Distribute System Summative Answers

This part relates to background concepts about active replication.
a) Describe the generic workflow of a passive replication based distributed system. [5 marks]

In *passive* replication for fault tolerance, there is at any one time a single server and one or more secondary backup servers. In the pure form of the model, front ends communicate only with the primary replica server to obtain the service. The primary server executes the operations and sends copies of the updated data to the backups. If the primary fails, one of the backups is promoted to act as the server.

The sequence of events when a client requests an operation to be performed is as follows:

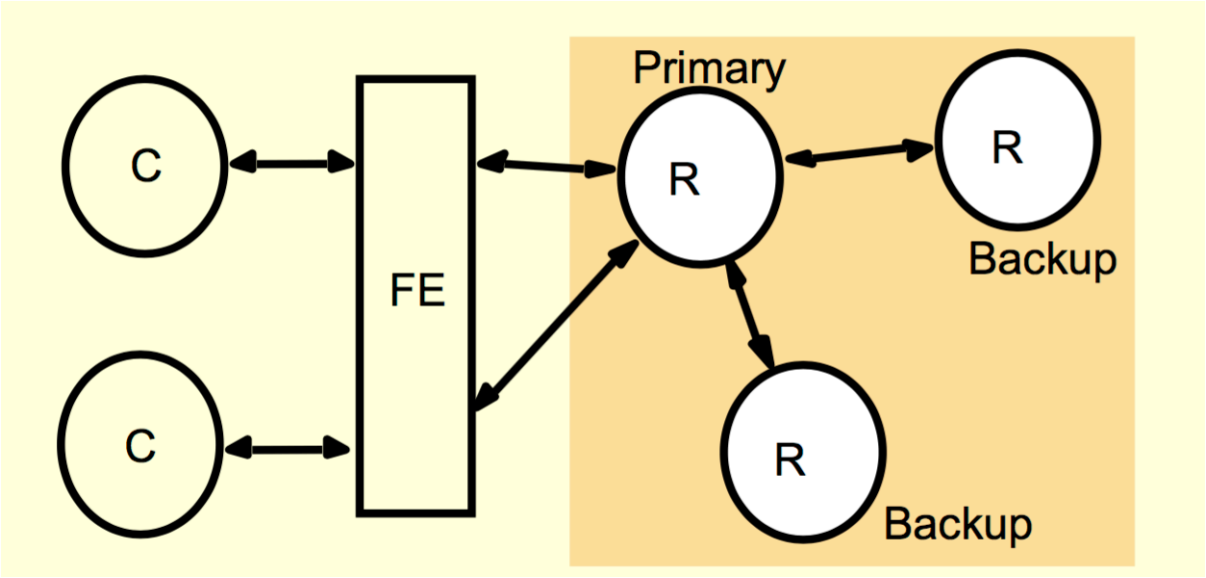
1. *Request*: The front end issues the request, containing a unique identifier, to the primary server.

2. *Coordination*: The primary takes each request in the order in which it receives it. It checks the unique identifier, in case it has already executed the request, and if so it simply resends the response.

3. *Execution*: The primary executes the request and stores the response.

4. *Agreement*: If the request is an update, then the primary sends the updated state, the response and the unique identifier to all the backups. The backups send an acknowledgement.

5. *Response*: The primary responds to the front end, which hands the response back to the client.



b) Explain how passive replication ensures information availability. [5 marks]

This system obviously guarantees information availability if the primary server is correct, since the primary sequences all the operations upon the shared objects.
If the primary fails, then the system ensures information availability if a single backup becomes the new primary and if the new system configuration takes over exactly where the last left off. That is if:

. The primary is replaced by a unique backup.

. The replica managers that survive agree on which operations had been performed at the point when the replacement primary takes over.

Both of these requirements are met if the primary and backups managers are organized as a group and if the primary uses view-synchronous group communication to send the updates to the backups.

When the primary crashes, the communication system delivers a new view to the backups, one that excludes the old primary. The backup that replaces the primary can be chosen in any way.

For example, the backups can choose the first member in that view as the replacement. That backup can register itself as the primary with a name that the clients consult when they suspect that the primary has failed.

