Leases: An efficient Fault-tolerant Mechanism for Distributed File Cache Consistency

Gray, Cary, and David Cheriton. *Leases: An efficient fault-tolerant mechanism for distributed file cache consistency*. Vol. 23. No. 5. ACM, 1989.

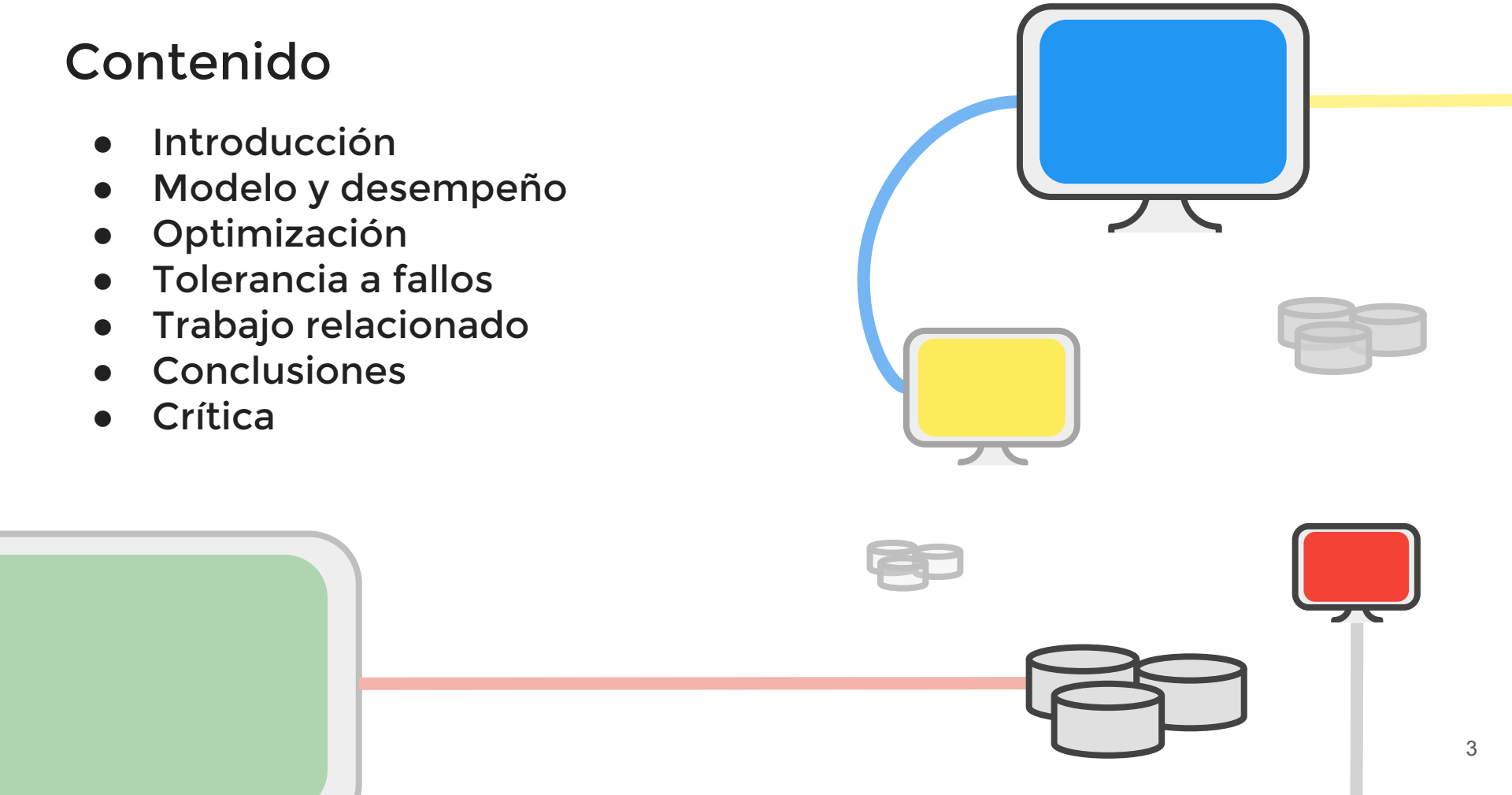
María Fernanda Mora Alba
10 noviembre, 2016

“In an ideal world there would be only one consistency model: when an update is made all observers would see that update”

- Werner Vogels

Contenido

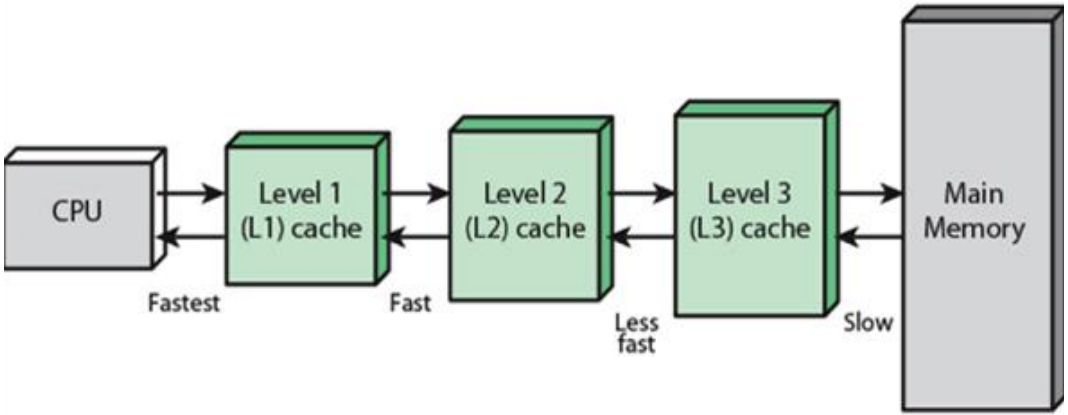
- Introducción
- Modelo y desempeño
- Optimización
- Tolerancia a fallos
- Trabajo relacionado
- Conclusiones
- Crítica



Introducción

Descripción del problema

- *Caching* es un mecanismo que permite mejorar la velocidad y desempeño al copiar en la memoria de rápido acceso (caché) los datos usados más frecuentemente.

