Distributed Systems

Distributed Transactions

Prof. Dr. Oliver Hahm

Frankfurt University of Applied Sciences
Faculty 2: Computer Science and Engineering
oliver.hahm@fb2.fra-uas.de
https://teaching.dahahm.de

30.06.2023

Motivation

Particular problem for the development of distributed applications:

Partial Failure Property

- Failure of single components in a distributed system

⇒ Complex error conditions in distributed applications

Motivation for transactions

- Atomic actions as generalization of the transaction concept of databases
- Reducing the complexity for the application developer in the presence of errors and concurrency
- Automatic backward error recovery, combination with forward error recovery possible

Agenda

- Transaction Concept
- Site Local Commit Protocols
  - Intention Lists
  - Shadowing
  - Write-Ahead Logging (WAL)
- Two-Phase Commit Protocol

Agenda

- **Transaction Concept**
- Site Local Commit Protocols
  - Intention Lists
  - Shadowing
  - Write-Ahead Logging (WAL)
- Two-Phase Commit Protocol

Transaction Concept

Definition

A transaction is a series of actions (i.e., operation on resources) with ACID properties.

- **Atomicity**:
  - Transaction is either executed completely or appears as never started
  - No intermediate result in between start and final state gets visible

Transaction Concept

Definition

A transaction is a series of actions (i.e., operation on resources) with ACID properties.

- **Atomicity**:
  - Transaction is either executed completely or appears as never started
  - No intermediate result in between start and final state gets visible
- **Consistency**:
  - A transaction takes the system from one consistent state into another consistent state
  - No application constraints is violated

Transaction Concept

Definition

A transaction is a series of actions (i.e., operation on resources) with ACID properties.

- **Atomicity**:
  - Transaction is either executed completely or appears as never started
  - No intermediate result in between start and final state gets visible
- **Consistency**:
  - A transaction takes the system from one consistent state into another consistent state
  - No application constraints is violated
- **Isolation**:
  - Each transaction must be performed without interference from other transactions

Transaction Concept

Definition

A transaction is a series of actions (i.e., operation on resources) with ACID properties.

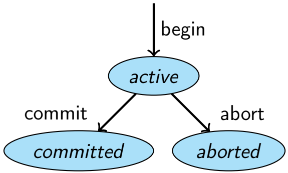
- **Atomicity**:
  - Transaction is either executed completely or appears as never started
  - No intermediate result in between start and final state gets visible
- **Consistency**:
  - A transaction takes the system from one consistent state into another consistent state
  - No application constraints is violated
- **Isolation**:
  - Each transaction must be performed without interference from other transactions
- **Durability**:
  - The effects of a completed transaction do not get lost (even in case of any (allowed) error wrt the error model)

Local and Distributed Transactions

Local transaction

- Effects are restricted to a single computer system

State diagram

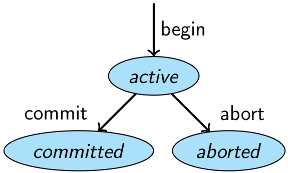


Local and Distributed Transactions

Local transaction

- Effects are restricted to a single computer system

State diagram



Distributed transaction

- Effect to multiple sites of a distributed system

?

to be developed

Failure Model – Abort and Site Failure

Definition

A failure model describes all anticipated failures to which a system reacts gracefully. All other failures are considered a disaster.

Failure Model – Abort and Site Failure

Definition

A failure model describes all anticipated failures to which a system reacts gracefully. All other failures are considered a disaster.

Transaction abort:

- Aborting single transactions, e. g., by …
  - explicit user aborts
  - errors in the application logic
  - as a consequence of a deadlock resolution

Failure Model – Abort and Site Failure

Definition

A failure model describes all anticipated failures to which a system reacts gracefully. All other failures are considered a disaster.

Transaction abort:

- Aborting single transactions, e. g., by ...
  - explicit user aborts
  - errors in the application logic
  - as a consequence of a deadlock resolution

Site failure:

- Failure of a participating system, e. g., by ...
  - transient or permanent hardware failures (including power outage)
  - crash of the OS with reboot
- All running processes crash
- All transactions in state **active** change to state **aborted**

Failure Model – Media Failure and Communication Failure

Media failure:

- Non-recoverable error on non-volatile storage medium used while processing the tranactions, e. g. ...
  - Hard disc (used for storing data)
  - Tape (used for logging)
- Standard treatment by using stable storage (redundant storage on multiple media, e. g., mirror disks, RAID, ...)
- Out of scope for this lecture

Failure Model – Media Failure and Communication Failure

Media failure:

- Non-recoverable error on non-volatile storage medium used while processing the tranactions, e. g. ...
  - Hard disc (used for storing data)
  - Tape (used for logging)
- Standard treatment by using stable storage (redundant storage on multiple media, e. g., mirror disks, RAID, ...)
- Out of scope for this lecture

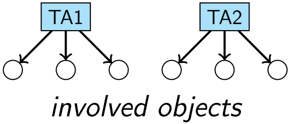
Communication failure:

- Errors of the messaging system which lead to the loss of messages
- Partitioning: Network disintegration into multiple isolated subnetworks

Flat Transactions

Flat transactions

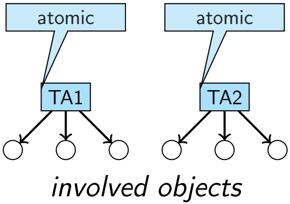
- Traditional model used in database context
- Transaction involves a set of objects (resources)
- Transactions may share objects (regulated by concurrency control mechanisms)
- Transactions cannot be nested



Flat Transactions

Flat transactions

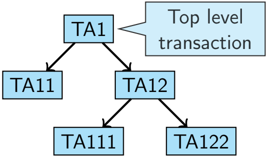
- Traditional model used in database context
- Transaction involves a set of objects (resources)
- Transactions may share objects (regulated by concurrency control mechanisms)
- Transactions cannot be nested



⇒ Disadvantage: No possibility to store intermediate results

Nested Transactions

- A transaction may include **inner transactions** (subtransactions)
- **Isolated resettability** of inner transactions: abort of an inner transaction results is an **exception** (not abort!) of higher-level transaction
- **Abort** of a transaction results in the **abort of all inner transactions**
- **Commitment** is **relative** to the parent transaction (final when the top-level transaction commits)
- Concurrency control at each nesting layer



