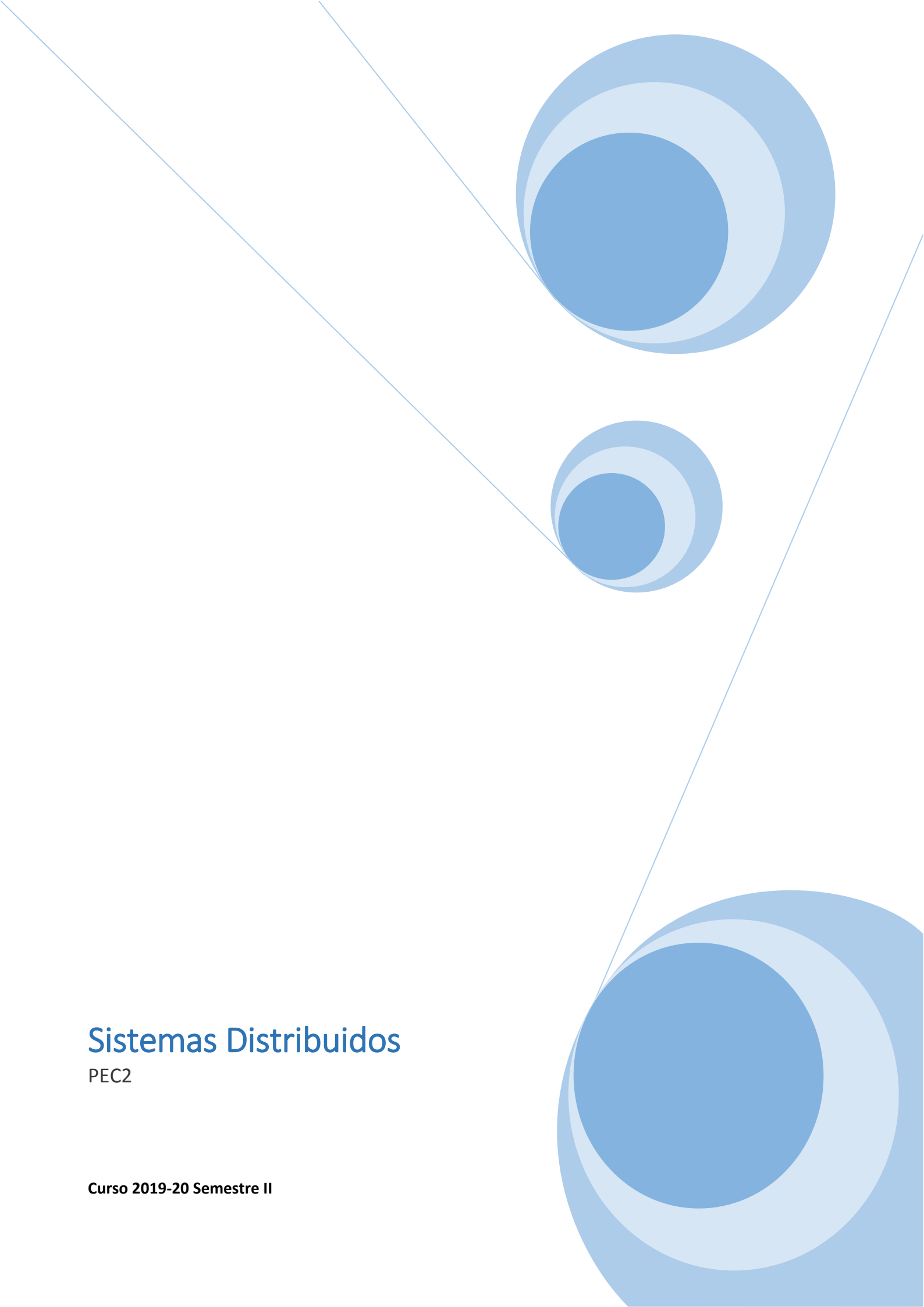


# Sistemas Distribuidos

PEC2

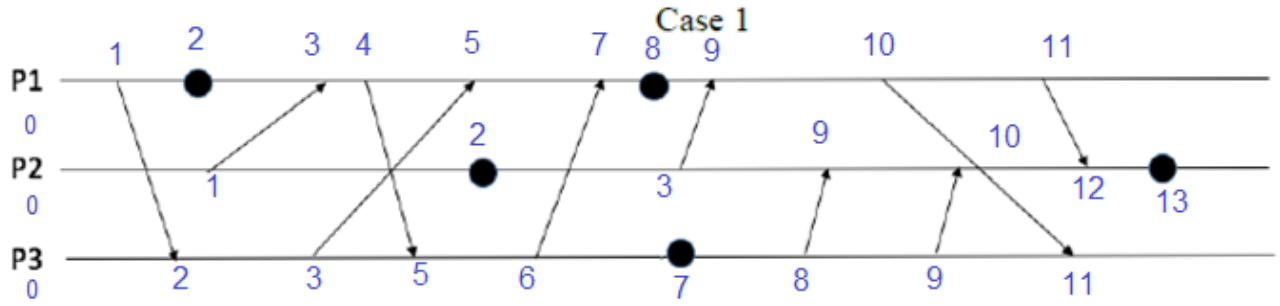
**Curso 2019-20 Semestre II**

# Índice

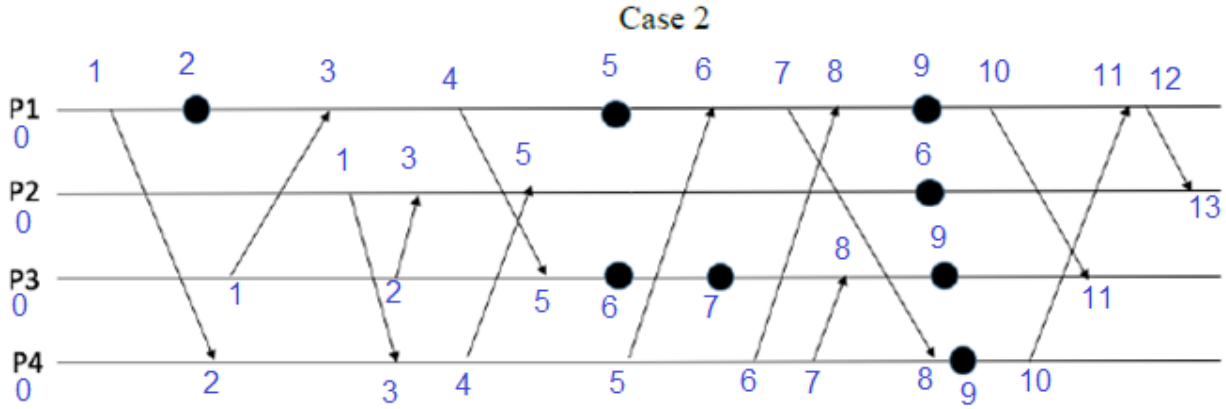
# Pregunta 1

1. In the next scenarios, where both, arrows and dots, indicate progress of the clock in each process, answer these questions:

a) With Lamport clocks. All clocks start at zero. Label both diagrams (case 1 and case 2) with the Lamport clock values. Show the final state of the clocks.

