Tarea 7 - Cómputo distribuido

Dalia Camacho, Gabriela Vargas

1 Teorema 3: de *Consensus in Synchronous Systems: A Concise Guided Tour*

Para introducir el Teorema 3 del artículo *Consensus in Synchronous Systems: A Concise Guided Tour* [2] recordamos los conceptos de fallas por *crash*, fallas por omisión y fallas bizantinas. Así mismo hacemos la distinción entre acuerdo y acuerdo uniforme para el problema de consenso.

Una falla de tipo *crash* se refiere a que los procesos fallidos se detienen (o mueren) y a partir de ese momento ya no forman parte de la toma de decisiones.

Una falla por omisión se refiera a que los procesos fallidos no envían o no reciben información de uno o varios procesos.

El consenso se logra si cumple con las propiedades de terminación, acuerdo y validez. El acuerdo consiste en que cualesquiera dos procesos correctos deciden el mismo valor. Sin embargo el acuerdo uniforme es más estricto que el acuerdo, ya que para que el acuerdo uniforme se logro todos los procesos correctos o fallidos deben elegir el mismo valor. Si un problema de consenso requiere acuerdo uniforme, entonces se le conoce como problema de consenso uniforme.

Theorem 1 (Teorema 3 en [2])**.** *No existe un protocolo que pueda resolver el problema de consenso uniforme en un sistema síncrono que acepte $f \geq \frac{n}{2}$ fallas por omisión.*

Proof. Este teorema se demuestra por contradicción, supongamos que existe un protocolo A tal que resuleva el problema de consenso uniforme en el caso binario. Ahora dividimos los procesos en dos conjuntos el P_0 y el P_1. El P_0 contiene $f \geq \frac{n}{2}$ procesos y el P_1 contiene $n - f \leq \frac{n}{2} \leq f$ procesos. Como f es el número máximo de fallas puede ocurrir que fallen todos los procesos de P_0 o todos los procesos de P_1. A continuación consideramos las siguientes ejecuciones.

- *Ejecución R_0:* Todos los procesos proponen el valor 0 y todos los procesos de P_0 son correctos, en cambió todos los procesos de P_1 tienen una falla tipo *crash*. Por lo que los procesos en P_0 no escuchan de los procesos en P_1 y todos los procesos en P_0 eligen el valor 0.

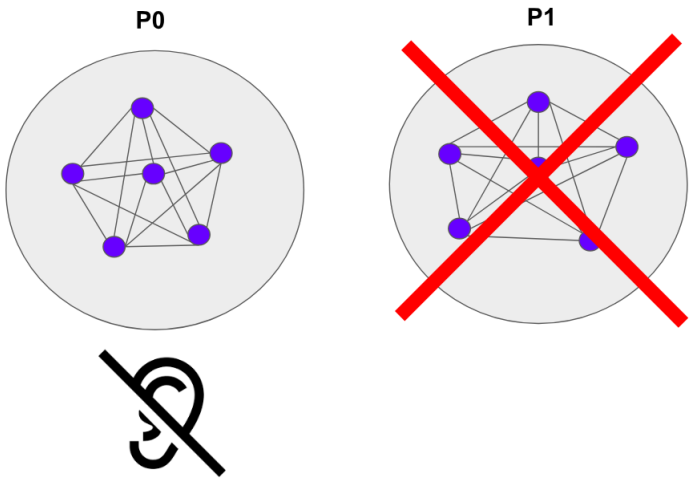

Figure 1: Se representa la ejecución E_0 y se muestra como los procesos P_0 no escuchan de los procesos en P_1 debido a una falla tipo *crash*.

- *Ejecución R_1:* Todos los procesos proponen el valor 1 y todos los procesos de P_1 son correctos, en cambió todos los procesos de P_0 tienen una falla tipo *crash*. Por lo que los procesos en P_1 no escuchan de los procesos en P_0 y todos los procesos en P_0 eligen el valor 0.

