Bienvenidos a una nueva presentación del bloque temático dedicado a bases de datos distribuidas, donde vamos a estudiar la relevancia del teorema CAP en el desarrollo de sistemas distribuidos, y cómo se relaciona con otros aspectos de gestión de dichos sistemas (entre otros, el modelo BASE).

Bases de datos distribuidas

- Gestión de réplicas
- Teorema CAP
- Modelo BASE

EIMT.UOC.EDU

La presentación se estructura en tres secciones. En primer lugar se explican las nociones básicas de replicación. Seguidamente, en la segunda sección, analizaremos el teorema CAP. Finalmente presentaremos el modelo BASE, su relación con el teorema CAP, y en qué se diferencia del modelo ACID.

Bases de datos distribuidas

- Gestión de réplicas
- Teorema CAP
- Modelo BASE

EIMT.UOC.EDU

Como ya sabemos las dos estrategias que se pueden utilizar en el diseño de bases de datos distribuidas son la fragmentación y la replicación. La fragmentación ya ha sido suficientemente tratada en presentaciones previas. A continuación vamos a complementar las nociones asociadas a la replicación.

Replicación

- La replicación consiste en almacenar todos o una parte de los fragmentos diseñados (y por lo tanto, los datos que éstos contienen) en más de un nodo de la BDD.
- La replicación permite maximizar la disponibilidad de los datos, así como mejorar el rendimiento de la BDD.
- El problema principal es que es necesario garantizar la consistencia de las réplicas.
- La consistencia se refiere a que todas las réplicas de unos mismos datos deben contener los mismos valores.

EIMT.UOC.EDU

La replicación consiste en almacenar todos o una parte de los fragmentos diseñados (y por lo tanto, los datos que éstos contienen) en más de un nodo de la base de datos distribuida.

La replicación maximiza la disponibilidad de los datos, es decir, la capacidad de responder a las peticiones que formulan las aplicaciones (y por lo tanto, los usuarios), especialmente en el caso que se produzcan averías. Recordemos que las situaciones de avería pueden ser de naturaleza diversa e incluyen: fallos en los dispositivos de almacenamiento, fallos de alguno de los nodos de la base de datos distribuida, el particionamiento del sistema debido a fallos en la red o la posibilidad de que se pierdan mensajes o que éstos no se entreguen a tiempo. Los mensajes se usan a efectos de coordinación entre los diferentes nodos que forman la base de datos distribuida.

La replicación también ayuda a mejorar el rendimiento de la base de datos, dado que permite acercar los datos allá donde se necesitan. Asimismo permite mejorar la eficiencia de las operaciones de consulta dado que es suficiente (en principio) con acceder a una de las réplicas para resolver dichas peticiones.

Evidentemente, no todo son ventajas. El principal problema es garantizar la consistencia de las diferentes réplicas (o copias) que de unos mismos datos (u objetos de mundo real) puedan existir. En este caso la consistencia se refiere al hecho que todas las réplicas de unos mismos datos deben contener los mismos valores.

Gestión de réplicas

- El mantenimiento de la consistencia de las réplicas (o gestión de réplicas) se puede realizar:
  - De manera síncrona o asíncrona
  - A través de una réplica destacada (réplica o copia primaria) o sin réplica destacada (todas las réplicas son iguales)

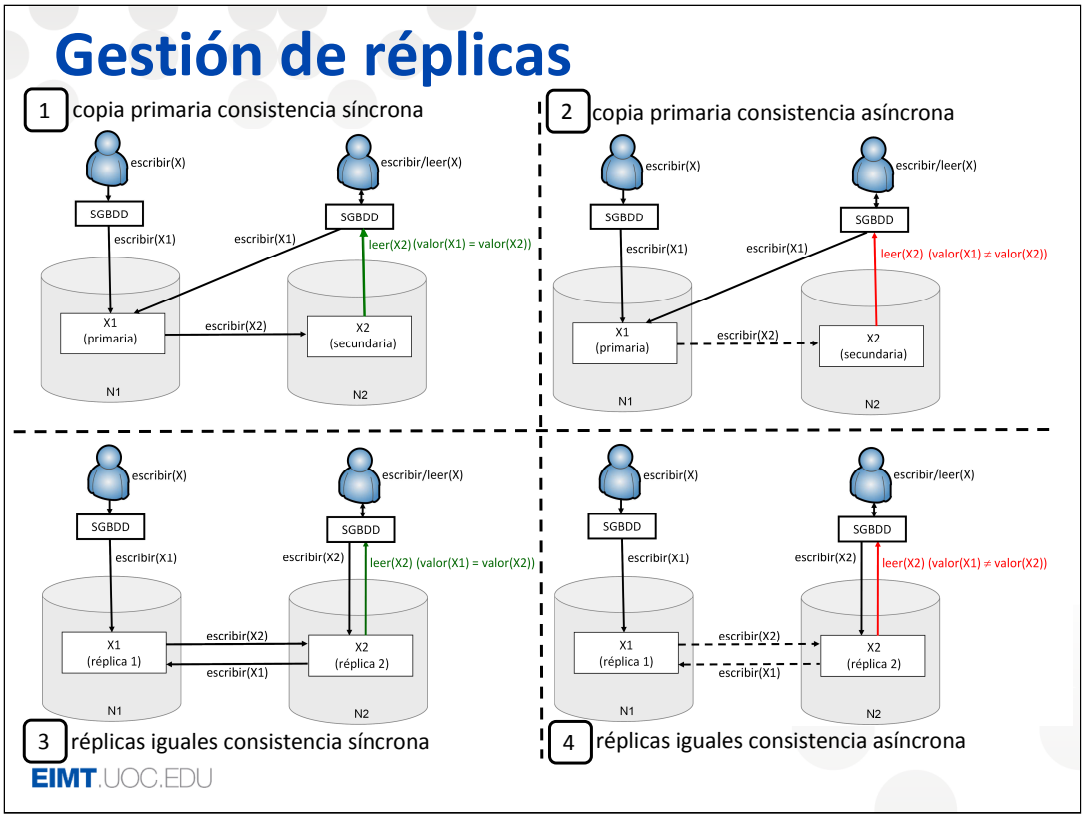
EIMT.UOC.EDU

Existen diferentes políticas (o mecanismos) para garantizar la consistencia de las réplicas. Antes de entrar en ellas, es necesario destacar que la existencia de réplicas debe ser transparente a los usuarios (y si es posible también para las aplicaciones). Esto implica que la responsabilidad de la gestión de las réplicas debe recaer en el sistema gestor de la base de datos distribuida.

Las políticas surgen de considerar dos dimensiones que son ortogonales entre sí.

La primera tiene que ver con el hecho de si la consistencia se mantiene de forma síncrona o asíncrona (es decir, si los cambios realizados sobre una réplica se propagan de forma inmediata o diferida al resto).

