

Tema 7 - Tiempo

Felipe Ortega, Gorka Guardiola

GSyC, ETSIT. URJC.

Sistemas Distribuidos (SD)

26 de noviembre, 2019





(cc) 2008- Grupo de Sistemas y Comunicaciones.,
 Algunos derechos reservados. Este trabajo se entrega bajo la licencia
 Creative Commons Reconocimiento - NoComercial - SinObraDerivada
 (by-nc-nd). Para obtener la licencia completa, véase
<https://creativecommons.org/licenses/by-nc-nd/3.0/es/>.

Contenidos

- 7.1 Principios y definiciones
- 7.2 Network Time Protocol
- 7.3 Tiempo en sistemas distribuidos
- 7.4 Ejemplos (I)
- 7.5 Relación binaria
- 7.6 Concurrencia y ejecución causal
- 7.7 Relojes vectoriales y lógicos
- Referencias

7.1 Principios y definiciones

Tiempo

- ▶ En otras asignaturas visto timeouts, noción de tiempo.
- ▶ ¿Cómo funciona el tiempo en sistemas distribuidos?

Deriva de reloj

- ▶ No todos los relojes avanzan al mismo ritmo.
- ▶ Razones técnicas (temperatura chip, etc.) y físicas (relatividad, ej. GPS).
- ▶ Velocidad relativa de los relojes además cambia, dos relojes sincronizados se desincronizan: deriva (drift).
- ▶ Para compensar se sincronizan los relojes periódicamente.
- ▶ Hay que usar la noción central de reloj lo menos posible.

Timeouts

- ▶ Ojo, esto no requiere comparar relojes de varios nodos.
- ▶ Usa sólo el reloj local.
- ▶ Si avanza o retrocede puede cambiar si uso el reloj inadecuado (pared vs. free running).
- ▶ Timeouts son fallos y los fallos son normales (conmuto de vuelta).
- ▶ Los timeouts introducen latencia, hay que elegirlos bien.

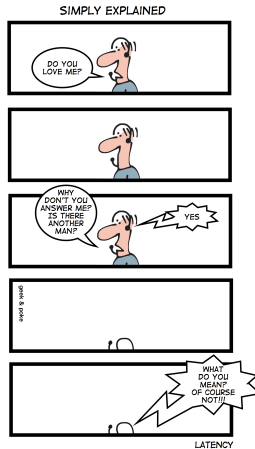
Sistemas asíncronos, vs. síncronos

- ▶ Modelo teórico, el menor número de límites posibles.
- ▶ Asíncronos: Los mensajes pueden tardar un tiempo indefinido en llegar, no hay noción central de reloj.
- ▶ Síncronos: El tiempo de entregar un mensaje está acotado, hay noción central de reloj.
- ▶ Viveza (liveness): algo bueno sucede antes o después (condición que debe suceder un número de veces, posiblemente infinita)
- ▶ Seguridad (safety): nada malo sucede (condición sobre un prefijo).

Reloj en un PC

- ▶ Suponemos un reloj central, hay que sincronizarlos todos.
- ▶ SAW (surface acoustic wave), oscilador de cuarzo (en la placa), PCC processor cycle counter (un por core).
- ▶ El PCC se suele usar para medir intervalos, se suelen sincronizar los de los cores, pero son distintos.
- ▶ Ambos cambian mucho con la temperatura (el PCC es mucho peor), no muy bueno.
- ▶ Fuentes externas, NTP, GPS, reloj externo conectado a un puerto (USB, serie...).
- ▶ Cada tic, una interrupción, actualizo una variable con lo que estime que es la hora con las diferentes fuentes.

Latencia (y su variabilidad: jitter)



De geek & poke, <http://geek-and-poke.com/>.