Agenda

- Transaction Concept
- Site Local Commit Protocols
  - Intention Lists
  - Shadowing
  - Write-Ahead Logging (WAL)
- Two-Phase Commit Protocol

Site Local Commit Protocols

Protocols to achieve local atomicity in failure cases and persistent effects:

- Intention Lists (Lampson 1981)
- Shadowing (Gray 1981)
- Write-Ahead Logging (WAL)

Agenda

- Transaction Concept
- Site Local Commit Protocols
  - Intention Lists
  - Shadowing
  - Write-Ahead Logging (WAL)
- Two-Phase Commit Protocol

Intention lists

Procedure

- **Intended changes** on data base objects are collected in a **list** (i. e., executed on copies of the original data)
- The list is written into **stable memory**
- Decision is made (**committed-aborted**)

  On aborted Discard list

  On committed Update originals of the object in non-volatile memory

Intention lists

Procedure

- **Intended changes** on data base objects are collected in a **list** (i. e., executed on copies of the original data)
- The list is written into **stable memory**
- Decision is made (**committed-aborted**)

  On aborted: Discard list

  On committed: Update originals of the object in non-volatile memory

In distributed systems

- Each node maintains a **tentative list** and knows the **coordinator**
- A coordinator maintains a **list of all nodes** and notifies these
- Notified nodes update the objects according to their list and delete it

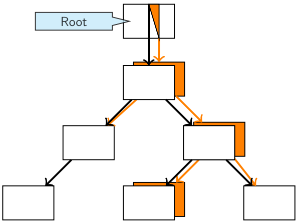
Agenda

Procedure

- Assumes **non-volatile memory** as **tree structure** with **reference blocks** (cf. UNIX file system)
- Create **after images** of all blocks as **shadow version** up to the root node
- Make decision (committed-aborted) by atomic **pointer swap** at root block (writing a block)

On aborted Discard shadow structure

On committed Release former original blocks, shadow blocks become the original ones



On site failure

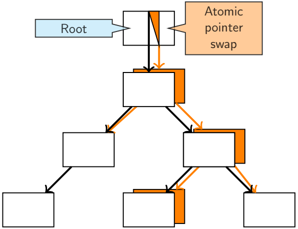
- Either old or new state is established

Procedure

- Assumes **non-volatile memory** as **tree structure** with **reference blocks** (cf. UNIX file system)
- Create **after images** of all blocks as **shadow version** up to the root node
- Make decision (committed-aborted) by atomic **pointer swap** at root block (writing a block)

On aborted Discard shadow structure

On committed Release former original blocks, shadow blocks become the original ones



On site failure

- Either old or new state is established

Pros & Cons

Advantages

- Atomic state change by writing the root block

Drawbacks

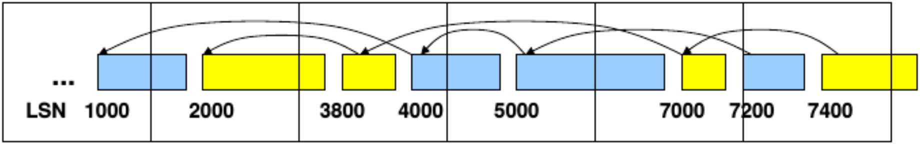
- No concurrency of commit processes
- Physical alignment of data blocks may get lost on commit since shadow blocks are from the free list

Agenda

Basics of Logging

- A log consists of a sequence of so called **log records**
- Each record of variable lengths is identified by a **log sequence number (LSN)** → Byte offset in log stream (analog to TCP sequence numbers)
- The log is **shared** among processes on a node
- **Linking** of related log entries of a **commit process**
- Realization of the log in replicated files (stable memory)
- Superordinate table with all log entries and SQL access is common

Block structure of the medium



older write direction⟶ younger

WAL: Writing, Reading, and Shortening

Writing of log entries

- Only **sequential writing** of log files (high performance)
- **Buffered writing** of log entries in main memory
- **Forced write** of a log entry enforces storage of all preceding log entries (with smaller LSN) and return from operation after a block has been physically stored on the medium

WAL: Writing, Reading, and Shortening

Writing of log entries

- Only **sequential writing** of log files (high performance)
- **Buffered writing** of log entries in main memory
- **Forced write** of a log entry enforces storage of all preceding log entries (with smaller LSN) and return from operation after a block has been physically stored on the medium

Reading of log entries

- Only in case of site failures and potentially on transaction failures (see below)

WAL: Writing, Reading, and Shortening

Writing of log entries

- Only **sequential writing** of log files (high performance)
- **Buffered writing** of log entries in main memory
- **Forced write** of a log entry enforces storage of all preceding log entries (with smaller LSN) and return from operation after a block has been physically stored on the medium

Reading of log entries

- Only in case of site failures and potentially on transaction failures (see below)

Log shortening

- Log can get arbitrarily long, but the duration for the restart after a site failure has to be limited
- Use of **checkpoints**

WAL: Local Commits and Protocol

Use of log entries as part of the local commitment

- Log records for each transaction are linked among themselves
- Writing of **before images (undo records)** for all objects whose persistent original object state is changed during the transaction (update in place)
- Writing of **after images (redo records)** for all achieved final object states of the transaction
- Store the final **outcome record** via forced write:
  - Enforces storage of all related log entries
  - Upon appearance in the log, the commitment is complete
  - May be realized as a flag in the last block

WAL: Local Commits and Protocol

Use of log entries as part of the local commitment

- Log records for each transaction are linked among themselves
- Writing of **before images (undo records)** for all objects whose persistent original object state is changed during the transaction (update in place)
- Writing of **after images (redo records)** for all achieved final object states of the transaction
- Store the final **outcome record** via forced write:
  - Enforces storage of all related log entries
  - Upon appearance in the log, the commitment is complete
  - May be realized as a flag in the last block

Write-Ahead Logging Protocol

- Secured writing of log records **before** modification of the original persistent state

Write-Ahead Logging: Error Handling

Handling of transaction failures

- (Potentially) create and store "aborted" outcome record
- If before records has been written, their content has to be re-established as persistent object state