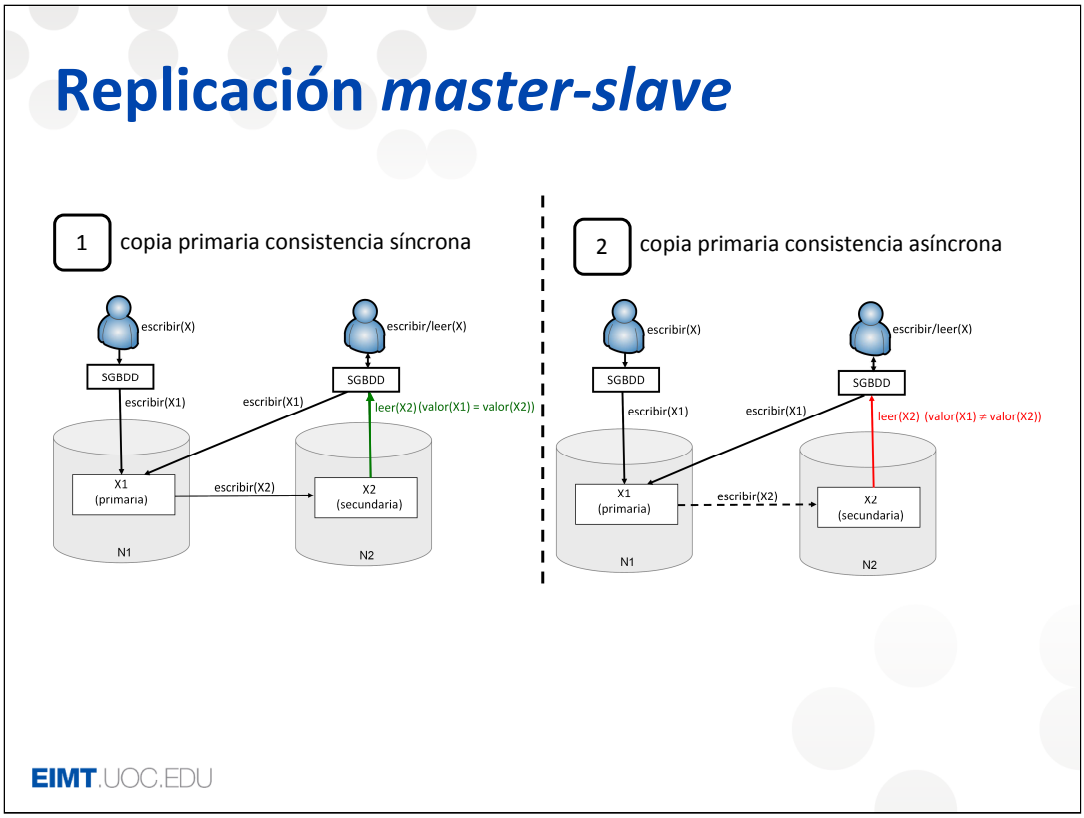
La segunda dimensión se relaciona con la existencia de diferentes tipos de réplicas (una copia primaria donde se realizan los cambios que luego serán propagados al resto) o no (todas las réplicas son iguales). Las políticas que consideran diferentes tipos de réplicas se conocen de forma genérica como *master-slave*, mientras que las que consideran todas las réplicas iguales se conocen bajo la denominación de replicación P2P (o replicación *multimaster*).



En la figura se muestra el resultado de combinar ambas dimensiones. En ella podemos ver un mismo conjunto de datos (llamado X) que está replicado en dos nodos de la base de datos distribuida (denotados como N1 y N2). La réplica que de X existe en el nodo N1 se ha identificado como X1, mientras que la que existe en N2 se ha denominado X2.

En los cuadrantes 1 y 2 (parte superior de la figura) se muestran las estrategias *master-slave.* Si nos fijamos, podemos ver que X1 es la réplica o copia primaria de X, mientras que X2 constituye la réplica o copia secundaria (o "esclava") de X. En la parte inferior de la figura (que se corresponde a las estrategias P2P, cuadrantes 3 y 4), podemos observar que no existe tal diferenciación (X1 y X2 son réplicas de X, no existe una denominación específica para cada una de ellas).

Por otro lado, en la parte izquierda de la figura (cuadrantes 1 y 3), se muestran las estrategias de consistencia síncrona. En ellas, las operaciones de escritura (o cambio) sobre todas las réplicas se ejecutan de manera atómica (es decir, o bien todas las operaciones tienen éxito y las réplicas pasan a contener los nuevos valores o bien ninguna de ellas los contiene, o sea, se mantienen los valores antiguos, reportándose el correspondiente mensaje de error a la aplicación. La parte derecha de la figura (cuadrantes 2 y 4), a su vez, muestra las estrategias de consistencia asíncrona. En ellas, una parte de las operaciones de cambio sobre unas mismas réplicas se puede retrasar en el tiempo. En otras palabras, las operaciones de escritura no se ejecutan de forma atómica sobre todas las réplicas como si de una sola operación se tratase, a pesar que el sistema confirme el éxito de la operación de escritura a la aplicación.



En el caso que se siga una política *master-slave* síncrona (véase la figura 1) tenemos una réplica primaria donde se ejecutan todas las operaciones de escritura. El resto de réplicas (denominadas secundarias) que puedan existir se mantienen de manera síncrona.

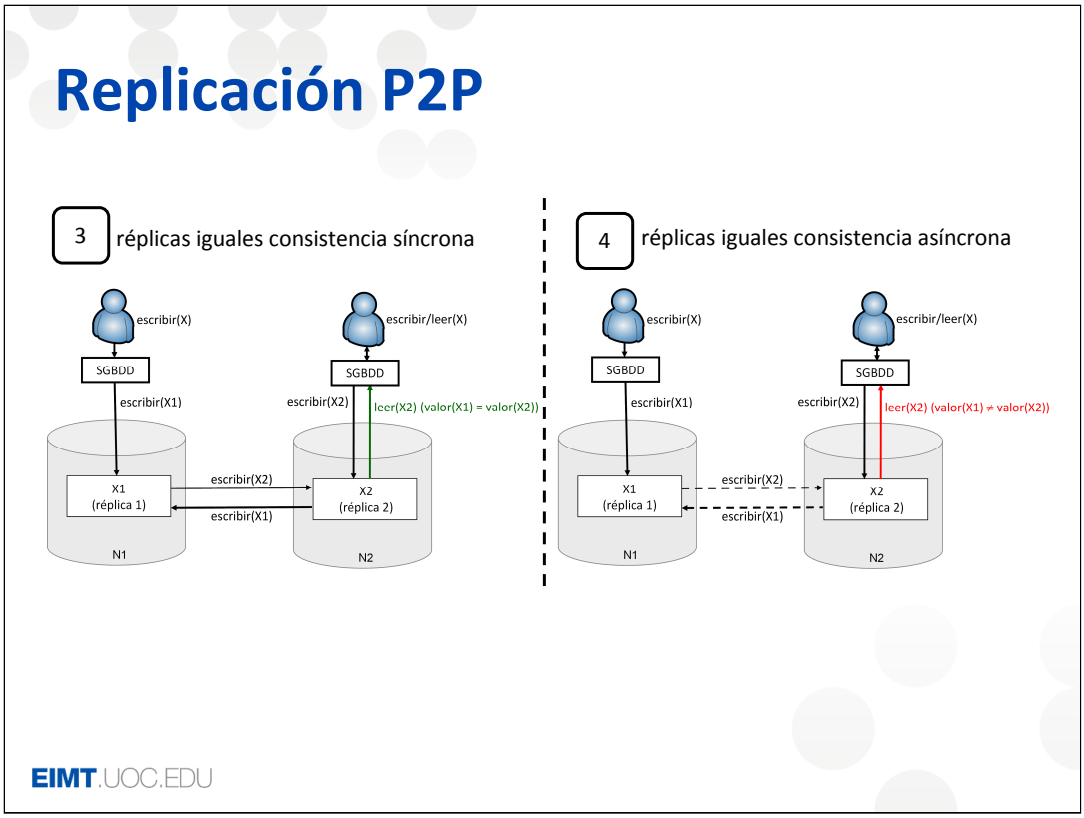
En esta estrategia no se pueden generar inconsistencias, dado que los cambios son inmediatamente propagados a todas las réplicas antes de dar por confirmada la operación. Si la propagación no es posible (por ejemplo, debido a que algún nodo que contiene una réplica secundaria no está accesible o disponible), la operación de escritura es rechazada por el sistema y la situación se reporta a la aplicación. Las operaciones de lectura se pueden servir a partir de cualquier réplica y siempre se obtendrá el valor más recientemente confirmado.

