

Department of Computer Science

PURDUE
UNIVERSITY

CS505: Distributed Systems

Lecture 11: Group Communication and
Replication

Outline

- ▶ **Group Communication, Replication Models**
- ▶ **From Consensus to Atomic Broadcast**
- ▶ **Virtual Synchrony**

Group Communication

- ▶ **E.g., we want to replicate a “server”**
 - Use a *group* of servers $G = \{p_1, p_2, \dots\}$
 - Address them as a single logical entity from outside (i.e., from application)
- ▶ **Static group/membership**
 - Failed processes are not removed, as they eventually recover; $G = \text{const.}$
 - Subset of processes of G are available at any time
- ▶ **Dynamic group/membership**
 - Failed processes are removed, new processes can be added; $G(t)$
 - Virtual synchrony
 - Time reflected by subsequent logical *views*
 - Processes are removed/added upon view changes
 - A messages is delivered within a given view

Active Replication

▶ State machine replication [Schneider'90]

- Every replica delivers every request
- Every replica processes every request

▶ Requirements

- Deterministic operations
 - Worst case single request executed at a time; no multi-threading
- Requests are delivered in “compatible” orders, e.g.,
 - Total order for updates, causal
 - Reliable delivery for pure reads

▶ Evaluation

- Little concurrent execution
- Little noticeable service degradation in case of failures

Passive Replication

- ▶ **Primary-backup replication [BMST'93]**
 1. **Primary**
 - Delivers requests
 - Processes requests
 - Computes state update
 2. **Backups**
 - Apply updates

- ▶ **Upon failure of the primary a new one is chosen**
 - Leader election; *P*

► Evaluation

- Non-deterministic operations can be supported
- Multi-threading within leader can be supported
- Noticeable service disruption in case of primary failure
- What if state updates are of significant size?

► Variant: coordinator-cohort

- Primary is not fixed

Semi-Active Replication

- ▶ **All replicas still process all requests**
 - Still need to be deterministic
- ▶ **However**
 - “Primary” replica computes order of requests
 - Requests replayed on backups in order of “occurrence” on primary
- ▶ **Evaluation**
 - Supports some multi-threading
 - Still relatively high fail-over time

Semi-Passive Replication

▶ No fixed primary [DSS'98]

- Coordinator of consensus plays role of primary
- Proposed value is result/update pair of a computed request
- Upon decision, other processes apply update and return reply to client

▶ Evaluation

- Needs “undo” facility for operations
- Can be mixed with active replication

Summary

| | Active | Passive | Semi-active | Semi-passive |
|----------|--|---|--|-----------------------------------|
| + | Fast fail-over | Supports non-determinism Supports multi-threading | Some multi-threading | Fast fail-over Non-determinism |
| ■ | Redundant computation Only deterministic operations Single threads | Slow fail-over Operations on large data yielding large state updates <i>P</i> | Redundant computation Only deterministic operations | Operations on large data |

State Machine Replication

- ▶ **Cornerstone is total order broadcast**
 - Possibly with causal order
- ▶ **If possible, closer look at semantics of operations**
 - Writes need to be totally ordered
 - Reads need to be reliable
 - Causal/FIFO depending on requirements

Total Order Broadcast

I. No duplication

- No message is delivered more than once

II. No creation

- No message is delivered unless it was broadcast

III. Validity

- If p_i and p_j are correct, then every message broadcast by p_i is eventually delivered by p_j

IV. (Uniform) agreement

- If a correct (or not) process delivers a message m , every correct process eventually delivers m

Total Order Broadcast (2)

V. (Uniform) total order

- Let p_i and p_j be any two correct (or not) processes that deliver two messages m_1 and m_2 . If p_i delivers m_1 before m_2 , then p_j delivers m_1 before m_2

▶ How about the following property instead of V.
[BCM'05]?