Reloj

- ▶ Tipos de error.
- ▶ Error de fase (offset) (constante, compensable).
- ▶ Error de frecuencia (constante, compensable).
- ▶ Error absoluto en el valor (respecto de una fuente fiable), la medida de su incertidumbre es la precisión (por ejemplo la desviación estándar).
- ▶ Fluctuación de la fase (jitter, dispersión), más corto plazo, segunda diferencia tiempo.
- ▶ Inestabilidad de frecuencia, más largo plazo (wander).
- ▶ Desviación de oscilador ideal, medido por la varianza de Allan.
- ▶ Dado un error, el tiempo que puede mantenerlo es la confiabilidad (reliability).

Reloj

- ▶ Resolución, cada cuanto actualizo el valor (qué bit se mueve en un tic).
- ▶ El error absoluto en el valor tiene que ser menor que la resolución.

Reloj: varianza de Allan

- ▶ Calcular la varianza no converge para el tipo de ruido (flicker $1/f$).
- ▶ Medida estándar de lo malo que es un reloj, media de la diferencia primera de las muestras.
- ▶ https://en.wikipedia.org/wiki/Allan_variance.
- ▶ <http://www.allanstime.com/AllanVariance/>.

7.2 Network Time Protocol

NTP (v4: rfc5905)

- ▶ Algoritmo de sincronización de relojes.
- ▶ Jerarquía de relojes (estratos o strata).
- ▶ Relojes de Cesio, rubidio en la raíz.
- ▶ <https://tools.ietf.org/html/rfc5905>.

NTP

- ▶ El cliente se conecta al servidor.
- ▶ El servidor contesta con una marca de tiempo NTP, precisión.
- ▶ El protocolo es seguro (cambiar el tiempo es un problema de seguridad, certificados etc.).
- ▶ Los mensajes tardan tiempo en llegar.
- ▶ El cliente debe compensar el RTT (lo que tarda el mensaje en ir y venir).
- ▶ Sistema con realimentación, PLL/FLL y filtrado (poco a poco, para no variar el reloj muy rápido).

7.3 Tiempo en sistemas distribuidos

Ajuste de hora en el kernel

- ▶ Dos interfaces *adjtime()* y *adjtimex()*.
- ▶ *adjtime()* Resolución de microsegundos (mal llamado microkernel, ojo no confundir).
- ▶ *adjtimex()* Resolución de nanosegundos (mal llamado nanokernel), realimentación mitad kernel mitad usuario.

El tiempo en sistemas distribuidos

- ▶ Los mensajes tardan tiempo variable en mandarse (latencia variable, jitter).
- ▶ Cada máquina tiene su reloj, deriva.
- ▶ No hay una noción de reloj único (se puede aproximar, por ejemplo NTP, pero surgen otros problemas).
- ▶ ¿Qué puede reemplazar la noción de tiempo?

El tiempo en sistemas distribuidos

- ▶ ¿Podemos arreglarlo con marcas de tiempo?
- ▶ No, el ahora es una ilusión.
- ▶ Problema relativista de simultaneidad (qué pasa antes o después depende del observador).
- ▶ La velocidad de la luz es constante, los relojes no.
- ▶ Relatividad especial, todavía peor (gravedad).

No hay un ahora



No hay un ahora



No hay un ahora



No hay un ahora



No hay un ahora



No hay un ahora



No hay un ahora



No hay un ahora



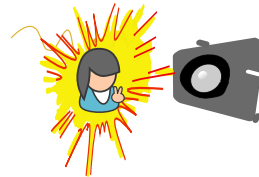
No hay un ahora



No hay un ahora



No hay un ahora



No hay un ahora

- ▶ Podría saber qué va antes, con el reloj.
- ▶ Porque las cámaras no se mueven.
- ▶ ¿Qué sucede si las cámaras van a la misma velocidad que la chica?
- ▶ La simultaneidad es tan relativa como la velocidad.