El principal problema de esta política es que el nodo que contiene la réplica primaria constituye un cuello de botella en el rendimiento, y éste se puede ver aún más degradado porque la consistencia de las réplicas se realiza de manera síncrona. Además, si el nodo que contiene la réplica primaria falla, la disponibilidad del sistema se verá limitada, dado que únicamente se podrán servir operaciones de lectura.

Por su parte, en el caso que se aplique una política *master-slave* asíncrona (figura 2), tenemos una réplica primaria donde se centralizan todas las operaciones de escritura. Los cambios realizados se propagan de manera asíncrona al resto de réplicas (las secundarias).

Respecto a la estrategia anterior se mejora el rendimiento, dado que la propagación de las operaciones de escritura a las réplicas secundarias puede esperar. Sin embargo, en el lapso de tiempo entre la escritura de la réplica primaria y su propagación al resto de réplicas, las operaciones de lectura pueden recuperar valores incorrectos (o sea, no consistentes), dado que las operaciones de lectura pueden consultar cualquier réplica. En consecuencia, por ejemplo, dos usuarios podrían obtener un valor diferente para unos mismos datos si sus peticiones son respondidas por nodos diferentes. Al igual que en la política *master-slave* síncrona, el nodo que contiene la réplica primaria sigue siendo un cuello de botella a efectos de rendimiento y disponibilidad.

A modo de conclusión destacar que las estrategias de replicación *master-slave* son especialmente adecuadas para aplicaciones que realizan muchas lecturas y pocas escrituras. Los problemas que se derivan del fallo del nodo que contiene la réplica primaria se pueden resolver eligiendo una nueva réplica (de entre las secundarias) como réplica primaria. El nodo que almacena dicha réplica pasaría a absorber las operaciones de escritura.



En la política P2P síncrona, tal y como muestra la figura 3, podemos ver que no existe diferenciación entre las diferentes réplicas que unos mismos datos puedan existir. En consecuencia, las operaciones de escritura pueden ser realizadas sobre cualquier réplica, al igual que las lecturas. Además podemos observar que todas las escrituras sobre unas mismas réplicas se realizan de manera síncrona (o atómica), hecho que implica que las lecturas siempre recuperarán valores consistentes (es decir, los cambios más recientemente confirmados). El problema principal es que el rendimiento se puede ver comprometido porque la consistencia de las réplicas se realiza de manera síncrona.

Finalmente, en la política P2P asíncrona (véase la figura 4), tampoco existen réplicas destacadas, y éstas se mantienen de forma asíncrona. En este caso, todos los cuellos de botella en el rendimiento y los problemas derivados de situaciones de fallo (disponibilidad limitada de la base de datos distribuida) desaparecen. El problema es que las réplicas pueden no ser consistentes. Por lo tanto, las operaciones de lectura pueden devolver valores obsoletos (en otras palabras, no se garantiza que se devuelva el valor más recientemente confirmado). Durante el proceso de propagación de las operaciones de escritura a las réplicas, se deben detectar y resolver dichas inconsistencias (es decir, una cierta réplica debe prevalecer sobre las otras).

A modo de conclusión, las políticas de replicación P2P resuelven los problemas de rendimiento y disponibilidad asociados a la replicación *master-slave*. Además, las aplicaciones tanto pueden ser intensivas en lecturas como en escrituras. A pesar de ello, cuando la replicación P2P es síncrona, el rendimiento puede disminuir (si acontecen situaciones de avería) a cotas similares a la replicación *master-slave*. Si deseamos evitar esto propagando los cambios de manera asíncrona, la consistencia puede peligrar. Leer valores incorrectos es transitorio, pero la posibilidad de perder cambios existe (como veremos más adelante), y puede resultar grave.

Los problemas se agravan cuando el número de réplicas de unos mismos datos crece, es decir, las políticas de gestión de réplicas que hemos visto se tienen que generalizar a un número cualquiera de réplicas (y no sólo dos como ha sido el caso de nuestros ejemplos). En el caso de replicación P2P, esto se concreta con una política intermedia entre la consistencia síncrona y asíncrona que hemos explicado, y que recibe el nombre de quórums. En esencia, la idea es configurar el número de réplicas que deben ser escritas de manera atómica (en lugar de todas) y el número de réplicas que deben ser recuperadas para resolver las operaciones de lectura. Entraremos en estas cuestiones con posterioridad.

Bases de datos distribuidas

- Gestión de réplicas
- Teorema CAP
- Modelo BASE

EIMT.UOC.EDU

El teorema CAP (también conocido como conjetura de Brewer) enuncia las propiedades deseables de un sistema distribuido, y cómo éstas se pueden ver comprometidas ante situaciones de avería (específicamente ante fallos de los nodos y/o fallos de la red de comunicaciones). Fue presentado en el año 2000 por Brewer en una conferencia invitada que tuvo lugar en el Symposium on Principles of Distributed Computing. Dos años más tarde, dos profesores del MIT (Gilbert y Lynch) probaron su veracidad.

El teorema CAP ha dado lugar a discusiones (incluso ha dado lugar a diversas interpretaciones), ha propiciado líneas de investigación y ha impulsado el desarrollo de implementaciones por parte de la industria. Desde la perspectiva de NoSQL, el teorema CAP captura la idea de que la consistencia puede no ser compatible con la gestión de datos altamente distribuidos. Es por ello que NoSQL aboga (entre otros aspectos) por la relajación de las propiedades ACID.

Teorema CAP

- **Consistencia** (*consistency*): los usuarios del sistema tienen que poder recuperar siempre los mismos datos en un mismo instante de tiempo.
- **Disponibilidad** *(availability):* las peticiones de servicio enviadas por los usuarios a un nodo que está disponible deben obtener su debida respuesta.
- **Tolerancia a particiones** (*tolerance to network partitions)*: el sistema debe proporcionar servicio a los usuarios a pesar de que se puedan producir situaciones de avería que causen que el sistema quede particionado en diferentes componentes.