VI. Strong (uniform) total order

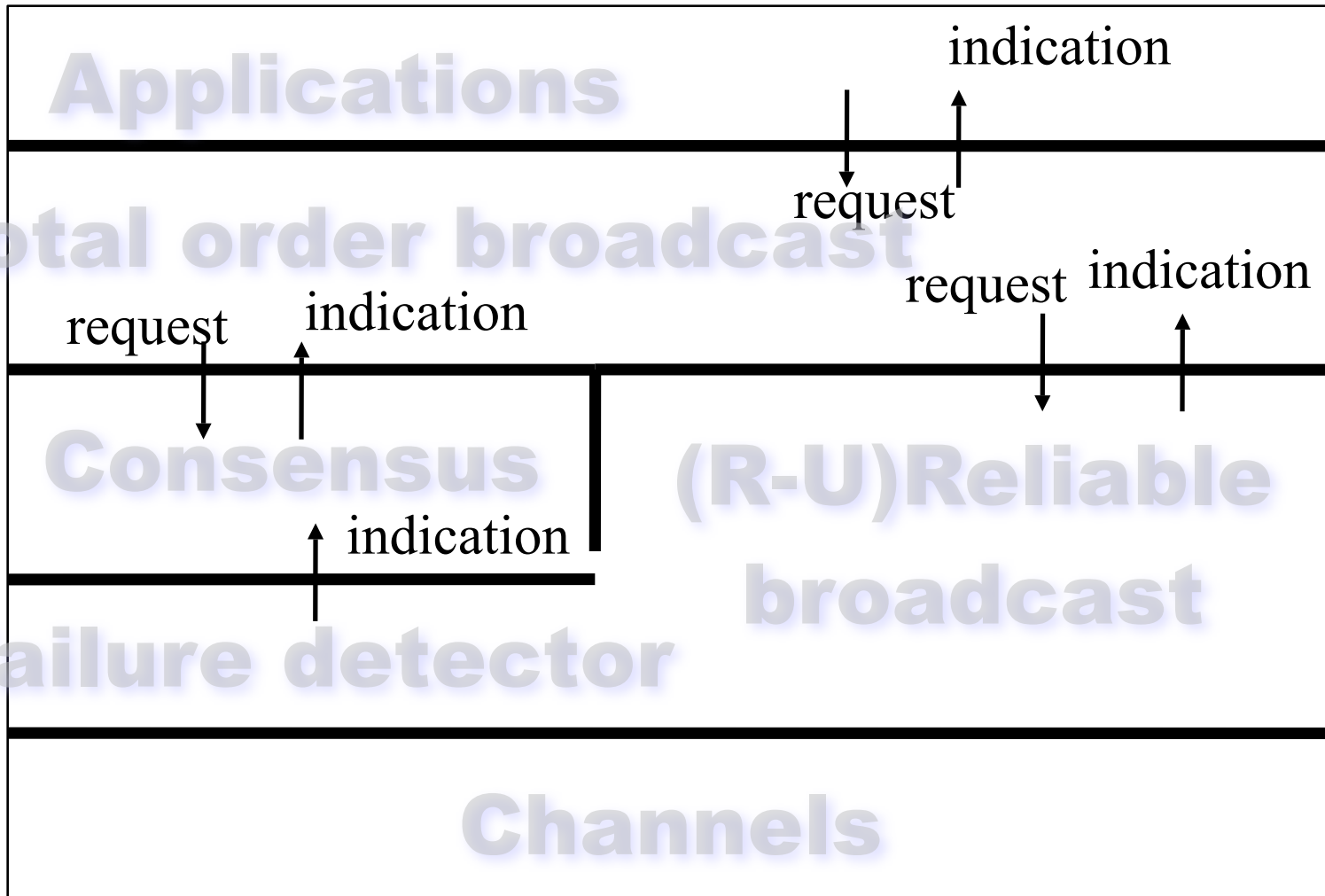
- Let p_i and p_j be two correct (or not) processes that deliver m_1 . If p_i delivers a message m_2 before m_1 , then p_j also delivers m_2 before m_1