Write-Ahead Logging: Error Handling

Handling of transaction failures

- (Potentially) create and store "aborted" outcome record
- If before records has been written, their content has to be re-established as persistent object state

Handling of site failures

- Read the log
- For each transaction which has not yet been completed, establish before image (if existing)
- For all transactions with an existing "committed" outcome record establish the last after image in each case of all objects (at some point) $\rightarrow$ idempotence

Advantages of WAL

- Several **interlocked commit operations** can take place simultaneously
- High **I/O performance** through buffering and sequential writing of logs
- Data **block alignment** of the persistent state remains unchanged (update in place)
- **Parallelization** of logs is possible
- After site failures and subsequent restart only the log has to be analyzed
- Log can be shared by commit protocols

Agenda

- Transaction Concept
- Site Local Commit Protocols
  - Intention Lists
  - Shadowing
  - Write-Ahead Logging (WAL)
- Two-Phase Commit Protocol

Commit Protocols

- Are used to coordinate a commit/abort decision of a set of processes in an distributed environment
- E. g., to enforce atomicity in failure cases and durability for distributed transaction environments
- Special cases of the so called **consensus protocols** ($\rightarrow$ yes/no)

Commit Protocols

- Are used to coordinate a commit/abort decision of a set of processes in an distributed environment
- E. g., to enforce atomicity in failure cases and durability for distributed transaction environments
- Special cases of the so called **consensus protocols** ($\rightarrow$ yes/no)

Two Phase Commit Protocol

- Most commonly used protocol
- Very high practical relevance and used in multiple products

Theoretical background

- Multiphase commit protocols (e. g., Skeen, 1981)
- Weak/strong termination conditions lead to blocking/non-blocking commit algorithms

Properties of a Commit Algorithm

A commit algorithm for a set of processes provides the following properties:

1. All processes which make a decision make the same decision

Properties of a Commit Algorithm

A commit algorithm for a set of processes provides the following properties:

1. All processes which make a decision make the same decision
2. Once a decision has been made, it is **binding** for any process and cannot be revoked

Properties of a Commit Algorithm

A commit algorithm for a set of processes provides the following properties:

1. All processes which make a decision make the same decision
2. Once a decision has been made, it is **binding** for any process and cannot be revoked
3. Only if **all processes** decide for **commit** the **common decision** is **commit**
   - ⇒ As soon as **one process** decides on **abort** the **common decision** must be **abort**

Properties of a Commit Algorithm

A commit algorithm for a set of processes provides the following properties:

1. All processes which make a decision make the same decision
2. Once a decision has been made, it is **binding** for any process and cannot be revoked
3. Only if **all processes** decide for **commit** the **common decision** is **commit**
   - ⇒ As soon as **one process** decides on **abort** the **common decision** must be **abort**
4. If at one point in time all errors that have occurred are repaired and no new errors occur for a sufficiently long time, the processes come to a common decision

Terminology

Window of vulnerability

- Interval between the local commit decision **of a process and the *notification of the common decision***
- Also called uncertainty period of a process

Terminology

Window of vulnerability

- Interval between the local commit decision **of a process and the *notification of the common decision***
- Also called uncertainty period of a process

Blocking protocol

- The protocol provides that a process must await the repair of a fault

Terminology

Window of vulnerability

- Interval between the local commit decision **of a process and the *notification of the common decision***
- Also called uncertainty period of a process

Blocking protocol

- The protocol provides that a process must await the repair of a fault

Independent recovery

- A process can make a decision on its own after a failure without communicating with another process

Terminology

Window of vulnerability

- Interval between the local commit decision **of a process and the *notification of the common decision***
- Also called uncertainty period of a process

Blocking protocol

- The protocol provides that a process must await the repair of a fault

Independent recovery

- A process can make a decision on its own after a failure without communicating with another process

Participants

- The processes that handle commit protocol

Background

Lemmas

- If communication failures or system failures are possible, there is no commit protocol which does not block a process
  Note: If only individual site failure occur a non-blocking commit protocol may still exist.
- No commit protocol can guarantee independent recovery of failed processes
  Note: There is no commit protocol without a uncertainty period for the participants

Basics of the Two Phase Commit Protocol

Two Phase Commit Protocol (2PC)

Blocking commit algorithm with a weak termination property ($\Rightarrow$ if no errors occur, all processes come to a decision at some point)

First published by *J. Gray, 1978*

Basics of the Two Phase Commit Protocol

Two Phase Commit Protocol (2PC)

Blocking commit algorithm with a weak termination property ($\Rightarrow$ if no errors occur, all processes come to a decision at some point)

First published by *J. Gray, 1978*

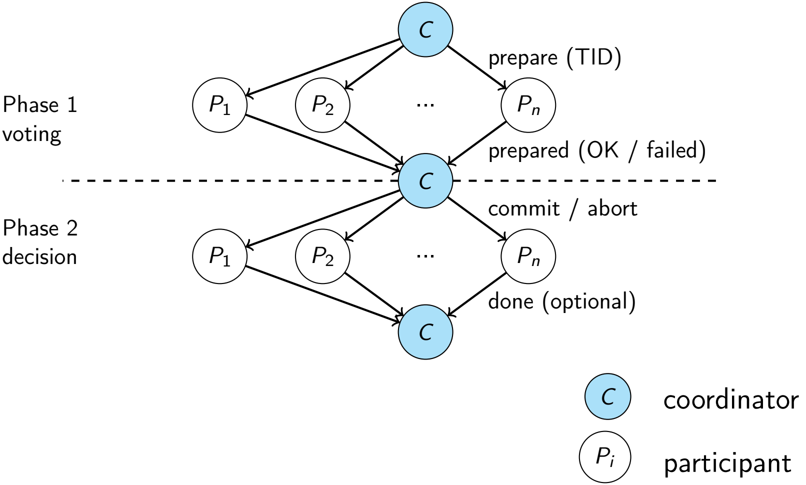
Roles

- **Participant**
- **Coordinator** as designated participant controlling the protocol
  Note: typically the participant which initiates the transaction
- Coordinator knows all participants
- Participants only know the coordinator

