Seminar
Fault Tolerant Systems

Group Communication
and
reliable Multicast

Mario Oschwald
mario.oschwald@hpi.uni-potsdam.de
Agenda

- Motivation
- ISIS background
- Scope of the presented communication system
- System characteristics and notions
- Site view management
- Broadcast primitives protocols and correctness
Motivation

- In replicated systems it is often desirable to communicate with a group of system components.
- A message sent to a group should either reach all or no components.
- Messages sent to a group should reach the members in the same order.
- Group members should have a consistent view of group membership.
- In case of failure of a group member these constraints should still be followed.
ISIS Background

- First in a series of four reliable distributed system developed at Cornell University, New York by Kenneth P. Birmann (Isis, Horus, Ensemble, QuickSilver)
- Developed mainly during 1987-1993
- Toolkit that provides mechanisms for distributed computing
  - managing replicated data
  - synchronizing distributed computing
  - automatic recovery
  - dynamic reconfiguration to manage changing workloads
- Marketed commercially after 1993 by Stratus Technologies
- Phased out of the market by 1998
Scope

- Only halting failures of system process are handled
- The system is absolutely cooperative and does not handle byzantine errors
- Only the following types of events are considered:
  - local computations by a process
  - broadcast from a process to a set of processes
  - broadcasts subject to predetermined ordering constraints
  - process failures
  - process recoveries
System Characteristics and Notions

- System consists of a collection of processes with a local state
- Processes communicate through messages
- No shared memory or synchronized clocks
- Failure is always a halting failure, a process always stops executing with no incorrect or malicious actions
- No information survives a failure
- Process groups are a set of processes that can be addressed directly
Site View Management

Overview

☐ Each site has a site view – a set of site it thinks operational
☐ A site view is changed when a site fails or recovers
☐ A site view sequence \( V_0, V_1, \ldots \) is a sequence reflecting these changes
☐ The site view management protocol assures that each site goes through the same sequence of site views
☐ All sites can be ordered by a site ID
☐ the “oldest” site is the view manager
☐ when the view manager detects a failure or recovery it initiates the following protocol:
Site View Management Protocol

1. The view manager calculates a proposed view extension $V_k$ and sends it to all sites in $V_k$.

2. A site receives this extension and stops accepting messages not in $V_k$.
   - If the site has no proposed extension pending that contradicts $V_k$, it saves the new view and sends a positive acknowledgment.
   - Otherwise, it responds with a negative acknowledgment and sends the in $V_k$ missing events along.

3. The view manager collects all acknowledgments.
   - If all are positive, it sends a commit message to all sites in $V_k$.
   - Otherwise, or if additional failures occurred, a new proposed extension is calculated, and the protocol continues a step (1).
Site View Management Protocol (3)

☐ If a view manager fails the following steps are taken:

1. If the new view manager has an uncommitted view extension, the previous view manager may have sent some commit messages. A new site view is calculated including this pending extension and the protocol continues at (1)

2. If the new view manager has a committed extension, some sites may have not received that commit. A new view is calculated including this pending extension and the protocol starts at (1)
Site View Management

Correctness

☐ If the view manager does not fail all sites obtain the same committed view extensions

☐ If the view manager fails and any site has a committed view extension, then all sites have acknowledged that extension. It will be committed eventually everywhere

☐ If a new view manager is unaware of a proposed extension sent by the old one it will get this extension eventually through a negative acknowledgment
Fault tolerant Process Groups

- Every member of a process group needs to have the same view of group membership and global state
- Process fail or recovery events have to be handled in the same logical order by every group member
- Three broadcast primitives are formally described:
  - $\text{GBCAST}$ – group broadcast
  - $\text{ABCAST (BCAST)}$ – atomic broadcast
  - $\text{CBCAST (OBCAST)}$ – causal broadcast
- Common property atomicity – broadcasts made to a group is eventually received by all operational members
Data Structures of the broadcast protocols

(Birnam & Joseph)
ABCAST (BCAST)

Overview

- Used when the order that messages are delivered must be the same at all destinations but is not predetermined

- Two necessary constraints:
  - atomicity – every operational destination receives the broadcast
  - two ABCASTs that have destinations in common will be delivered in the same order

- Invoked as ABCAST(msg, label, dests)
  - msg – the message
  - label – a string grouping ABCASTs (only ABCASTs with the same label have to follow constraint 2
  - dests – the destinations
ABCAST (BCAST) Protocol

(1) The sender sends message to its destinations

(2) Each recipient adds message to a priority queue associated with label and tags it as undeliverable. It is tagged with a priority larger than any other message in the queue. This priority returned to the sender

(3) The sender collects all priorities and sends the maximum priority to all destinations

(4) The recipients change the priority to the new one and tag the message as deliverable. The queue is resorted and all messages at the beginning of the queue tagged deliverable are delivered to the process

☐ In case of failure of a recipient the sender doesn't wait anymore for its answer

☐ In case of failure of the sender a recipient takes over the protocol requerying all recipients for their assigned priority.
ABCAST (BCAST)

Correctness

☐ The protocol is atomic because all recipients must have received a copy before the messages can be tagged deliverable.