No hay un ahora



No hay un ahora



No hay un ahora



No hay un ahora



No hay un ahora



No hay un ahora



No hay un ahora



No hay un ahora

- ▶ ¿Podría saber qué va antes?, ¿con el reloj?
- ▶ ¿Qué es un reloj? ¿luz reflejada en unas manecillas?
- ▶ La velocidad de la luz es constante → los relojes no.
- ▶ Qué sucede a la vez depende del observador, su reloj, que depende de su velocidad.

No hay un ahora

- ▶ Aunque los relojes fuesen perfectos y todo el mundo estuviese quieto, los relojes no van igual (relatividad general).
- ▶ En un pozo gravitatorio (superficie de la tierra, fuera), aceleración (principio de equivalencia).
- ▶ Causa problemas en GPS.

No hay un ahora

- ▶ Pero, en cualquier caso, los relojes no son perfectos.
- ▶ En particular los de los PCs, que son especialmente malos.
- ▶ Podemos usar NTP, pero dependemos del jitter del reloj y de la latencia de red.
- ▶ <http://queue.acm.org/detail.cfm?id=2745385>.

El tiempo en sistemas distribuidos

- ▶ ¿Qué nos queda?
- ▶ Se preserva la causalidad (qué afecta a qué).
- ▶ Se sabe que el flash es antes de que la luz le llegue a la chica.
- ▶ Se sabe que la respuesta viene después de la pregunta.

7.4 Ejemplos (I)

Ejemplo: replicación de ficheros

- ▶ Tengo mi cuenta con mis ficheros en dos ordenadores A, B y un portátil C.
- ▶ Llego al trabajo y sincronizo C con A, me voy a mi casa y sincronizo B con C.
- ▶ En cualquier momento trabajo en A, B o C.
- ▶ Quiero tener mi cuenta replicada en todos ellos y no perder datos.

Ejemplo: replicación de ficheros

- ▶ Problema: mi ordenador de casa va adelantado.
- ▶ Sincronizo $A \leftrightarrow B$.
- ▶ Trabajo en B, las marcas de tiempo son anteriores a la que genera A.
- ▶ Sincronizo $A \leftrightarrow B$.
- ▶ Pierdo mi trabajo (piensa que lo que había en A era más nuevo).

Ejemplo: notificaciones

- ▶ Me llega una notificación, por ejemplo de Twitter.
- ▶ Pero no me ha llegado todavía el mensaje.
- ▶ Está usando relojes y causa inconsistencias, si preservase la causalidad. . .

Ejemplo: grupos de Whatsapp, facebook

- ▶ Quito a mi jefe de un grupo de Whatsapp.
- ▶ Pongo un post y le llega.
- ▶ Last Writer Wins (LWW) + relojes malos.
- ▶ Si preservase la causalidad...

Ejemplo: grupo de gente chateando

- ▶ Todas las respuestas.
- ▶ Después de los posts que las originaron.

Dos tipos de problemas

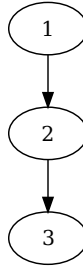
- ▶ El problema de la conversación: muchos hablando a la vez, respuestas después de preguntas → Historias causales.
- ▶ El problema del fichero, capturar la idea de versión con varios modificándola → Versiones.
- ▶ Veremos que el problema de las versiones es una simplificación del otro.

7.5 Relación binaria

Relación binaria (paréntesis matemático)

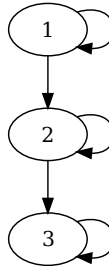
- ▶ Conjunto de pares.
- ▶ O un conjunto y un operador (que dice si un par pertenece al conjunto de arriba).
- ▶ Fácil imaginar como un grafo (si hay flecha entre los nodos, pertenece a la relación).
- ▶ Ejemplo $a > b$ para el conjunto $\{1, 2, 3\}$.

Relación binaria, ejemplo



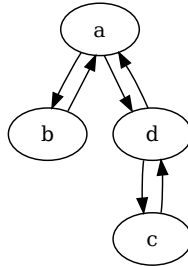
- Ejemplo $a > b$ para el conjunto $\{1, 2, 3\}$.

