What type of properties are the following?

- Once you have sent a request to the server, you will receive a response within 10 seconds.

- All client requests that are not preceded by an identical request will be eventually processed.
PREVIOUSLY ON DISTRIBUTED SYSTEMS
Atomic commit

Preserve data consistency for distributed transactions in the presence of failures

- Setup
  - one coordinator
  - a set of participants
- Each process has access to a Distributed Transaction Log (DT Log) on stable storage
- Each process $p_i$ has an input value $vote_i$
  \[vote_i \in \{Yes, No\}\]
- Each process $p_i$ has an output value $decision_i$
  \[decision_i \in \{Commit, Abort\}\]
AC SPECIFICATION

AC-1: All processes that reach a decision reach the same one
AC-2: A process cannot reverse its decision after it has reached one
AC-3: The **Commit** decision can only be reached if all processes vote **Yes**
AC-4: If there are no failures and all processes vote **Yes**, then the decision must be **Commit**
AC-5: If all failures are repaired and there are no more failures, then all processes will eventually decide
AC-1: All processes that reach a decision reach the same one.

AC-2: A process cannot reverse its decision after it has reached one.

AC-3: The **Commit** decision can only be reached if all processes vote **Yes**.

AC-4: If there are no failures and all processes vote **Yes**, then the decision will be **Commit**.

AC-5: If all failures are repaired and there are no more failures, then all processes will eventually decide.

**AC-1:**
- AC-1 does not require all processes to reach a decision.
- It does not even require all correct processes to reach a decision.

**AC-4:**
- Avoids triviality.
- Allows **Abort** even if all processes have voted **Yes**.

**Note:**
- A process that does not vote **Yes** can unilaterally **Abort**.
UNCERTAINTY

- A process is in **uncertain** if it has voted **Yes** but does not have sufficient information to **Commit**.

- While uncertain, a process cannot decide unilaterally.

- **uncertainty**
  + communication failures
  
  blocking
INDEPENDENT RECOVERY

• Suppose process $p$ fails while running Atomic Commit

• If, during recovery, $p$ can reach a decision without communicating with other processes, we say that $p$ can independently recover

• total failure (= all processes fail)
  - independent recovery
    _____________________________
    blocking
A FEW CHARACTER-BUILDING FACTS

Proposition 1

If communication failures or total failures are possible, then every AC protocol may cause processes to become blocked.

Proposition 2

No AC protocol can guarantee independent recovery of failed processes.
Our first Atomic Commit protocol

2-Phase Commit (2PC)

- The simplest and most popular AC protocol
- Important assumption: synchrony
2-Phase Commit

Coordinator $c$
Participant $p_i$

I. sends VOTE-REQ to all participants
2-Phase Commit

Coordinator \( c \)  
1. sends VOTE-REQ to all participants

Participant \( p_i \)  
2. sends \( vote_i \) to Coordinator  
   if \( vote_i = \text{No} \) then  
   \( decision_i := \text{Abort} \)  
   halt
2-Phase Commit

Coordinator $c$

1. sends VOTE-REQ to all participants

Participant $p_i$

2. sends $vote_i$ to Coordinator

if $vote_i = No$ then

$decision_i := Abort$

send $Abort$ to all who voted $Yes$

halt

else

$decision_c := Commit$

send $Commit$ to all

halt

3. if (all votes are $Yes$) then

else

$decision_c := Abort$

send $Abort$ to all who voted $Yes$

halt
2-Phase Commit

Coordinator \( c \)

1. sends VOTE-REQ to all participants

2. sends \( vote_i \) to Coordinator
   - if \( vote_i = \text{No} \) then
     - \( decision_i := \text{Abort} \)
     - halt
   - else
     - \( decision_c := \text{Commit} \)
     - send Commit to all
     - \( decision_i := \text{Commit} \)
     - halt

Participant \( p_i \)

3. if (all votes are \text{Yes}) then
   - \( decision_c := \text{Commit} \)
   - send Commit to all
   - halt

4. if received Commit then
   - \( decision_i := \text{Commit} \)
   - else
     - \( decision_i := \text{Abort} \)
     - halt
Notes on 2PC

- Satisfies AC-1 to AC-4
- But not AC-5 (at least "as is")
  - A process may be waiting for a message that may never arrive
    - Use **Timeout Actions**
  - No guarantee that a recovered process will reach a decision consistent with that of other processes
    - Processes save protocol state in DT-Log

AC-5: If all failures are repaired and there are no more failures, then all processes will eventually decide
**Timeout actions**

Coordinator $c$

Participant $p_i$

**Step 2:** $p_i$ is waiting for VOTE-REQ from Coordinator

**Step 3:** Coordinator is waiting for vote from participants

**Step 4:** $p_i$ (who voted Yes) is waiting for Commit or Abort
Timeout actions

Coordinator $c$

Step 3: Coordinator is waiting for vote from participants

Participant $p_i$

Step 2: $p_i$ is waiting for VOTE-REQ from Coordinator

Since it has not cast its vote yet, $p_i$ can decide *Abort* and halt

Step 4: $p_i$ (who voted *Yes*) is waiting for *Commit* or *Abort*
TIMEOUT ACTIONS

