Fault-Tolerant Broadcasts and Related Problems

Tom Austin
Prafulla Basavaraja

CS 249 – Fall 2005

Introduction

- It’s not an easy task to design and verify a fault-tolerant distributed application. Consensus and several types of reliable broadcasts play an important role in simplifying this task.

- Consensus algorithms can be used to solve many problems that arise in practice, such as electing a leader.

- Reliable broadcasts and its variants are convenient when the processes should agree on a set of messages they deliver and its order.

Models of distributed computation

(Message passing model)

Synchrony

Synchrony is an attribute of both processes and communication. We say that a system is synchronous if it satisfies the following properties:

- There is a known upper bound δ on message delay; this consists of the time it takes for sending, transporting, and receiving a message over a link.

- Every process p has a local clock Cp with known bound rate of drift p>=0 with respect to real time. That is, for all p and all t > t',

$$(1+\rho)-1 \leq Cp(t) - Cp(t') / (t-t') \leq (1+\rho)$$

Where Cp (t) is the reading of Cp at the real-time t.

- There are known upper bounds on the time required by a process to execute a step.

Process Failures

The following is a list of models of failures :

- Crash: A faulty process that stops prematurely and does nothing from that point on. Before stopping, however, it behaves correctly.

- Send omission: A faulty process that stops prematurely, or intermittently omits to send messages it was supposed to send, or both.

- Receive omission: A faulty process that stops prematurely, or intermittently omits to receive messages it was sent to it, or both.

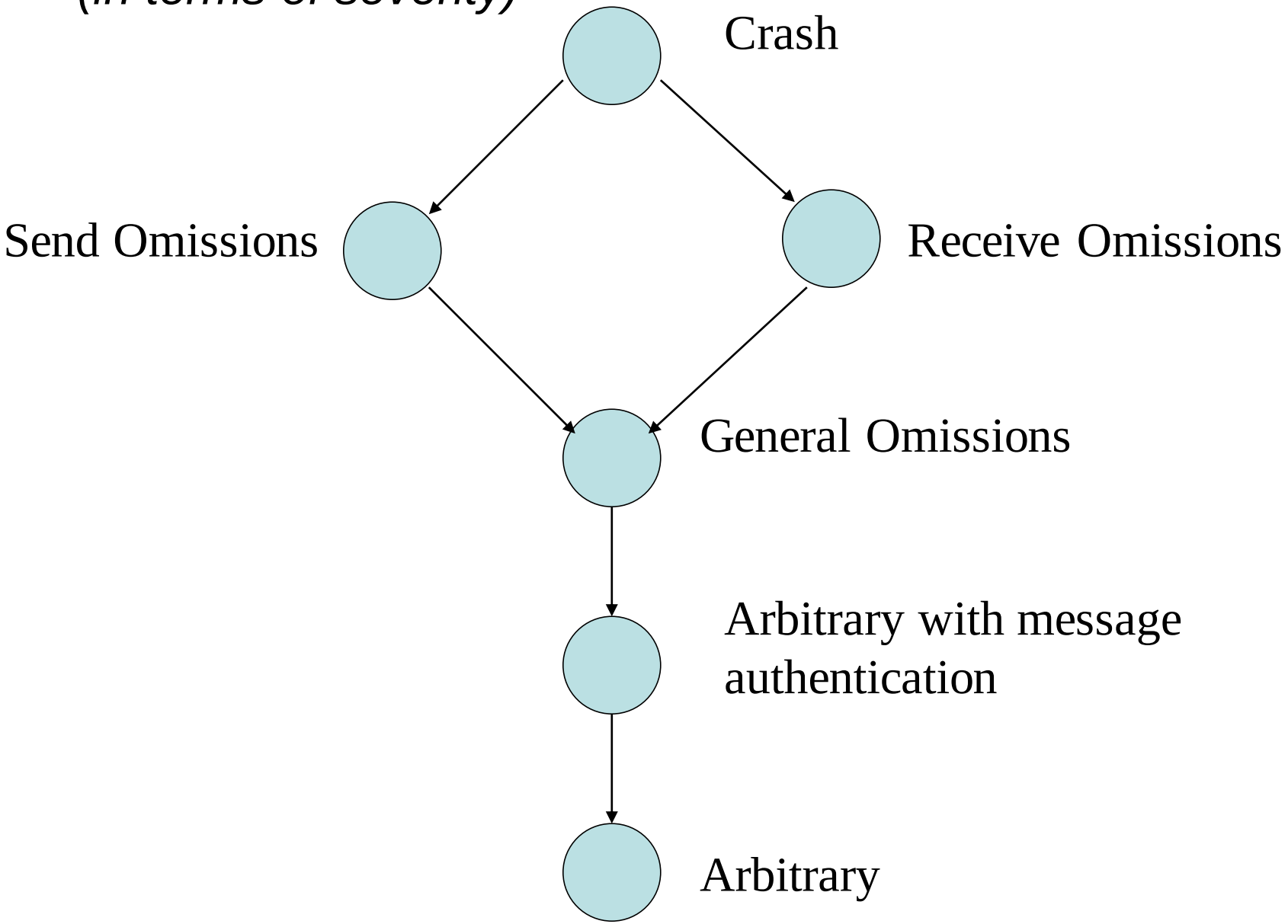
- General omission: A faulty process is subject to send or receive omission failures, or both.