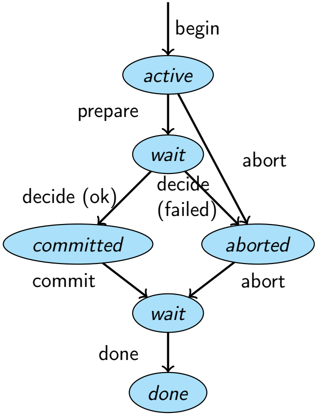
Use of local logs of each participant to update the status of the commit process

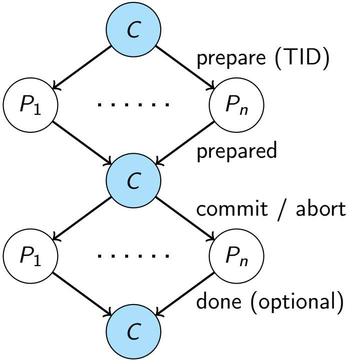
Two Phase Commit Protocol

Message flow (normal case without failures)

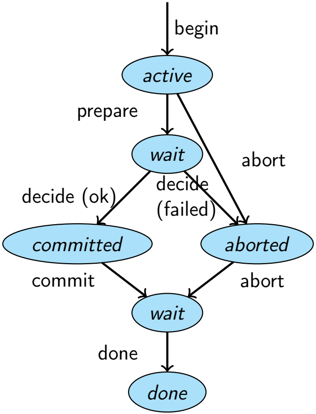


State Diagram of the Coordinator

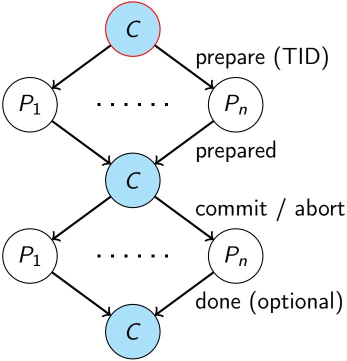




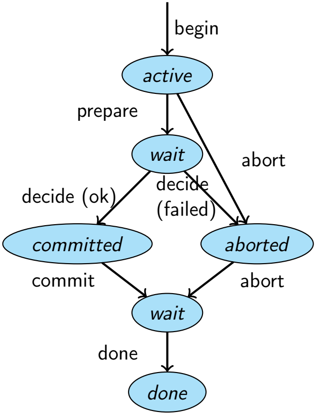
State Diagram of the Coordinator



Asks

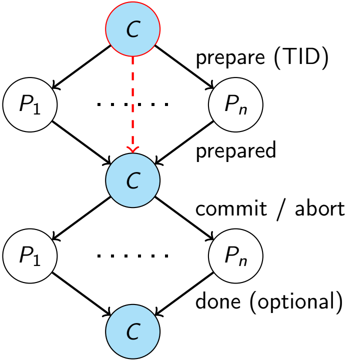


State Diagram of the Coordinator

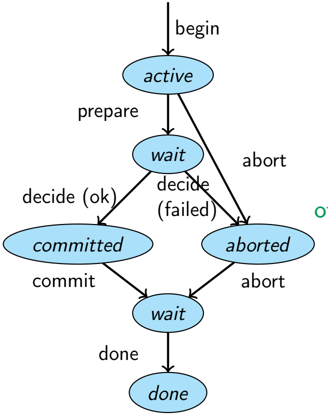


Asks

Waiting
for responses



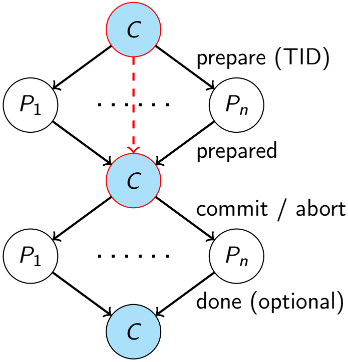
State Diagram of the Coordinator



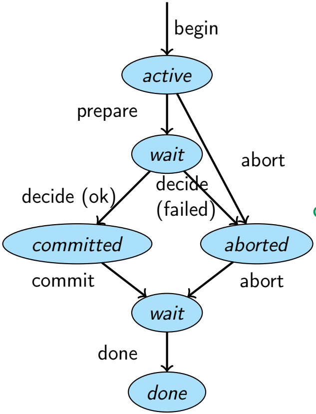
Asks

Waiting
for responses

local
decision
of the coordinator



State Diagram of the Coordinator



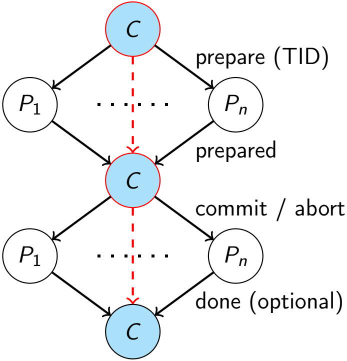
Asks

Waiting
for responses

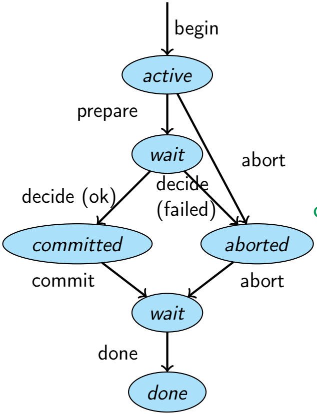
local
decision
of the coordinator

Propagate

Waiting
for confirmation



State Diagram of the Coordinator



Asks

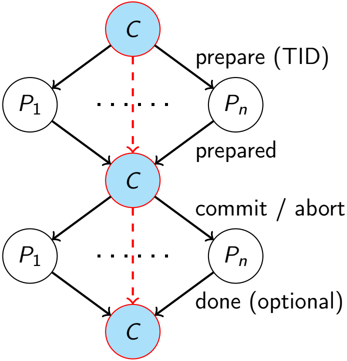
Waiting
for responses

local
decision
of the coordinator

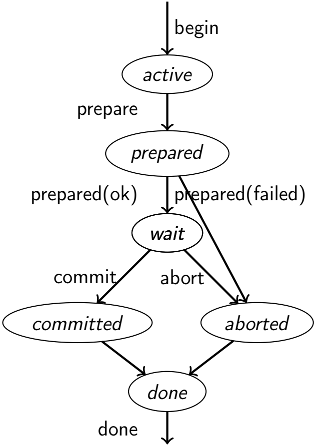
Propagate

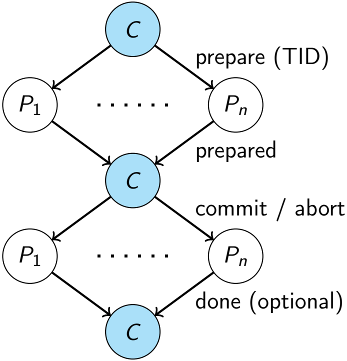
Waiting
for confirmation

Forget

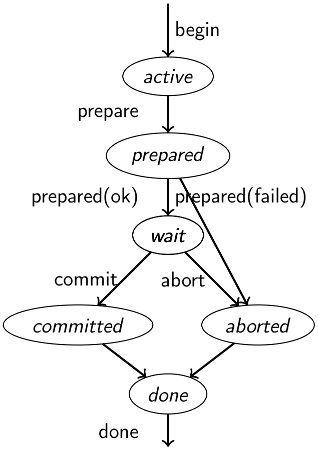


State Diagram of the Participants

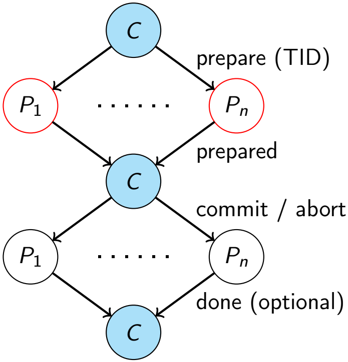




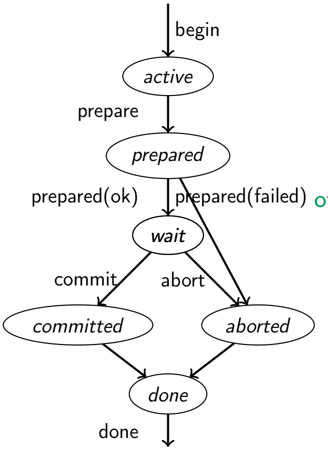