Relación binaria, ejemplo



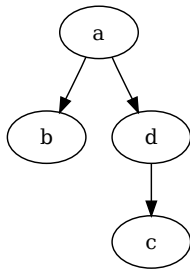
- Ejemplo $a \geq b$ para el conjunto $\{1, 2, 3\}$.

Relación binaria simétrica



- $a \diamond b \longrightarrow b \diamond a, \forall a, b.$
- Todas las flechas tienen una de vuelta.

Relación binaria antisimétrica



- ▶ $a \diamond b \longrightarrow \neg b \diamond a, \forall a, b.$
- ▶ Todas las flechas no tienen una de vuelta.

Relación binaria transitiva

- ▶ $a \diamond b \wedge b \diamond c \longrightarrow a \diamond c.$
- ▶ ¿Eres capaz de dibujar el grafo?

Orden parcial, orden total

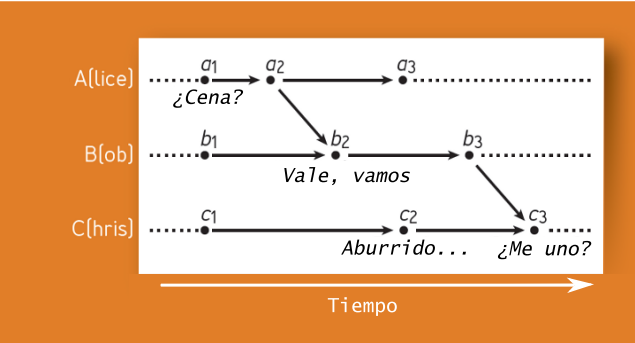
- ▶ reflexivo $a \leq a$.
- ▶ antisimétrico $a \leq b, b \leq a \rightarrow a = b$.
- ▶ transitivo $a \leq b, b \leq c, a \leq c$.
- ▶ todos comparables, orden total, si no, orden parcial (ejemplo, vectores).

7.6 Concurrencia y ejecución causal

Causalidad

- ▶ Vamos a ver un ejemplo para centrar el problema.
- ▶ Tenemos a tres personas, Alice, Bob y Chris.
- ▶ Las flechas son mensajes que intercambian (por ejemplo en un chat).
- ▶ Alice decide salir a cenar.
- ▶ Se lo dice a Bob (que acepta).
- ▶ Mientras, Chris está en casa aburrido.
- ▶ Bob se lo dice a Chris.

Causalidad



Quedando para cenar

Causalidad

- ▶ Si vemos el conjunto de eventos
- ▶ Hay una relación causal entre la decisión de Alice, el mensaje a Bob y el mensaje a Chris.
- ▶ Entre c_2 (Chris se aburre) y a_1 (Alice decide una cena) no hay relación causal.
- ▶ Ver si c_2 va antes o después o después de a_1 requiere un reloj (orden total).
- ▶ c_2 y a_1 son concurrentes.

Relación binaria de “sucede antes de”

- ▶ Relación binaria entre eventos.
- ▶ $x \Rightarrow y$.
- ▶ Si es en el mismo nodo, x pasa antes que y.
- ▶ x es el evento de mandar un mensaje e y el de recibirlo.
- ▶ $\exists z$ tal que $x \Rightarrow z, z \Rightarrow y$.
- ▶ Es un orden parcial, porque hay sucesos concurrentes.

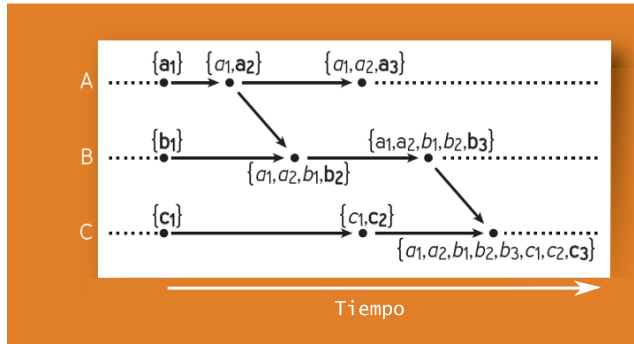
Concurrencia

- ▶ x e y son concurrentes $x \parallel y$ si:
 - ▶ $x \not\bowtie y$.
 - ▶ $y \not\bowtie x$.

