Grado en ingeniería informática

2021-22

Universitat Oberta de Catalunya

PEC 2. Trabajar juntos: comunicar y coordinar acciones

75.589 - Sistemas distribuidos

1. *Discute el problema de replicación activa y pasiva en base a la información presentada en Wikipedia:*
   https://en.wikipedia.org/wiki/Replication_(computing)
   Basándote en el contenido del libro de Coulouris, busca y propón al menos una información que podrías agregar a esta página web de Wikipedia para mejorar su contenido real.

Se define como replicación al proceso de compartir información entre diferentes fuentes para asegurar la consistencia de esta, para así incrementar la fiabilidad, accesibilidad y tolerancia a fallos del sistema.

En los modelos que realizan replicación, en el contexto de los sistemas distribuidos, existen dos componentes fundamentales:

- Gestor de réplicas, siendo el componente encargado de almacenar las réplicas en cuestión.
- Frontal, que actúa como interfaz entre los clientes y los gestores de replicas, recibiendo las peticiones de los clientes y comunicándolas a los gestores de réplicas.

La replicación es habitualmente categorizada dentro de dos grandes grupos, la replicación activa y la replicación pasiva:

- **Replicación activa:** Se entiende que una replicación es activa, cuando esta se lleva a cabo ejecutando las peticiones que llegan en cada una de las réplicas existentes.
  En este tipo de replicación el cliente interactúa con los frontales al realizarle una petición, la cual es difundida por estos a todos los gestores de replicas por igual. Estos gestionan de forma concurrente estas peticiones, tras lo cual el cliente se sincroniza con la primera respuesta recibida de la misma.

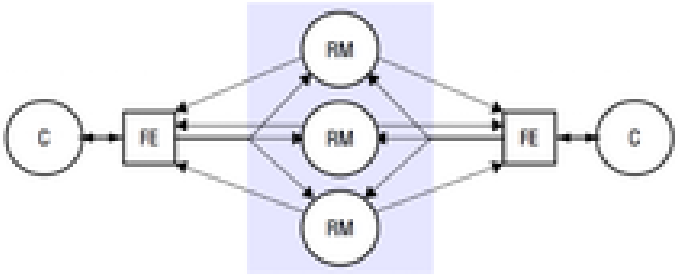


Ilustración 1. Replicación activa

En términos de ejecución, el proceso de replicación activa comienza cuando el frontal recibe la petición del cliente y la difunde a todos los gestores de réplicas utilizando las herramientas de multidifusión fiable y de ordenación total, tras lo cual se queda a la espera de recibir una respuesta antes de continuar enviando más peticiones. Una vez que el frontal ha ejecutado la multidifusión, el sistema de comunicación se encargara de entregar la petición a los gestores según lo establecido en

la ordenación total. Cuando el sistema de comunicación ha entregado las peticiones los gestores ejecutarán la petición de forma independiente, aunque idéntica, y responderán al frontal.

- **Replicación pasiva:** En contraposición se entiende que la replicación es pasiva cuando la petición se ejecuta únicamente en una réplica y se transfiere su resultado al resto de las réplicas.
  Este enfoque utiliza habitualmente una arquitectura maestro esclavo en la que se designa a un gestor de réplicas primario que ejecuta las peticiones (que actúa como maestro) y se envían los resultados de estas ejecuciones a los gestores segundarios (Esclavos). En este caso los frontales actuarán como intermediarios entre los clientes y los gestores primarios, al ser estos los que ejecutarán las peticiones.

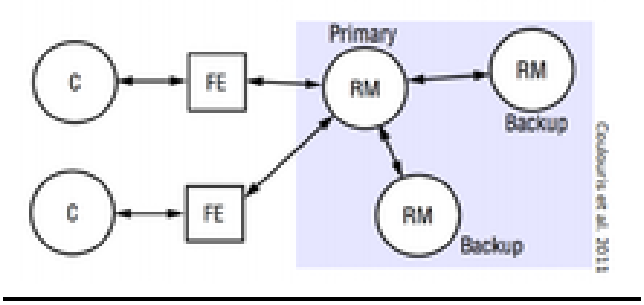


Ilustración 2. Replicación pasiva

Por su parte, el proceso de replicación comenzará tras la recepción de la petición del cliente por parte del frontal, la cual enviará al gestor primario quien ejecutará la petición, de acuerdo al esquema FIFO (first in first out), y almacenará la respuesta. En este punto si la petición era de actualización el gestor primario enviará está a todos los gestores secundarios, que confirmarán la recepción mediante el uso de mensajes de tipo ACK. Finalmente, el gestor primario comunicará la respuesta al frontal y este al cliente.