- Arbitrary (sometimes called Byzantine or malicious): A faulty process can exhibit any behavior whatsoever.

- Arbitrary with message authentication: A faulty process can exhibit arbitrary behavior but a mechanism for authenticating messages are provided using *unforgeable signatures* is available.

Classification of Failure Models

(in terms of severity)



- These models of failures are applicable to both asynchronous and synchronous systems.

- Timing failure is only pertinent to synchronous system, it is more severe than general omission but less severe than arbitrary failures with message authentication.

Communication failures

- Crash: A faulty link stops transporting messages. Before stopping, however, it behaves correctly.
- Omission: A faulty link intermittently omits to transmit messages sent through it.
- Arbitrary (sometimes called Byzantine or malicious) **:** A faulty link can exhibit any behavior whatsoever. For eg, it can generate spurious messages.

In synchronous systems, we also have :

- Timing failures: A faulty link transports messages faster or slower then its specification.

Network Topology:

- The communication network can be modeled as a graph, where nodes are processes and edges are communication links between processes. This model encompasses point-to-point as well as broadcast networks

- The problems we consider are not solvable if failures result in partition of network. Thus the underlying network must have sufficient connectivity to allow correct process to communicate directly or indirectly despite process and communication failures.

Broadcast Specifications
Reliable Broadcast

- The weakest type of broadcast
- Guarantees three properties
  - Agreement: all correct processes agree on the set of messages they deliver.
  - Validity: all messages broadcast by correct process are delivered
  - Integrity: no spurious messages are ever delivered

Order of the messages are not preserved.

Reliable Broadcast using send and Receive (by message diffusion)

Every process P executes the following :

To execute **broadcast(R, m)**:

tag m with sender(m) and seq#(m) // These tag makes m unique

send m to all neighbors including p

deliver(R, m) occurs as follows:

upon receive(m) do

If p has not previously executed deliver(R, m)

then

If sender(m) ≠ p then send(m) to all neighbors

deliver(R, m)

FIFO Broadcast

- This is Reliable broadcast that guarantees that messages broadcast by the same sender are delivered in the order they were broadcast.

Using Reliable Broadcast to build FIFO Broadcast

Every process p executes the following:

Initialization:

msgBag := Ø //set of msgs that p R-delivered but not yet F-delivered

Next[q] := 1 for all q // seq num of next msg from q that p will F-deliver

To execute broadcast(F,m):

broadcast(R, m)

deliver(F, m) occurs as follows:

upon deliver(R,m) do

q := sender(m)

msgBag := msgBag U {m}

While ($\exists$ m^1 Є msgBag : sender(m^1) = q and seq#(m1) = next[q]) do

deliver(F, m^1)

next[q] := next[q] + 1

msgBag := msgbag – {m^1}

Causal Broadcast

- Causal broadcast is a strengthening of FIFO broadcast.
- This requires that messages be delivered according to the causal precedence relationship.
- If a message m depends on m^1 then this broadcast requires that m^1 be delivered before m.

Causal Broadcast using FIFO Broadcast

Every process p executes the following:

Initialization:

prevDlvrs := $\perp$

To execute broadcast(C,m):

broadcast(F, (prevDlvrs || m))

prevDlvrs := $\perp$

deliver(C, m) occurs as follows:

upon deliver(R,(m1,m2 …… m_n) for some n do

for i := 1….n do

If p has not previously executed deliver(C, m_i)

then deliver(C, m_i)

prevDlvrs := prevDlvrs || m_i

Timed Broadcasts

- For some algorithms, time is a critical factor. All of the Broadcasts previously mentioned can also have a timed variant.
- Guarantees that a message will be delivered within a bounded time, or not at all.
- This is called Delta-Timeliness.
  - Real Time variant: Messages must be delivered within the delta of real time.
  - Local-Time variant: Real time does not matter, as long as all processes agree.