By the ordering property of view-synchrony and the use of stored identifiers to detect repeated requests. The view-synchronous semantics guarantee that either all the backups or none of them will deliver any given update before delivering the new view. Thus the new primary and the surviving backups all agree on whether any particular client's update has or has not been processed. Consider a front end that has not received a response. The front end retransmits the request to whichever backup takes over as the primary. The primary may have crashed at any point during the operation. If it crashed before the agreement stage (4), then the surviving replica managers cannot have processed the request. If it crashed during the agreement stage, then they may have processed the request. If it crashed after that stage, then they have definitely processed it. But the new primary does not have to know what stage the old primary was in when it crashed. When it receives a request, it proceeds from stage 2 above. By view-synchrony, no consultation with the backups is necessary, because they have all processed the same set of messages.

a) Suppose the primary server may fail sometimes. Explain what procedures should be introduced to the distributed system in order to resolve this failure situation. In addition, if the failed primary server recovers, describe what should be done to allow this server to run as part of the distributed system again.

The client should have no idea that there is more than one server, this is the goal of replication transparency. The connection to the servers should be handled by the front end.

The front-end should hold a list of all servers, i.e. the main server and then the 2 backup servers. If the primary server is available, it should be at the top of the list and the front end should connect to this server. The front end requires little functionality to achieve fault tolerance. It just needs to be able to look up the new primary when the current primary does not respond.

If the front-end can't connect to a server, we assume the server has failed and is down.

As a result, front end should try and connect to primary server. If this fails, tries to connect to one of the backup servers. When the primary crashes, the communication system eventually delivers a new view to the surviving backups, one that excludes the old primary.

Consider a front end that has not received a response. The front end retransmits the request to whichever backup takes over as the primary. The primary may have crashed at any point during the operation. If it crashed before the agreement stage (4), then the surviving replica managers cannot have processed the request. If it crashed during the agreement stage, then they may have processed the request. If it crashed after that stage, then they have definitely processed it. But the new primary does not have to know what stage the old primary was in when it crashed. When it receives a request, it proceeds from stage 2 above. By view-synchrony, no consultation with the backups is necessary, because they have all processed the same set of messages.

Fault Tolerance Services

1. Detect error and diagnose what has failed, i.e. which server.
2. Distributed system must isolate the failure to the offending component.
3. System must reconfigure, i.e. promote a backup server to primary server.
4. If primary server recovers, wait until server has replied to a request to client and then swap recovered server back into primary server slot.

If a front end is unable to locate the primary server, should throw an exception but should then check out available servers from a directory service/list. I.e. the front end should have a list of servers to connect to with the primary server at the top of the list.

If a front ends request to primary server is lost, wait for an arbitrary amount of time,

i.e. apply timeout to await server reply. Then re-send. If multiple requests appear to get lost assume that the ‘cannot locate server error’.

If primary server crashes after receiving request from the client, front end should store this client request. Try a different server and continue as normal.

If server reply to client is lost, apply timeout to await server reply. Then front end should get another server to send the reply.

If the failed primary server recovers, it is placed to the bottom of the server list, i.e. acts as a backup as another server is promoted to be the primary server.

If a failed server recovers, it needs to get the current data of the server components in the distributed system, i.e. it needs to get up to speed. It does this by searching through the list of other servers and checking each one to see if it is working. When a working server is found, the new recovered server updates its data to be consistent. Due to the consistency of the distributed system, it doesn’t matter which server it gets the data from.

b)

Client

The client program should not change at all. This is due to replication transparency: irrespective of what server is being used, the client is just making requests via the front end.

Front End

The front-end portion of the program is modified so that when there is a request for some service by a client/user, the front-end tries to connect to the primary server. If this throws an error, then goes through the list of backup servers and tries to connect to one of them.

Through a method connectNewServer() which gets the server list, unbinds server 1 i.e. the primary server from the registry. Then goes through the list of servers, i.e. all the backup servers. When it finds a working server, it changes its stub name to be server 1 and then binds it to the registry. If no servers found, exception thrown.

It doesn’t matter which of the backup servers is chosen to be promoted as they are identical due to consistency of the distributed system.

Server

The server has been modified that whenever a server starts it checks for other existing serves to sync with. This goes through all the servers in the RMI registry, looking at all the servers in the registry to see if they are working. The gets the order history of one of these servers and sets its order history to be the same, ensuring consistency.

References

Lectures Slides

Distributed Systems Concepts and Design Coulouris, Dollimore, Kindberg and Blair.