La replicación activa tiene las ventajas de conseguir reducir la latencia, ser tolerante a los fallos, debido a el uso de una multidifusión fiable y totalmente ordenada, y ser capaz de resolver fallos bizantinos al ser equivalente a un algoritmo de consenso en el que el frontal recoge f+1 respuestas antes de responder al cliente.
Como desventaja requiere un gran esfuerzo computacional para que todos los gestores ejecuten todas las peticiones y gestionen la comunicación para que todas las peticiones sean transmitidas a todos los gestores de réplicas y todas las respuestas de sus ejecuciones de vuelta a los frontales.

Por su parte, la replicación pasiva tiene las ventajas de que el frontal requiere de poca funcionalidad y que se consigue mantener una consistencia secuencial al controlar el orden de modificación mediante un único gestor primario. Pero a su vez adolece de las desventajas de no soportar fallos bizantinos y de sufrir de problemas de cuello de botella en gestor el primario.

En lo que respecta a una información contenida en el libro Coulouris que podría mejorar el contenido real de Wikipedia, se podría proponer la información relativa a los requisitos que se deben cumplir para que un sistema que utiliza la replicación pasiva siga conservando la linealidad en caso de que su gestor primario caiga. Estos se podrían dividir en dos fundamentalmente:

- El primer requisito es satisfecho siempre y cuando el gestor primario sea remplazado únicamente por uno de los servidores secundarios.
- El segundo se basa en que los gestores de replicas que sobreviven tras la caída del gestor primario deben llegar a un acuerdo en el orden de las operaciones hasta el momento de la caída, para que así cuando se remplace este gestor primario se pueda seguir manteniendo la consistencia secuencial.

2. *Apache Zookeeper es uno de los códigos más utilizados para la coordinación distribuida.*
   Contesta las siguientes preguntas:
   a. *Explica cómo consigue Apache Zookeeper la elección de líder. Mira el siguiente vídeo (inicio en el segundo 4:26).*
   https://www.coursera.org/lecture/cloud-computing-2/1-3-election-in-chubby-andzookeeper-IDKhR

Apache Zookeper es un proyecto de código abierto, basado en un servidor centralizado utilizado para mantener la información de configuración, por lo que necesitará mantener un líder electo entre los nodos en todo momento.

La forma principal con la que se escoge a este líder es haciendo rondas de votación, en las cuales cada servidor crea un número de secuencia (id) para sí mismo y lo manda al resto de nodos a modo de proposición de líder, obteniéndose este de sumar uno al del número de secuencia más alto contenido en su sistema de ficheros Zookeeper.

Una vez todos los servidores han realizado el proceso, como todos los servidores han recibido el id más alto de los demás, en la siguiente ronda todos los servidores ya tienen en su sistema de ficheros Zookeeper el valor más alto de cada uno de los servidores y por consiguiente la elección de todos deberá ser unánime, correspondiéndose al valor más alto de entre todos los servidores que han promocionado a un líder.
De esta forma se garantiza que mientras los ficheros en los servidores sean escritos de manera atómica, o lo que es lo mismo que se escriben de manera completa en cada servidor sin que existan sistemas de ficheros con información parcial, la elección del líder siempre será única y unánime, salvo que exista un conflicto porque existen dos Ids iguales o porque el líder falle durante la elección.

En el caso de que exista este tipo de conflictos, Zookeeper lo soluciona mediante un protocolo de *Commit* en dos fases, consistente en los pasos:

1. El líder manda un mensaje de NEW_LEADER al resto de servidores
2. Cada proceso responde con un mensaje de ACK al menos a un líder (al servidor con el ID más alto)
3. El líder espera para la mayoría de ACK y después manda un mensaje de COMMIT a todos.
4. Una vez se ha recibido el COMMIT, los procesos actualizan su líder.