> El teorema CAP enuncia que es imposible garantizar simultáneamente las tres características.

EIMT.UOC.EDU

El acrónimo CAP se refiere a tres propiedades: consistencia, disponibilidad y tolerancia a particiones. El teorema expresa que en un sistema distribuido es imposible garantizar simultáneamente las tres características.

La consistencia, disponibilidad y tolerancia a particiones tienen un significado específico en el contexto del teorema CAP, que pasamos a comentar a continuación.

En primer lugar, la noción de consistencia se relaciona con la existencia de réplicas, y se usa en el sentido de que todos los usuarios del sistema tienen que recuperar siempre los mismos datos (en concreto los más recientemente confirmados) en un mismo instante de tiempo, con independencia de cuál sea su punto de acceso al sistema distribuido.

Por su parte, la disponibilidad se refiere a que las peticiones de servicio enviadas por los usuarios a un nodo que está disponible (u operativo) deben obtener respuesta.

Finalmente, la tolerancia a particiones significa que el sistema debe proporcionar servicio a los usuarios a pesar que se puedan producir situaciones de avería (por ejemplo, fallos en la red de comunicaciones) que causen que el sistema quede particionado en diferentes componentes temporalmente aislados entre sí.

Teorema CAP

- En un sistema altamente distribuido se van a producir situaciones de avería que causen que el sistema se particione.
- En este escenario, garantizar P constituye una necesidad. En consecuencia, hay que decidir que se quiere priorizar:
  - La consistencia (C además de P)
  - La disponibilidad (A además de P)

EIMT.UOC.EDU

En un sistema altamente distribuido, con cientos o miles de nodos interconectados trabajando de forma colaborativa, y que almacena una gran cantidad de datos se van a producir situaciones de avería que pueden particionar el sistema. A modo de ejemplo, las posibilidades de fallo en las comunicaciones son altas (aunque se resuelvan rápidamente) cuando consideramos redes WAN (*wide area network* en inglés).

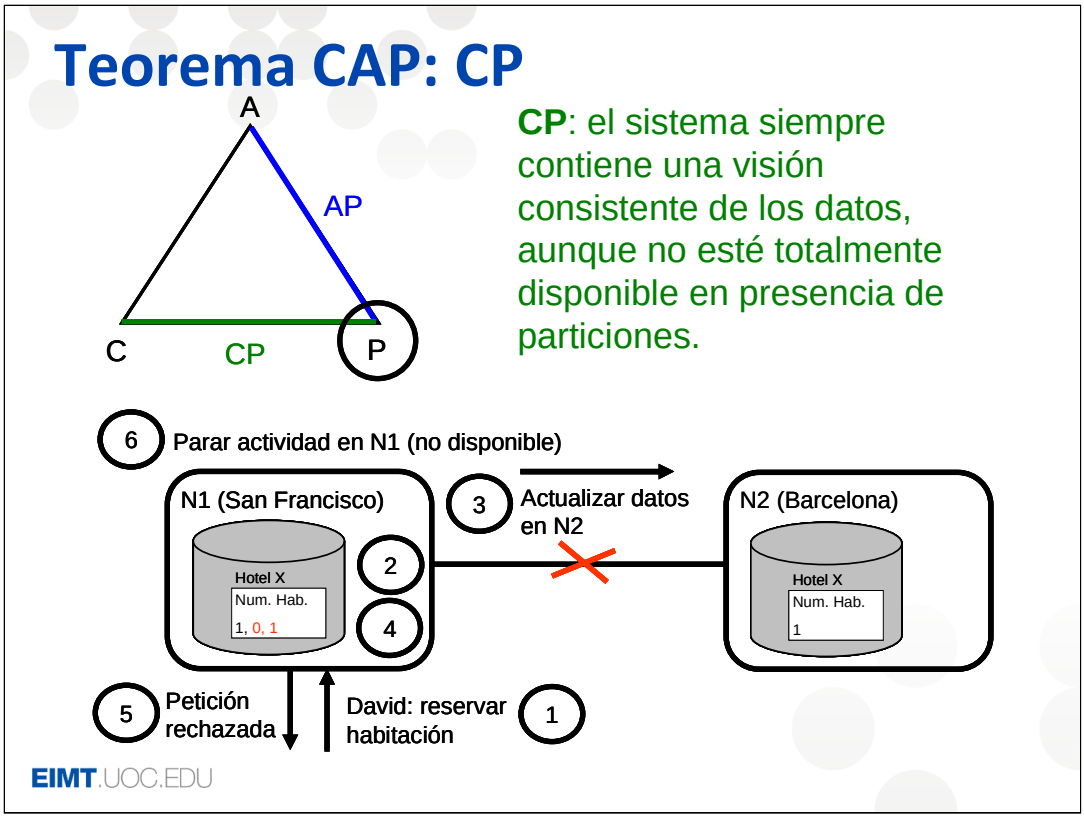
Un fallo en las comunicaciones causa el particionamiento del sistema. A pesar de ello, es necesario que el sistema pueda seguir suministrando servicio a los usuarios a unas cotas de calidad aceptable, es decir, se desea garantizar la propiedad P del teorema CAP. Los motivos pueden ser diversos, pero al menos hay uno bastante convincente: desde un punto de vista empresarial, no suministrar servicio, implica una pérdida de ingresos.

Por lo tanto, y teniendo en cuenta el teorema CAP, es necesario decidir que se desea priorizar: la consistencia o la disponibilidad.

Cuando se prioriza la consistencia (es decir, cuando trabajamos con un sistema CP), el sistema siempre puede mostrar una visión consistente de los datos, aunque no esté totalmente disponible en presencia de particiones.

En el caso que se priorice la disponibilidad (o sea, cuando el sistema es AP), el sistema siempre está disponible, aunque temporalmente puede mostrar datos inconsistentes cuando existen particiones.

A continuación vamos a ver, de manera intuitiva, las implicaciones del teorema CAP.

