Script generated by TTT

Title: Distributed_Applications (12.06.2012)

Date: Tue Jun 12 14:31:37 CEST 2012

Duration: 72:44 min

Pages: 28

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

Operations

Communication in the two-phase commit protocol

Problems

Generated by Targeteam

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

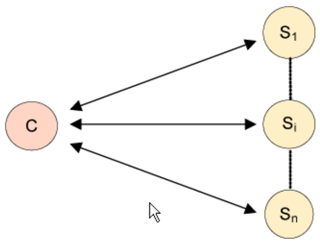
Operations

Communication in the two-phase commit protocol

Problems

Generated by Targeteam

One component (e.g. the client initiating the transaction or the first server in the transaction) becomes the ***coordinator*** for the commit process. In the following we assume, client C is the coordinator.



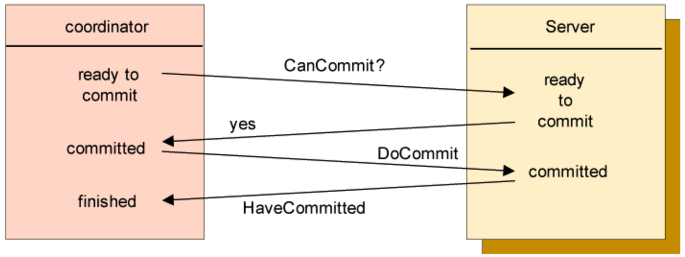
1. Coordinator C contacts all servers S_i of the distributed transaction trans requesting their status for the commit (CanCommit?)
   - if server S_k is not ready, i.e. it votes no, then the transaction part at S_k is aborted;
   - $\exists$ i with S_i is not ready

     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.

Generated by Targeteam

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 * N messages for N servers.

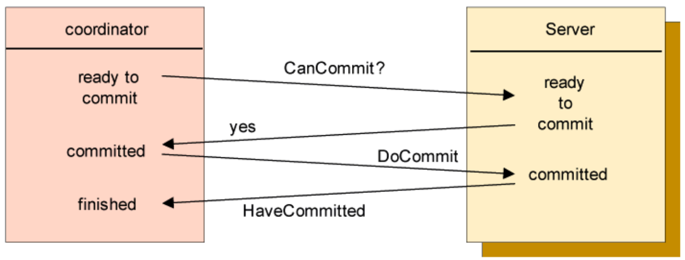
During the 2PC process several failures may occur

- 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 * N messages for N servers.

```
Coordinator:
    multicast: ok to commit?
    collect replies
        all ok =>
                log commit to outcomes table
                wait until saved to persistent store
                send commit
        else   => send abort
    collect acknowledgements
    garbage collect data from outcomes table

After Failure:
    for each pending protocol in outcomes table
        send outcome (commit or abort)
        wait for acknowledgements
        garbage collect data from outcomes table
```

```
Server: first time message (CanCommit) received
    ok to commit =>
        save data to temp area (persistent store)
        reply ok
    commit =>
        make change permanent
        send acknowledgement
    abort  => delete temp area

message is a duplicate (recovering coordinator)
    send acknowledgement

After Failure:
    for each pending protocol
        contact coordinator to learn outcome
```


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.

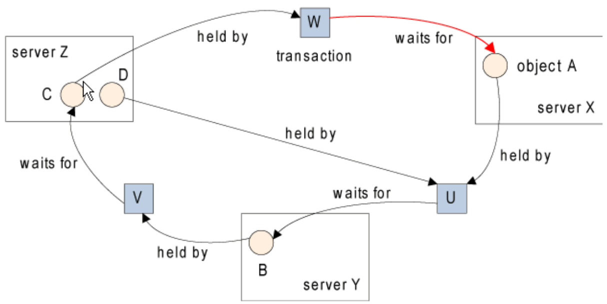
Distributed transactions

Distributed transactions are an important paradigm for designing reliable and fault tolerant distributed applications; particularly those distributed applications which access shared data concurrently.

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.



theory: 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.

Edge Chasing

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 ⇒ the ***coordinator*** of T.

