Today's Topics - Chapter 15

- Replication 15.1
- Group Communications 15.2
- Fault Services 15.3

Replication

Replication can provide the following:

- performance enhancement
  - e.g. several web servers can have the same DNS name and the servers are selected in turn. To share the load.
- Replication of read-only data is simple, but replication of changing data has overheads

Replication Increases Reliability

- Suppose you have a unit with a failure probability of p.
- If you have a system with n units and $n - m$ units are required so that it functions properly. Then what is the new probability of failure?

Example - Triple Module Redundancy

- Assume that a unit either works or stops working with probability p.
- Have three copies and a voting circuit, pick the most popular result.
- Probability of failure is now $3p^2(1-p) + p^3 = 3p^2 - 2p^3$

Byzantine Faults

Two types of failures:

- Failure by omission: The system stops working, it fails to provide some service. You know that the system does not work because it isn't responding.
- Byzantine Failure. The system starts producing incorrect output. It is not always easy to distinguish between the system failing and it correctly running.

Fault-tolerant service

- The goal guarantees correcth behaviour in spite of certain faults (can include timeliness)
  - if f of f+1 servers crash then 1 remains to supply the service
  - if f of 2f+1 servers have Byzantine faults then they can supply a correct service
- availability is hindered by server failures replicate data at failure-independent servers and when one fails, client may use another.

Requirements for Replicated Data

- Replication transparency clients see logical objects (not several physical copies) they access one logical item and receive a single result
- Consistency specific to the application, e.g. when a user of a diary disconnects, their local copy may be inconsistent with the others and will need to be reconciled when they connect again. But connected clients using different copies should get consistent results.

System Model

- each logical object is implemented by a collection of physical copies called replicas the replicas are not necessarily consistent all the time (some may have received updates, not yet conveyed to the others)
- we assume an asynchronous system where processes fail only by crashing

Replica Managers

- an RM contains replicas on a computer and access them directly
- RMs apply operations to replicas recoverably i.e. they do not leave inconsistent results if they crash
- objects are copied at all RMs unless we state otherwise
- static systems are based on a fixed set of RMs in a dynamic system: RMs may join or leave (e.g. when they crash)

State Machine Approach

A RM can be a state machine with the following properties:

- applies operations atomically
- its state is a deterministic function of its initial state and the operations applied
- all replicas start identical and carry out the same operations
- Its operations must not be affected by clock readings etc.

Basic Architectural Model

- Clients see a service that gives them access to logical objects, which are in fact replicated at the RMs
- Clients request operations: those without updates are called read-only requests the others are called update requests
- Clients request are handled by front ends. A front end makes replication transparent.
- What can a front end hide from a client?

Insert Figure 15.1 here.

Five Phases in performing a request

- Issue Request: The Front End either:
  - sends the request to a single RM which passes it on to all the others.
  - Multicasts the message to all RM (in the state machine approach)
- Coordination: The RM apply the request; and decide on its ordering relative to other request decide whether to apply the request. (according to FIFO, causal or total ordering)
- Execution: The RMs execute the request (often tentatively)
- The RMs *agree* on the effect of the request.

Five Phases Continued

- Response: One or more RMs reply to the FE for:
  - for high *high availability* the fastest response is delivered.
  - to tolerate Byzantine faults, take a vote.

Ordering

- Fifo Ordering: If a front end issues request r and then request r' then any correct RM that handles r' handles r before it.
- Causal Ordering: If $r \rightarrow r'$ then any correct replica manager handles r before r'.
- Total Ordering: If a correct RM handles r before r' then any correct replica manager does the same.

Total Order is too strong. Causal Ordering is desirable, FIFO ordering often implemented. Later on we will look at the differences in detail.

Group Communication

- The basic idea is that we have a group of processes which participate in the replica.
- If the processes are fixed and no process fails then there is no problem.
- But if we have a number of processes that can join/leave or fail we have to keep track of who belongs to the group.
- The problem is made more complicated, because the might be messages in transit while processes join or leave.

Role of a group membership service

- Provide an interface for group membership changes.
- Implementation of a failure detector.
- Notifying members of group membership changes: The services notifies the group's members when a process is added, or when a process is excluded.
- Performing group address expansion.

View Delivery

- One way of managing all this is with the idea of a view.
- A view is a the set of processes that belong to the group. The group manager delivers a series of views to each process.
- We require some consistency requirements with views and messages. Messages are associated with views.

View Synchronous group Communication

- *Agreement:* In any given view, processes deliver the same set of messages.
- *Integrity* If a process delivers a message, then it it will not deliver that message again.
- *Validity* Correct process always deliver the messages that they send. If the the system fails to deliver a message to any process q, then in the next view q will not be there.

It is essentially a consistency requirement that messages delivered from certain views arrive all before or all after a view change.

Insert figure 15.3.

Fault-tolerant Services

- If data is distributed and faults can occur some care has to be taken so that things don't get inconsistent.
- A system is correct if a user can see no difference between one copy and multiple copies.

Bank Account Example

- Consider a naive replication system, in which two RMs at computers A and B each maintain replicas of two bank accounts x and y.
- Clients read and update the accounts at their local RM and the other one in case of failure.
- Replica managers propagate updates to one another in the background after responding to each client.

Bank Account Example

- Client 1 updates the balance of x at its local replica manager B to be 1 Euro and then updates to update y's balance to be 2 but discovers that B has failed, so Client 1 updates it A instead.
- But Client 2 reads the balance of y to be 2 at A but since B crashed the setting the balance of x did not get through.

Bank Account Example

Client 1:	Client 2:
$setBalance_B(x, 1)$	
$setBalance_A(y, 1)$	
	$getBalance_A(y) \rightarrow 2$
	$getBalance_A(x) \rightarrow 0$

Consistency

Basic idea.

- We would like some sort of temporal consistency, if s happens before t then on all copies s happens before t. But in the presence of network delays this is not possible.
- So various weaker notions of consistency are introduced.
- One common criterion is sequential consistency. A sequence of operations all allowed there is an interleaving of the individual sequences that produces that interleaving.

Sequential Interleaving

- Consider two sequences of operations s_1, s_2, s_3, t_1, t_2. Then some of the possible interleaving would be:
  - $s_1 t_1 s_2 s_3$
  - $t_1 s_1 t_2 s_2 s_3$
  - $t_1 t_2 s_1 s_2 s_3$

How you make a system do this is a hard question. We will look at this in more detail when we look at transactions.

Active vs Passive Replication

- Passive Single process acts as a primary which propagates data to the backups.
- Active Front End sends the same message to every process in the group.