Open Questions: Groupy

pongo en rojo cosas a comentar, corregir y comentar de todo esto

GMS1 - incapacidad para manejar el fallo (crash) del líder

a) What happens if the leader worker crashes?

En la implementación GMS1, no existe ningún mecanismo de detección de fallos ni de elección de líder. Los esclavos simplemente envían mensajes {mcast, Msg} y {join, Peer} al líder que conocen inicialmente y esperan recibir {msg, Msg} o {view, …} de él.

Si el líder falla (crash):

- Los esclavos no detectan el fallo, y continuarán enviando mensajes {mcast, ...} y {join, ...} al PID del líder caído. Estos mensajes se perderán.
- No se procesarán nuevas solicitudes mcast ni join, ya que solo el líder puede hacerlo.
- No se distribuirán nuevos mensajes {msg, ...} ni nuevas vistas {view, ...} a los esclavos.
- El sistema se **paraliza funcionalmente** para cualquier operación que requiera coordinación central (multicast, gestión de miembros), ya que al no haber coordinación, los estados de los trabajadores divergen. Los esclavos existentes no detectarán el fallo y no se elegirá un nuevo líder.

Mirar Apéndice A) (abajo)

GMS2 - Detección de fallos de líder y un mecanismo de elección -> falladas de procesos (se desincronizan)

b) What happens now if the leader worker crashes?

Ahora en esta implementación gms2, se introduce la detección de fallos del líder y un mecanismo de elección. Cada esclavo monitoriza al líder, que genera un mensaje 'DOWN' cuando este crashea.

Si el líder falla (crash):

1. Detección de fallo:
    a. Los esclavos que lo estaban monitorizando recibirán un mensaje {'DOWN', Ref, process, Leader, Reason}.
    b. Al recibir este mensaje, cada esclavo transita a la función election(Name, Master, Slaves).
2. Proceso de elección automática
    a. Dentro de election/3, cada esclavo examina su lista local Slaves. El primer proceso en esta lista es designado como el nuevo líder potencial.
    b. Si un proceso descubre que él mismo es el primero en la lista ([Self|Rest]), asume el rol de líder, posiblemente retransmite el último mensaje conocido (en gms3, no en gms2), y finalmente envía una nueva vista ({view, Self, Rest}) al resto (Rest). Luego continúa como leader/3.
    c. Si un proceso descubre que otro nodo (NewLeader) es el primero en la lista ([NewLeader|Rest]), permanece como esclavo, empieza a monitorizar al NewLeader (erlang:monitor(process, NewLeader)) y continúa como slave/5 con el nuevo líder y la nueva lista de esclavos (Rest).

En resumen, el sistema es capaz de recuperarse del fallo del líder eligiendo al siguiente nodo en la lista como nuevo líder y estableciendo una nueva vista.

c) How can the membership service tolerate the crash of a slave worker if we are not monitoring slaves?

¿Cómo puede el servicio de afiliación tolerar la caída de un trabajador esclavo si no estamos supervisando a los esclavos?

En GMS2, explícitamente sólo se monitoriza al líder; los *slaves* **no se monitorizan entre sí ni son monitorizados por el líder**.
Sin embargo, el sistema tiene cierta tolerancia implicita al fallo de *slaves*:

- El grupo puede seguir funcionando **mientras haya un proceso vivo que actúe como líder**.

Como no estamos monitorizando a los esclavos, no podemos detectar sus fallos y, por esta razón, permanecen en la lista de esclavos después de estrellarse.

En consecuencia, un esclavo colapsado podría convertirse eventualmente en el nuevo líder, pero en ese momento, los otros procesos comenzarían a monitorizarlo, detectarían que ha fallado y elegirían un nuevo líder.

d) Why do the workers desynchronize?

Los *workers* se *desincronizan*, debido la fallada del *leader* antes de enviar los mensajes multicast, ya que tan solo una porción de *slaves* recibirán este último mensaje, dando lugar a que el respectivo *worker* no tendrá disponible el cambio de vista (color) correcto, quedando desincronizado.

La desincronización de los workers ocurre principalmente si el líder falla durante la multidifusión, logrando enviar un mensaje solo a un subconjunto de esclavos. Como el nuevo líder elegido no retransmite este mensaje perdido, los estados de los workers divergen.

