Unai Gil, Iñigo Rojo & Jon Ander Ruiz (GRR)
DS_2020-2021_GRR_A3.3a
02/12/2020 - Distributed Systems, UPV/EHU

As we have seen, a fault-tolerant service based on passive replication requires a mechanism to deal with the possible failure of the primary server. When this occurs, the secondary must (1) detect the fault and (2) elect a new primary.

In general, agreeing on the election of a particular process or node (which we will call a coordinator or leader) among a group of processes is a very common problem in distributed systems. In this activity we will analyze different protocols that allow the election of a unique leader. Amon them, we can choose the most suitable for our cloud file service.

What is there to be done

1. **The teacher will propose the approaches to detect the failure of the coordinator and to solve the election of a new leader, described in Section 3.3 of the Notes.**

2. **Compare the coordinator election approaches from the point of view of response time and scalability. For this second aspect, consider the number of messages exchanged as a parameter of the complexity of the protocol.**
   1. **Bully Algorithm**
      a) Response time:

         Due to being hierarchical in nature, every process will have to send the election message to all of its superiors, and these superiors will start an election process sending the message to its superiors. If initiated by the lowest ranking one it will have to be sent N! Times, multiplied by two(response from the superior). This will cost (n^2 - 1) *△ , where△ is the maximum response time of the system and where n will be the number of the initiating process in the hierarchy(1 being the highest identity process). It is linear though it can grow very big if there is docens of processes. If we optimize the algorithm we can have an upper process that received the message that can send the election message confirmation to lower process ones to stop the messages of lower processes from being sent multiple times.

      b) Scalability:

         About the number of messages exchanged, we have seen that it is linear in its complexity. Despite that the number of messages can grow a lot if we don't implement the aforementioned optimization. This will cost (n^2 - 1) *△ messages in a worst case scenario, where n will be the number of the initiating process in the hierarchy(1 being the highest identity process).

2. **Ring Algorithm**
    a) Response Time: By definition, in the ring algorithm, all the processes are put in a circular form. The process P_i that wants to be "promoted" to coordinator status has to pass the election message to the process $P_{(i+1) \text{ mod } N}$. If P_{i+1} does not reply, then sends the message to P_{i+2}, and so until it finds a process that acknowledges the reception of the message. When a new process receives an election message, it forwards it to its successor including in the message its identifier. So, after a complete round, all the processes are informed that the new coordinator is the process with the highest identifier included in the message. So, in any case, the message has to make an entire circle, passing by all the n nodes at least once. So its response time would be $3 * n * \triangle - 1$ where $\triangle$ is the maximum response time of the system, n is the number of nodes in the circle and its multiplied by 3 because each process has to sent 3 messages, the response, the coordinator message and the notice message.

    b) Scalability:

    About the number of messages exchanged, in the worst case scenario we would be sending n * 3 - 1 messages (election message, response from the next process and the notice message). This means that our growth is polynomial which is good from an scalability point of view.

3. **Taking into account that our cloud file service will be modest in terms of number of servers (consider a primary and two secondary servers), implement a concrete solution for failure detection and the election of a new primary. The final implementation is part of the Project, but for this activity prepare a brief report explaining your design and presenting the pseudo-code.**

1) Failure detection:

First prototype: Heartbeat

Be the processes P1, P2, P3, …, Pn and Pp where Pp is the primary process and P1, …, Pn are the secondary processes grouped in a group called G.

```
Init
    While true
        For each process in G:
            heartbeat(p, m)
        End for each
        If allProcessesUp() == true
            //Case in which all processes are running.
            continue()
        Else
            //Case in which there is an error, either on the primary
        or on a secondary.
            If checkPrimary() == true
                //Case primary running. Failure on secondary.

            Else
                //Case primary down. Elect new primary.
                electNewPrimary(G)
            End If Else
        End If Else
        wait(t)
    End While
End
```

heartbeat(p, m, s) ⇒ Function where a process sends a message m every s time to the process p. This function also sends the processes id.

allProcessesUp() ⇒ Function where the primary checks if it received a heartbeat from every process in G. True if all processes are up, false otherwise.

checkPrimary() ⇒ Function where a process checks if the primary process is running. True if the primary is running, false otherwise.

electNewPrimary(G) ⇒ Function where the group of processes G, elects a new primary process.

wait(t) ⇒ Function that pauses the system t milliseconds.

2) **Election of a new primary:**

Where L is the list of process identifiers of others and id is our own

```
electNewPrimary(G)
Init
        For each identifier in L
                If L > id
                        m= R-broadcast(ElectionMessage)
                End if
        End for
        If m == empty
                R-Broadcast(ElectedMessage)
        End if
End
```

Upon receiving ElectionMessage the other processes will execute this function. If m is empty, meaning that there was no response to a message from a superior(or there is no superior) they will send ElectedMessage letting the rest of the processes know which process is acting as primary. This will be implemented as part of the message handling of the server by introducing 2 new orders (EED,EON short for elected and election).