Two-phase commit protocol (2PC)

This protocol supports the communication between all involved servers of the distributed transaction in order to jointly decide if the transaction should commit or abort.

We can distinguish between two phases:

- **Voting phase**: the servers submit their vote whether they are prepared to commit their part of the distributed transaction or they abort it.
- **Completion phase**: it is decided whether the transaction can be successfully committed or it has to be aborted; all servers must carry out this decision.

**Steps of the two-phase commit protocol**

1. Coordinator C contacts all servers $S_i$ of the distributed transaction trans requesting their status for the commit ($CimCommit$):
   - if server $S_i$ is not ready, i.e. it votes no, then the transaction part at $S_i$ is aborted;
   - $\exists i$ with $S_i$ is not ready
     then trans is aborted; the coordinator sends an abort message to all those servers who have voted with ready (i.e. yes).
2. $\forall i$ with $S_i$ is ready, i.e. commit transaction trans. Coordinator sends a commit message to all servers.
3. Servers send an acknowledgement to the coordinator.
The coordinator communicates with the participants to carry out the two-phase commit protocol by means of the following operations:

- **canCommit(trans)** ⇒ Yes/No: call from the coordinator to ask whether the participant can commit a transaction; participant replies with its vote.
- **doCommit(trans)**: call from the coordinator to tell participant to commit its part of a transaction.
- **doAbort(trans)**: call from the coordinator to tell participant to abort its part of a transaction.
- **haveCommitted(trans, participant)**: call from participant to coordinator to confirm that it has committed the transaction.
- **getDecision(trans)** ⇒ Yes/No: call from participant to coordinator to ask for the decision on trans.

Number of messages: \( 4 \times N \) messages for \( N \) servers.

During the 2PC process several failures may occur. One of servers crashes.

- The coordinator crashes.

Depending on their state, this may result in blocking situations, e.g., the coordinator waits for the commit acknowledge of a server, or a server waits for the final decision (commit or abort).

**Extended 2PC**

- Three-Phase Commit protocol (3PC) is another approach to overcome blocking of servers until the crashed coordinator recovers.

Number of messages: \( 4 \times N \) messages for \( N \) servers.
Distributed transactions are an important paradigm for designing reliable and fault tolerant distributed applications; particularly those distributed applications which access shared data concurrently.

**General observations**

**Isolation**

**Atomicity and persistence**

**Two-phase commit protocol (2PC)**

**Distributed Deadlock**

Multiple transactions may access objects of multiple servers resulting in a distributed deadlock. At object access the server lock manager locks the object for the transaction.

Deadlock detection schemes try to find cycles in a wait-for graph.

- edge chasing: construct a global wait-for graph from all local wait-for graphs of the involved servers. Problems: the central server is a single point of failure, communication between servers take time.
distributed approach to deadlock detection

no global wait-for graph is constructed.

each involved server has some knowledge about the edges of the wait-for graph.
servers attempt to find cycles by forwarding messages (called probes).
each distributed transaction T starts at a server \( \rightarrow \) the coordinator of T.
the coordinator records whether T is active or waiting for a particular object on a server.
lock manager informs coordinator of T when T starts waiting for an object and when T acquires finally the lock.

**Edge Chasing Algorithm**

**Transaction Priorities**

---

The algorithm consists of 3 steps: initiation, detection and resolution.

\textbf{deadlock detected}

\begin{align*}
    W & \rightarrow U \rightarrow V \rightarrow W \\
    \text{server} Z & \rightarrow \text{coordinator} W \\
    \text{object} A & \rightarrow \text{held by} \\
    \text{server} X & \rightarrow \text{held by} \\
    \text{initiation} & : \text{server} X \text{ notes that } W \text{ is waiting for another transaction } U; \text{ it sends the probe } "W \rightarrow U" \text{ to the server of } B \text{ via the coordinator of } U. \\
    \text{detection} & : \text{detection consists of receiving probes and deciding whether a deadlock has occurred and whether to forward the probes.} \\
    \text{resolution} & : \text{when a cycle is detected, a transaction in the cycle is aborted to break the deadlock.} \\
\end{align*}

deadlock detected

\begin{align*}
    W & \rightarrow U \rightarrow V \rightarrow W \\
    \text{server} Z & \rightarrow \text{coordinator} W \\
    \text{object} A & \rightarrow \text{held by} \\
    \text{server} X & \rightarrow \text{held by} \\
    \text{initiation} & : \text{server} X \text{ notes that } W \text{ is waiting for another transaction } U; \text{ it sends the probe } "W \rightarrow U" \text{ to the server of } B \text{ via the coordinator of } U. \\
    \text{detection} & : \text{detection consists of receiving probes and deciding whether a deadlock has occurred and whether to forward the probes.} \\
    \text{resolution} & : \text{when a cycle is detected, a transaction in the cycle is aborted to break the deadlock.} \\
\end{align*}

\begin{align*}
    W & \rightarrow U \rightarrow V \rightarrow W \\
    \text{server} Z & \rightarrow \text{coordinator} W \\
    \text{object} A & \rightarrow \text{held by} \\
    \text{server} X & \rightarrow \text{held by} \\
    \text{initiation} & : \text{server} X \text{ notes that } W \text{ is waiting for another transaction } U; \text{ it sends the probe } "W \rightarrow U" \text{ to the server of } B \text{ via the coordinator of } U. \\
    \text{detection} & : \text{detection consists of receiving probes and deciding whether a deadlock has occurred and whether to forward the probes.} \\
    \text{resolution} & : \text{when a cycle is detected, a transaction in the cycle is aborted to break the deadlock.} \\
\end{align*}