Por ejemplo, en el video se muestra que los 6 servidores en primera ronda promocionan N3, N5, N6, N12, N32 y N80, pero evidentemente en segunda ronda solo quedará **N80**

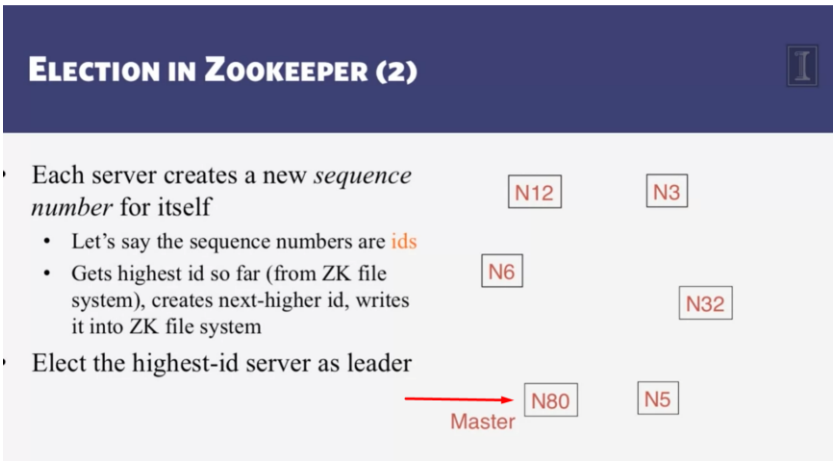


Ilustración 3. Elección de lider en Zookeeper

Otro punto importante a considerar para mantener la condición necesaria de *Zookeeper* de que siempre exista un líder electro son los mecanismos necesarios para manejar los posibles fallos o caídas del líder.
A este respecto el sistema Zookeeper tiene varias alternativas, el primer mecanismo consiste en que cada servidor monitorice el correcto funcionamiento del líder, mediante algún mecanismo de detección de fallos como un ping-poning, y una vez que el fallo esta confirmado se comience otra nueva elección. Este mecanismo, aunque en principio lógico, puede suponer que cuando el líder falle muchos servidores comiencen elecciones de forma simultánea, conllevando esto un aluvión de mensajes y saturación de las comunicaciones.

Este problema que lleva implícito este mecanismo lleva a Zookeeper a que en realidad implemente un método diferente, en el cual cada vecino monitoriza el ID siguiente más alto, de forma que en caso anterior, por ejemplo, N3 monitorizará N6, que monitorizará N6 que monitorizará N12… De forma que si alguno de los procesos de esta cadena circular fallará el proceso inmediatamente anterior lo detectaría y se declararía de forma automática el siguiente a ser líder en la cadena de orden de IDs

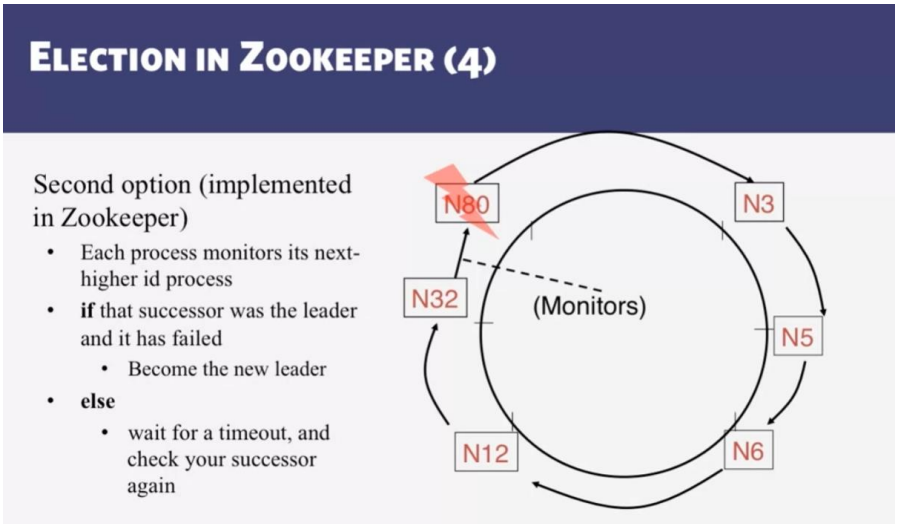


Ilustración 4. Mecanismo para la detección y sustitución de líder en Zookeeper

b. Compara el algoritmo de Zookeeper con otros dos algoritmos de elección de líder y explica ventajas y desventajas.