Ejecución causal

- ▶ Una forma de ordenar el grafo de eventos.
- ▶ Todos los eventos están en orden causal (los que suceden de forma concurrente, no).
- ▶ Hay muchas combinaciones (con la libertad que deja el intercambio de mensajes).

Historias causales



Se guarda el conjunto de eventos por el que he pasado.

Historias causales

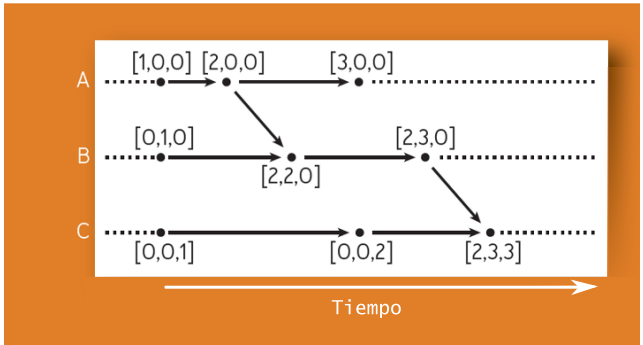
- ▶ En lugar de una marca de tiempo.
- ▶ Vamos a guardar un conjunto de eventos que me precedieron.
- ▶ La causalidad se puede ver por inclusión estricta $x \Rightarrow y \iff H_x \subsetneq H_y$.
- ▶ <https://www.vs.inf.ethz.ch/publ/papers/holygrail.pdf>.

Historias causales con índice (dotted)

- ▶ Más fácil si marcamos el último, que es al que pertenece la historia.
- ▶ La causalidad se puede ver por pertenencia $x \Rightarrow y \iff x \in H_y$.
- ▶ Porque la historia causal incluye todos los eventos que le preceden causalmente.

7.7 Relojes vectoriales y lógicos

Relojes vectoriales



Cuento cuanto he visto de cada evento.

Relojes vectoriales

- ▶ Comprimo las historias causales.
- ▶ Comprimo $\{a_1, a_2, b_1, b_2, b_3, c_1, c_2, c_3\}$ a $[2_a, 3_b, 3_c]$.
- ▶ Un vector con el nombre del nodo como índice.
- ▶ Traduzco las reglas a esta representación compacta.

Relojes vectoriales

- ▶ Sabíamos que $x \Rightarrow y \iff H_x \subsetneq H_y$.
- ▶ Pero sabemos que eso es ver si hay al menos un elemento en H_y no contenido en H_x .
- ▶ y el resto están contenidos.
- ▶ Eso se traduce a que $\forall i : V_x[i] \leq V_y[i] \wedge \exists j : V_x[j] < V_y[j]$.
- ▶ O en otras palabras $V_x < V_y$.

Relojes vectoriales

- ▶ Un evento nuevo es incrementar el índice del nodo en el que sucede.
- ▶ Se combinan dos con el máximo de ambos (unión en las historias)
 $\forall i : V[i] = \max(V[i], V_y[i]).$
- ▶ La recepción es un evento y una combinación.

Relojes vectoriales

- ▶ Se pueden comparar si todos los valores son $<$ o \leq .
- ▶ Si no, son incomparables $x \parallel y$.
- ▶ Si son comparables, ya sabemos $x \Rightarrow y \iff V_x < V_y$.
- ▶ Es un orden parcial (hay algunos incomparables).