Coordinator $c$

Step 3: Coordinator is waiting for vote from participants

Coordinator can decide **Abort**, send **Abort** to all participants who voted **Yes**, and halt

Participant $p_i$

Step 2: $p_i$ is waiting for VOTE-REQ from Coordinator

Since it has not cast its vote yet, $p_i$ can decide **Abort** and halt

Step 4: $p_i$ (who voted **Yes**) is waiting for **Commit** or **Abort**
Timeout actions

Coordinator $c$

Step 3: Coordinator is waiting for vote from participants

Coordinator can decide **Abort**, send **Abort** to all participants who voted **Yes**, and halt

Participant $p_i$

Step 2: $p_i$ is waiting for VOTE-REQ from Coordinator

Since it has not cast its vote yet, $p_i$ can decide **Abort** and halt

Step 4: $p_i$ (who voted **Yes**) is waiting for **Commit** or **Abort**

$p_i$ cannot decide: it must run a termination protocol
**Termination protocols**

A. Wait for coordinator to recover
   - it always works, since the coordinator is never uncertain
   - may block recovering process unnecessarily

B. Ask other participants
Cooperative termination

- Coordinator appends list of participants to VOTE-REQ
- When an uncertain process $p$ times out, it sends a DECISION-REQ message to every other participant
- If $q$ has decided, it sends its decision to $p$, which acts accordingly
- If $q$ has not yet voted, it decides Abort and sends Abort to $p$
- What if $q$ is uncertain?
LOGGING ACTIONS

- When $c$ sends VOTE-REQ, it writes START-2PC to its DT Log
- When $p_i$ is ready to vote Yes,
  - $p_i$ writes Yes to DT Log, along with a list of participants
  - $p_i$ sends Yes to $c$
- When $p_i$ is ready to vote No, it writes Abort to its DT Log
- When $c$ is ready to Commit, it writes Commit to its DT Log
  before sending Commit to participants
- When $c$ is ready to decide Abort, it writes Abort to its DT Log
- After $p_i$ receives a decision value, it writes it to its DT Log
$p$ recovers

- if DT Log contains START-2PC, then $p = c$
  - if DT Log contains a decision value, decide accordingly
  - else, decide **Abort**

- otherwise, $p$ is a participant
  - if DT Log contains a decision value, decide accordingly
  - else if it does not contain a **Yes** vote, decide **Abort**
  - else (**Yes** but no decision) run a termination protocol
2PC AND BLOCKING

- Blocking occurs whenever the progress of a process depends on the repairing of failures.
- No AC protocol is non-blocking in the presence of communication or total failures.
- But 2PC can block even with non-total failures and with no communication failures among operating processes!

Enter 3PC!
ADMINISTRIVIA

- Problem set #1 will be released on Monday
  - Due Monday 9/27 before class, by email to Tony and me
- **Individual work only**
  - No collaboration with classmates
  - No looking up solutions online
  - No handwritten-and-scanned answers
- Take a look at list of papers we will read in part 2
  - Start thinking about what you want to do
Blocking and uncertainty

Why does uncertainty lead to blocking?

An uncertain process does not know whether it can safely decide **Commit** or **Abort**, because some of the processes it cannot reach could have decided either.

**Non-blocking property**
If any operational process is uncertain, then no process has decided **Commit**
2PC REVISITED

In U, both A and C are reachable
2PC REVISITED

In U, both A and C are reachable.

U

\[ p_i \]

\[ \text{VOTE-REQ Yes} \]

\[ \text{ABORT} \]

\[ \text{VOTE-REQ No} \]

A

C

\[ \text{COMMIT} \]
In **PC**, a process knows that it will Commit unless it fails.

**2PC REVISITED**

- **U** votes Yes for **pi**.
- **pi** then sends a VOTE-REQ to **U**.
- **U** sends a VOTE-REQ to **PC**.
- **PC** sends a VOTE-REQ to **FS**.
- **FS** sends a VOTE-REQ to **A**.
- **A** sends a VOTE-REQ to **C**.
- **C** then sends a COMMIT message to **FS**.
- **FS** sends a COMMIT message to **PC**.
- **PC** sends a COMMIT message to **U**.

The process knows its decision based on these interactions.
3-Phase Commit (3PC)

- Important assumption: *synchrony*

- For most of our discussion, we’ll only consider non-total failures. Total failures will require special care.
3-Phase Commit

Coordinator $c$

Participant $p_i$

1. sends VOTE-REQ to all participants
3-Phase Commit

Coordinator $c$
1. sends VOTE-REQ to all participants

Participant $p_i$
2. sends $vote_i$ to Coordinator
   if $vote_i = \text{No}$ then
   $decision_i := \text{Abort}$
   halt
3-Phase Commit

