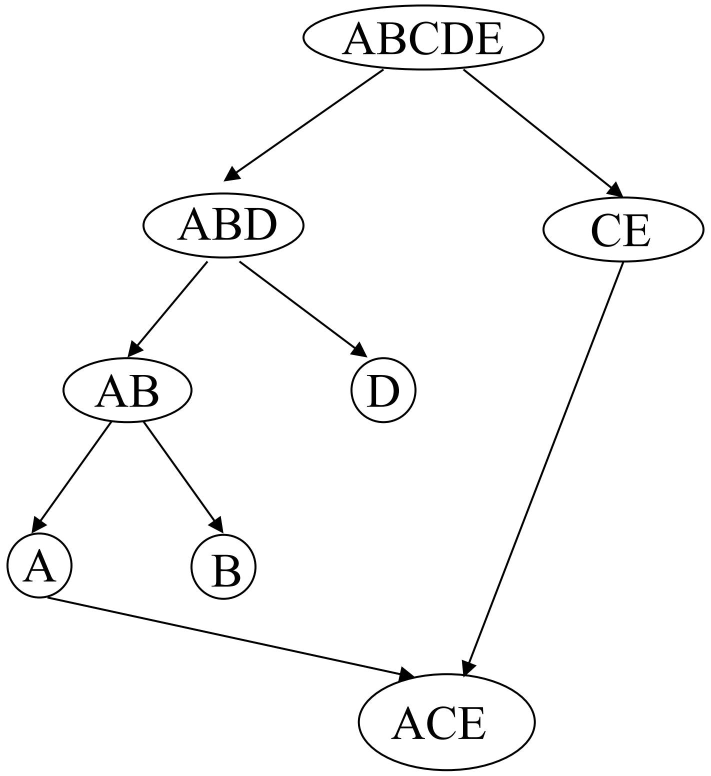
Bầu cử tĩnh...

- Vote Assignment ...:
  - Above values are determined so that there is a non-null intersection between every read and write quorum, i.e., at least 1 current copy in any reading quorum gathered.
  - Write quorum is high enough to disallow simultaneous writes on 2 distinct subset of replicas.
  - The scheme can be modified to collect write quorums from non-current replicas. Another modification: obsolete replicas updated.
  - (e.g.,) System with 4 replicas at 4 sites. Votes assigned: V1 = 1, V2 = 1, V3 = 2, & V4 = 1.
    - Let disk latency at S1 = 75 msec, S2 = 750 msec, S3 = 750 msec. & S4 = 100 msec.
    - If r = 1 & w = 5, and read access time is 75 ms and write access is 750 msec.

Bầu cử động

- Assume in the above example, site 3 becomes unreachable from other sites.
- Sites 1, 2, & 4 can still collect a quorum, however, Site 3 cannot.
- Another partition in {1,2,4} will make any site unavailable.
- Dynamic voting: adapt the number of votes or the set of sites that can form a quorum, to the changing state of the system due to sites & communication failures.
- Approaches:
  - *Majority based approach*: set of sites change with system state. This set can form a majority to allow access to replicated data.
  - *Dynamic vote reassignment*: number of votes assigned to a site changes dynamically.

Tiếp cận dựa trên số đông



- Figure indicates the partitions and (one) merger that takes place
- Assume one vote per copy.
- Static voting scheme: only partitions ABCDE, ABD, & ACE allowed access.
- Majority-based approach: one partition can collect quorums and the other cannot.
- Partitions ABCDE, ABD, AB, A, and ACE can collect quorums, others cannot.

Tiếp cận số đông...

- Notations used:
  - Version Number, *VNi*: of a replica at a site *i* is an integer that counts the number of successful updates to the replica at *i.* Initially set to 0.
  - Number of replicas updated, *RUi*: Number of replicas participating in the most recent update. Initially set to the total number of replicas.
  - Distinguished sites list, *DSi*,: at *i* is a variable that stores IDs of one or more sites. *DSi* depends on *RUi*.
    - *RUi* is even: *DSi* identifies the replica that is greater (as per the linear ordering) than all the other replicas that participated in the most recent update at i.
    - *RUi* is odd: *DSi* is nil.
    - *RUi* = 3: *DSi* lists the 3 replicas that participated in the most recent update from which a majority is needed to allow access to data.

Ví dụ: tiếp cận số đông

- Example:
  - 5 replicas of a file stored at sites A,B,C,D, and E.State of the system is shown in table. Each replica has been updated 3 times, RUi is 5 for all sites. DSi is nil (as RUi is odd and != 3).

	A	B	C	D	E
VN	3	3	3	3	3
RU	5	5	5	5	5
DS	-	-	-	-	-

  - B receives an update request, finds it can communicate only to A & C. B finds that RU is 5 for the last update. Since partition ABC has 3 of the 5 copies, B decides that it belongs to a distinguished partition. State of the system:

	A	B	C	D	E
VN	4	4	4	3	3
RU	3	3	3	5	5
DS	ABC	ABC	ABC	-	-

Ví dụ: tiếp cận số đông...