Relojes vectoriales con índice (dotted)

- ▶ Marco el índice del último que he actualizado.
- ▶ $[2_a, \mathbf{3}_b, 3_c]$.
- ▶ Esto se hace con otra variable que guarda ese índice: $\{[2_a, 2_b, 3_c], b\}$.
- ▶ Sabiendo cual es el último, podemos hacer lo mismo que con las historias.
- ▶ Para saber si $x = [2_a, 0_b, 0_c] \Rightarrow y = [2_a, \mathbf{3}_b, 0_c]$ basta con ver $x[a] \leq y[a]$, ($2 \leq 2$).

Relojes lógicos

- ▶ <http://research.microsoft.com/en-us/um/people/lamport/pubs/time-clocks.pdf>.
- ▶ Comprimo más los relojes vectoriales, menos garantías.
- ▶ La marca de tiempo es el número de elementos en la historia causal (o la suma del reloj vectorial).
- ▶ $x \Rightarrow y$ entonces $LT(x) < LT(y)$.
- ▶ El mensaje pone el reloj local al máximo del que viene en el mensaje y el local.

Relojes lógicos

- ▶ Nos garantizan que $x \Rightarrow y \rightarrow LT(x) \leq LT(y)$.
- ▶ es decir $LT(x) > LT(y) \rightarrow x \not\Rightarrow y$.
- ▶ el mayor no causa al menor.
- ▶ No nos garantizan $LT(x) < LT(y) \rightarrow x \Rightarrow y$.
- ▶ Puede ser menor pero no causar al mayor, hemos perdido información.

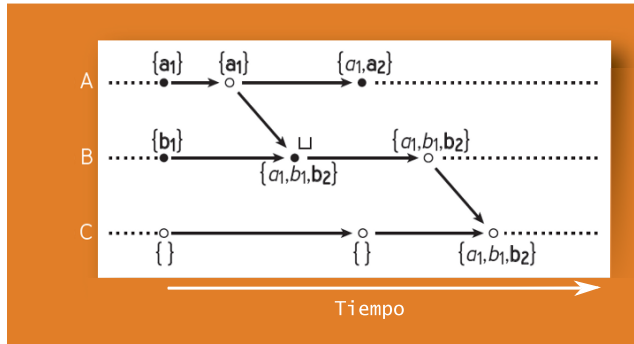
Sistemas Distribuidos (SD)

- ▶ Para mantener marcas de versión en un documento (el segundo problema).
- ▶ No necesito mantener más que un hilo canónico (la versión).
- ▶ Hay un sabor de todo lo anterior para versiones.

Historias causales sólo con los eventos relevantes

- ▶ Historias causales pero sólo para la versión de un documento.
- ▶ Los eventos son las modificaciones del documento.
- ▶ Cuanto dos versiones del documento se encuentran.
 - ▶ Si uno precede a otro, lo sustituye.
 - ▶ Si son concurrentes, merge \sqcup (si son compatibles, automático).
 - ▶ Hay que definir qué significa compatibles, puedo juntarlos, elegir uno al azar, requerir a un humano....
- ▶ Podemos verlo como un árbol genealógico.
- ▶ <http://www.allthingsdistributed.com/les/amazon-dynamo-sosp2007.pdf>.

Historias causales sólo con los eventos relevantes

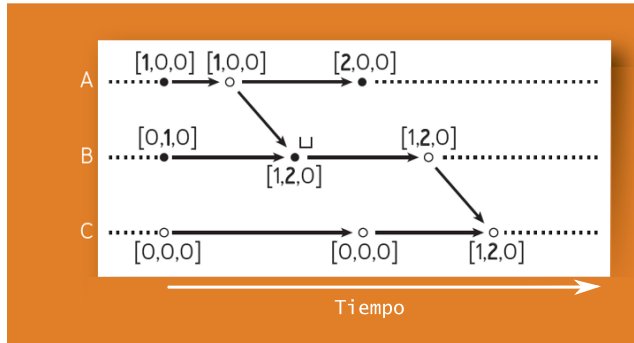


Se sigue sólo el árbol genealógico del documento.

Vector de Versión

- ▶ Relojes vectoriales, pero para versiones.
- ▶ <http://zoo.cs.yale.edu/classes/cs422/2013/bib/parker83detection.pdf>.

