Universidad Nacional Autónoma de México

Facultad de Ciencias

Computación distribuida

Práctica 7

Consenso

Baños Mancilla Ilse Andrea - 321173988

Flores Arriola Edson Rafael - 423118018

Rivera Machuca Gabriel Eduardo - 321057608

Reporte de práctica

Introducción

Como hemos visto a lo largo del curso, el problema de consenso en sistemas distribuidos consiste en lograr que un conjunto de procesos comunicados mediante mensajes, lleguen a un acuerdo sobre un valor único, aún en presencia de fallos. Dentro de los algoritmos de este tipo hemos visto esquemas básicos de votación síncrona y protocolos como el del Rey o versiones reducidas del Problema de los Generales.

En esta práctica, implementamos un protocolo de consenso síncrono basado en gráficas completas, donde cada proceso puede comunicarse directamente con el resto. Además, utilizaremos un modelo de fallas basado en desconexión (“crash”), el cual es un escenario más restringido que el modelo bizantino, por ejemplo, pero sigue siendo realista en aplicaciones prácticas.

Este diseño coincide con el modelo que hemos utilizado repetidamente en las prácticas: un sistema en rondas perfectamente sincronizadas, en el cual cada proceso recibe todos los mensajes enviados en la ronda anterior antes de avanzar a la siguiente. En este entorno, el consenso es alcanzable en un número de rondas finito. Dicho esquema funciona mediante un protocolo de votación iterada, similar al del algoritmo del Rey, donde cada nodo mantiene un conjunto de valores conocidos y los propaga al resto en cada ronda.

La implementación final utiliza el modelo síncrono de pasos discretos basado en SimPy, donde el algoritmo se ejecuta por rondas, propagando conjuntos de valores en estructuras `New` y `V`, siguiendo el patrón del Algoritmo 1 proporcionado. El modelo de fallas por apagado se implementa deshabilitando el envío y recepción de mensajes en nodos con `id < f`.

Implementación de NodoConsenso

Para implementar este algoritmo se usó de base lo ya mencionado en la introducción utilizando SimPy y la estructura de simulación modular proporcionada en prácticas anteriores (clase Nodo, Canales, y el motor de eventos env). El sistema simula una red síncrona en rondas, donde todos los nodos envían mensajes al inicio de cada ronda. Después esperan un tiempo fijo (TICK) para permitir que los mensajes lleguen. Posteriormente procesan todos los mensajes recibidos en esa ronda.La implementación usa CanalRecorridos, que ya maneja la cola de mensajes. Esto reproduce el modelo muy similar

al estudiado, el cual consiste en comunicación síncrona con recepción garantizada dentro de cada ronda (excepto si el proceso falla).

En el código, cada nodo mantiene:

- V: vector donde cada posición k contiene el valor propuesto por el nodo k (cuando se conoce).
- New: conjunto de pares (v, k) que representan “nuevo conocimiento” que el nodo debe propagar.
- rec_from[j]: los valores nuevos recibidos desde el vecino j en la ronda actual.
- fallare: flag que indica si el nodo dejará de funcionar.
- lider: la decisión final.

Respecto al manejo de fallas, el algoritmo simula fallas crash definidas por:

```
if self.id_nodo < f:
    self.fallare = True
```

Los nodos que fallan siguen “vivos” en la simulación pero ya no envían ni procesan mensajes.

En cuanto a la propagación de mensajes, durante cada ronda:

1. Si New no está vacío, enviar el contenido a todos los vecinos.
2. Esperar TICK.
3. Recibir todos los mensajes de la ronda.
4. Incorporar nuevos valores descubiertos.
5. Actualizar la nueva New.

Después de recibir mensajes:

- El nodo examina todos los pares (v,k) recibidos.
- Si nunca ha visto al nodo k (V[k] is None):
    - entonces actualiza su información
    - y añade ese nuevo par a New.

Este mecanismo asegura que la información fluye transitivamente a lo largo de la red, pues si un nodo conoce algo nuevo sobre otro nodo, lo reenvía hasta que todos los nodos correctos lo conocen. Después de completar las rondas los nodos ejecutan:

```
for valor in self.V:
    if valor is not None:
        v_decision = valor
        break
```

Con esto, eligen el valor con menor índice en su conocimiento, el cual corresponde al ID menor en V. En otras palabras, el líder es elegido como el primer valor no vacío de V, propuesto necesariamente por algún nodo funcional.

Resultados, Observaciones y Conclusiones

En esta Práctica pudimos probar que cuando no hay fallas, todos los nodos convergen al mismo valor después de unas cuantas rondas. Además, los nodos correctos aún pueden llegar a un acuerdo común aún si algunos nodos fallan. En concreto, si ejecutamos *r* rondas podemos tolerar hasta *r - 1* fallos, pues garantizamos que los nodos sobrevivientes se ejecuten en la ronda adicional. Finalmente, la lista V mantiene el conjunto de valores de los nodos correctos, y el valor elegido es consistente en ellos.

En cuanto a las observaciones, notamos que el atributo V funciona como un registro de conocimiento acumulado, que permite la coordinación eficiente. Además, el algoritmo no ocupa comparaciones para decidir el resultado de la “votación”, pues lo hace automáticamente al seleccionar al primer nodo no vacío de V.

En conclusión, fue posible implementar un algoritmo de consenso bajo fallas por apagado en un modelo síncrono. El uso de SimPy simplificó la simulación de rondas y el manejo de mensajes, permitiendo centrarse en la lógica del protocolo.