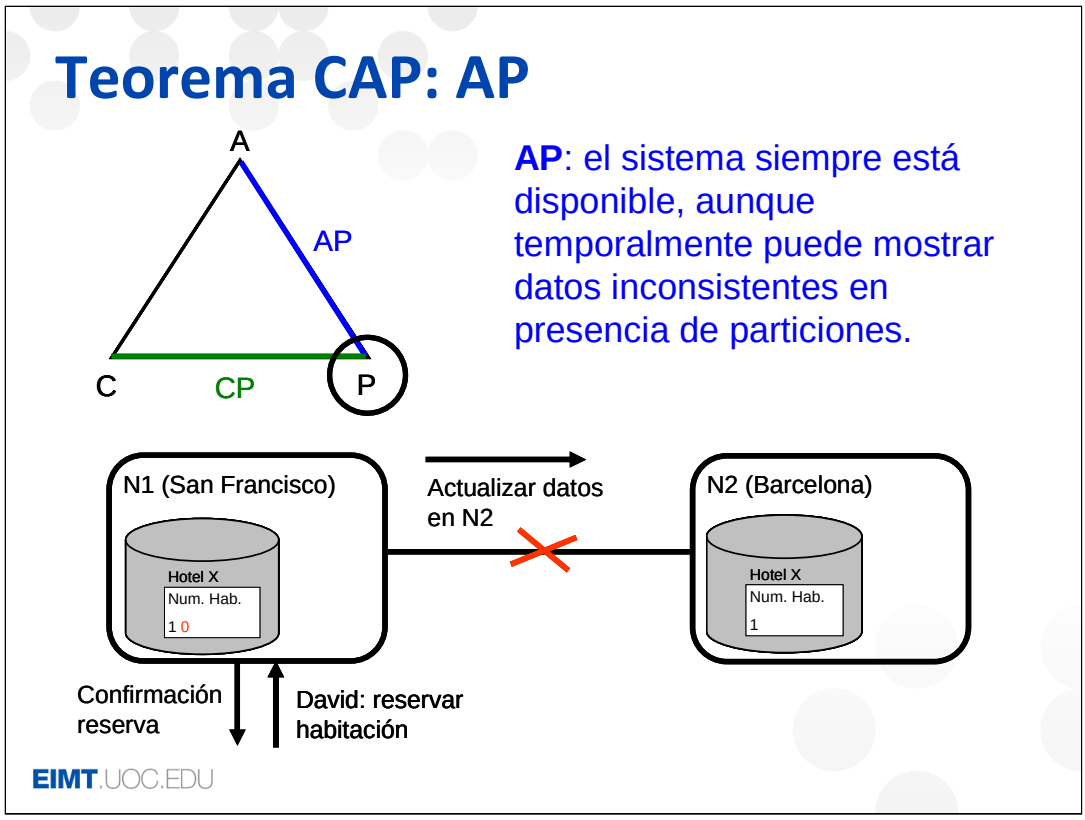


Un sistema CP siempre contiene una visión consistente de los datos aunque no esté totalmente disponible cuando se producen particiones del sistema, tal y como muestra nuestro ejemplo de reservas de habitaciones de hotel.

En él se muestran dos nodos (N1 y N2) que guardan datos de un mismo objeto (el hotel X). Podemos ver que se guarda el número de habitaciones disponibles (en concreto, queda una habitación disponible). Partimos de una situación inicial donde las réplicas son consistentes. Finalmente asumiremos que para garantizar la consistencia se usa replicación P2P síncrona.

Imaginemos que el nodo N1 recibe una petición de un usuario (David) que intenta reservar una habitación en el hotel X (en la figura se trata de la petición 1). Para procesar la petición, en primer lugar, el sistema comprobará que la reserva es factible (hay habitaciones libres) y pasará a modificar los datos disminuyendo en una unidad el número de habitaciones disponibles (es la operación número 2 que deja el número de habitaciones a cero). Antes de confirmar la reserva al usuario, y dado que la consistencia de las réplicas se mantiene de forma síncrona, N1 envía la petición de actualización del número de habitaciones libres al nodo N2 (operación número 3). En este punto, supongamos que se produce un fallo en las comunicaciones que causa que el sistema quede particionado. El nodo N1 está esperando la confirmación por parte de N2 de que su petición de actualización se ha procesado con éxito, pero no recibe respuesta (a pesar que el nodo N2 está disponible, no está accesible). Pasado un tiempo razonable (*timeout*), N1 cancelará la petición del usuario. Esto implica, en primer lugar, anular los resultados producidos (es la operación 4 que restaura el número de habitaciones disponibles a una habitación), y en segundo lugar, avisar al usuario de que su petición de reserva ha sido rechazada (posiblemente le sugerirá que lo intente un poco más tarde). A partir de aquí, el sistema puede optar por dejar de dar servicio en el nodo N1 (se trata de la operación número 6), es decir, el nodo deja de estar disponible. Esta última decisión es un tanto drástica, y se muestra para ilustrar que la propiedad de disponibilidad (la A del teorema CAP) se ve comprometida. De hecho, en un entorno real, N1 estaría parcialmente disponible, y procesaría cualquier tipo de petición que se pueda resolver accediendo únicamente al nodo N1.

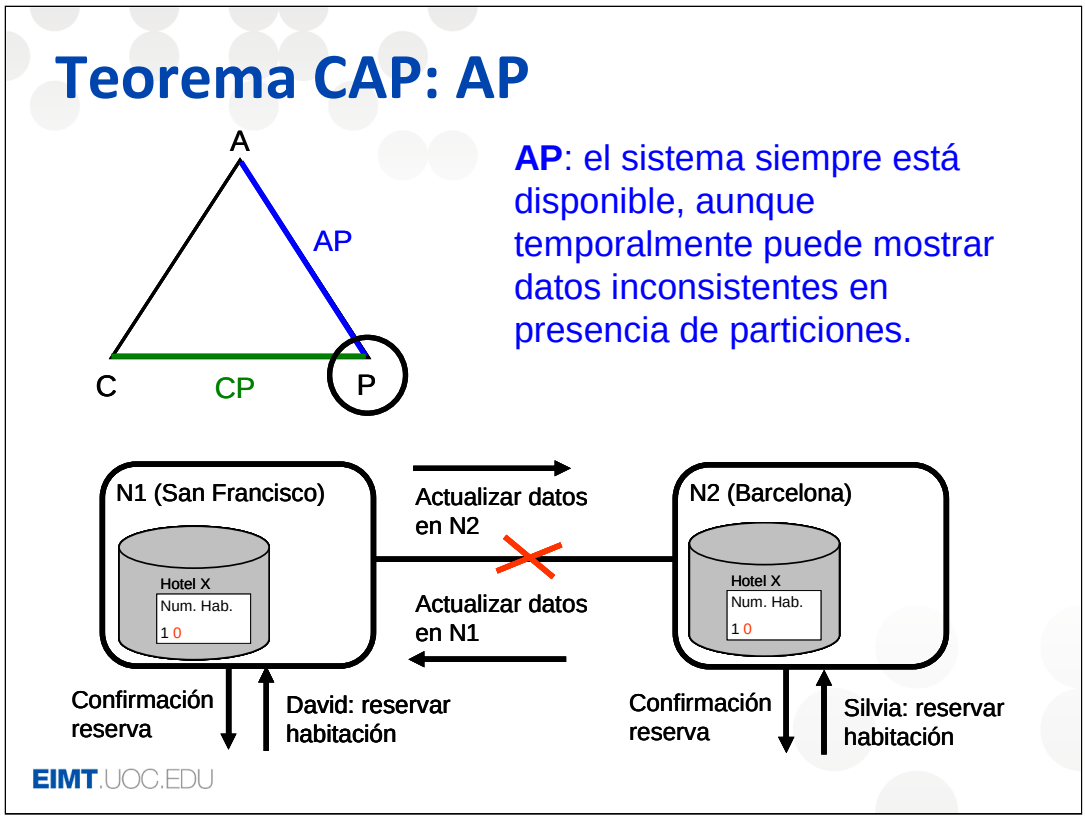
El ejemplo también puede dar la sensación que el sistema no proporciona servicio a los usuarios debido a su particionamiento (es como si sólo se cumpliese la propiedad C del teorema CAP). Esto es así porque el ejemplo sólo incluye dos nodos. En un escenario real existirían múltiples nodos, y éstos podrían quedar en una de las dos particiones. Las peticiones de servicio sobre los nodos que no necesiten acceder a nodos de la otra partición se podrían procesar con normalidad.



