AMQP breakdown for clustering

AmqpBreakdown

Breakdown of AMQP 0-10 commands and controls for clustering.

Definitions

Replica: broker member of the cluster.

Connected replica: replica directly connected to the client.

Connected session: Attahced session on the connected replica.

Shadow session: Backup of active session state on non-connected replica.

Detached session: Session not attached to any replica, awaiting timeout.

Replicate: Multicast to cluster, defer completing some action till cluster responds.

Inferred: Change on the connected replica that can be inferred by other replicas because it is a deterministic response to some replicated change (e.g. responding to a query command)

Types of state.

- shared state
  - wiring.
  - queue contents.
- session state.
  - command ids.
  - subscriptions.

Cluster qualities of service:

- shared state only: replicate shared state but does not support failover.
- failover: Replicate shared state and sessions state, supports failover.

We examine the impact on cluster state of each AMQP control & command, and the implications for what needs replication.

Connection controls

None of the connection controls MUST be replicated, they do not affect state.

Connection creation/destruction MAY be replicated if using shadow connections to organize sessions.

Session controls

Performance note: Session controls other than completed are not sent on a per-message basis so are not critical path.

Failover requires that all replicas know:

- state of command IDs to correlate completions.
- outgoing commands in doubt for replay.

This means:

- all outgoing commands must be inferred or replicated
- all incoming commands are replicated OR additional command-id info is replicated.

Recieve

attach,detach: MUST replicate. Replicas must know all sessions & attachment.

request-timeout: MUST replicate - replicas should respect timeout. See other events below.

command-point, gap, expected: MUST replicate for consistent command numbering.

confirmed: Can be ignored as per spec.

flush: No need to replicate, only attached replica need respond.

known-completed: MUST replicate so replicas can avoid wrap-around. on their unknown-complete set.

completed: Replicas need client completions to bound their replay list. They do not need immediate replication since completions will be re-sent as part of failover. They MUST known of completions before sending a known-complete to client since the client will no longer notify completion of known-complete commands.

So we can do one of

- repliacate all completions (performance risk)
- replicate when sending known-complete (risk memory growth in replay lists)
- combination: replicate sending known-completed and replicate completions when replay list is large.

Send

attached, detached, timeout, command-point, expected,confirmed, flush: No effect on replicated state.

known-completed: See receive completed above.

gap: Not used.

completed: Only need replication for *persistent commands*, i.e. commands that change persistent shared state.

Persistent commands and the async store.

Persistent commands guarantee that once completed their effects are stored persistently and will survive total broker shutdown.

For the strongest guarantee it must be persisted on all persistent replicas in the cluster:

- Connected broker receives command, replicates and initiate async store.
- All persistent replicas initiate async store.
- On async completion, replicas mcast an async store confirmation.
- The connected broker waits for all store confirmations before sending completion.

A more performant implementation with a weaker guarantee would send completed when the local async store completes with no async notifications from the cluster. The risk: client receives completed, local disk is destroyed, rest of cluster shuts down before storing, message is lost. Need to determine if that's an acceptable risk in general, or perhaps offer configurable choice.

Detached session timeouts.

Having each replica indepenedently destroy timed-out sessions creates a race: a client could resume a session on ne replica concurrently with the timeout expiring and session state being destroyed on other replicas.

To avoid this we choose an arbitrary cluster member to mcast "session timeout" events when sessions time out. This woudl be the primary in active /passive mode or an arbitrarily chosen member (e.g. the oldest member) in active-active mode.

Execution commands

Receive

sync: No need to replicate, no effect on replicated state. **exception**: MUST replicate, causes destruction of session state. **transfer**: MUST replicate before sending completed.

Send

sync,result, exception: inferred.

Message commands

Receive

Definition: an incoming message transfer is *finished* when:

- accept=none: completed sent for incoming transfer.
- accept=explicit: received accept for the message and sent completed for the accept.

Before a message is finished it can be re-queued in the event of a client disconnect.

transfer: MUST replicate, update queue content.

acquire, release, accept, reject: See dequeue management below.

resume: Not implemented.

subscribe, cancel, set-flow-mode, flow, flush, stop: MUST replicate subscription state for replay.

Send

transfer: MUST replicate for shared state & replay. Need not replicate content if it can be inferred. See "Persistent commands and the async store" above and "Dequeue management" note below.

acquire, release, accept, reject: See dequeue management below.

stop:

Dequeue management.

Enqueue is straightforward: incoming transfers are replicated.

Dequeue provides more options:

- active/passive mode: active replica is "owner" and controls order of enqueue dequeue. Information about dequeues can be delayed/compressed /batched.
- active/active mode, no queue owners: all dequeue information must be replicated.
- active/active mode with queue owners: queues have an owner like active/passive but ownership can be transferred.

Active-active no owners: Replicate all incoming message commands. Replicate dequeue descisions that cannot be inferred. In our current IO-driven model this means all dequeues.

Active/passive: Active broker does dequeues. Can avoid/defer replication till of incoming acquire, release, accept, reject and outgoing transfer till messages are *finished* - i.e. till sending completion for accept or for transfer in implicit-accept mode. At this point it may be possible to batch the events.

Note that events must still be sent even if batched: all replicas need to know which messages were removed from which queues, and the order to replay them in the event of fail-over.

Tx commands

select, commit, rollback: MUST replicate. All replicas must commit/rollback consistently.

Dtx commands

select,start,end,commit,forget,get-timeout,prepare,recover,rollback,set-timeout: MUST replicate so all replicas know which commands are within transactions.

All replicas must join the DTX so all will commit/fail under control of DTX manager.

Exchange commands

declare,delete,bind,unbind: MUST replicate, update wiring.

bound,query: MUST replicate replicas can respond in the event of failover.

Queue commands

declare,delete,purge: MUST replicate, update wiring/queue content. query: MUST replicate replicas can respond in the event of failover.

File & Stream commands not implemented.