State Diagram of the Participants



Local preparation

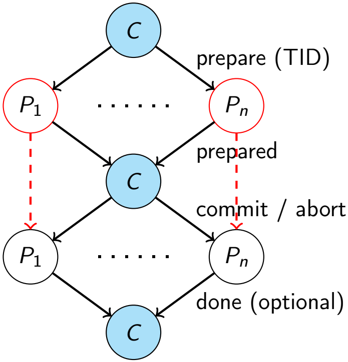


State Diagram of the Participants

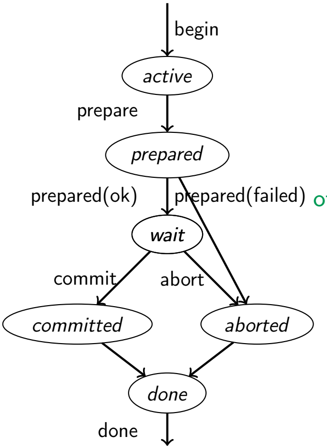


Local preparation

Waiting on decision of the coordinator



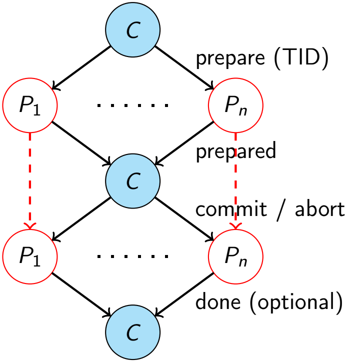
State Diagram of the Participants



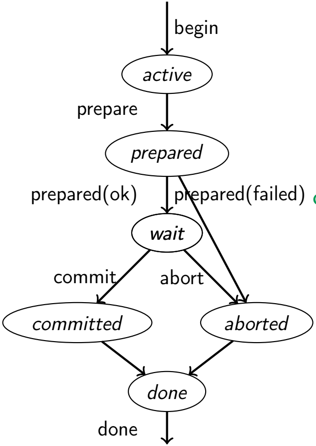
Local preparation

Waiting on decision of the coordinator

Execute actions according to decision



State Diagram of the Participants

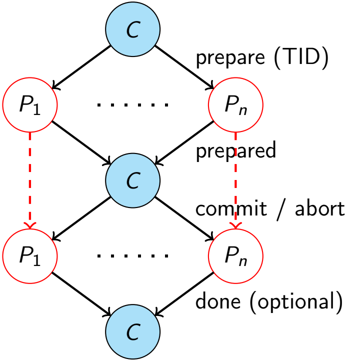


Local preparation

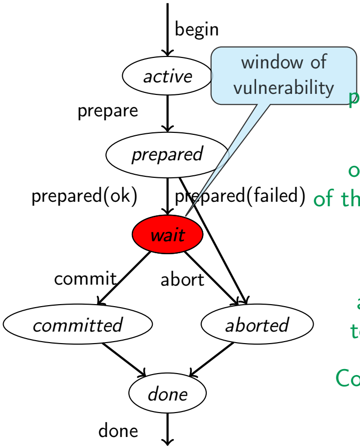
Waiting on decision of the coordinator

Execute actions according to decision

Communicate end



State Diagram of the Participants

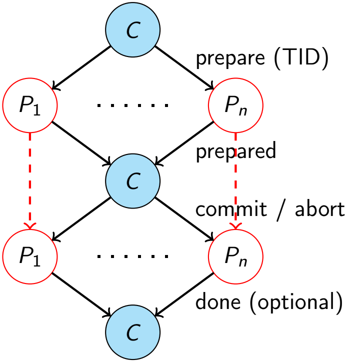


Local preparation

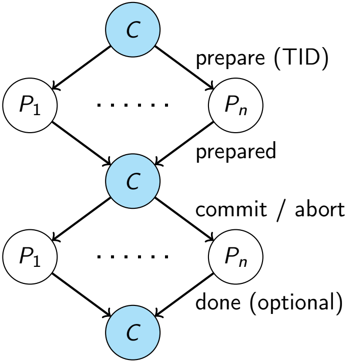
Waiting on decision of the coordinator

Execute actions according to decision

Communicate end



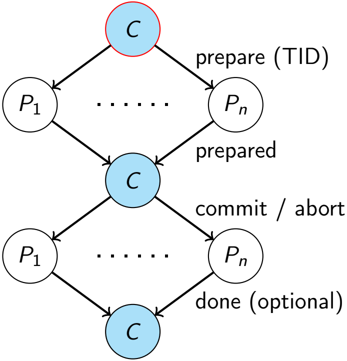
Logging of the Coordinator



Logging of the Coordinator

$\boxed{TID : begin(P_1, \ldots, P_n)}$

Asks

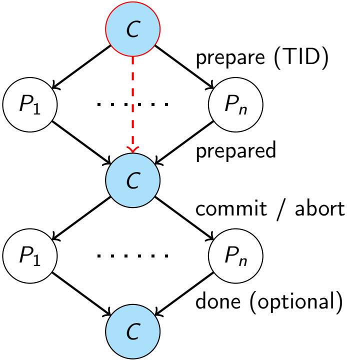


Logging of the Coordinator

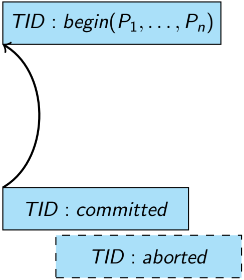
$TID : begin(P_1, \ldots, P_n)$

Asks

Waiting
for responses



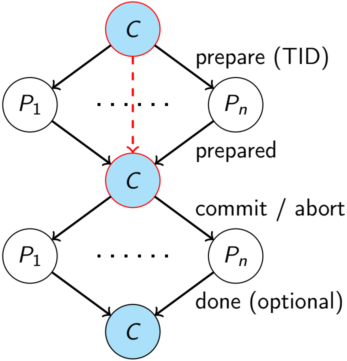
Logging of the Coordinator



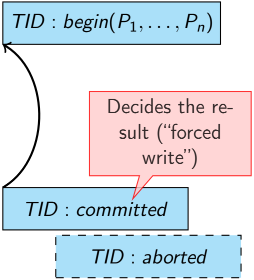
Asks

Waiting
for responses

local