Un ejemplo de otro algoritmo de elección de líder es el caso del algoritmo de consenso *Raft*, que es el mecanismo de elección de líder más utilizado en Amazon. Este ofrece de manera integrada la tolerancia a errores y funciona con una base de datos única que almacena al líder actual.

Este algoritmo es de líder único, y en este un nodo puede estar en estado seguidor, candidato o líder.

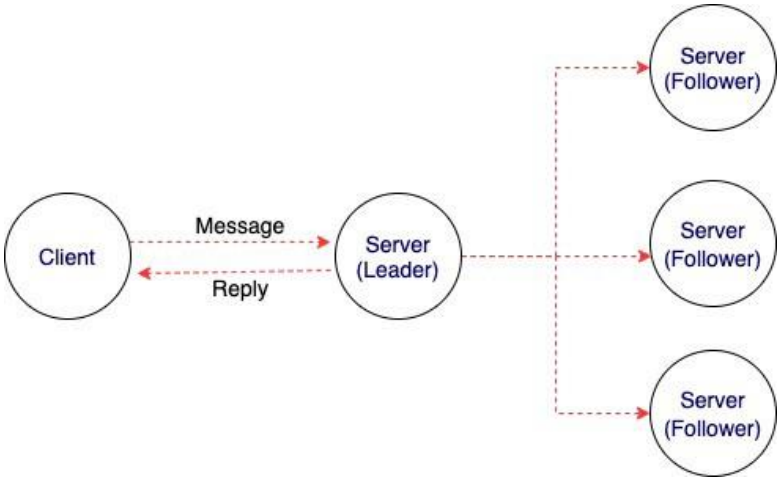


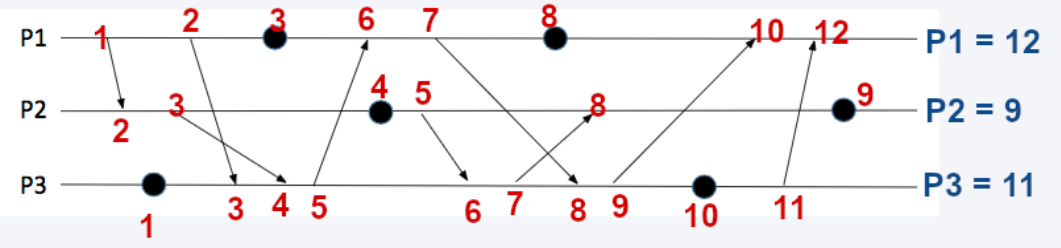
Ilustración 5. Algoritmo de consenso Raft

En este algoritmo es necesario que el líder se muestre activo al resto de nodos de forma periódica, de forma que si no lo hace en un periodo de tiempo determinado (*timeout*), otros nodos que ostenten el rol de candidatos pueden tratar de obtener el rol de líder. De esta manera se trata de no depender del tiempo, dada la difícil tarea de que los nodos dentro de un *cluster* se mantengan sincronizados para ordenar o coordinar las operaciones distribuidas.