**Coordinator** $c$

1. sends VOTE-REQ to all participants

2. sends $vote_i$ to Coordinator
   
   if $vote_i = \text{No}$ then
   
   decision$_i := \text{Abort}$
   
   halt

3. if (all votes are **Yes**) then
   
   send Precommit to all

   else

   decision$_c := \text{Abort}$
   
   send Abort to all who voted **Yes**

   halt

**Participant** $p_i$
3-Phase Commit

Coordinator $c$

1. sends VOTE-REQ to all participants

3. if (all votes are Yes) then send Precommit to all
   else $\text{decision}_c := \text{Abort}$
   send Abort to all who voted Yes

Participant $p_i$

2. sends $vote_i$ to Coordinator
   if $vote_i = \text{No}$ then $\text{decision}_i := \text{Abort}$
   halt

4. if received Precommit then send Ack
   halt
3-Phase Commit

Coordinator $c$

1. sends VOTE-REQ to all participants

Participant $p_i$

2. sends $vote_i$ to Coordinator

if $vote_i = \text{No}$ then

$decision_i := \text{Abort}$

halt

3. if (all votes are $\text{Yes}$) then

send $\text{Precommit}$ to all

else

$decision_c := \text{Abort}$

send $\text{Abort}$ to all who voted $\text{Yes}$

halt

4. if received $\text{Precommit}$ then

send $\text{Ack}$

5. collect $\text{Ack}$ from all participants

When all $\text{Ack}$’s have been received:

$decision_c := \text{Commit}$

send $\text{Commit}$ to all
3-Phase Commit

Coordinator $c$

1. sends VOTE-REQ to all participants

Participant $p_i$

2. sends $vote_i$ to Coordinator

if $vote_i = No$

then $decision_i := Abort$

else

3. if (all votes are Yes) then

send Precommit to all

else

$decision_c := Abort$

send Abort to all who voted Yes

halt

4. if received Precommit then

send Ack

halt

5. collect Ack from all participants

When all Ack's have been received:

$decision_c := Commit$

send Commit to all

6. When $p_i$ receives Commit, sets $decision_i := Commit$ and halts
When all Ack's have been received:

Commit

send Commit to all

I. sends VOTE-REQ to all participants

2. sends Precommit to Coordinator if decision\(_i\) = No then

Abort

else

decision\(_c\) := Abort

send Abort to all who voted Yes

halt

3. if (all votes are Yes) then

send Precommit to all

4. if received Precommit then

send Ack

5. collect Ack from all participants

When all Ack's have been received:

decision\(_c\) := Commit

send Commit to all

6. When \(p_i\) receives Commit, sets decision\(_i\) := Commit and halts

Some messages are known before they are sent. So why are they sent?
Some messages are known before they are sent. So why are they sent?

They inform the recipient of the protocol’s progress

5. collect Ack from all participants

When all Ack’s have been received:

\[ \text{decision}_c := \text{Commit} \]

send Commit to all

4. if received Precommit then send Ack

6. When \( p_i \) receives Commit, sets \( \text{decision}_i := \text{Commit} \) and halts

3-Phase Commit

- When \( c \) receives Ack from \( p_i \), it knows that \( p_i \) is not uncertain
- When \( p_i \) receives Commit, it knows no participant in uncertain, so it can commit
**Timeout actions**

Coordinator $c$

Step 2: $c$ is waiting for VOTE-REQ from the coordinator

Step 3: Coordinator is waiting for vote from participants

Step 5: Coordinator is waiting for Ack’s

Participant $p_i$

Step 2: $p_i$ is waiting for VOTE-REQ from the coordinator

Step 4: $p_i$ is waiting for **Precommit**

Step 6: $p_i$ is waiting for **Commit**
### Timeout actions

**Coordinator** $c$

- **Step 3:** Coordinator is waiting for vote from participants

**Participant** $p_i$

- **Step 2:** $p_i$ is waiting for VOTE-REQ from the coordinator
  - Same as in 2PC
- **Step 4:** $p_i$ is waiting for Precommit
- **Step 6:** $p_i$ is waiting for Commit

**Step 5:** Coordinator is waiting for Ack's
Timeout actions

Coordinator $c$

Step 3: Coordinator is waiting for vote from participants

Same as in 2PC

Participant $p_i$

Step 2: $p_i$ is waiting for VOTE-REQ from the coordinator

Same as in 2PC

Step 4: $p_i$ is waiting for Precommit

Step 5: Coordinator is waiting for Ack's

Step 6: $p_i$ is waiting for Commit
## Timeout actions

<table>
<thead>
<tr>
<th>Coordinator $c$</th>
<th>Participant $p_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step 2:</strong> $p_i$ is waiting for VOTE-REQ from the coordinator</td>
<td>Same as in 2PC</td>
</tr>
<tr>
<td><strong>Step 3:</strong> Coordinator is waiting for vote from participants</td>
<td><strong>Step 4:</strong> $p_i$ is waiting for Precommit</td>
</tr>
<tr>
<td>Same as in 2PC</td>
<td>Run termination protocol</td>
</tr>
<tr>
<td><strong>Step 5:</strong> Coordinator is waiting for Ack's</td>
<td><strong>Step 6:</strong> $p_i$ is waiting for Commit</td>
</tr>
</tbody>
</table>