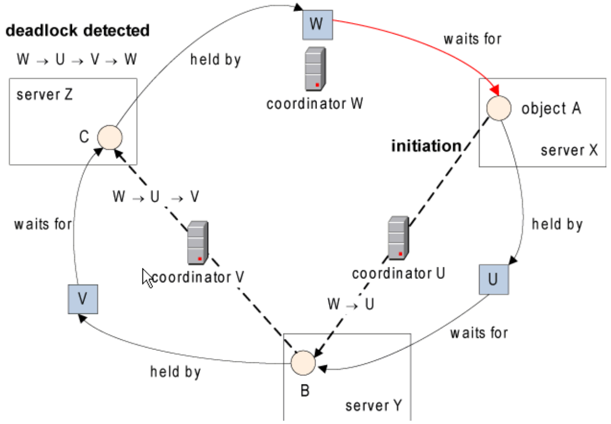
the coordinator records whether T is active or waiting for a particular object on a server.

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.

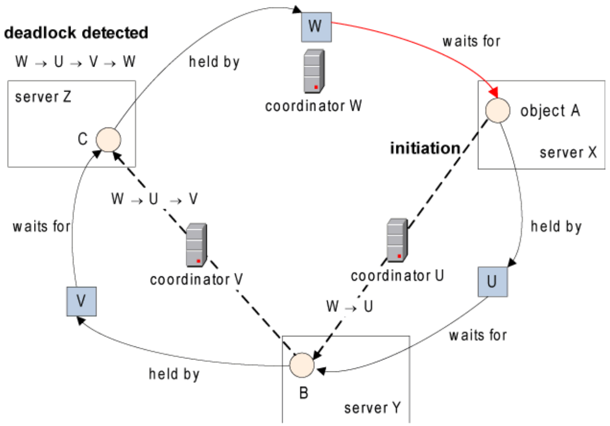


initiation : server X notes that W is waiting for another transaction U; it sends the probe "W → U" to the server of B via the coordinator of U.

detection : detection consists of receiving probes and deciding whether a deadlock has occurred and whether to forward the probes.

Server Y receives the probe "W → U"; it notes B is held by transaction V and appends V to the probe to produce "W → U → V"; probe is forwarded to server Z via coordinator of V.

Edge Chasing Algorithm



initiation : server X notes that W is waiting for another transaction U; it sends the probe "W → U" to the server of B via the coordinator of U.

detection : detection consists of receiving probes and deciding whether a deadlock has occurred and whether to forward the probes.

Server Y receives the probe "W → U"; it notes B is held by transaction V and appends V to the probe to produce "W → U → V"; probe is forwarded to server Z via coordinator of V.

resolution : when a cycle is detected, a transaction in the cycle is aborted to break the deadlock.

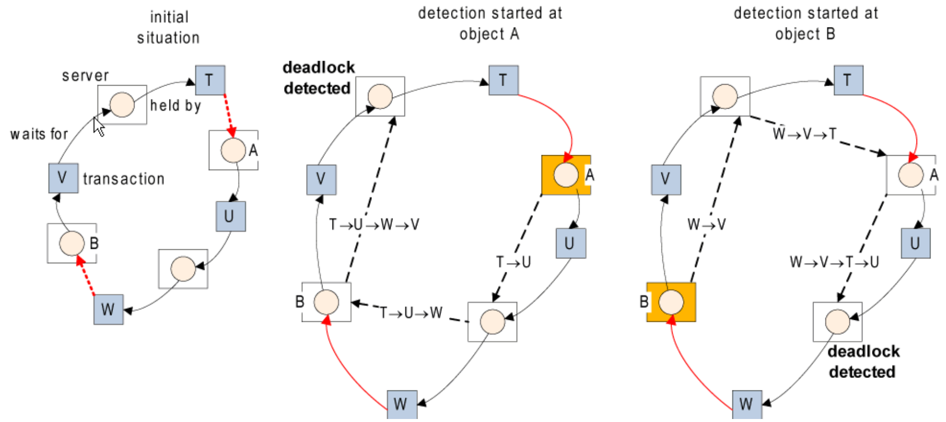
Transaction Priorities

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.

