Script **generated by TTT**

Title: Distributed_Applications (11.06.2013)

Date: Tue Jun 11 14:30:54 CEST 2013

Duration: 61:38 min

Pages: 23

Distributed transactions

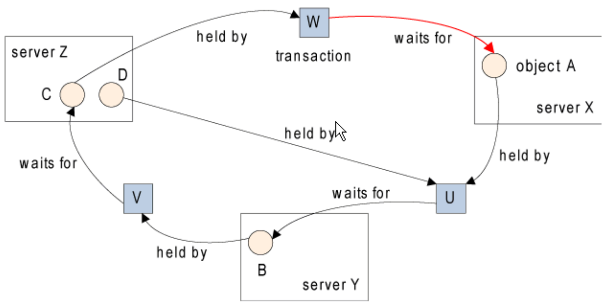
Distributed transactions are an important paradigm for designing reliable and fault tolerant distributed applications; particularly those distributed applications which access shared data concurrently.

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](#)

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.

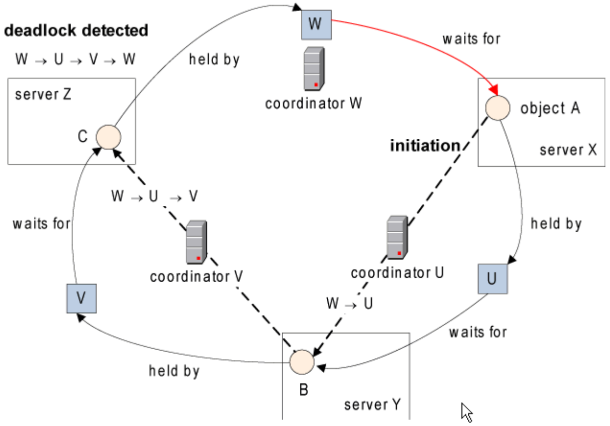
lock manager informs coordinator of T when T starts waiting for an object and when T acquires finally the lock.

[Edge Chasing Algorithm](#)

[Transaction Priorities](#)

Edge Chasing Algorithm

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

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.

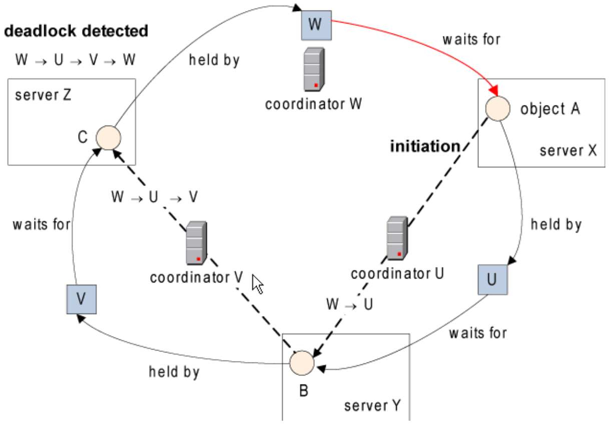
lock manager informs coordinator of T when T starts waiting for an object and when T acquires finally the lock.

Edge Chasing Algorithm

Transaction Priorities

Edge Chasing Algorithm

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.

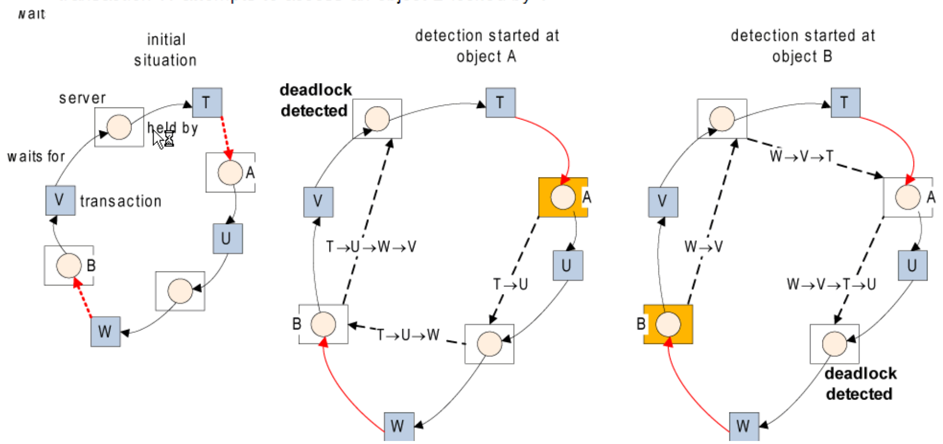
Transaction Priorities

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



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.

wait 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.