☐ The ordering constrained is also correctly implemented because:
  - Before a message goes to the delivery queue it has the same priority at all sites
  - If a message gets the same priority, they can be ordered using the assigners process ID
  - The priority is unique to every deliverable message and fully ordered
  - If a deliverable message is at the head of the queue no other message can ever get a lower priority
Potential Causality Revisited

(Lamport - Graphics based on J. Scheerer's)
CBCAST (OBCAST) Overview

- CBCASTs are used to enforce a certain delivery order when desired
- Only potentially causally related messages can be ordered this way
- B is potentially causally related to B' if:
  - Both are sent by the same sender and B was sent before B'
  - B was delivered to the sender of B' before B' was sent
- Invoked as CBCAST(msg, clabel, dests)
  - clabel is used to indicate that potentially causally related messages should indeed be ordered and in which order
- B precedes B' if it is potentially causally related to B' and the clables indicate a ordering
- Two necessary constraints:
  - atomicity – every operational destination receives a message or none does
  - If B precedes B' B will be delivered before B' at all overlapping destinations
CBCAST (OBCAST) Protocol

- Every process $p$ keeps a buffer $\text{BUF}_p$ of all messages it has every received or sent.
- Along with each message $B$ the remaining destinations $\text{REM}_\text{DESTS}(B)$ are stored.
- A message $B$ is transmitted with the following protocol from $p$ to $q$.

1. All messages $B_i$ preceding $B$ are sent along with $B$ to $q$.
2. $q$ is removed from all $\text{REM}_\text{DESTS}(B_i)$ if listed there.
3. For every message received $q$ does the following steps in the precedence order:
   4. If $B_z$ was already received it is discarded.
   5. If $q$ is in $\text{REM}_\text{DESTS}(B_z)$, $B_z$ is delivered and $q$ removed, a copy of $B_z$ is placed in $\text{BUF}_q$.
   6. Otherwise $B_z$ is simply put in $\text{BUF}_q$. 

Every process $p$ keeps a buffer $\text{BUF}_p$ of all messages it has every received or sent.
CBCAST (OBCAST)
Correctness

☐ Any process that receives a message adds it to its buffer from which it is eventually delivered. Thus if any site has received the message and does not fail every destination will eventually receive the message. This ensures atomicity.

☐ The order is correct because for any pair of messages with B preceding B' the following holds:
  - A copy of B will always be in the sender's buffer before B' is sent.
  - Therefore any packet containing B' will also contain B.
  - A recipient will deliver all messages in a packet ordered by precedence.
GBCAST Overview

- Used to organize a process group
- Informs group members about changes in the process view: when a member fails, recovers, joins or withdraws or when another global group property changes
- Group members should be able to act directly on GBCASTs without any further agreement protocols, therefore all GBCASTs must be ordered relative to other events in the same way at every group member
- Three necessary constraints:
  - atomicity – every operational destination receives a message or none does
  - The order of delivery relative to all other broadcasts (including GBCASTs) is the same at all destinations
  - A GBCAST indicating a failure of process p must be delivered after all messages sent by p before failing are delivered
GBCAST Protocol (1)

- Special subprotocol for failure GBCASTs (process f has failed)

1. The sender of the GBCAST first sends a message to all processes in the system informing that a GBCAST is pending.
2. Every process then schedules all CBCASTs sent by f that have nonempty REM_DESTs for transmission and waits until they are delivered.
3. Every process then waits until all available ABCASTs from f become deliverable and are delivered.
4. An acknowledgment is returned to the initiator of the GBCAST that all messages from f were delivered.
GBCAST Protocol (2)

- Subprotocol to ensure ordering relative to ABCASTs

1. The sender sends the GBCAST to all group members
2. A recipient places the GBCAST on all ABCAST priority queues
3. The priority is established just like for ABCASTs
4. The GBCAST is not tagged as deliverable!
5. The recipient waits until all ABCASTs are at the head of their queue
Subprotocol to ensure ordering relative to CBCASTs

1. The sender sends the GBCAST to all group members
2. A recipient establishes a wait queue. All CBCASTs that would have been delivered are put into this queue until the GBCAST completes
3. All participants agree on a before list of CBCASTs that should be delivered before the GBCAST
4. All not delivered CBCASTs on the before list are delivered

1. After the two subprotocols have completed
2. The GBCAST is delivered
3. All GBCASTs are removed from ABCASTs priority queues
4. All remaining CBCASTS in the wait queue are delivered
References

- Ken Birman's Homepage http://www.cs.cornell.edu/ken/
- http://www.stratus.com/