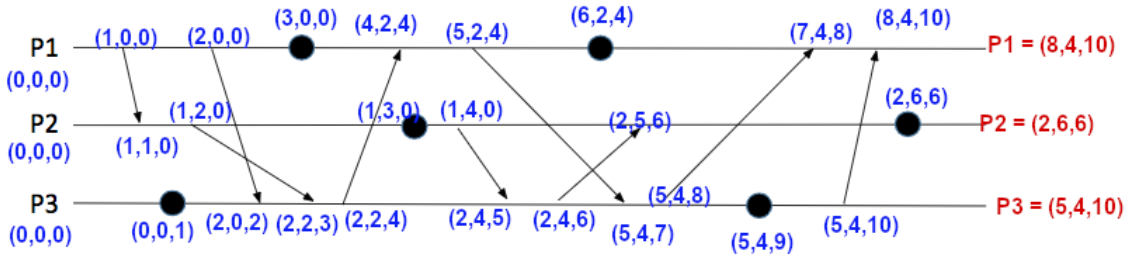
Otro ejemplo podría ser el del algoritmo del abusón (*bully*) con el cual se realiza la elección dinámica del líder según el valor de ID de los procesos, siendo el ID de mayor valor el elegido como líder. Este sistema se caracteriza por ser síncrono y utilizar tiempo de espera para la identificación de fallos en los procesos.

En comparación a algoritmos de elección en anillo como el utilizado en *Zookeeper*, la elección basada en el algoritmo abusón supone que el sistema es síncrono, utiliza el tiempo de espera para detectar fallos o caídas de los procesos y en que cada proceso conoce que otros procesos tienen el ID más alto que el suyo y pueden comunicarse entre ellos.

3. *Sincronización de los relojes:*
   a. *En el siguiente caso, donde tanto las flechas como los puntos indican el progreso del reloj en cada proceso, considera:*
      i. *Relojes Lamport. Todos los relojes comienzan en cero. Etiqueta el diagrama con los valores del reloj de Lamport. Muestra el estado final de los relojes.*



      ii. *Relojes vectoriales. Todos los relojes comienzan en (0,0,0,0). Etiqueta el diagrama con los valores del reloj vectorial. Muestra el estado final de los relojes.*



¿Cuál es la diferencia entre relojes vectoriales y de Lamport? ¿Cómo los relojes vectoriales mantienen las relaciones de causalidad entre eventos? ¿Cómo se aplicaron a Amazon Dynamo para detectar la necesidad de reconciliación semántica? https://dl.acm.org/doi/abs/10.1145/1323293.1294281

Por un lado los relojes de Lamport se definen como un contador software monótonamente creciente que no tiene relación con ningún reloj físico y que funciona de forma que cada proceso mantiene su propio reloj lógico, que actualiza cada vez que sucede un evento, siendo así capaz de mostrar el orden de los procesos mediante el empleo de relaciones *happened-before*, o de ordenamiento causal, habilitándose la determinación de que si un evento sucedió antes que otro tendrá una marca temporal menor que este.

Sin embargo, este sistema tiene varios inconvenientes, primero el problema de que no es capaz de demostrar que si un evento tiene una marca temporal en su reloj de Lamport menor a otro este ocurrió antes y el segundo que al capturar únicamente el flujo de datos entre dos procesos puede mostrar el orden, pero no la causalidad entre eventos.

Por su parte los relojes vectoriales nacieron como respuesta al problema de los relojes de Lamport de no poder asegurar que si un evento tiene una marca temporal de su reloj de Lamport más pequeña sucedió antes.

Los relojes vectoriales, en lugar de almacenar un único contador como en el caso de los relojes de Lamport, almacenan un vector de N enteros, uno para cada uno de los procesos implicados en la sincronización, de forma que siendo Vi el vector de relojes de un proceso, Vi[i] será el contador respectivo a su proceso y el resto respectivo al resto de procesos implicados.

De esta forma, aun teniendo la desventaja evidente respecto a los relojes de Lamport de necesitar de más almacenamiento para los contadores, los relojes vectoriales son capaces de solucionar el problema de la causalidad entre eventos, asignando a cada uno de los procesos del vector un valor por el estado del resto de procesos, de forma que, mediante la inspección de las marcas de tiempo entre dos eventos, se podrá corroborar que uno sucedió antes que el otro.

Amazon Dynamo utiliza los relojes vectoriales para detectar la causalidad entre diferentes eventos del mismo objeto, de forma que se identificó que un vector de reloj correspondía a un par nodo-contador y se atribuyó cada versión de un objeto almacenado en Dynamo a un vector de reloj, de forma que para determinar la causalidad entre dos versiones únicamente era necesario inspeccionar la causalidad entre sus respectivos vectores de reloj.