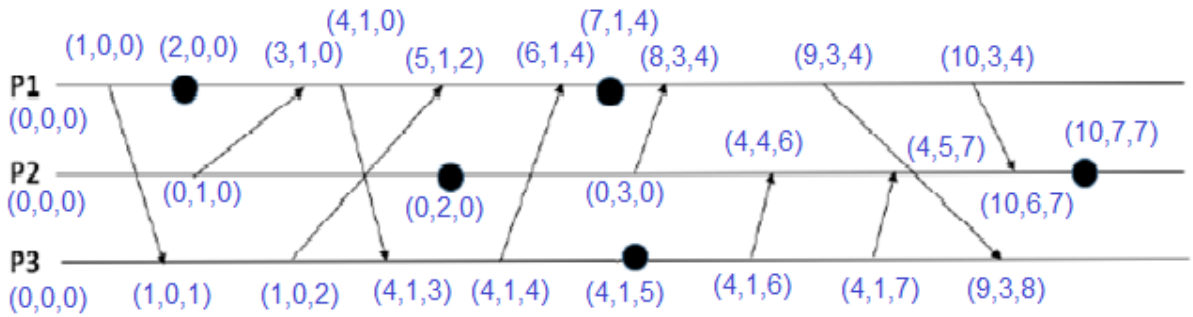



**Caso 1:** Estado final de los relojes → $\mathbf{C(P_1) = 11, C(P_2) = 13, C(P_3) = 11}$

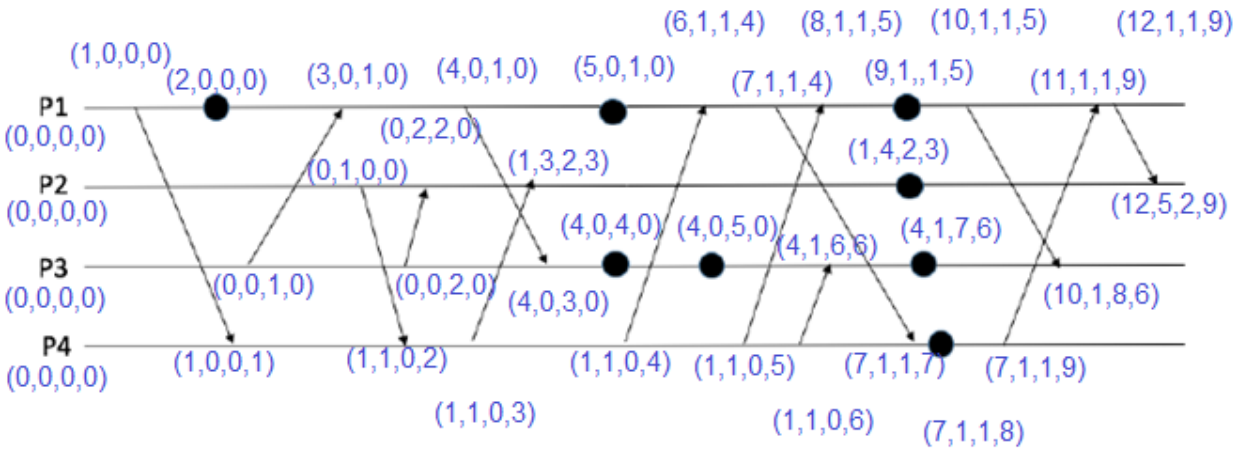



**Caso 2:** Estado final de los relojes → $\mathbf{C(P_1) = 12, C(P_2) = 13, C(P_3) = 11, C(P_4) = 10}$

b) With vector clocks. All clocks start at (0,0,0,0). Label both diagrams (case 1 and case 2) with the vector clock values. Show the final state of the clocks.




**Caso 1:** Estado final de los relojes → $\mathbf{V(P_1) = (10, 3, 4), V(P_2) = (10, 7, 7), V(P_3) = (9, 3, 8)}$




**Caso 2:** Estado final de los relojes → **$V(P_1)$ = (12, 1, 1, 9), $V(P_2)$ = (12, 5, 2, 9), $(P_3)$ = (10, 1, 8, 6), $V(P_4)$ = (7, 1, 1, 9)**

c) Using the happened before relationship, explain what can be achieved with vector clocks that is impossible with Lamport clocks.

Ambos relojes se pueden categorizar en dos grupos: Los **relojes lógicos** (*Lamport*) y **relojes vectoriales.**

Los **relojes de *Lamport*** muestran el orden de los procesos pero a partir de su esquema, no es posible conocer el momento en que sucedieron. Por lo tanto, se puede estar seguro que un suceso se produce en el tiempo antes que otro. Este hecho se representa con el símbolo flecha "→".

Por ejemplo a → b representa que el suceso "a" se ha producido antes que el suceso "b". Esta relación es transitiva. Si a → b y b → c entonces se puede afirmar que a → c. Cuando no se conoce cual se los eventos sucedió antes, se dice que son **concurrentes**, a || b.

Si "a" ocurre antes que "b", el contador del evento "a" es seguro menor que el contador del elemento "b", C(a) < C(b). Sin embargo, no ocurre lo mismo al revés. Si C(a) < C(b) **no se puede afirmar** si "a" es menor que "b" o si "a" y "b" son concurrentes.

Lo **relojes vectoriales** por el contrario, permiten conocer si un evento **sucede antes que otro** o si por el contrario, ambos son concurrentes. Ya que cada vector contiene N (el número de nodos) marcas temporales.

Por lo tanto si un evento "a" ocurre antes que un evento "b", a → b, entonces el reloj del evento "a" es menor que el del evento "b", V(a) < V(b). Y al contrario de los relojes de *Lamport*, si el reloj del evento "a" es menor que el del evento "b", se puede afirmar que "a" ocurre antes que "b".

Si no se da que V(a) < V(b) y tampoco se da que V(b) < V(a), entonces los eventos "a" y "b" son concurrentes.

En conclusión, con los relojes vectoriales es posible trazar el orden que sucedieron todos los eventos del proceso, a costa de un mayor consumo de recursos en comparación a los relojes de *Lamport*.

# Pregunta 2

Briefly explain 4 different algorithms for mutual exclusion. Discuss their scalability and fault tolerance.

La exclusión mutual se encarga de que los procesos sean capaces de **compartir** los recursos del sistema distribuido de manera organizada. Se consigue dando acceso a la denominada **sección crítica,** donde conviven los **recursos** y los **algoritmos** que regulan el control de los mismos.

**Algoritmo del servidor central**

Tiene un carácter centralizado ya que uno de los dos nodos es elegido como líder, el cual, se encarga de administrar el acceso a la sección crítica. El líder conoce en todo momento el estado de la sección crítica, mientras el resto de nodos pueden estar en varios estados: haciendo uso de la sección crítica, -a la espera de que sea liberada para usarla- o no interesado en acceder. Para realizar las peticiones, todos los nodos envían mensajes al líder de manera colaborativa y este, almacena las peticiones en su cola de peticiones donde se escoge la solicitud más antigua.