Example:

- transaction T attempts to access an object A locked by U
- transaction W attempts to access an object B locked by V



Transaction Priorities

Every transaction involved in a deadlock cycle may cause the initiation of deadlock detection

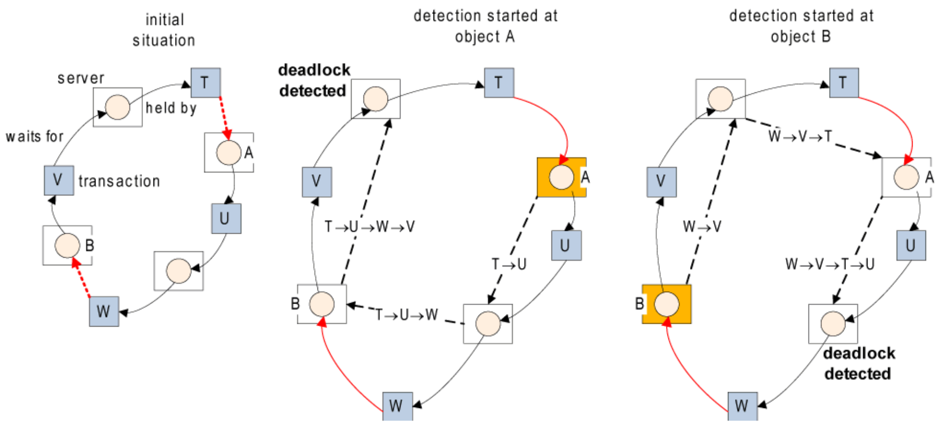
several servers initiate deadlock detection in parallel

⇒ possible more than one transaction in a cycle is aborted.

Example:

transaction T attempts to access an object A locked by U

transaction W attempts to access an object B locked by V



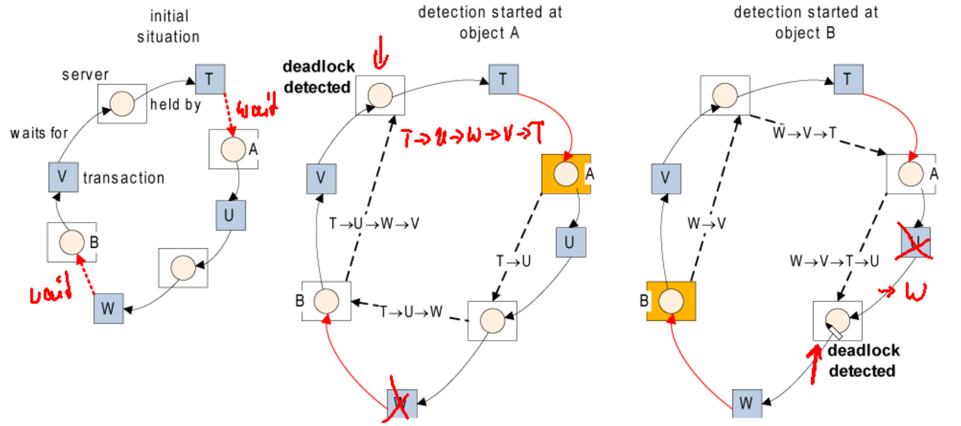
Transaction Priorities

⇒ possible 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



transactions are totally ordered by priorities.

in a cycle, transaction with lowest priority is aborted.

Edge Chasing

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 ⇒ 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]

Introduction

Group communication facilities the interaction between groups of processes.

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

- unicasting, 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.

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.
  - Datagrams (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.

Groups of components

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.

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.

Generated by Targeteam

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.

requires replication of group membership information, i.e. consistency must be maintained.

joining and leaving a group must happen synchronously.

Hybrid approach

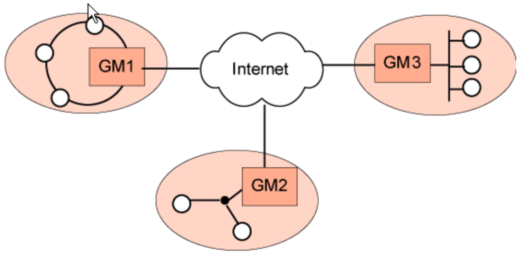
Generated by Targeteam

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.



Generated by Targeteam

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

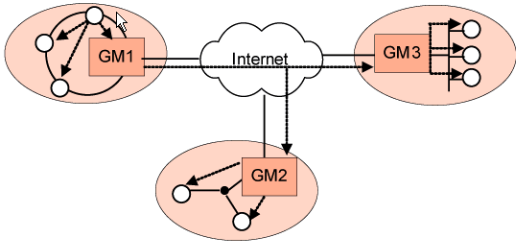
Unicast: send and receive messages addressed to individual group members.

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



Generated by Targeteam

Group communication

Introduction

Group communication facilities the interaction between groups of processes.