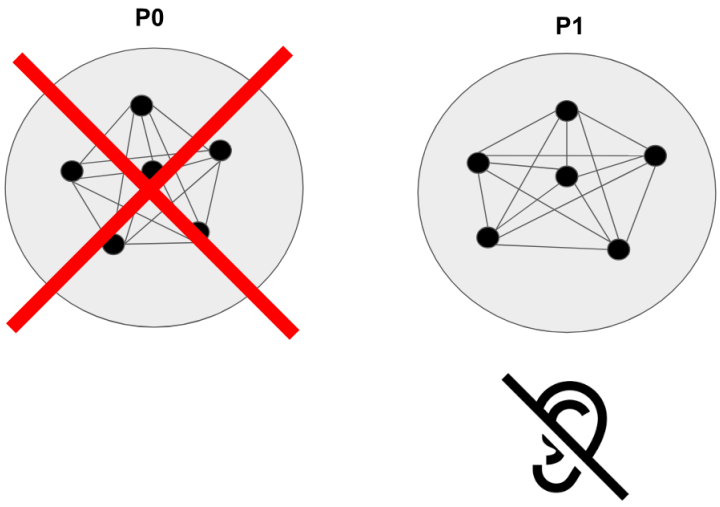

Figure 2: Se representa la ejecución E_1 y se muestra como los procesos P_1 no escuchan de los procesos en P_0 debido a una falla tipo *crash*.

En los dos casos anteriores los procesos correctos de un conjunto no escuchan de los procesos del otro conjunto y por lo tanto eligen su valor inicial. Como las fallas fueron por *crash* no se genera ningún conflicto. El tercer caso $R_{0,1}$ que presentamos más adelante tiene fallas por omisión. Más aún se logra replicar la visión que tienen los procesos P_0 en E_0 y la que tienen los procesos P_1 en $E_{0,1}$.

- *Ejecución* $R_{0,1}$*:* Los procesos en P_0 proponen 0 y los procesos en P_1 proponen 1. Todos los procesos en P_1 tienen fallas por omisión, estos no escuchan a los procesos en P_0 ni envían información a los procesos en P_0. Por lo que los procesos en P_0 sólo escuchan de otros procesos en P_0 con valor 0 y los de P_1 sólo escuchan de otros procesos en P_1 con valor 1. Debido a esto la visión de los procesos en P_0 es la misma que en el caso E_0 y no hay forma en que puedan distinguir la ejecución $E_{0,1}$ de la ejecución E_0, estas son indistinguibles. Lo mismo ocurre con los procesos en P_1 para los cuales E_1 es indistinguible de $E_{0,1}$, por lo que todos eligen 1. Con esto tenemos que los procesos en P_0 eligen 0 y los procesos en P_1 eligen 1 y el acuerdo no fue uniforme. Por lo que no existe un protocolo que logre resolver el problema de consenso uniforme.

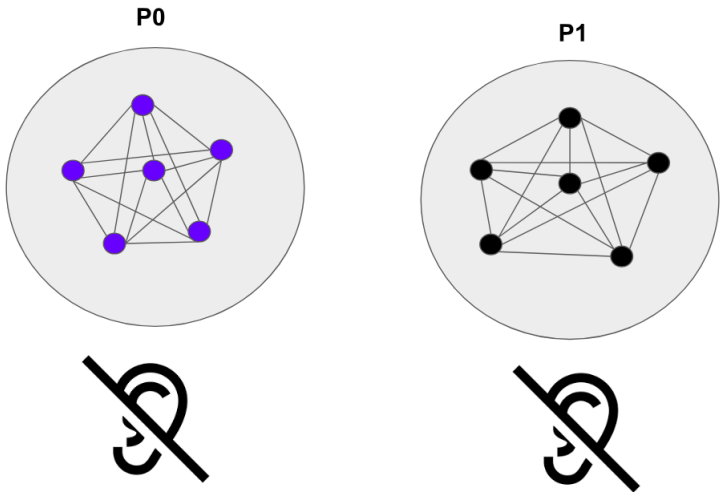

Figure 3: Se representa la ejecución $E_{0,1}$ y se muestra como los procesos P_0 no escuchan de los procesos en P_1 y los P_1 no escuchan de los procesos en P_0. Si comparamos la visión de los P_0 con la ejecución E_0 ésta es la misma. De la misma forma la visión de los procesos en P_1 es igual a la de los procesos en la ejecución E_1.

□

2 Describir lo visto en clase[1]

2.1 Teorema CAP

La clase comenzó recordando el teorema de imposibilidad FLP, el cual indica que nos es posible resolver el problema de consenso en entornos asíncronos sin cambiar las condiciones de agreement haciendo ajustes de consenso aproximado. La justificación de este teorema se basa en que es imposible garatizar simultáneamente 3 propiedades:

1. **Consistencia:** Todos los nodos deben contar con una copia de la misma versión de los datos, es decir, todos los nodos deben estar viendo lo mismo.
2. **Disponibilidad:** Las fallas de algún nodo o del canal de comunicación no deben privar a los nodos sobrevivientes de seguir operando.
3. **Tolerancia a las particiones:** En caso de que la red sufra una partición, el conjunto debe seguir operando.