- Example...:
  - Now, C needs to do an update and finds it can communicate only to B. Since RUc is 3, it chooses the static voting protocol and so DS & RU are not updated. System state:

	A	B	C	D	E
VN	4	5	5	3	3
RU	3	3	3	5	5
DS	ABC	ABC	ABC	-	-

  - Next, D makes an update, finds it can communicate with B,C, & E. Latest version in BCDE is 5 with RU = 3. A majority from DS = ABC is sought and is available (i.e., BC). RU is now set to 4. RU is even, DS set to B (highest lexicographical order). System state:

	A	B	C	D	E
VN	4	6	6	6	6
RU	3	4	4	4	4
DS	ABC	B	B	B	B

Ví dụ tiếp cận số đông...

- Example...:
  - C receives an update, finds it can communicate only with B. BC has half the sites in the previous partition and has the distinguished site B (DS is used to break the tie in the case of even numbers). Update can be carried out in the partition. Resulting state:

	A	B	C	D	E
VN	4	7	7	6	6
RU	3	2	2	4	4
DS	ABC	B	B	B	B

Giao thức: dựa trên số đông

- Site *i* receives an update and executes following protocol:
  - *i* issues a Lock_Request to its local lock manager
  - Lock granted? : i issues a Vote_Request to all the sites.
  - Site *j* receives the request: *j* issues a Lock_Request to its local lock manager. Lock granted? : *j* sends the values of *VNj, RUj,* and *DSj* to *i*.
  - Based on responses received, *i* decides whether it belongs to the distinguished partition procedure.
  - *i* does not belong to distinguished partition? : issues Release_Lock to local lock manager and Abort to other sites (which will issue Release_Lock to their local lock manager).
  - *i* belongs to distinguished partition? : performs *update* on local copy (current copy obtained before update is local copy is not current). i sends a commit message to participating sites with missing updates and values of *VN, RU*, and *DS*. Issues a Release_Lock request to local lock manager.

Giao thức: dựa trên số đông

- Site *i* receives an update and executes following protocol...:
  - Site *j* receives commit message: updates its replica, *RU, VN,* & *DS*, and sends Release_Lock request to local lock manager.
- Distinguished Partition: Let P denote the set of responding sites.
  - *i* calculates *M* (the most recent version in partition), *Q* (set of sites containing version *M*), and *N* (the number of sites that participated in the latest update):
    - *M* = max{*VNj : j in P*}
    - *Q* = {*j in P : VNj* = *M*}
    - *N = RUj, j in Q*
  - |Q| > N/2 ? : i is a member of distinguished partition.
  - |Q| = N/2 ? : tie needs to be broken. Select a j in Q. If DSj in Q, i belongs to the distinguished partition. (If *RUj* is even, *DSj* contains the highest ordered site). i.e., i is in the partition containing the distinguished site.

Giao thức: dựa trên số đông

- Distinguished Partition...:
  - If *N* = 3 and if *P* contains 2 or all 3 sites indicated by *DS* variable of the site in *Q*, *i* belongs to the distinguished partition.
  - Otherwise, *i* does not belong to a distinguished partition.
- Update: invoked when a site is ready to commit. Variables are updated as follows:
  - *VNi* = *M* + 1
  - *RUi* = cardinality of P (i.e., |P|)
  - *DS*i is updated when N != 3 (since static voting protocol is used when N = 3).
    - *DSi* = K if *RUi* is even, where K is the highest ordered site
    - *DSi* = P if *RUi* = 3
- Majority-based protocol can have deadlocks as it uses locks. One needs a deadlock detection & resolution mechanism along with this.

Vượt lỗi - Failure Resilience

- Resilient process: is one that masks failures and guarantees progress despite a certain number of failures.
- Approaches:
  - Backup process:
    - A primary process + 1 or more backup processes
    - Primary process executes while backups are inactive
    - If primary fails, one of the backup processes take over the functions of the primary process.
    - To facilitate this takeover, state of the primary process is checkpointed at appropriate intervals.
    - Checkpointed state stored in a place that will be available even in the case of primary process failure.
    - Plus:
      - Little system resources are consumed by backup processes as they are inactive.

Vượt lỗi - Failure Resilience

- Resilient Approaches:
  - Backup process...:
    - Minus...:
      - Delay in takeover as (1) backups need to detect primary’s failure (mostly by timeouts of “heartbeat” messages) (2) backup needs to recompute the system state based on checkpoint.
      - Backup should not reissue IOs and resend messages already sent by the primary process. Also, messages processed by the primary after the checkpoint must be available for back up process during the recomputation.
      - Which backup to act as primary needs to be solved.
  - Replicated execution:
    - Several processes execute simultaneously. Service continues as long as one of them is available.
    - Plus:
      - Provides increased reliability and availability

Vượt lỗi - Failure Resilience

- Resilient Approaches...:
  - Replicated execution...:
    - Minus:
      - More system resources needed for all processes
      - Concurrent updates need to be handled