Vector de Versión

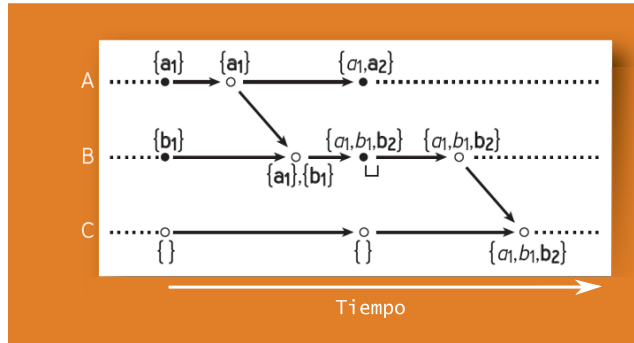


Historias causales con eventos relevantes, pero comprimido.

Vector de versión postergando el merge

- ▶ A veces se guardan varias versiones juntas incompatibles.
- ▶ Más tarde quizás se hace el merge (o de un subconjunto).

Vector de versión postergando el merge



Se sigue sólo el árbol genealógico del evento.

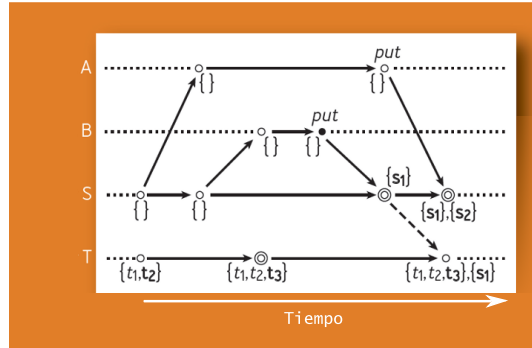
Vector de versión sparse

- ▶ Puedo tener muchos nodos.
- ▶ Pero pocos en la historia de cada documento.
- ▶ En lugar de guardar $[0, 3, 0, 0, \dots, 0, 0, 0]$.
- ▶ Puedo guardar un diccionario (map, en golang).
- ▶ Los que no están son cero.
- ▶ Cuidado al comparar/merge, hay que recorrerse ambos.

Vector de versión/Historia de versión con índice

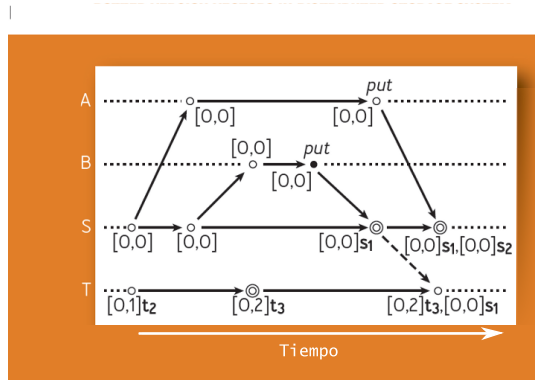
- ▶ Se puede hacer el mismo truco, guardar el último índice separado.

Historia de versión con índice (dotted)



Se apunta el último separado.

Vector de versión con índice (dotted)




Se apunta el último separado, (el servidor reparte IDs).

Corte causal

- ▶ Para obtener un snapshot.
- ▶ No puede haber nada después del corte que preceda a algo antes.
- ▶ No puede haber nada sin causa.

Bibliografía I

 [Birman, 2015] Birman, Kenneth P.
Reliable distributed systems: technologies, web services, and applications.
Springer, 2005.

 [Lamport, 2001] Lamport. L.
Paxos made simple.
ACM SIGACT News, 32.4 (2001): 18-25.

 [van Renesse, 2011] van Renesse, R.
Paxos Made Moderately Complex.
Cornell University, 2011.

 [van Steen & Tanenbaum, 2017] van Steen, M., Tanenbaum, A. S.
Distributed Systems.
Third Edition, version 01. 2017.