Total Order Broadcast with Consensus

- ▶ **Successive executions of consensus**
 - Increasing numbers

- ▶ ***Multi-valued* consensus**
 - Participants propose (sets of) messages

Overview



Proposition with Uniform Consensus

```
rdelivered ← empty
todelivered ← empty
wait ← false
sn ← 1

to execute broadcastTO(m) do
  broadcastR(m)

upon deliverR(m) do
  rdelivered ← rdelivered ∪ {m}

upon (rdelivered - todelivered not empty) and not(wait) do
  wait ← true
  propose(rdelivered - todelivered, sn)

upon decide(decided, sn) do
  todelivered ← todelivered ∪ decided
  ordered ← deterministicSort(decided) \\ use unique ids of messages
  for all m ∈ ordered in order
    deliverTO(m)

  sn ← sn + 1
  wait ← false
```

▶ **What if a broadcaster fails?**

- And a single process `deliverRs` a message and fails?

▶ **What if a process `deliverTOS` `m` before it `deliverRs` `m`?**

▶ **Specification?**

- Uniform agreement or agreement?
- Uniform total order or total order? Strong?

▶ **Limitations? Optimizations?**

Equivalences

- 1. One can build total order broadcast with consensus and reliable broadcast**
 - 2. One can build consensus with total order broadcast**
 - How?
- ▶ **Therefore, consensus and total order broadcast are equivalent problems in a system with reliable channels**

Causal Atomic Broadcast Revisited

- ▶ With FIFO Atomic Broadcast (implemented with FIFO Broadcast on Consensus, yielding uniform agreement)
- ▶ Proposition

to execute broadcast_{CA}(m) by p_i :

broadcast_{FA}(m)

deliver_{CA}(m) occurs as follows:

upon deliver_{FA}(m) do

deliver_{CA}(m)

► Claim

- If m from $\text{deliver}_{CA}(m)$ influenced $\text{broadcast}_{CA}(m')$
- $\text{deliver}_{CA}(m)$ preceded $\text{broadcast}_{CA}(m')$
- (locally) $\text{deliver}_{CA}(m')$ can not precede $\text{deliver}_{CA}(m)$ anymore
- Thus $\text{deliver}_{CA}(m')$ can not precede $\text{deliver}_{CA}(m)$ anymore on any process

► Uniform agreement?

Virtual Synchrony

▶ *Dynamic* groups

- Processes which fail are excluded
- New processes can join
- Set of processes in a group varies over time

▶ Problem

- Absence of synchrony, global time
- Which processes are part of the group at what point?
- *Views* may vary at different processes, e.g., due to asynchrony but also diverging failure detector outputs

Idea

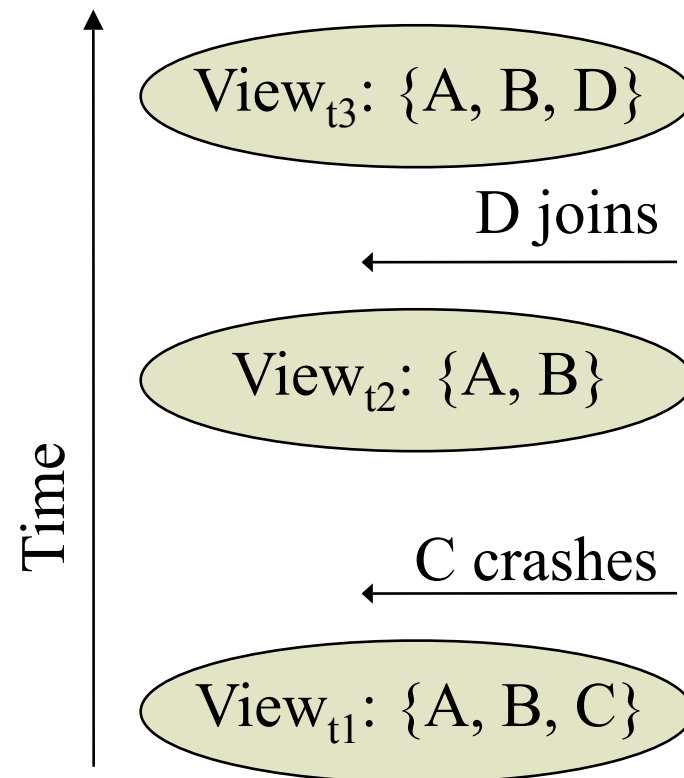
► *Logical time*

- Define group composition according to messages delivered
 - Messages may impact “distributed” state
- Sequence of views, uniquely identified by monotonically increasing counter
 - View is set M of processes, uniquely identified
 - Initially $M = \{p_1, \dots, p_n\}$