4. Teoremas CAP y ACID

a. Explica los dos teoremas, compáralos e indica cuál ha sido su evolución

ACID es la denominación de un teorema para las transacciones en bases de datos, entendiendo por transacción a una interacción, compuesta por varios procesos, con una estructura de datos compleja de forma que toda ella debe ser realizada de una sola vez sin que la estructura de datos en cuestión pueda ser accedida por el resto del sistema.

De forma que se podrá afirmar que una transacción sigue el teorema ACID cuando cumple los principios que se detallan en sus siglas (*Atomicity, Consistency, Isolation, Durability*), debiéndose garantizar:

- *Atomicity*: Cuando la transacción se ejecuta de forma atómica, o lo que es lo mismo, teniendo la garantía de que esta en su totalidad se realiza por completo o no se realiza, pero nunca queda como incompleta.
- *Consistency*: Indicando que una vez se ha realizado la transacción la estructura de datos quedara en un estado consistente o integro, de acuerdo a las normas establecidas por la base de datos en cuestión.
- *Isolation*: Una transacción se entiende como un evento sin estado, de forma que esta será completamente independiente del resto de transacciones, por lo que nunca se afectarán entre sí.
- *Durability*: Una vez la transacción ha sido realizada, los cambios ocasionados por ella perdurarán en el tiempo.

Respecto a su evolución, el término ACID nace en 1983 de manos de Theo Härder y Andreas Reuter, con el propósito de proveer una garantía para un entorno seguro para el procesamiento de datos, ya que ofrecía una garantía de que los datos se mantenían consistentes y estables y se podían utilizar aún estando repartidos en diferentes localizaciones de memoria. De esta forma muchos de los sistemas de bases de datos NoSQL utilizaban este modelo para asegurar que los datos estaban guardados de forma segura y consistente.

Por su parte CAP es un teorema que trata la implantación de sistemas distribuidos de forma que cumplan dos de las tres propiedad deseadas y expuestas en sus siglas, *Consistency, Availabiliyu y Partitions*:

- *Consitency*: A diferencia de lo que significaba este término para el teorema ACID, en CAP significa que si un valor es alterado en un sistema distribuido inmediatamente después si un proceso intenta leer este valor obtenga el mismo valor modificado. De forma que esta propiedad se esta incumpliendo si se da el caso que un proceso obtenga un valor diferente del que ha cambiado otro proceso (o nodo) anteriormente.
- *Availability*: Se cumplirá esta propiedad cuando exista la garantía de que si algún número de nodos en el sistema fallan se siga manteniendo operativo el sistema.
- *Partitions*: Esta propiedad está relacionada a la tolerancia a las particiones del sistema distribuido, de forma que, si los nodos pertenecientes al sistema o datos distribuidos son separados en dos grupos y uno de ellos pierde la comunicación, el sistema siga permaneciendo operativo. De esta forma se garantiza que si un nodo falla el sistema siga funcionando y no se extienda el fallo al resto del sistema hasta hacerlo no operativo.

Por su parte el teorema de CAP fue propuesto por Eric Brewer, estableciendo que los sistemas distribuidos no pueden ofrecer de forma simultánea más de dos de los tres principios propuestos. Este teorema se presentó en 1998, para en 2002 tener su prueba real de concepto en el MIT. El movimiento de las bases de datos NoSQL han aplicado el teorema CAP como un argumento contra el más tradicional enfoque de ACID, en el que se prioriza la consistencia de los datos, y la partición frente a la potencial baja disponibilidad de los datos.

b. Qué caracterísicas tienen en este aspecto: Cassandra i MongoDB
https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/dml/dmlDatabaseInternalsTOC.html
https://docs.mongodb.com/manual/replication/

MongoDB es un popular sistema de gestión de bases de datos NoSQL que guarda datos como ficheros BSON (bynary JSON), ampliamente utilizado en entornos de BigData y tiempo real en aplicaciones que se ejecutan en diferentes localizaciones.