Para la concesión de la sección crítica, a pesar de no estar en uso, es necesario el envío de dos mensajes (la petición y la concesión). Este hecho, produce retrasos en el proceso de petición. La liberación de la sección crítica implica otro mensaje. Si se supone que el paso de mensajes es asíncrono, no debería producir retrasos en el proceso.

Si el servidor recibe muchas solicitudes de concesión, y no es capaz de gestionarlas de manera ágil, puede convertirse en un cuello de botella para el rendimiento del sistema. La sincronización puede sufrir retrasos al aumentar el tiempo que transcurre con un mensaje de ida y otro de vuelta.

Por ejemplo, un mensaje de liberación hacia el servidor, seguido de otro mensaje de concesión al siguiente proceso que va a tomar la sección crítica. Por lo tanto, contra mayor sea el número de host realizando peticiones al servidor, más retrasos se pueden producir.

- **Escalabilidad.** Consume muy poco ancho de banda y el envío de mensajes para cada acceso es solo de 3. Pueden producirse cuellos de botella.
- **Tolerancia a fallos.** Puede tolerar los fallos de los nodos, pero si falla el servidor se cae el sistema completo.

**Algoritmo basado en anillo**

Es un algoritmo sencillo, donde los procesos se unen formando un ciclo cerrado. Se consigue la exclusión a través de un mensaje testigo que se envía de un proceso a otro en un único sentido.
Cada nodo tiene un identificador y el que tenga el identificador más alto tendrá el papel de coordinador. Por defecto, cada nodo se cataloga como no participante. Si quiere comenzar una elección, al recibir el mensaje de elección compara el identificador del mensaje con el suyo. Si el recibido es mayor, reenvía el mensaje al vecino, si por el contrario, es menor, sustituye el identificador por el suyo y reenvía el mensaje. Si el mensaje contiene su identificador, es que es el mayor de todos y toma el papel de coordinador.

Cumple los requisitos de seguridad y concesión de peticiones de la exclusión mutua y en principio es robusto a fallos. En el peor de los casos, puede que el mensaje de elección recorra todo el circuito cuando el vecino anterior tenga el identificador más alto. Luego habrá que hacer otro circuito para anunciar la elección y un último recorrido para tomar los mensajes de los nodos. Por lo tanto, se requieren 3N -1 mensajes para la elección de coordinador. En este caso también, contra más nodos existan, más mensajes hay que enviar y más tiempo tardará la elección del coordinador.

- **Escalabilidad.** Uso continuo del ancho de banda por los nodos que generan x * N (nodos) mensajes. La sincronización necesita N -1 mensajes.
- **Tolerancia a fallos.** Un fallo de un nodo rompe el anillo.

**Algoritmo que usa multidifusión y relojes lógicos**

Se basa en que cuando un proceso necesita acceder a una sección crítica, envía un mensaje (que contiene el un *id* y un *timestamp*) por multidifusión y solamente entra cuando los demás hayan contestado a ese mensaje. Está diseñado para cumplir todas las condiciones de la exclusión mutua.

Los procesos cuentan con identificadores y una variable "estado" que puede ser LIBERADA, BUSCADA o TOMADA. Si un proceso quiere acceder y los demás están en estado LIBERADA, obtendrá el acceso de manera inmediata. Si algún proceso se encuentra en estado TOMADA, no contestará hasta estar LIBERADA, por lo tanto el proceso solicitante tendrá que esperar.

Si dos procesos lo solicitan a la vez, accederá el que tenga la marca temporal más baja de los relojes de *Lamport*. Conseguir la entrada en este logaritmo supone 2(N -1) mensajes. En caso de existir un hardware encargado de la multidifusión, solo sería necesario un mensaje para la petición, generando N mensajes. Esto produce un mayor consumo en recursos, como el ancho de banda.

La ventaja de este algoritmo es que su retraso de sincronización es de solamente el tiempo de envío de un mensaje, mientras que los anteriores algoritmos necesitan envíos de ida y vuelta. Este protocolo sufrió mejoras para requerir N mensajes en el peor de los casos.

- **Escalabilidad.** Para acceso a la zona crítica se necesitan 2(N -1) mensajes. Puede sufrir retrasos de 2N -2 mensajes. La sincronización se realiza en un único tiempo.
- **Tolerancia a fallos.** Si un nodo falla se reconfigura el sistema para que se mantenga la solicitud de peticiones.