Atomic Broadcasts

- Atomic Broadcasts guarantee a Total Ordering of all messages.
- Unlike Causal broadcasts, this requires all processes to deliver all messages in the same order.
- Note that this does not enforce a causal order.

Timed Atomic Broadcast using Timed Reliable Broadcast

Every Process p executes the following:

To execute **broadcast**(A delta, m):
    broadcast(R delta, m)

deliver(A delta, m) occurs as follows:
    upon deliver(R delta, m) do
        schedule deliver(A delta, m)
            at time ts(m)+delta

FIFO Atomic Broadcast

- Guarantees Total Ordering requirement of Atomic broadcasts.
- Also satisfies FIFO requirement.
- No algorithm given in text, but Atomic Broadcast can be converted to FIFO Atomic Broadcast by using sequence numbers. (Essentially the same logic used as to convert Reliable Broadcast to FIFO Broadcast).

Causal Atomic Broadcast

- Satisfies the Total Ordering requirement of Atomic broadcasts.
- Maintains causal ordering of messages.
- Algorithm can be built from Timed Causal Broadcast or from FIFO Atomic Broadcast

Timed Causal Atomic Broadcast using Timed Causal Broadcast

Every process p executes the following:

To execute broadcast(CA delta, m):

broadcast(C delta, m)

deliver(CA delta, m) occurs as follows:

upon deliver(C delta, m) do

schedule deliver(CA delta, m)

at time ts(m) + delta

Causal Atomic Broadcast using FIFO Atomic Broadcast

- Logic is mostly the same as for converting Causal Broadcast to Atomic Broadcast.
- The change is that a list of suspected faulty processes is kept. If messages arrive out of causal order, the sender is added to the suspects list, and all of its future messages will be discarded.

Broadcast portion of the algorithm is unchanged. New deliver is:

deliver(CA, m) occurs as follows:

upon deliver(FA,<m,D>) do

If sender(m) not in suspects and

p has previously executed

deliver(CA,m') for all m' in D

then deliver(CA, m)

prevDlvrs := prevDlvrs || m

else discard m

suspects := suspects || m

Uniform Broadcasts

- This attempts to keep all processes in sync, even at the expense of errors. (Easier to correct one coherent system than multiple machines in different states.)
- Modifies other types of broadcasts mentioned before.
- Prevents a system getting out of sync due to one faulty process delivering an extra message.
- Does not resolve problem of a faulty process failing to deliver a message.
- Note that this only attempts to address benign failures.

Broadcast summaries

copied from p. 114 of Mullender text

- Reliable Broadcast = Validity + Agreement + Integrity
- FIFO Broadcast = Reliable Broadcast + FIFO Order
- Causal Broadcast = Reliable Broadcast + Causal Order
- Atomic Broadcast = Reliable Broadcast + Total Order
- FIFO Atomic Broadcast = Reliable Broadcast + FIFO Order + Total Order
- Causal Atomic Broadcast = Reliable Broadcast + Causal Order + Total Order

All of these have Timeliness and Uniform variants.

Algorithms

- Author offers pseudo code algorithms for the various broadcast specifications.
- Algorithms are “layered”
  - Weaker broadcasts used to build stronger ones.
  - Modular approach makes for simpler algorithms.
  - Increases portability--only the lowest levels need to worry about specific features of the network.
  - May be less efficient that non-layered solutions.

Implementation of Algorithms.

- To better demonstrate these algorithms, we have converted several of them to Java.
- Code is available for download at http://www.bias2build.com/broadcast/

Amplification of Failures

- Broadcast algorithms tend to amplify the failures of a single process.
- If process crashes, system may still be left in an incorrect state (even if it ran correctly until it crashed).
- Crashes will, however, do a great deal to prevent the system from reaching a faulty state.

Bibliography

- “Distributed Systems”, 2nd edition, 1993, Mullender. (Ch 5: Fault-Tolerant Broadcasts and Related Problems, by Vassos Hadzilacos and Sam Toueg. Algorithms and diagrams were taken from here).
- “Distributed Systems - Principles and Paradigms”, 2002, Andrew S. Tanenbaum and Maartin van Steen.
- Java Networking Tutorial, http://java.sun.com/docs/books/tutorial/networking/index.html