En el caso de un sistema AP, el sistema siempre está disponible, aunque puede mostrar temporalmente datos inconsistentes en presencia de particiones. Vamos a ver la situación tomando de nuevo el ejemplo de reservas de habitaciones de hotel.

Dado que las peticiones de servicio de los usuarios recibidas por un nodo que está operativo deben ser procesadas (es decir, se quiere garantizar la propiedad de disponibilidad, la A del teorema CAP) asumiremos que se aplica replicación P2P asíncrona.

El ejemplo muestra la petición de un usuario (de nuevo, David) que intenta reservar una habitación en el hotel X. La petición se procesa en el nodo N1. Dado que hay habitaciones libres, la reserva se efectúa y se actualizan los datos en la réplica que del hotel X existe en N1. Asimismo el usuario recibe la confirmación de que su reserva se ha realizado con éxito. En consecuencia, se verifica la propiedad de disponibilidad. Además, como la consistencia de las réplicas se mantiene de manera asíncrona, las operaciones previas se pueden ejecutar con anterioridad a la petición de actualización de los datos del hotel X en el nodo N2. Esta petición puede no llegar a N2, si se produce una partición del sistema. En este caso, la consistencia se ve comprometida (las réplicas no contienen los mismos valores, en el nodo N1 no quedan habitaciones libres, mientras que en N2 existe una habitación disponible). Se supone que esta situación es transitoria. Cuando se restablezca la comunicación se detectará el problema y el sistema garantizará que los valores de ambas réplicas acabarán convergiendo a unos mismos valores.



La situación descrita podría empeorar, si un usuario (en concreto Silvia) intenta reservar de forma concurrente una habitación en el hotel X, y esta petición se recibe en el nodo N2. Dado que la réplica no ha sido actualizada, Silvia recibe la confirmación de que su reserva se ha procesado con éxito. A pesar de que las réplicas contienen los mismos valores (no quedan habitaciones libres), éstos son incorrectos dado que se han aceptado (y confirmado) dos peticiones de reserva cuando únicamente quedaba una habitación libre. En resumen, se ha producido *overbooking*.

Cuando se reestablezca la comunicación, el sistema detectará que se ha vulnerado la consistencia. Ahora la situación es más grave, dado que una de las reservas tiene que ser descartada. En otras palabras, se van a perder cambios, probablemente los que correspondan a la última petición de reserva efectuada (es decir, la de Silvia). La responsabilidad de resolver la situación con el cliente recaerá en el hotel. Éste puede contar con habitaciones de clientes que finalmente no se presentarán, asignar una habitación que no ha puesto en oferta (en general, es frecuente que los hoteles guarden habitaciones para solucionar los problemas de *overbooking*), ofrecer una habitación de una categoría superior sin incrementar el precio etc. En el caso peor algún responsable del hotel contactará con el cliente para disculparse e informarle que, por culpa de los informáticos ☺, su reserva ha tenido que ser anulada.

Teorema CAP: quórums

- Son un mecanismo de replicación P2P.
- Se basan en el ajuste de tres parámetros:
  - N (factor de replicación): número de réplicas que existen de unos mismos datos
  - W: número de réplicas que deben ser escritas de forma atómica por una operación de escritura
  - R: número de réplicas que deben ser recuperadas para la resolución de una operación de lectura

EIMT.UOC.EDU

Los quórums son un mecanismo de replicación P2P que siguen un enfoque intermedio entre las políticas de replicación P2P síncrona y asíncrona que hemos explicado anteriormente.

Se basan en el ajuste de tres parámetros que pueden ser fijados por el administrador de la base de datos para cumplir los requerimientos de las aplicaciones. Frecuentemente también se permite que sean las propias aplicaciones quienes los configuren.

El primero es el factor de replicación N que indica el número de réplicas que de unos mismos datos existe en la base de datos distribuida. Cada una de las réplicas se almacena en un nodo. Por lo tanto, N también denota el número mínimo de nodos del sistema distribuido (podrían existir más dado que, además de replicación, podemos usar fragmentación como estrategia de distribución).

El segundo parámetro es W e identifica el número de réplicas que deben ser escritas de manera atómica (es decir, síncrona) para que el gestor de la base de datos confirme que la operación se ha ejecutado con éxito. El resto de réplicas se actualizarán de forma asíncrona. Fijémonos que los quórums derivan en la política de replicación P2P síncrona que hemos explicado cuando los valores de W y N coinciden. Si W se fija a 1 estaríamos en el extremo contrario, es decir, en una estrategia de replicación P2P asíncrona a efectos de escritura de réplicas.

Finalmente, el parámetro R constituye el número de réplicas que es necesario recuperar para que el gestor considere que la operación de lectura se ha ejecutado con éxito.

Dependiendo de la flexibilidad que ofrezca un sistema gestor de la base de datos basado en esta estrategia de replicación, tanto podremos trabajar en un sistema que sea de tipo CP o AP desde la perspectiva del teorema CAP.

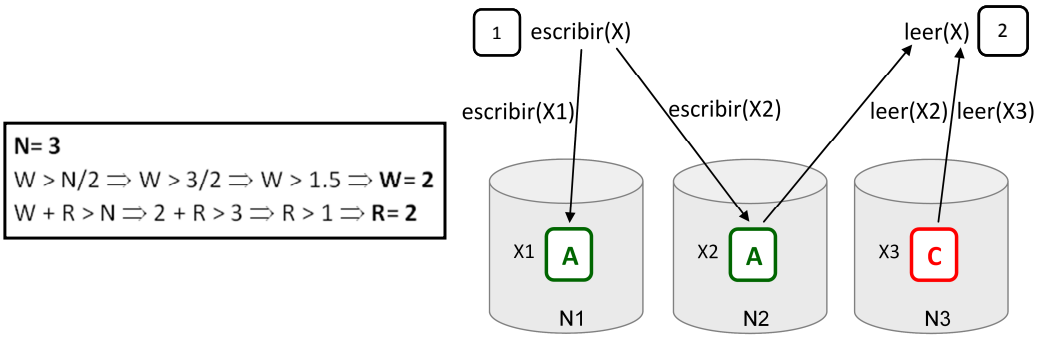
Teorema CAP: quórums

- Sistema CP (*strong consistency*):

  [1] W > N/2

  [2] W + R > N

- Ejemplo: N= 3, W= 2, R= 2



EIMT.UOC.EDU

Para garantizar que el sistema es CP a través de quórums se deben verificar las inecuaciones que se muestran en la transparencia.

Para preservar la consistencia se deben dar dos condiciones. La primera, tal y como muestra inecuación [1], es que es necesario escribir de forma atómica al menos la mitad más una de las réplicas que de unos mismos datos existen. Esto evita que se puedan perder cambios y asegura que al menos existe una réplica que contiene el valor correcto (es decir, el más recientemente confirmado).