**Algoritmo Bully**

La elección de un líder simplifica los algoritmos y reduce el número de mensajes. Cuando un líder falla o deja de estar activo se debe asignar este papel al host que tenga el identificador más alto.

Cuando un nodo quiere realizar una elección, envía un mensaje "ELECCIÓN" a los nodos con un identificador más alto que el suyo, si estos le contestan con un OK, el nodo que envío el mensaje deja de participar.

Esta secuencia se repite hasta que quedan los dos nodos con lo identificadores más altos y el mayor le envía su OK al segundo mayor. Este nodo asume el papel de coordinador el cual, envía un mensaje "COORDINADOR" al resto de nodos para que sean informados.

Es imposible que dos procesos decidan ser coordinador y la entrega de mensajes es fiable pero, no se garantiza la seguridad. El peor de los casos se dará cuando sea el nodo con el identificador más bajo el primero en detectar el fallo del coordinador, en el que serán N -1 los host que iniciarán el proceso de elección.

- **Escalabilidad.** En el peor de los casos se necesitan O($N^2$) cuando el proceso con el ID más bajo detecta el fallo del coordinador.
- **Tolerancia a fallos.** Si falla el coordinador y se recupera al mismo tiempo que otro nodo asume el puesto, se pueden enviar mensajes COORDINADOR con diferentes ID´s. El sistema no es asíncrono, por lo que los tiempos de espera pueden variar bastante.

## Pregunta 3

Describe the Byzantine generals' problem. When is it impossible to detect a failure? Imagine two scenarios with different number of processes that a) work and detect failures and b) cannot detect failures (one scenario for each case). Discuss the efficiency of the algorithm.

Su nombre se basa en la historia de un general que debe tomar la decisión de retirarse o atacar y posteriormente, comunicar su decisión a sus lugartenientes. Se sabe que un número desconocido de los implicados pueden ser traidores -incluido el General-. No se puede confiar en que los traidores comuniquen correctamente las órdenes, o lo que es peor, pueden alterar intencionadamente los mensajes para desestabilizar el proceso.

En la vida real, los generales son los procesos que originan los mensajes y las solicitudes enviadas a otros procesos son los mensajes. Los generales y tenientes traidores son procesos defectuosos y los generales y tenientes leales son procesos correctos. La orden de retirada o ataque es un mensaje con un solo *bit*, cero o uno. Al aplicar el problema del **General Bizantino,** la solución debe pasar tres pruebas: terminación, acuerdo y validez.

Se debe garantizar que todos los procesos correctos lleguen a un consenso con respecto el valor de la solicitud pedida.
Si el proceso raíz es correcto, los demás procesos tienen que decidir sobre el valor original dado por el proceso fuente. Hay que tener en cuenta que si el proceso de origen es defectuoso, todos los demás procesos aún deben acordar el mismo valor. Por lo tanto, aunque el general sea desestabilizador, todos los lugartenientes tendrán que tomar una decisión común y unánime.

Esta situación no es fácil de solucionar. Por ejemplo, si el proceso fuente es el único defectuoso y dice a la mitad de los procesos que el valor de su petición es cero y a la otra mitad que su valor es uno. Al recibir el mensaje, el resto de procesos deben acordar un valor decidido entre todos. Los procesos podrían preguntarse entre sí para comparar el valor recibido del proceso fuente.

El algoritmo de un proceso que ha recibido el mensaje con cero, debe tomar una decisión y comprueba que uno de los otros procesos indica que el valor correcto es uno. Se produce un **conflicto**. En este momento, el proceso tiene dos opciones: Asumir que el proceso de origen es defectuoso -ya que ha entregado valores diferentes a procesos diferentes- o que el proceso consultado es defectuoso y miente sobre el valor que recibió del proceso de origen. Parece imposible tomar una decisión final sobre quién es el traidor. Y de hecho, así lo es en algunos casos.

Se puede observar que independientemente del número de procesos que participen, un proceso fuente inestable junto a un colaborador, puede producir que la mitad de los procesos elijan atacar, mientras que la mitad de los procesos opten por retirarse, lo que lleva a una **máxima confusión**.

El algoritmo de *Lamport*, *Pease* y *Shostak* proporciona una solución sencilla al problema. Si se dispone de n procesos, con m procesos defectuosos, postula que se puede detectar y solucionar el problema siempre que se cumpla la siguiente condición: **n > 3m**. El algoritmo de *Lamport* tiene dos etapas. En la primera, los procesos iteran a través de m + 1 rondas de envío y recepción de mensajes. En la segunda etapa, cada proceso toma toda la información que se le ha dado y la usa para tomar una decisión.

Si se toma como referencia la condición n > 3m , se puede crear los siguientes escenarios.