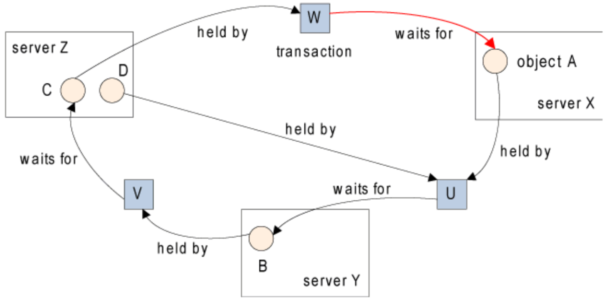
[Edge Chasing Algorithm](#)

[Transaction Priorities](#)

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](#)

Group communication

Introduction

Group communication facilities 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

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.

Introduction

Group communication facilities the interaction between groups of processes.

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.

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

Group communication

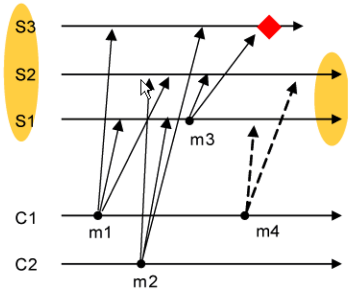
Introduction

Group communication facilities the interaction between groups of processes.

Sequence of message delivery

It is desired to deliver all messages sent to the group G to all group members of G in the ***same sequence*** , because otherwise we might get non-deterministic system behavior.

Example for group reconfiguration



m4 is sent by C1 before the group composition is modified. However, in order to guarantee atomicity, m4 should not be delivered to S1 and S2 (since, due to the crash, it is no longer possible to deliver m4 to S3).

Message delivery

Message delivery is an important issue of group communication; two aspects are relevant:

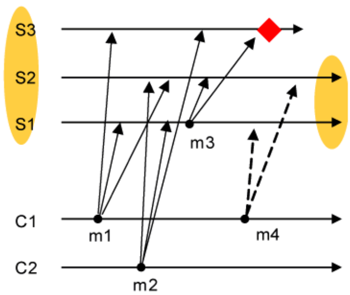
a) ***who*** gets the message, and

b) ***when*** is the message delivered.

Sequence of message delivery

It is desired to deliver all messages sent to the group G to all group members of G in the ***same sequence*** , because otherwise we might get non-deterministic system behavior.

Example for group reconfiguration



m4 is sent by C1 before the group composition is modified. However, in order to guarantee atomicity, m4 should not be delivered to S1 and S2 (since, due to the crash, it is no longer possible to deliver m4 to S3).

Message delivery

Message delivery is an important issue of group communication; two aspects are relevant:

a) ***who*** gets the message, and

b) ***when*** is the message delivered.

Atomicity

Sequence of message delivery

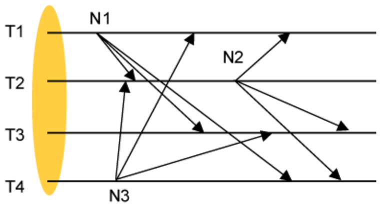
Ordering for message delivery

Virtually synchronous ordering

determination of a correct sequence based on the ***before*** relation between two events modeling their causal dependency (see causally distributed breakpoints).

Example

1. T_1 sends N_1 , and T_2 sends N_2 with N_2 dependent on N_1
2. T_4 sends N_3 with N_1 and N_3 concurrent
3. at T_2 : N_3 is received before N_1
4. at T_3 : N_3 is received after N_1



sync-ordering

This approach for message delivery introduces synchronization points. Synchronously ordered messages are delivered to all group members ***in-sync*** .

let N_i be a synchronously ordered message

all other messages N_k are delivered either before or after N_i has been delivered to all group members.

The ordering method enables the group to synchronize their local states (at synchronization points the group members have a common consistent state).

Ordering for message delivery

Delivery of messages without delay in the same sequence is not possible in a distributed system ⇒ ordering methods for message delivery.

synchronously, i.e. there is a system-wide global time ordering.

loosely synchronous, i.e. consistent time ordering, but no system-wide global (absolute) time.

Total ordering by sequencer

Virtually synchronous ordering

sync-ordering