The Totem Single-Ring Ordering and Membership Protocol

Y. Amir, L.E. Moser,
P.M. Melliar-Smith, D.A. Agarwal,
P. Ciarfella

Introduction

- Features of Totem
  - Supports high-performance fault-tolerant distributed systems
  - Provides totally ordered message delivery with low overhead, high throughput, and low latency
  - Key to its high performance is an effective flow control mechanism
  - Rapid detection of network partitioning and processor failure
  - Reconfiguration and membership services

Introduction

- Totem provides delivery in agreed order
  - Guarantees delivery in consistent total order, and when a message is delivered, all prior messages have been delivered from that configuration
- Totem provides delivery in safe order
  - Guarantees that when a processor delivers a message, it has determined that every processor in the current config has received and will deliver the message

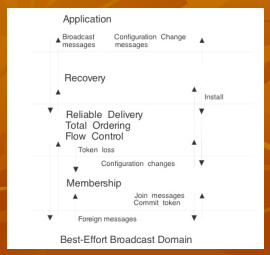
Introduction

- Delivery in consistent total order is not easy in the presence of processor failure and network partitioning
- Recovery when partitioning has occurred can be difficult if there are inconsistencies
- Totem protocol does not guarantee every message will be delivered, but that if two processes do deliver a message, it will be in the same total order

Introduction

- Configuration changes may also need to be known by the application programs
- Different processors may learn of a config change at different times, but need to form a consistent view of the change and the messages related
- Totem introduces something called extended virtual synchrony, in which processors can fail and restart with stable storage intact despite a network partitioning having occurred

Protocol Hierarchy



- Totem protocol hierarchy

Protocol Hierarchy

- Arrows on the left represent passage of messages through the hierarchy
- Arrows on the right represent Configuration Change messages and installation

Services - Membership

- Uniqueness of Configurations
  - Each config identifier is unique, processors are only members of one configuration
- Consensus
  - All processors that install a config determine that the all members of the config have a consensus on the membership
- Termination
  - If the config ceases to exist, every processor either installs a new config with a Configuration Change message or fails.
- Configuration Change Consistency

Services - Reliable Ordered Delivery

- Reliable Delivery
  - Every message has a unique identifier
  - Messages will not be duplicated and two messages from one processor will strictly be delivered one before the other
  - Messages will be delivered before a Configuration Change message to install a new config
- Delivery in Causal Order
  - Reliable Delivery
  - If a processor delivers two messages, and one message precedes the other in causal order, the processor will deliver it first

Services - Reliable Ordered Delivery

- Delivery in Agreed Order
  - Causal Order
  - When a process delivers a message, it must have delivered all preceding messages in the total order
- Delivery in Safe Order
  - Agreed Order
  - If a process in configuration C delivers a message, all members of C must have installed C, so that no members miss the message.

Totem Protocol

- Four Components
  - Total Ordering Protocol
  - Membership Protocol
  - Recovery Protocol
  - Flow Control Mechanism

Total Ordering Protocol

- A token-passing ring is imposed on a broadcast domain.
- Only the possessor of the token can broadcast messages.
- Multiple messages can be sent for each visit of the token.
- When no processor has a message to broadcast, the token continues to circulate.

Total Ordering Protocol - Token Fields

- Important fields in the token
  - *ring_id*: identifier of the ring the token is on
  - *token_seq*: sequence number to recognize duplicate tokens
  - *seq*: largest sequence number of any message broadcast, high-water mark
  - *aru*: sequence number used to determine if all processors on the ring have received ever message with a sequence number <= *aru*, low-water mark
  - *rtr*: retransmission request list

Total Ordering Protocol

- On getting the token
  - Empty the input buffer, delivering messages or retaining them until they can delivered in order
  - Broadcast any requested retransmissions or new messages, update the token
  - For each new message broadcast, increment the *seq* field of the token, and set the sequence number of the message to that value

Total Ordering Protocol

- On getting the token
  - Comparison between *aru* on the token and the processor's *my_aru*. If *my_aru* is smaller, replace *aru* with *my_aru* and set *aru_id* on the token with its identifier
  - If *aru_id* is the processor's identifier, set *aru* to *my_aru* (No other processor changed the *aru* in the rotation)
  - If *seq* and *aru* are equal, increment *aru* and *my_aru* in step with *seq*, and set *aru_id* to null

Total Ordering Protocol

- On getting the token
  - These rules are setup so that this final case works
    - If the *seq* field of the token is greater than its *my_aru*, the processor has not received all of the messages that have been broadcast on the ring, so it augments the *rtr* field of the token with the missed messages.
    - If the processor with the token has received any of the messages in the *rtr*, it retransmits them before broadcasting any new messages.
    - A retransmitted message is removed from the *rtr*

Membership Protocol

- Objective is to ensure consensus, so that every member of a configuration agrees on the membership of the configuration
- Detects such failures as processor failure, network partitioning, and loss of the token.
- Generates a new token and can recover messages that were not received by all processes prior to a failure

Membership Protocol

- Data Structures
  - Join Message
  - Configuration Change Message
  - Commit Token
  - Local variables at each processor