El teorema CAP propuesto por Gilbert y Lynch[1], indica que aunque en el problema de consenso no se pueden cumplir simultáneamente estas 3 propiedades, pero es posible satisfacer 2 de 3.

2.1.1 Descripción del sistema distribuido

Se considera un sistema distribuido compuesto por dos servidores G_1 y G_2. Ambos servidores operan con la misma variable v, con valor inicial v_0. Hay canales de comunicación entre servidores y entre cliente-servidor, por lo que este sistema se puede representar con siguiente figura:

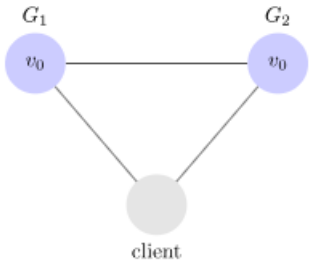

Figure 4: Sistema distribuido con dos servidores.

El cliente puede generar solicitudes de lectura y escritura sobre la variable v. Una operación de escritura se representa de la siguiente manera: el servidor recibe una solicitud de escritura, ejecuta la operación y manda una confirmación.

[1] Imágenes de esta sección tomadas de https://mwhittaker.github.io/blog/an$_i llustrated_p roof_o f_t he_c ap_t heorem/$

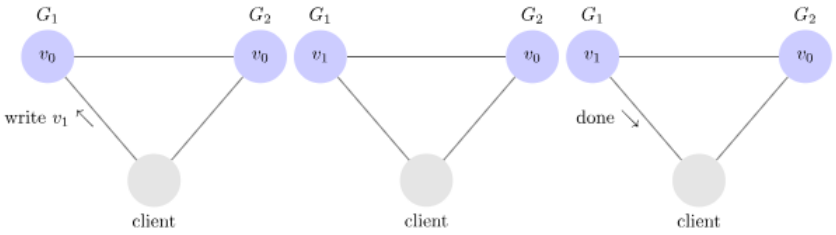

Figure 5: Operación de escritura en un sistema cliente-servidor.

La operación de lectura se ve de la siguiente forma: el servidor recibe una solicitud de lectura y envía el dato solicitado.

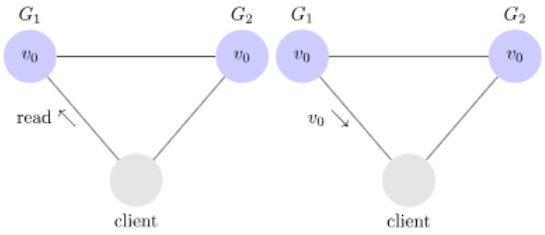

Figure 6: Operación de lectura en un sistema cliente-servidor.

2.2 Propiedad de consistencia

En un sistema fuertemente consistente, el cliente envía una solicitud de escritura a cualquier servidor para modificar la variable v y una vez que reciba confirmación por parte del servidor que realizó la operación, todos los demás sevidores deberán tener el nuevo valor de v. Como consecuencia, una solicitud de lectura de v hecha a cualquier servidor de la red, debería tener el mismo valor como respuesta.

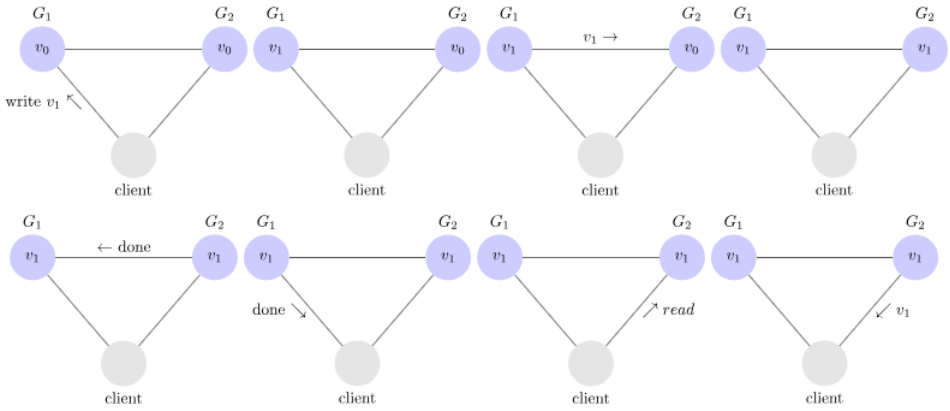

Figure 7: Operación de escritura en un sistema consistente.