1. **Detecta fallas.**
    - Número de procesos (n = 10)
    - Número de procesos defectuosos (m = 3)
    - $10 > 3 * 3 \rightarrow 10 > 9$

    Por lo tanto, en un sistema con 10 procesos y 3 de ellos defectuosos, sería capaz de detectar los fallos.

2. **No detecta fallas**
    - Número de procesos (n = 14)
    - Número de procesos defectuosos (m = 5)
    - $14 > 3 * 5 \rightarrow 14 < 15$

    Por lo tanto, en un sistema con 14 procesos y 5 de ellos defectuosos, no sería capaz de detectar los fallos.

En el caso general ($f \geq 1$ ), el algoritmo de *Lamport* para el caso de mensajes no firmados realiza f + l rondas. En cada ronda, un proceso envía a un grupo de los procesos los valores que ha recibido en la ronda previa. El coste del algoritmo es muy elevado, ya que implica enviar O(Nf + 1) mensajes.

La complejidad y el coste de la solución obligan que sea requiera cuando la amenaza sea grande. Si el origen de esta amenaza es un fallo en el hardware, entonces la posibilidad de conducta arbitraria es baja. Si el origen de la amenaza son usuarios malintencionados y un sistema quisiera hacerles frente, se utilizaría otras técnicas como firmas digitales.

# Pregunta 4

Compare the indicated concepts in the following cases:

## a) Database transactions: ACID and BASE

ACID y BASE son los modelos más habituales de consistencias de datos. Dependiendo de las necesitadas del sistema, uno de adaptará mejor que el otro.

**ACID.** Formado por 4 características que garantizan que las transacciones sobre las BBDD se realicen de forma confiable.
***Atomic*** (A) se basa en el todo o nada. Si falla una parte de la transacción, fallan las demás operaciones de la transacción y los cambios no se producen en la BBDD.
**Consistency** (C). Cualquier interacción sobre la BBDD la mantendrá siempre en un estado válido bajo las reglas establecidas.
**Isolation** (I). La ejecución concurrente de las transacciones debe producir el mismo estado en el sistema que si se ejecutaran de manera independiente y seguidas las mismas transacciones.
**Durability** (D). Una vez que se produce una transacción, esta será persistente, incluso ante cualquier tipo de fallas, como pérdida de alimentación errores o caídas del sistema.

**BASE. Basic Availability** (BA). El sistema debe estar en funcionamiento de manera ininterrumpida.
**Soft state** (S)**.** Cualquier mínimo cambio en el estado del sistema puede cambiar incluso con acciones que no producen inserción de datos. El estado del sistema es sensible a cambios.
**Eventual consistency** (E). En el momento que el sistema realice cambios, debe ser consistente al propagar los datos por todas las ubicaciones posibles.

Se pueden diferenciar en aspectos como el rendimiento o la disponibilidad. Los sistemas ACID pueden ser poco eficientes en sistemas grandes de producción. Este hecho se debe por su alta consistencia a la hora de insertar datos en la BBDD, que al garantizar el almacenamiento de los datos, ante muchas consultas simultáneas, puede producir retrasos o incluso bloqueos.

Una de las grandes características de los sistemas BASE su gran flexibilidad ante diferentes tipos de acciones sobre la BBDD. Su prioridad es la disponibilidad de los datos en todo momento.

## b) Highly available services: Gossip and Bayou

### Gossip

Es un algoritmo de comunicación que se puede presentar como "todo el mundo sabe todo". Se utiliza en Sistemas Distribuidos a gran escala, con servicios en alta disponibilidad mediante la replicación pasiva de datos.
La información se envía a grupos pequeños formados por elementos vecinos. Estos repetirán el proceso hasta expandir la información por todo el sistema. A veces se llama “protocolo epidémico”.
Gracias a su escalabilidad y sencillez, cada vez están más extendidos entre los Sistemas Distribuidos.

- Coordinación: Con *timestamps* y actualizaciones regulares entre los participantes.
- Disponibilidad: Actualizaciones no replicadas hasta alcanzar el estado deseado.
- Transparencia: Sistema transparente.

**Bayou**

Se trata de un sistema de almacenamiento de datos replicado. Todas las replicas pueden ser consultadas o editadas por cualquier usuario del sistema, siempre que estas estén disponibles. Es una manera de garantiza la alta disponibilidad (*High Availability*), ya que si una réplica sufre alguna falla, otra puede seguir ofreciendo el mismo servicio.

- Coordinación: Mediante operaciones permitidas como *undo* o *redo*.
- Disponibilidad: Actualizaciones aplicadas de manera inmediata.
- Transparencia: Sistema no transparente.