La segunda condición (véase la inecuación [2]) garantiza que, a efectos de operaciones de lectura, siempre se recuperará, al menos, una réplica que contendrá el valor correcto. Esto es así porque entre los conjuntos de réplicas accedidos para escritura y lectura siempre existe intersección.

Para calcular los valores de los parámetros, en primer lugar, es necesario decidir el factor de replicación N, tal y como se muestra en la parte izquierda del ejemplo. A continuación se calcula W y por último R. En nuestro ejemplo, los valores que se han asignado a W y R son los mínimos que hacen que se cumplan las inecuaciones. En la parte derecha de la figura se muestra la ejecución de dos operaciones (una de escritura y otra de lectura) sobre unos mismos datos (denotados como X) para los que existen 3 réplicas (N=3) que se identifican como X1, X2 y X3. Partimos de una situación inicial donde todas las réplicas tienen los mismos valores (ese valor es C). En primer lugar se ejecuta la operación de escritura. El nuevo valor para la réplica es A. Atendiendo a W se deberán escribir de forma síncrona dos réplicas cualesquiera. Las réplicas elegidas han sido X1 y X2. La operación de lectura (que se ejecuta en segundo lugar) lee también dos réplicas. Una de ellas (en concreto X2) contiene el valor correcto que será el que finalmente se devolverá al usuario.

Para concluir destacar que la estrategia garantiza que los usuarios tengan una visión consistente de la base de datos (bajo su punto de vista, todas las réplicas contienen los mismos valores). Sin embargo, realmente, esto no es así tal y como muestra la figura. Este nivel de consistencia se conoce como *strong consistency* (o consistencia fuerte). En cualquier caso, todas las réplicas acabarán convergiendo a unos mismos valores, es decir, la réplica X3 acabará mostrando el valor correcto, dado que se actualiza de forma asíncrona.

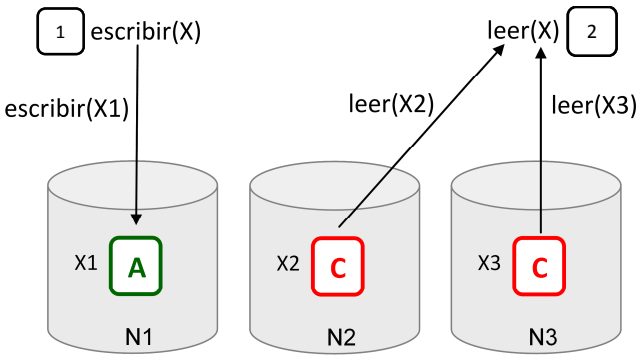
Teorema CAP: quórums

- Sistema AP (*eventual consistency*):

  [1] $W + R \leq N$

  Valores pequeños para W y R

- Ejemplo: N= 3, W= 1, R= 2



EIMT.UOC.EDU

Los quórums también se pueden aplicar en sistemas AP. En este caso se eligen valores pequeños para W y R. La relación entre W, R y N verifica la inecuación [1].

Como consecuencia, se leen algunas réplicas (en el ejemplo se leen 2) y también se escriben algunas réplicas de forma atómica (en nuestro ejemplo 1). El sistema no garantiza que una lectura devolverá el valor más recientemente confirmado. Además, en esta configuración, se pueden confirmar operaciones de escritura conflictivas en diferentes nodos. El tiempo transcurrido entre la confirmación de una operación de escritura y el fin de la propagación de dicha escritura a todas las réplicas disponibles se denomina ventana de inconsistencia.

Durante la ventana de inconsistencia, y a diferencia del caso anterior, las anomalías pueden ser visibles a los usuarios. Es más, en algún caso, se tendrán que resolver de forma explícita. Esto representa una complicación en la lógica de la aplicación.

En algún momento se sincronizarán todas las réplicas y el sistema gestor garantizará que converjan a unos mismos valores, realizando las reparaciones oportunas. El nivel de consistencia que se deriva de esta estrategia se denomina *eventual consistency* (consistencia final en el tiempo).

Recordemos que todos estos problemas se han ilustrado en nuestro ejemplo de reserva de habitaciones de hotel que presentaba *overbooking*.

Bases de datos distribuidas

- Gestión de réplicas
- Teorema CAP
- Modelo BASE

EIMT.UOC.EDU

A continuación vamos a presentar, en primer lugar, los principios del modelo BASE y sus diferencias con el modelo ACID. Seguidamente, veremos la aproximación que realizan las bases de datos NoSQL y las relacionales con respecto al teorema CAP.

Modelo BASE

- Los sistemas que garantizan AP, se basan en un modelo de transacciones diferente al modelo ACID, en concreto se trabaja con el denominado modelo BASE:
  - **Disponibilidad limitada** (*Basic Availability*)
  - **Estado flexible** *(Soft-state)*
  - **Consistencia final en el tiempo** (*Eventual consistency*)

EIMT.UOC.EDU

Los sistemas distribuidos de tipo AP se basan en un modelo transaccional que recibe el nombre de BASE. El acrónimo captura las características principales del modelo.

Primero, la noción de disponibilidad limitada: el hecho de que una parte del sistema no esté disponible, no significa que todo el sistema deje de funcionar. En segundo lugar la posibilidad de que los datos, temporalmente, puedan estar en un estado no consistente (estado flexible). Por último, la noción de consistencia final en el tiempo. En este último caso, se espera que, dado un período de tiempo suficiente sin cambios en unos mismos datos, todas las réplicas disponibles de esos datos converjan hacia unos mismos valores. En caso de conflictos que no puedan resolverse, se acepta que se puedan perder algunos datos.

En relación al término *eventual consistency*, un pequeño apunte. Aunque siempre (o casi siempre) lo podamos encontrar traducido al castellano como consistencia eventual, se trata de una traducción errónea, confusa y que se debería evitar. La palabra “eventual” existe en inglés y castellano, pero tienen significados totalmente diferentes. En inglés se refiere a algo que sucederá en un punto final del tiempo (quizá indefinido) tras bastantes esfuerzos, problemas etc. Por su parte, en castellano significa potencialidad, es decir, se relaciona con la posibilidad de que algo suceda o no suceda.

ACID, BASE y CAP

- Los modelos ACID y BASE representan dos enfoques para el mantenimiento de la consistencia situados en polos opuestos: ACID constituye un enfoque pesimista, mientras que BASE es optimista.
- La noción de consistencia (C) en CAP y ACID difieren:
  - En teorema CAP la C se refiere a la consistencia de las réplicas de unos mismos datos.
  - La consistencia en el modelo ACID es una noción que abarca múltiples elementos. Se conoce como consistencia estricta (*strict consistency*).

EIMT.UOC.EDU

Los modelos ACID y BASE representan dos enfoques para el mantenimiento de la consistencia situados en polos opuestos. El modelo ACID es pesimista, es decir, siempre previene (y por lo tanto, evita) que la base de datos pueda caer en un estado inconsistente. Por contra, el modelo BASE es optimista. Esto significa que la base de datos puede o no puede caer en un estado inconsistente, pero si eso pasa, se detecta y se resuelve.

De hecho, el acrónimo escogido (BASE) pretende reflejar la diferencia de enfoque escogido. Se trata de un juego de palabras, relacionado con el significado de los acrónimos cuando son vistos como palabras en el ámbito de la ciencia química (base y ácido).