La falta de un mecanismo de acuses de recibo (ACKs) impide además saber si un esclavo procesó un mensaje antes de fallar. La simulación de fallos (`crash/2`) demuestra experimentalmente cómo esta entrega parcial durante el `bcast` causa la inconsistencia.

GMS3 - Mitigar gms2 haciendo que el nuevo líder reenvíe el Last mensaje conocido, asumiendo que el primer nodo de la lista (el nuevo líder) lo recibió si alguien lo hizo.

e) How should we change the implementation to handle the potential loss of messages in the network? How would this change impact performance?

Para ello es necesario implementar **reliable multicast** .
Cuando algún proceso detecta que pierde algún mensaje (el número de secuencia de un mensaje entrante *m* es superior al número de secuencia esperado del siguiente mensaje), el proceso mantiene el mensaje en una cola de espera y solicita a la retransmisión de los mensajes perdidos mediante el envío de NACKs al líder.

Cuando un proceso entrega un mensaje a su trabajador, también debe revisar la cola de espera y entregar cualquier otro mensaje listo (después de comprobar sus números de secuencia). Si el proceso utiliza la implementación básica de la multidifusión fiable , envía también un ACK al líder por cada mensaje entregado a su trabajador.

f) What would happen if we wrongly suspect the leader to have crashed?

Cuando se sospecha erróneamente que un líder ha fallado, se inicia un proceso de elección innecesario que nombra a un nuevo líder mientras el original sigue funcionando. Esto crea una situación de líderes simultáneos donde:

- El líder original continúa operando sin saber que ha sido "reemplazado"
- El nuevo líder elegido comienza a actuar como tal

Esta situación divide efectivamente el sistema en dos partes:

1. Nodos que siguen al líder original (posiblemente solo él mismo)
2. Nodos que siguen al nuevo líder (la mayoría que detectó el falso fallo)

Como consecuencia, el antiguo líder se desincronizaría al entregar solo sus propios mensajes, que seguiría enviando al resto de procesos. Esto podría provocar que los procesos se desincronicen si reciben mensajes con números de secuencia solapados del antiguo y nuevo líder en distinto orden, resultando en:

- Mensajes duplicados
- Órdenes contradictorias
- Estados inconsistentes entre los nodos

En el código de manejo de mensajes de vista, existen diferentes cláusulas para diferentes situaciones:

- La primera maneja mensajes de vista ya entregados
- La segunda maneja mensajes donde la lista de esclavos cambió pero el líder sigue siendo el mismo (un nuevo proceso se unió)
- La tercera maneja cambios de liderazgo debido a fallos (cuando el proceso recibe el mensaje de vista antes de detectar el fallo por sí mismo)

Una posible solución sería implementar un mecanismo donde el líder original, al intentar comunicarse con los esclavos, pueda detectar que ya no es reconocido como líder (por respuestas inesperadas o falta de ellas) y renunciar a su rol.

g) When is it possible that a process crashes after it has delivered a message that will not be delivered by any correct node?

¿Cuándo es posible que un proceso se bloquee después de haber entregado un mensaje que no será entregado por ningún nodo correcto?

Un proceso puede fallar después de entregar un mensaje que no será entregado por otros nodos correctos en el siguiente escenario crítico:

El problema ocurre durante una entrega parcial, cuando el líder comienza a difundir un mensaje pero solo lo envía a algunos esclavos antes de fallar. El momento exacto del fallo es crucial - el líder debe fallar justo después de:

- Haber entregado el mensaje a su propio proceso worker
- Haber enviado el mensaje solo a una parte de los esclavos
- No haber almacenado el mensaje en ningún buffer de recuperación

La situación se vuelve irrecuperable porque **GMS3 solo reenvía el último mensaje conocido** durante la elección de nuevo líder. Si este mensaje específico no fue el último recibido por el nuevo líder, se pierde definitivamente.

El resultado es una **inconsistencia permanente** en el sistema: los nodos que recibieron el mensaje tendrán un estado diferente a los que no lo recibieron, y esta discrepancia no puede resolverse automáticamente con los mecanismos implementados.

Este problema ilustra por qué los sistemas distribuidos con fuertes garantías de consistencia implementan protocolos de consenso más robustos como Paxos o Raft, donde un mensaje solo se considera "entregado" cuando una mayoría de nodos ha confirmado su recepción.