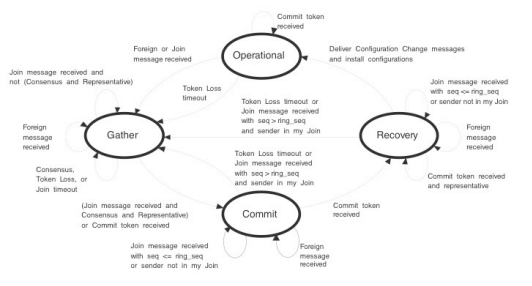
Membership Protocol

- Seven Events in the Protocol
  - Receiving a Foreign Message
    - A message broadcast by a processor not in the ring, and activates the membership protocol
  - Receiving a Join Message
    - Informs the receiver of the sender's proposed membership
  - Receiving a Commit Token
    - On first receipt, the member updates the Commit token
    - On second receipt, obtains updated info that the other members have supplied
  - Token Loss Timeout
    - Indicates that a processor did not receive the token or a message within the required time, which activates the membership protocol

Membership Protocol

- Seven Events in the Protocol
  - Token Retransmission Timeout
    - Indicates that a processor should retransmit the token because it has not received the token or a message broadcast from another processor
  - Join Timeout
    - Used to determine the interval after which a Join Message is rebroadcast
  - Consensus Timeout
    - Indicates that a processor failed to reach consensus on the new ring in the require amount of time
  - Recognizing Failure to Receive
    - If the *aru* field has not advanced in several rotations of the token, the processor that set the *aru* has repeatedly failed to receive a message

Finite State Machine for Membership



Four States of the Membership

- Operational State
  - Messages are broadcast in agreed or safe order
  - If a Token Loss timeout occurs, or a Join message or foreign message is received, we move to Gather state
  - A message is buffered until it has been acknowledged as received by the other processors

Four States of Membership

- Gather State
  - A processor collects information about operational and failed processors, and broadcasts that info in Join messages
  - Upon receipt of a Join message, a processor updates two local variables, *my_proc_set* and *my_fail_set*.
  - If they changed, the previous membership is abandoned and the new info is broadcast in Join message
  - Consensus is reached when a processor receives Join messages with *proc_set* and *fail_set* equal to its own from every processor in the difference set, *my_proc_set - my_fail_set*

Four States of Membership

- Commit State
  - First Rotation of the Commit token
    - Confirms all members in the token's *memb_list* are committed to the membership.
    - Collects info to correctly handle messages from the old ring that need retransmission
  - Second Rotation of the Commit token
    - Disseminate all the info collected in the first rotation
    - On receipt of the Commit token a second time, a processor enters the Recovery state

Four States of Membership

- Recovery State
  - Commit token gets converted into the regular token for the new ring, replacing the *memb_list* and *memb_index* with the *rtr* field
  - New ring is formed, but not yet installed
  - Processors use the new ring to deliver messages from their old rings to meet any agreed or safe delivery guarantees
  - In an atomic action, each proc. delivers the exchanged messages to the application along with Config Change Messages, installs the new ring, and shifts to Operational state.

Recovery Protocol

- Six Steps
  - Exchange messages with other processors from the old ring to ensure they have the same set of messages to be rebroadcast on the new ring
  - Deliver to the application messages from the old ring needed for agreed or safe delivery guarantees
  - Deliver the first Configuration Change message
  - Deliver to the application any other messages that can now be delivered in agreed of safe delivery in the transitional ring
  - Deliver second Configuration Change message
  - Shift to Operational state

Recovery Protocol

- Exchange Messages from the Old Ring
  - Determine the lowest *my_aru* of any processor from the old ring, broadcast all messages for the old ring with a sequence number greater than the lowest *my_aru* on the new ring
- Delivery of Messages on the Old Ring
  - Sorts the messages for the old ring and delivers messages in order until it encounters a gap in the sorted sequence or a message requiring safe delivery with a sequence number higher than *high_ring_delivered* (highest sequence number of a message delivered safe on the old ring)

Recovery Protocol

- Delivery of Messages in the Transitional Configuration
  - Deliver all remaining messages from the old ring by processors in *my_deliver_memb* (Set of identifiers of processors whose messages must be delivered in the transitional config)
- Failure of Recovery
  - Some may have installed the new ring while others have not. If a processor must deliver a message in safe order in a transitional config, all other processors must deliver it before installing the new ring. If some processor fails before the new ring is installed, it must be delivered during the transitional config of the next transitional config to be created

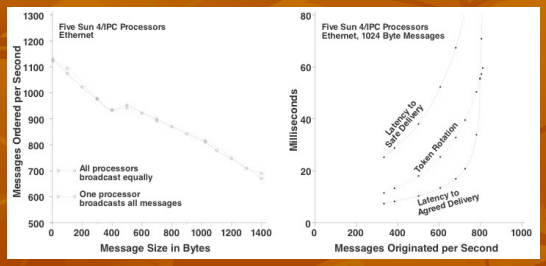
Flow Control Mechanism

- Simple mechanism to control the maximum number of messages broadcast during one token rotation
- Before processing the token and broadcasting messages, a processor must empty its input buffer
- Thus, the rate of broadcast will be limited to the processing rate of the slowest processor

Flow Control Mechanism

- The number of messages broadcast must not exceed the maximum number allowed to be broadcast during one visit of the token
- The number of messages broadcast must not exceed the window size minus the number of messages broadcast in the previous rotation of the token
- Must not exceed its fair share of the window size, based on the ratio of its backlog to the sum to the backlogs of all the processors as they released the token during the previous rotation.
  - *My_trc <= window_size x my_tbl(backlog + my_tbl - my_pbl)*

Performance Graph



Conclusion

- Provides fast reliable ordered delivery in a broadcast domain where processors may fail and the network may partition
- Membership protocol handles processor failure and recovery, as well as network partitioning and remerging
- Flow control mechanism avoids message loss due to buffer overflow and provides significantly higher throughput than prior total ordering protocols
- Current development is attempting to extend Totem to work on multiple rings interconnected by gateways, and to examine whether its exceptional performance extends to these multiple ring situations