decision
of the coordinator



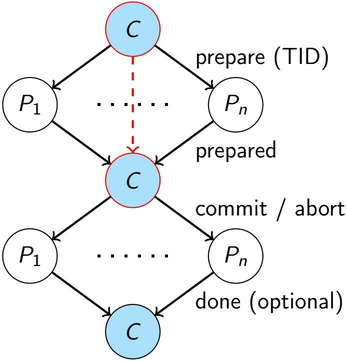
Logging of the Coordinator



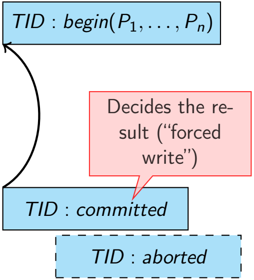
Asks

Waiting for responses

local decision of the coordinator



Logging of the Coordinator



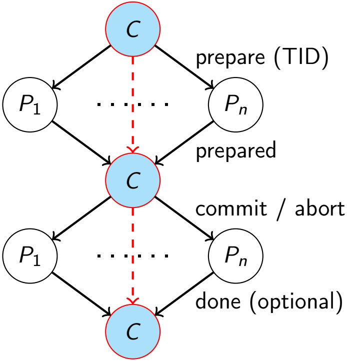
Asks

Waiting for responses

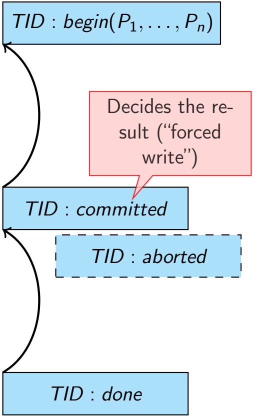
local decision of the coordinator

Propagate

Waiting for confirmation



Logging of the Coordinator



Asks

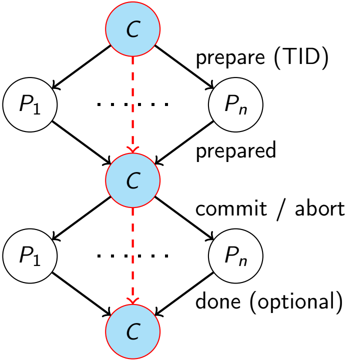
Waiting
for responses

local
decision
of the coordinator

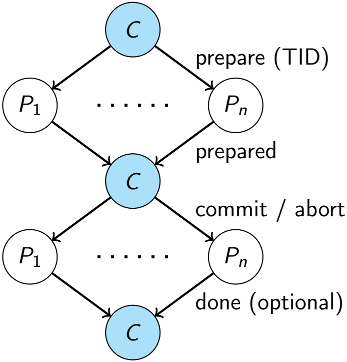
Propagate

Waiting
for confirmation

Forget



Logging of the Participants

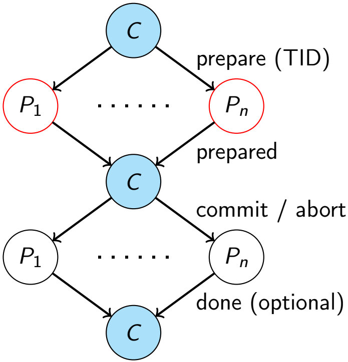


analog on unilateral abort

Logging of the Participants

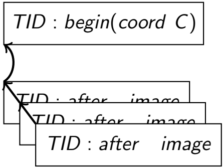
$\boxed{TID : begin(coord\ C)}$

Local
Preparation



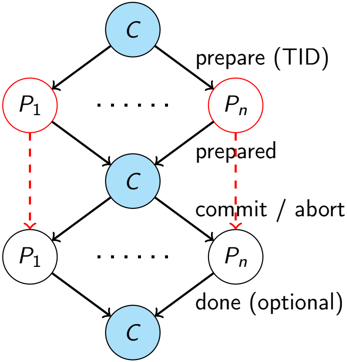
analog on unilateral abort

Logging of the Participants



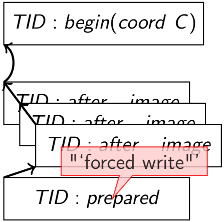
Local
Preparation

Waiting
on the decision
of the coordinator



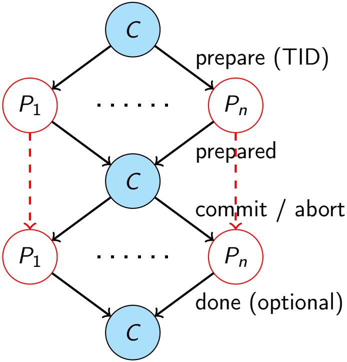
analog on unilateral abort

Logging of the Participants



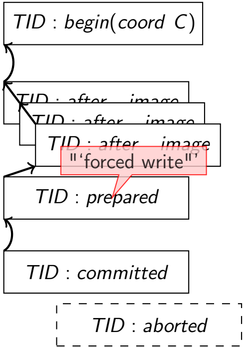
Local
Preparation

Waiting
on the decision
of the coordinator



analog on unilateral abort

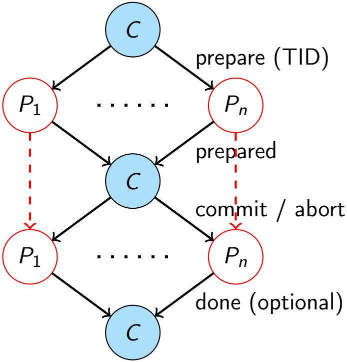
Logging of the Participants



Local
Preparation