## c) Concurrency control: Wiquipedia and Dropbox

En las BBDD de acceso simultaneo, se implementa una estrategia llamada **control de concurrencia.** Este acceso simultáneo puede producir información inconsistente o simplemente incorrecta durante los proceso de lectura y escritura simultáneas. Este problema ha llevado a diseñar diferentes estrategias de **control de concurrencia**. Dos de estos modelos son el optimista y el pesimista.

**Dropbox** utiliza la concurrencia optimista. Es un servicio que ofrece almacenamiento en la nube y permite la edición compartida de diferentes documentos, donde los cambios se realizan en tiempo real. La primera actualización es la aceptada, las que vengan después se almacenarán como versiones del mismo archivo.
Gracias al historial de versiones, los propios usuarios son capaces de restaurar o fusionar distintas ediciones de las diferentes versiones.

**Wikipedia** también implementa la concurrencia optimista. Es una enciclopedia *on-line* que permite editar sus páginas de manera colaborativa por cualquier usuario que quiera participar.
Se pueden realizar ediciones simultáneas aunque al final solo acepta la primera edición. Muestra el resto de cambios como conflictos.
El sistema deja en las manos del usuario solucionar los posibles conflictos que se producen.

# Pregunta 5

Discuss the concept of two-phase commit protocol (2PC) basing on the wikipedia information:
https://en.wikipedia.org/wiki/Two-phase_commit_protocol
Based on the Coulouris book content, search and propose at least one information that you could add to this Wikipedia web page in order to improve its actual content.

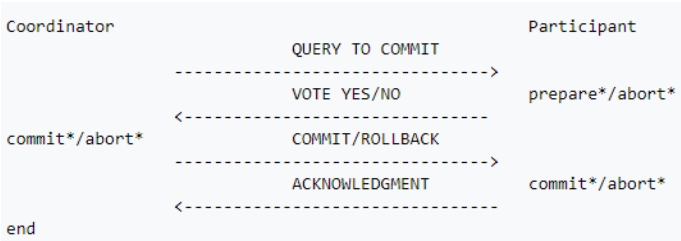
Se trata de un protocolo atómico utilizado en transacciones, BBDD y redes computacionales. Coordina los procesos para confirmar o anular las transacciones, incluso durante fallas temporales del sistema. Es una razón de porqué se ha extendido su implantación. Sin embargo, en casos excepcionales si necesita de intervención manual para solventar el fallo.

Para resolver una falla de manera autónoma, los elementos del protocolo utilizan **registros de estado**. Estos registros se utilizan para la recuperación del protocolo y aunque se utilizan en contadas ocasiones, es uno de sus puntos fuertes, que componen una parte sustancial del protocolo. Existen muchos escenarios de falla al que el protocolo se debe adaptar y respaldar.

Cuando se produce un fallo, el protocolo ejecuta dos partes diferenciadas.
**Fase solicitud de compromiso**. El proceso coordinador prepara al resto de procesos que participan en la transacción. Cada participante votará "si", si su parte se ha ejecutado de manera correcta. Y "no", si ha detectado problemas.
**Fase de compromiso**. En base a la votación, el coordinador decide si se realiza (si todos han votado "si") o se aborta la operación. Tras la decisión, los participantes realizan las acciones oportunas sobre los recursos locales en los que participan (como datos de una BBDD).

**Funcionamiento**
Se asigna un nodo coordinador. El protocolo asume que hay un almacenamiento estable en cada nodo con un registro de escritura anticipada y que se mantiene la comunicación entre nodos.

Una vez finalizada la transacción, el coordinador inicia el protocolo. Luego, los participantes responden con un mensaje de acuerdo o un mensaje de cancelación dependiendo de si la transacción se ha procesado con éxito en el participante.



*Flujo de mensajes*

Su mayor desventaja es que se trata de un **protocolo de bloqueo**. Si el coordinador falla permanentemente, algunos participantes nunca resolverán sus transacciones y quedarán bloqueados.

**Implementación**
En sistemas con BBDD, durante las transacciones, los participantes se registran para cerrar TM´s (procesamiento de transacción) para terminar la transacción utilizando el **algoritmo 2PC**. Existe un coordinador de cada transacción para gestionar el algoritmo. Aunque la función del coordinador se

puede traspasar a otra TM. En lugar de intercambiar mensajes 2PC entre ellos, los participantes intercambian los mensajes con sus respectivos TM.

Los TM relevantes se comunican entre ellos para terminar esa transacción. Con esta arquitectura, el protocolo se distribuye completamente (no necesita ningún componente centralizado) y se amplía con el número de nodos de red (tamaño de red) de manera efectiva. Esta arquitectura también es efectiva para la distribución de otros protocolos de compromiso atómico ya que todos usan el mismo mecanismo de votación y propagación de resultados a los participantes del protocolo.