2.3 Propiedad de disponibilidad

En un sistema con alta disponibilidad, si el cliente envía una solicitud a un servidor que no presenta falla, entonces ese servidor deberá enviar una respuesta al cliente, ya que el servidor no está autorizado a ignorar solicitudes del cliente.

2.4 Propiedad de tolerancia a particiones

En el contexto del sistema distribuido descrito anteriormente, una partición se describe como una falla en el canal de comunicación entre los servidores, pero un sistema con alta tolerancia a particiones sería capaz de funcionar correctamente aunque haya pérdida arbitraria de mensajes en la red.

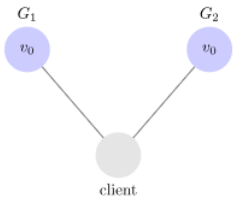

Figure 8: Partición en sistema.

2.5 Ejemplos de estrategias

De acuerdo con lo mencionado sobre el teorema CAP, un sistema distribuido será capaz de cumplir simultáneamente dos de tres propiedades. Por lo que es necesario definir una estrategia para asignarle mayor paso a la propiedades más críticas, dependiendo del tipo de problema a tratar.

Algunas estrategias incluyen los siguientes ejemplos:

- **Modelo de consistencia de Facebook:** Es un modelo débil en consistencia que prioriza la disponibilidad y la tolerancia a particiones, de tal forma que los usuarios obtengan repuesta del servidor la mayor parte del tiempo, aunque la información que reciban no sea la más actualizada.
- **Modelo de consistencia de cajeros automáticos:** Es un modelo fuerte en consistencia que sacrifica la disponibilidad del cajero cuando hay problemas de comunicación en los servidores.
- **Modelo de consistencia en sistemas de reserva:** Es un modelo que cambia de estrategia dependiendo de la disponibilidad de lugares a reservar. Si se tienen muchos lugares disponibles, entonces se le da preferencia a la disponibilidad del sistema, y en el caso contrario, donde hay pocos lugares disponibles, se priorizaría la consistencia.

2.6 Prueba simple del teorema CAP

Se pretende demostrar que un sistema distribuido no puede mantener las tres propiedades de consistencia, disponibilidad y tolerancia a particiones.

Asumimos por contradicción que existe un sistema consistente, disponible y tolerante a fallas. Entonces el sistema se vería de la siguiente manera:

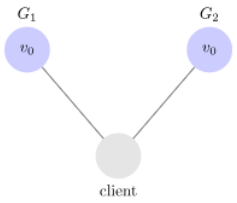

Figure 9: Sistema que capaz de cumplir con las tres propiedades.

Ahora consideremos una solicitud de escritura del cliente sobre la variable v. Como el sistema está disponible, el servidor G_1 debería responder, pero como el sistema está particionado, G_1 no podrá compartir el nuevo valor de v_1 a G_2.

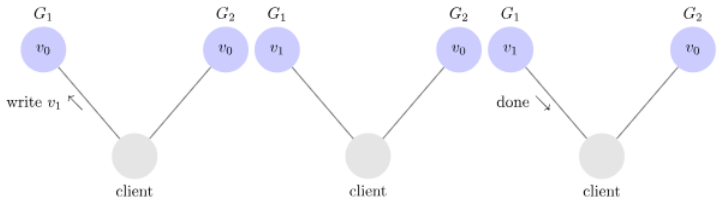

Figure 10: Solicitud de escritura en un sistema con partición.

En una segunda ejecución, el cliente solicita una lectura sobre v al servidor G_2, como el sistema tiene la propiedad de disponibilidad, entonces G_2 está obli-

gado a responderle con un valor incorrecto, ya que no recibió la actualización v_1 del servidor G_2 y entregará el valor inicial v_0, lo cual viola la propiedad de consistencia, y por lo tanto hemos demostrado que existe una ejecución inconsistente del sistema, lo cual contradice el supuesto de que existe un sistema que puede cumplir simultáneamente con las tres propiedades.

References

[1] Seth Gilbert and Nancy Lynch. Perspectives on the cap theorem. *Computer*, 45(2):30–36, 2012.

[2] M. Raynal. Consensus in synchronous systems: a concise guided tour. In *2002 Pacific Rim International Symposium on Dependable Computing, 2002. Proceedings.*, pages 221–228, Dec 2002.