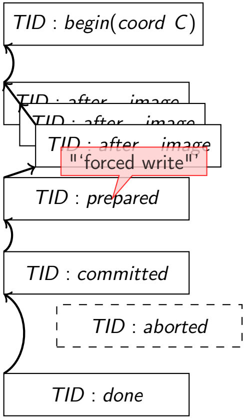
Waiting
on the decision
of the coordinator

Execute
actions
according
to decision



analog on unilateral abort

Logging of the Participants

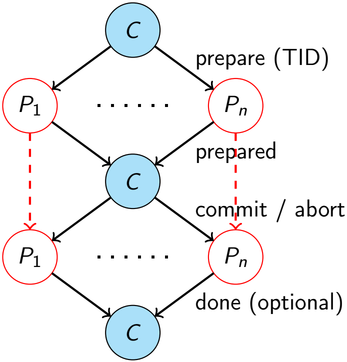


Local Preparation

Waiting on the decision of the coordinator

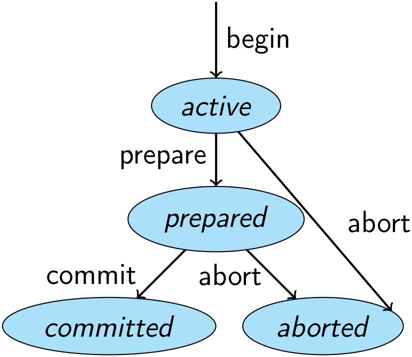
Execute actions according to decision

Communicate end

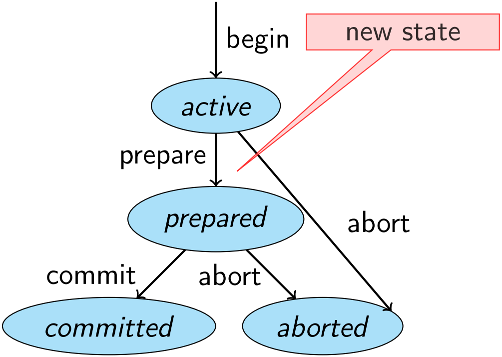


analog on unilateral abort

Summarized State Diagram for Distributed Transactions



Summarized State Diagram for Distributed Transactions



Failure Handling

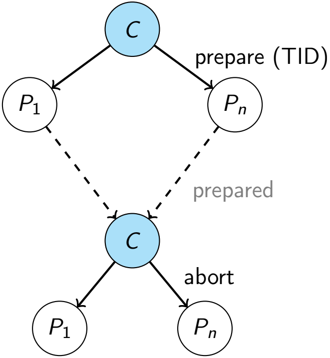
Handling of Transaction Failures

- Transition of active transactions into the state aborted
- Unilateral abort decision of the coordinator and every participant is possible
  - Coordinator sends abort later on, even if all participants have replied with prepared
  - Participant reply prepared or failed to the coordinator's prepare request

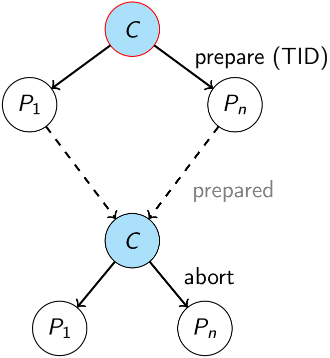
Handling of communication failures for active transactions

- Identification via timeouts (as for other events)
- Transition on transaction failures with the same handling as above

Example for Abort Decision of the Coordinator

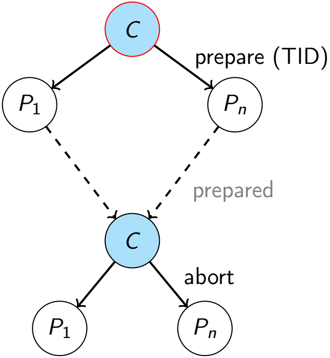


Example for Abort Decision of the Coordinator



Asks

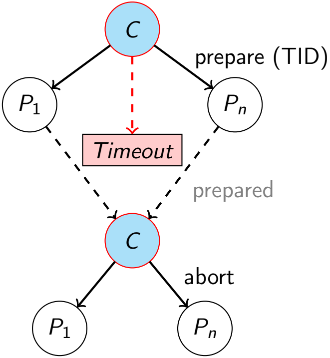
Example for Abort Decision of the Coordinator