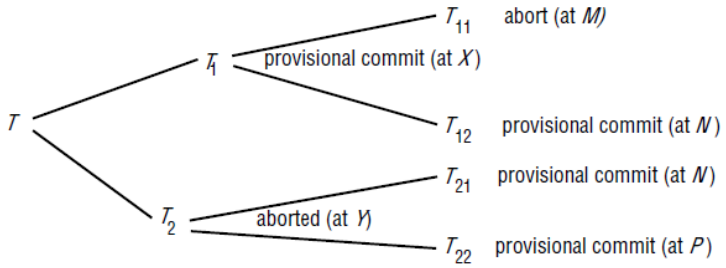
Una suposición sobre el resultado de las transacciones (confirmar o cancelar), puede guardar los mensajes y las operaciones de registro de los participantes durante la ejecución del protocolo 2PC. Por lo general, se paga una penalización por operaciones adicionales durante la recuperación de una falla. Por lo tanto, la mejor variante de optimización, si la hay, se elige de acuerdo con las estadísticas de resultados de transacciones y fallas.

El **protocolo *Tree* 2PC** (recursivo o anidado) es una variante común de 2PC en una red informática, que utiliza mejor la infraestructura de comunicación. Los participantes de organizan en un estructura jerárquica en forma de árbol donde, el coordinador se denomina "raíz" y participantes al resto de nodos. Los mensajes se propagan de forma escalonada, tanto hacía abajo como hacia arriba de la jerarquía.

El **protocolo de compromiso dinámico en dos fases** (D2PC) es otra variante, donde no hay un coordinador preestablecido. Los mensajes de votación son reenviados entre los nodos. El coordinador se determina dinámicamente mediante mensajes de acuerdo de transacciones, en el lugar donde colisionan. Al elegir un coordinador óptimo, D2PC compromete tanto al coordinador y cada participante en el tiempo mínimo posible, lo que permite una liberación más rápida de recursos bloqueados en cada participante de transacción (nodo de árbol).

Información que añadir del ***Coulouris book*** al documento de la ***Wikipedia***

- Se puede completar las fases en las que se desarrolla el protocolo, ya que en el *Coulouris book* se explica con más detalle el funcionamiento del protocolo 2PC.
- Las acciones de espera cuando se producen *timeouts*. En el libro se explican varias acciones que realizan los procesos cuando de consumen los tiempos de espera y tanto el coordinador como el resto de participantes no pueden avanzar hasta recibir la solicitud de respuesta de alguno de los otros participantes.
- Introducir el rendimiento del protocolo. Por ejemplo que el costo en mensajes es proporcional a 3N y el costo en tiempo es de tres rondas de mensajes.
- No añadiría más información del protocolo 2PC anidado, ya que es como un extra del artículo y se trata de una información complementaria al protocolo original. Sin embargo, si añadiría el esquema del protocolo 2PC anidado para que se vea algo más claro.




- Añadir el apartado que se narra la comparación entre los dos enfoques (jerárquico y plano).
- Añadir el cuarto paso que puede producir retrasos en las trasnsacciones.

# 6-Bibliografía


- Libro George-Coulouris-Distributed-Systems-Concepts-and-Design-5th-Edition. Autor: George Coulouris, Jean Dollimore, Tim Dollimore, Gordon Blair. Fecha de publicación [2012], Fecha de consulta [febrero-marzo 2020]

- marknelson.us Publicación de post sobre informática. Link1 Autor: Mark Nelson. Fecha de publicación [23 de julio de 2007]. Fecha de consulta [marzo 2020]

- dosideas.com Blog informática. Link2 Autor: Leonardo de Seta. Fecha de publicación [19 de febrero de 2013]. Fecha de consulta [marzo 2020]

- cnx.org Artículos sobre informática. Link3 Autor: Miguel Ángel Sicilia. Fecha de consulta [marzo 2020]

- dropbox.com Web *Dropbox*. Link4 Fecha de consulta [marzo 2020]

- wikipedia.org Web *Wikipedia* Link5 Fecha de consulta [marzo 2020]

- wikipedia.org Web *Wikipedia*, artículo Control de concurrencia. Link6 Fecha de consulta [marzo 2020]

- wikipedia.org Web *Wikipedia*, artículo Algoritmo *Bully*. Link 7 Fecha de consulta [marzo 2020]

- wikipedia.org Web *Wikipedia*, artículo Problema Generales bizantinos. Link8 Fecha de consulta [marzo 2020]

- binancevision El mundo del *Blokchain* Link9 Fecha de consulta [marzo 2020]