► More precisely

- Every suspicion or addition of process leads to a view change
- Messages are delivered w.r.t. to views
 - Any message delivered by a correct process within a view is delivered by other correct processes in that view
- Views are delivered in total order (to those involved)

Views



Reliable Broadcast with Virtual Synchrony

I. Validity

- If a correct process executes $\text{broadcast}^v(m)$, then it eventually $\text{delivers } m$

II. Termination

- If a process executes $\text{broadcast}^v(m)$, then eventually (1) every process in the view v does $\text{deliver}^v(m)$ or (2) every correct process in v installs a new view

III. View synchrony

- If process p_i belongs to two consecutive views v and v' , and does $\text{deliver}^v(m)$, then every process p_j in $v \cap v'$ that installs v' , also does $\text{deliver}^v(m)$ before installing v'

IV. Sending view delivery

- If $\text{deliver}^{v'}(m)$ and $\text{broadcast}^v(m)$ occur, then $v' = v$

V. Integrity

- For any message m , any process $\text{delivers } m$ at most once, and only if m was previously broadcast

Implementation

▶ New primitive

- `install(v)` notifies process of adoption of new view v

▶ Upon suspicion

- View change is proposed; consensus

▶ Flush phase

- All “preceding” messages must be delivered

▶ Failure detector?

- ... “program-controlled crash”
- Improvement: different time-outs for suspicions leading to view changes and for such leading to progressing in consensus

Alternatives

- ▶ Sending view delivery is strong
- ▶ Alternative specification for IV.

VI. Same view delivery

If process p_i does $\text{deliver}^v(m)$ and process p_j does $\text{deliver}^{v'}(m)$, then $v' = v$

Extended Virtual Synchrony

▶ View changes are costly

- Require agreement on message sets *and* membership
 - During view change “new” application messages are blocked/buffered
- If some messages can be delivered after the membership change the view change overhead can be reduced

▶ Further variants

- Semantically correct (“obsolescence”) [PRO02]
- Optimistic VS [SKM00]
 - Intention of view change is indicated
 - Process may then still broadcast (optimistic) if same view delivery is sufficient
 - Idea subsequently extended to have “transitional views”
- ...

References

- ▶ ***Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial.*** F.B. Schneider. ACM Surveys 22(2): 299-319, 1990.
 - Or: *Replication Management Using the State Machine Approach.* F.B. Schneider. Distributed Systems, 169-197, 1993, Addison-Wesley.
- ▶ ***The Primary-Backup Approach.*** N. Budhiraja, K. Marzullo, F.B. Schneider, S. Toueg. Distributed Systems, 199-216, 1993, Addison-Wesley.
- ▶ ***Lightweight Causal and Atomic Multicast.*** K.P. Birman, A. Schiper, P. Stephenson. ACM TOCS 3(9): 272-314, 1991.
- ▶ ***Semi-Passive Replication and Lazy Consensus.*** X. Defago, A. Schiper. JPDC 64(12): 1380-1398, 2004. Xavier Défago and André Schiper.
- ▶ ***Group Communication Specifications: A Comprehensive Survey.*** G. Chockler, I. Keidar, R. Vitenberg. CSUR 23(4): 427-469, 2001.
- ▶ ***Optimistic Virtual Synchrony.*** J. B. Sussman, I. Keidar, K. Marzullo. SRDS 2000, 42-51.
- ▶ ***Reducing the Cost of Group Communication with Semantic View Synchrony.*** J.O. Pereira, L. Rodrigues, R. C. Oliveira. DSN 2002, 293-302.
- ▶ ***Total Order Communications: A Practical Analysis.*** R. Baldoni, S. Cimmino, C. Marchetti. EDCC-5, 38-54, 2005.