---

**Edge Chasing Algorithm**

**Transaction Priorities**

---

Every transaction involved in a deadlock cycle may cause the initiation of deadlock detection
several servers initiate deadlock detection in parallel.

possible more than one transaction in a cycle is aborted.

\begin{align*}
    \text{Example:} \\
    \text{transaction } T \text{ attempts to access an object } A \text{ locked by } U \\
    \text{transaction } W \text{ attempts to access an object } B \text{ locked by } V
\end{align*}
**Transaction Priorities**

Every transaction involved in a deadlock cycle may cause the initiation of deadlock detection. Several servers initiate deadlock detection in parallel. It is possible that more than one transaction in a cycle is aborted.

**Example:**
- Transaction T attempts to access an object A locked by U
- Transaction W attempts to access an object B locked by V

![Diagram of deadlock detection]

**Edge Chasing**

A distributed approach to deadlock detection is implemented where no global wait-for graph is constructed. Each involved server has some knowledge about the edges of the wait-for graph. Servers attempt to find cycles by forwarding messages (called probes). Each distributed transaction T starts at a server as the coordinator of T. The coordinator records whether T is active or waiting for a particular object on a server. When a lock manager informs coordinator of T when T starts waiting for an object and when T acquires a lock.

**Edge Chasing Algorithm**

**Transaction Priorities**

---

**Introduction**

Group communication facilitates the interaction between groups of processes.

- **Motivation**
- **Important issues**
- **Conventional approaches**
- **Groups of components**
- **Management of groups**
- **Message dissemination**
- **Message delivery**
- **Taxonomy of multicast**
- **Group communication in ISIS**
- **JGroups**
Motivation

Many application areas such as CSCW profit immensely if primitives for a group communication are supported properly.

- Typical application for group communication
  - Fault tolerance using replicated services, e.g. a fault-tolerant file service.
  - Object localization in distributed systems; request to a group of potential object servers.
  - Conferencing systems and groupware.

- Functional components (e.g. processes) are composed to a group; a group is considered as a single abstraction.

Important issues of group communication are the following:

- **Group membership**: the structural characteristics of the group; composition and management of the group.
- **Support of group communication**: the support refers to group member addressing, error handling for members which are unreachable, and the message delivery sequence.
- Communication within the group
  - Unicast, broadcasting, multicasting
- Multicast messages are a useful tool for constructing distributed systems with the following characteristics
  - Fault tolerance based on replicated services.
  - Locating objects in distributed services.
  - Multiple update of distributed, replicated data.

- Synchronization
  - The sequence of actions performed by each group member must be consistent.

Groups of components

**Group addressing**
- Central approach: There is a central group server which knows the current state of the group composition.
- Decentralized approach: Each group member is aware of the group structure and its members.

**Communication services**
- This issue refers to the technology used for the communication between group members.
  - Packets (for example UDP).
  - Reliable data stream (for example TCP).

In order to get a consistent global group behavior, even in case of errors, a special group communication support is needed, for example ISIS (and the succeeding project Horus) by Cornell University.

**Classification of groups**

- **Closed vs. open group**
  - Distinction between flat and hierarchical group. A flat group may also be called a peer group.
  - Distinction between implicit (anonymous) and explicit group.
  - In the first case, the group address is implicitly expanded to all group members.
Classification of groups

Groups can be categorized according to various criteria.

Closed vs. open group

- Distinction between flat and hierarchical group. A flat group may also be called a peer group.
- Distinction between implicit (anonymous) and explicit group.

In the first case, the group address is implicitly expanded to all group members.

Group management architecture

Again, there are different approaches for providing the group management functionality. Centralized group managers, realized as an individual group server, decentralized approach, i.e., all components perform management tasks.

Hybrid approach

for each LAN cluster, there is a central group manager. Replication of group membership information and consistency control is limited to the group managers. A group manager knows all local components, as well as the remote group managers;
on executing a group function (e.g., a modification of the group membership), it contacts the local components and also propagates the information to all other group managers.

For message dissemination to the group members the following mechanisms are possible options:

- Unicast: send and receive messages addressed to individual group members.
- Group multicast: send and receive messages addressed to the group as a whole.
- Inter-group multicast: send and receive messages addressed to several groups.
- Broadcast: send and receive messages addressed to all components (requires filtering).

Hybrid approach for wide-area networks
Introduction

Group communication facilitates the interaction between groups of processes.

Motivation

Important issues

Conventional approaches

Groups of components

Management of groups

Message dissemination

Message delivery

Taxonomy of multicast

Group communication in ISIS

JGroups