Asks

Waiting
for responses

Example for Abort Decision of the Coordinator

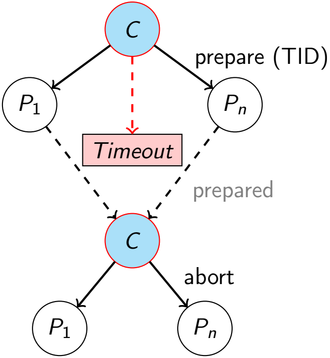


Asks

Waiting
for responses

Abort decision
of the coordinator

Example for Abort Decision of the Coordinator



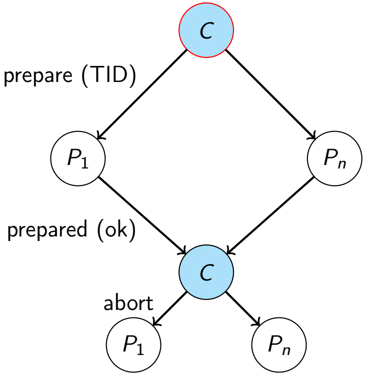
Asks

Waiting
for responses

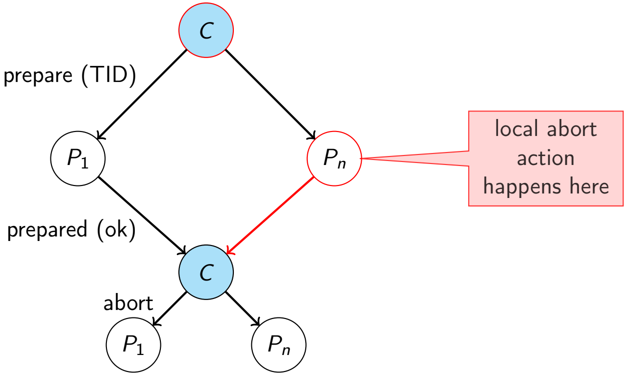
Abort decision
of the coordinator

Propagate

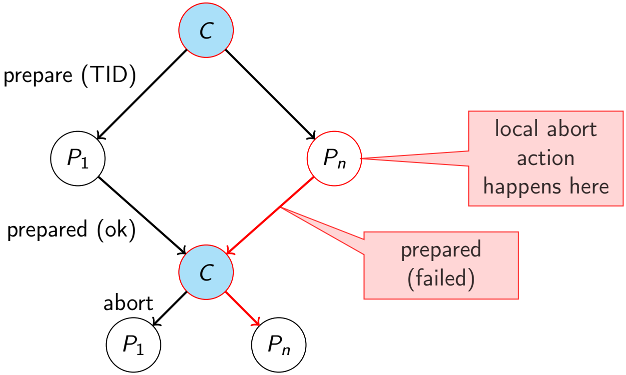
Example for Abort Decision of a Participant



Example for Abort Decision of a Participant



Example for Abort Decision of a Participant



Handling of a Site Failure by the Coordinator

- Processing depends on the information found in the log

a) $\boxed{TID : begin(P_1, \ldots, P_n)}$ $\Rightarrow$ abort decision, continue protocol

Handling of a Site Failure by the Coordinator

- Processing depends on the information found in the log

a) $TID : begin(P_1, \ldots, P_n)$ ⇒ abort decision, continue protocol

b) $TID : begin(P_1, \ldots, P_n)$ ⇒ decided already before site failure
continue protocol,
send commit/abort message
to all participants

$TID : committed$

$TID : aborted$

Handling of a Site Failure by the Coordinator

- Processing depends on the information found in the log

a) TID : $begin(P_1, \ldots, P_n)$ $\Rightarrow$ abort decision, continue protocol

b) TID : $begin(P_1, \ldots, P_n)$ $\Rightarrow$ decided already before site failure continue protocol, send commit/abort message to all participants

TID : $committed$

TID : $aborted$

c) TID : $begin(P_1, \ldots, P_n)$ $\Rightarrow$ nothing to do

TID : $committed$

TID : $done$

Handling of a Site Failure by the Participant

- Processing also depends on the information found in the log

a) $\boxed{TID : begin(coord\ C)}$ $\Rightarrow$ abort

Handling of a Site Failure by the Participant

- Processing also depends on the information found in the log

a) $\boxed{TID : begin(coord\ C)}$ $\Rightarrow$ abort

b) $\boxed{TID : begin(coord\ C)}$ $\Rightarrow$ abort

$\boxed{TID : after\quad image}$

$\boxed{TID : after\quad image}$

Handling of a Site Failure by the Participant

- Processing also depends on the information found in the log

a) $TID : begin(coord\ C)$ $\Rightarrow$ abort

b) $TID : begin(coord\ C)$

$TID : after\ \ image$

$TID : after\ \ image$

$\Rightarrow$ abort

c) $TID : begin(coord\ C)$

$TID : after\ \ image$

$TID : after\ \ image$

$TID : after\ \ image$

$TID : prepared$

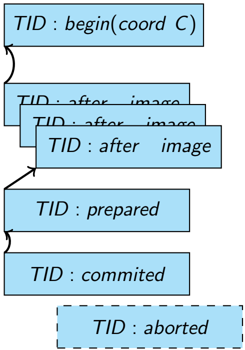
$\Rightarrow$ no unilateral decision possible

ask coordinator C
$getDecisision(TID)$

continue protocol

Handling of a Site Failure by the Participant (2)

d)



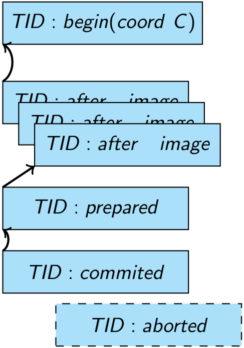
⇒ independent recovery possible

repeat local commit/abort action

continue protocol

Handling of a Site Failure by the Participant (2)

d)

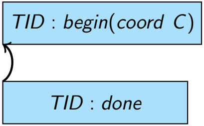


⇒ independent recovery possible

repeat local commit/abort action

continue protocol

e)



⇒ nothing to do

Extensions

Coordinator migration

- The coordinator role is transferred to a highly reliable computer
- Consequently, blocking of the participants due to a unavailability of the coordinator becomes less likely

Group commitment

- Common commitment of multiple transactions
- Less forced write operations
- Increased throughput (with slightly increased protocol runtime)

Cooperative termination

- Participants know each other
- Participants can ask other participants in case of site failures to avoid blocking

Extensions (2)

Presumed abort/presumed commit

- Set a default value for the result of a transaction if no specific outcome record in the log exists
- Simplification possible for log shortening

Decentralized two phase commit protocol

- No central coordinator
- Communication among participants, e. g., preferable via broadcast transmissions
- Reduces time complexity

Important takeaway messages of this chapter

- Transactions reduce the complexity of distributed applications
- The corresponding error model must distinguish between transaction, site, media and communication errors
- The two phase commit is the most commonly used commit protocol