Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial

Outline

- Introduction
- Definitions
- Fault Tolerance
    - State Machines
    - Output Devices
    - Clients
    - Reconfiguration
- Questions

Introduction

- A *State Machine* is a representation of a process that consists of a set of *states* and a set of *commands* for transitioning between them.
    - A *state* consists of assignments of values to all of the process's *state variables*.
    - A *command* is an atomic action that modifies the values of one or more *state variables*.
    - A *client* issues requests to a *state machine*, which could cause *commands* to be executed, and could cause output to a *client* or *output device*.

Introduction

- Clients of a process can make the following causality assumptions about the process.
  - O1: Requests issued by a single client to a given state machine sm are processed in the order they are issued.
  - O2: If the fact that request r was made to a state machine sm by client c could have caused a request r ’ to be made by a client c’ to sm, then sm processes r before r’.

Definitions – Failure Types

- Byzantine Failures
    - A *Byzantine failure* is a failure in which the failing component can perform arbitrary behavior.
- Fail-stop Failures
    - A *Fail-stop failure* is a failure in which the failing component transitions to a state which enables other components to become aware of the failure, then halts.

Definitions – *t* Fault Tolerance

- A system is said to be *t Fault Tolerant* for some integer *t* if, given that *t* or fewer components fail, the system will continue to operate correctly according to its specification.

Fault Tolerant State Machines

- A State Machine can be *replicated* and run concurrently on multiple processors to become fault tolerant.
    - If only fail-stop failures are possible, *t+1* replicas are sufficient for a *t* fault tolerant system.
    - If Byzantine failures are possible, *2t+1* replicas are necessary for a *t* fault tolerant system.

Fault Tolerant State Machines

- For replication to work, all the replicated state machines must receive and process the same sequence of requests. This requires *agreement* and *order*:
  - The a*greement* requirement is achieved when every non-faulty state machine replica receives every request.
  - The *order* requirement is achieved when every non-faulty state machine replica processes the requests it receives in the same relative order.

Agreement and Order

- Agreement
  - Satisfied by having a designated *transmitter* transmit values such that
    - IC1: All non-faulty processors agree on the same value.
    - IC2: If the *transmitter* is non-faulty, then all non-faulty processors agree on its value.
- Order
  - Satisfied by having each replica always process the next *stable* request with the lowest unique ID.
    - A request is *stable* if it is no longer possible to receive a request from the same, non-faulty client with a lower unique ID.

Logical Clock

- Keep a counter at each process.
- Increment the counter at each event.
- If a message is sent, include the current value of the counter in the message.
- If a message is recieved, update the counter to MAX(recv, counter)+1

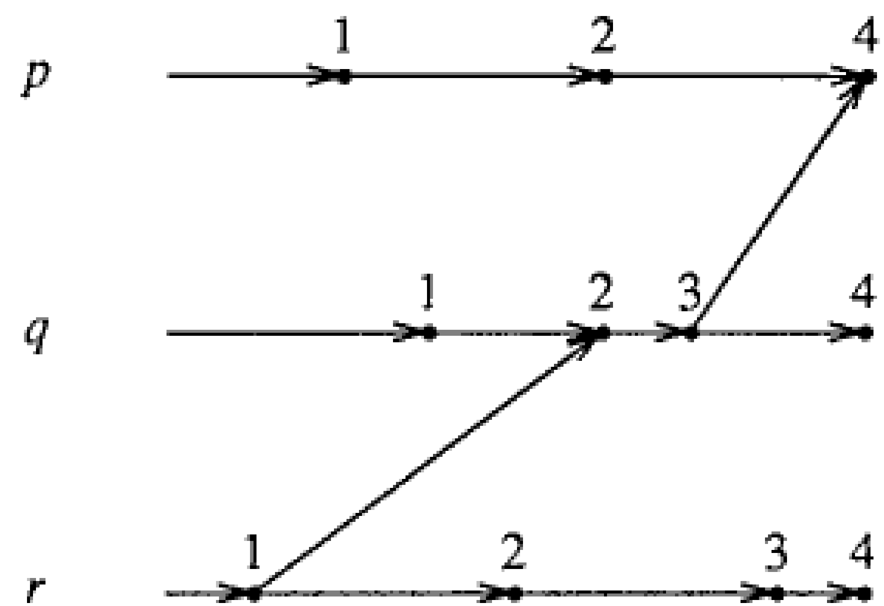

Figure 4. Logical clock example.

Synchronized Real-Time Clock

- Assume that all the processors have access to a local clock that differs from a remote clock by at most d seconds, for some value d.
- The key property of this system is that if a request has timestamp t, then the request is guaranteed to be stable when the local clock has time $t+d$.

Replica-Generated ID

- For each request, every state machine replica generates a unique *candidate uid* (*cuid*) for that request. Then using an agreement protocol, the non-faulty replicas agree on a single *cuid* to become the actual *uid* of the request.

Fault-Tolerant Output

- External Output
  - Add fault-tolerance in the same way as with state machines – with replication!
  - Assume the external consumer of the output can deal with replicated outputs.
- Internal Output
  - Client acts a voter, waits for enough identical responses before using an output.

Fault-Tolerant Clients

- Client Replication
  - When in doubt, run replicas in parallel.
  - Add code to state machines to handle replicated clients.
  - Not possible for all clients.
- Defensive Programming
  - Restrict the effects of a faulty client on the state machine replicas.

Reconfiguration

- Recover from failures by *disabling* faulty processors when detected, and reinstalling them once they have been *repaired*.
- Have a *configurator* that satisfies the following two properties run alongside each element of the system.
  - C1: Only a faulty element is removed from the configuration.
  - C2: Only a non-faulty element is added to the configuration.

Questions?