En lo que respecta a ACID, en MongoDB se añadió soporte para transacciones ACID multi-documento desde la versión 4.0 en 2018 y se extendió ese soporte para transacciones ACID multidocumento distribuidas en la versión 4.2 expuesta en 2019. Sin embargo, el modelo de documento de MongoDB permite que los datos relacionados se almacenen juntos en un solo documento, de forma que este formato junto con las actualizaciones atómicas de documentos obvia la necesidad de transacciones en la mayoría de los casos de uso.

Respecto a su relación con el teorema CAP, MongoDB es un sistema de almacenamiento de datos CP, que significa que resuelve las particiones de la red manteniendo la consistencia a riesgo de comprometer su disponibilidad.
En relación con el concepto de disponibilidad CAP, MongoDB es un sistema de un único master en el que cada set de réplicas puede tener un único nodo primario que recibe todas las operaciones de escritura. Cuando un nodo primario deja de estar disponible, el nodo secundario con el registro de operaciones mas reciente será elegido como nodo primario, y cuando el resto de nodos secundarios sean notificados del nuevo Master el sistema volverá a estar disponible.

Por su parte, Apache Cassabdra es también un sistema de bases de datos NoSQL, de software libre. Está basado en una base datos de *wide-column* en el que se permite almacenar los datos en una red distribuida, pero a diferencia de MongoDB, se distribuye en una estructura sin maestros, resultando en tener múltiples puntos de fallo en lugar de un punto único.

En lo que respecta al teorema CAP, Apache Cassandra es una base de datos AP, lo que significa que proporciona disponibilidad y tolerancia a particiones, pero no puede asegurar la consistencia en todo momento. Puesto que al ser un sistema que no tiene un nodo que actúa como maestro, todos los nodos deben estar disponible en todo momento. Aun con esto, Apache Cassandra sí que es capaz de proveer consistencia finalmente, permitiendo que lo cliente escriban en cualquier nodo en cualquier momento y trabajando en corregir inconsistencia de la manera más rápida posible.

En lo relativo a la relación de Apache Cassandra con ACID, no es capaz de soportar transacciones que cumplan totalmente ACID, pero sí que ofrece atomicidad y transacciones duraderas y aisladas con eventuales fallos de consistencias.

5. Explica y compara el control de concurrencia optimística y pesimístic. Propón ejemplos de aplicación real.

El control de concurrencia pesimista es un enfoque para los algoritmos de control de concurrencia en el cual las transacciones son retrasadas cuando existe algún conflicto. En ese caso se bloquean los registros de la base de datos frente a accesos para actualizaciones y se permite a otros usuarios acceder únicamente con permisos de solo lectura, o esperar a que el registro pase a estar desbloqueado. De esta manera la programación de aplicaciones con un enfoque pesimista de control de concurrencia es más complicado y complejo por la posibilidad de existencia de abrazos mortales.

De esta manera cuando se aplica el enfoque pesimista las operaciones de validación son ejecutadas en primer lugar, y si se puede validar la consistencia con la compatibilidad del bloqueo, se realizan las operaciones de lectura, procesamiento y escritura. O lo que es lo mismo, se sigue el ciclo: Validar → Leer → Procesar → Escribir

Por su parte el enfoque optimista responde a un enfoque de los algoritmos de control concurrencia en base a la asunción de que los conflictos en las operaciones y las bases de datos ocurren de forma muy aislada, por lo cual es recomendable ejecutar las transacciones hasta su completitud y después revisar si existen conflictos, sin que existan comprobaciones durante la ejecución de las transacciones.
Por consiguiente, este enfoque no necesita de ningún mecanismo de bloqueo o retraso, puesto que como ya se ha expuesto, las transacciones son ejecutadas sin comprobación, permitiendo a las transacciones proceder de forma asíncrona y permitir la comprobación de conflictos al final del proceso.
Esto hace que durante la ejecución de un proceso con el enfoque optimista solo se ejecuten las operaciones de lectura y procesamiento antes de las de validación y escritura.

Un ejemplo real de utilización de enfoque optimista en el control de concurrencia podría ser el caso de Wikipedia, en el cual se permite que varias personas puedan editar al mismo tiempo una misma página, de forma que hasta que no se llegue al momento de la validación no se compruebe este hecho y se permita guardar únicamente a la primera persona que ha introducido una modificación.