Otro hecho destacable es que la noción de consistencia en el teorema CAP y en el modelo ACID son diferentes. En el caso del teorema CAP únicamente se refiere a la consistencia de las réplicas, mientras que en el caso del modelo ACID se relaciona con el nivel más exigente de consistencia (la consistencia estricta o *strict consistency* en inglés) que abarca diversas situaciones que pueden comprometer la calidad y corrección de los datos (réplicas, violación de restricciones de integridad, pérdida de escrituras por acceso concurrente etc.).

BD distribuidas y el teorema CAP: consideraciones finales

- Las BD NoSQL son, en general, sistemas CP o AP:
  - Sistemas CP: MongoDB, BigTable, Hbase, Redis
  - Sistemas AP: Riak, Dynamo, Cassandra, CouchDB
- En ocasiones pueden trabajar indistintamente en ambas modalidades.
- Las BD relacionales distribuidas son sistemas CA.

EIMT.UOC.EDU

Las bases de datos NoSQL (excluimos de la discusión las orientadas a grafos) son sistemas CP o AP. Entre las primeras tenemos, por ejemplo, a MongoDB y BigTable. En las segundas, entre otras, tenemos a Riak. Las BD NoSQL de tipo AP se basan en el modelo BASE.

Debemos interpretar esta clasificación como la modalidad de trabajo por defecto. Una de las características atribuidas a las bases de datos NoSQL es su versatilidad. Por ejemplo, Riak utiliza quórums para gestionar las réplicas. Dependiendo de la configuración elegida de los parámetros N, W y R podría comportarse como un sistema CP. Otro ejemplo podría ser MongoDB que utiliza una variante de la replicación *master-slave* que hemos visto y que se denomina *replica sets*. Todas las escrituras sobre una réplica se realizan sobre la copia primaria y se propagan de forma asíncrona y eficiente (según sus especificaciones) al resto de réplicas (las secundarias). Las lecturas también se realizan sobre la copia primaria. En caso de fallo del *master*, se elige otro de forma dinámica. A pesar de ello, MongoDB también permite que las aplicaciones puedan efectuar lecturas sobre las copias secundarias. En este caso no se garantiza que las operaciones de lectura devuelvan el valor más recientemente confirmado. En consecuencia estaría exhibiendo un comportamiento similar a un sistema AP.

Finalmente las bases de datos distribuidas relacionales son sistemas CA. Esto significa que la calidad del servicio se puede ver comprometida cuando el sistema se particiona (es la P del teorema CAP) . Por ello es aconsejable disponer de una red de comunicaciones fiable y realizar un diseño cuidadoso de la base de datos. Recordemos que el sistema gestor de la base de datos (al menos desde un punto de vista teórico) debe proveer consistencia estricta (*strict consistency*). Esto significa que la ejecución de las transacciones (que se subdividen en subtransacciones que se ejecutan en los diferentes nodos), vistas en global, deben producir los mismos resultados que una ejecución en serie (o en secuencia). En otras palabras, el comportamiento del sistema debe ser equivalente al de una base de datos centralizada (como si no se hubiese aplicado ni fragmentación ni replicación). Una manera de conseguirlo (a efectos de los contenidos explicados en esta presentación) es aplicar estrategias de replicación síncrona que se incluyen en el protocolo de confirmación en dos fases. Dicho protocolo, además de asegurar la atomicidad y definitividad de las transacciones distribuidas, también garantiza que las modificaciones efectuadas sobre datos replicados se aplicarán en todas las réplicas antes de dar por finalizada la transacción.

Bases de datos distribuidas

- Gestión de réplicas
- Teorema CAP
- Modelo BASE

EIMT.UOC.EDU

En esta presentación hemos analizado la relevancia del teorema CAP desde la perspectiva de bases de datos distribuidas y de los sistemas que las gestionan.

El teorema CAP establece que cuando el sistema particiona (debido a una avería en la red de comunicaciones) se tiene que decidir entre reducir la disponibilidad o la consistencia. La probabilidad de que esto suceda depende de diversos detalles de la implementación del sistema, por ejemplo: ¿se trata de un sistema distribuido en una red WAN o en un clúster local? ¿Cuál es la calidad del hardware que se utilizará? ¿Qué procesos existirán para asegurar una gestión correcta de la red?

En el caso que la probabilidad de que el sistema se particione sea una realidad: ¿Qué impacto negativo puede tener en la calidad del servicio percibida por los usuarios? ¿Qué nivel de consistencia requieren las aplicaciones? ¿Y de disponibilidad? ¿Cuál es el nivel de redundancia necesario?

Las respuestas a las preguntas previas se deben de tener en cuenta en la toma de decisiones sobre qué base de datos elegir.

Referencias

D. J. Abadi (2012). "Consistency Tradeoffs in Modern Distributed Database System Design", *Computer* 45(2), pp 37-42. (http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6127847&tag=1)

A. Abelló (2012). Transaction Models and Concurrency Control. Material docente UOC, asignatura Arquitectura de bases de datos.

P.A. Bernstein & E. Newcomer (2009). *Principles of Transaction Processing*. 2nd edition. Morgan Kaufmann.

E. A. Brewer (2000). Towards Robust Distributed Systems, Keynote speech in the "Symposium on Principles of Distributed Computing (PODC)". (http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf).

E. A. Brewer (2012). "CAP Twelve Years Later: How the "Rules" Have Changed", *Computer* 45(2), pp 23-29. (http://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed)

U. Friedrichsen (2014). Self-healing Data: An Introduction to Consistency models, quorums and CRDTs. (http://www.slideshare.net/ufried/self-healing-data)

EIMT.UOC.EDU

Esperamos que hayáis disfrutado y aprendido con este vídeo. A continuación encontraréis algunas referencias que os permitirán profundizar más en los conceptos que hemos tratado.

Que tengáis un buen día.

Referencias

S. Gilbert & N. Lynch (2002). “Brewer’s Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web”, *ACM SIGACT News* 33(2), pp 51-59. (http://lpd.epfl.ch/sgilbert/pubs/BrewersConjecture-SigAct.pdf)

S. Gilbert & N. Lynch (2012). “Perspectives on the CAP Theorem”, *Computer* 45(2), pp 30-36. (http://groups.csail.mit.edu/tds/papers/Gilbert/Brewer2.pdf)

L. Liu & M.T. Özsu (Eds.) (2009). *Encyclopedia of Database Systems*. Springer.

M.T. Özsu & P. Valduriez (2011). *Principles of Distributed Systems*. 3rd edition. Springer.

D. Pritchett (2008). “BASE: an ACID Alternative”, *Queue ACM* 6(3), pp 48-55. (http://queue.acm.org/detail.cfm?id=1394128)

P.J. Sadalage & M. Fowler. (2013). *NoSQL Distilled. A brief Guide to the Emerging World of Polyglot Persistence*, Pearson Education.

W. Vogel (2009). “Eventually Consistent”, *Communications of the ACM* 52(1), pp 40-44. (http://dl.acm.org/citation.cfm?id=1435432)