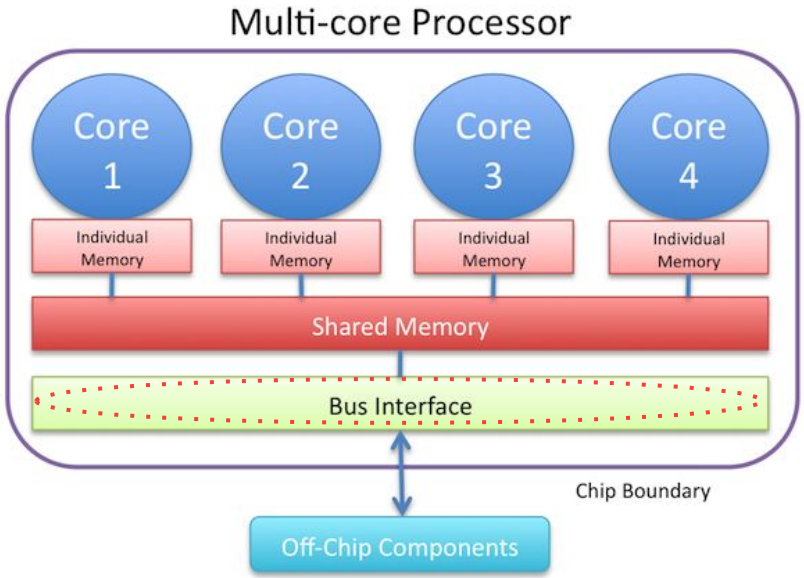


La jerarquía de memoria explota los principios de localidad espacial y temporal

La escritura a memoria principal se maneja con políticas de escritura

Descripción del problema

- ¿Cuál es el problema?
- Arquitecturas multicore:



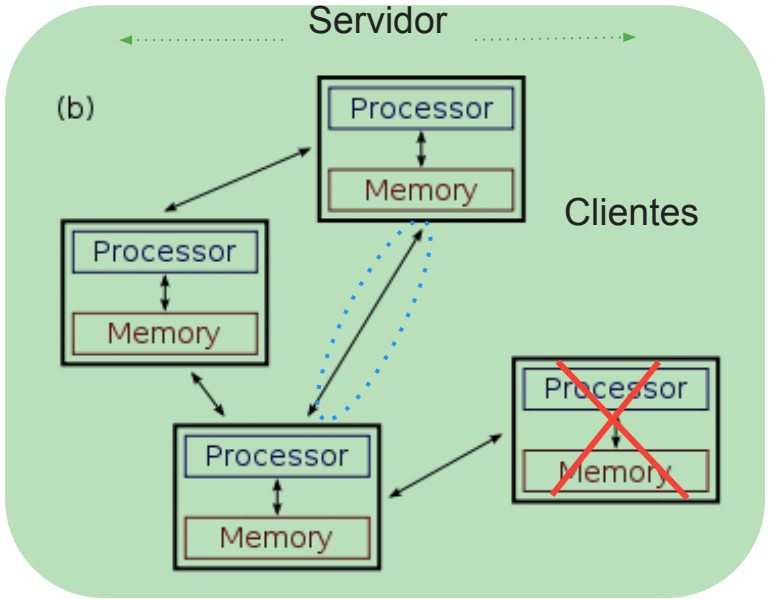
En arquitecturas multicore un core puede modificar un dato D, ¿cómo garantizar que otro core ve la versión modificada de D?

Suponían una comunicación confiable y síncrona

Estos protocolos ya habían sido estudiados

Descripción del problema

- Sistemas distribuidos:



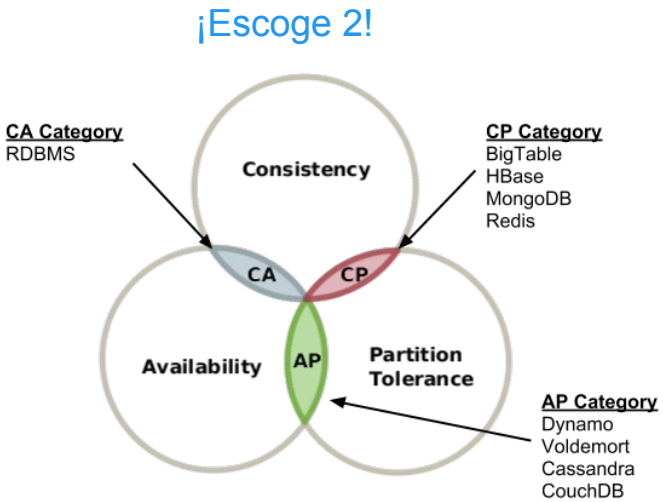
En un sistema distribuido los cores se convierten en clientes con cierta autonomía

Tenemos el problema adicional de la pérdida de mensajes y del fallo de clientes (además de otros)

Modelos de consistencia: contratos en donde el sistema le garantiza al programador que si sigue las reglas, la memoria será consistente y los resultados de las operaciones de memoria serán predecibles

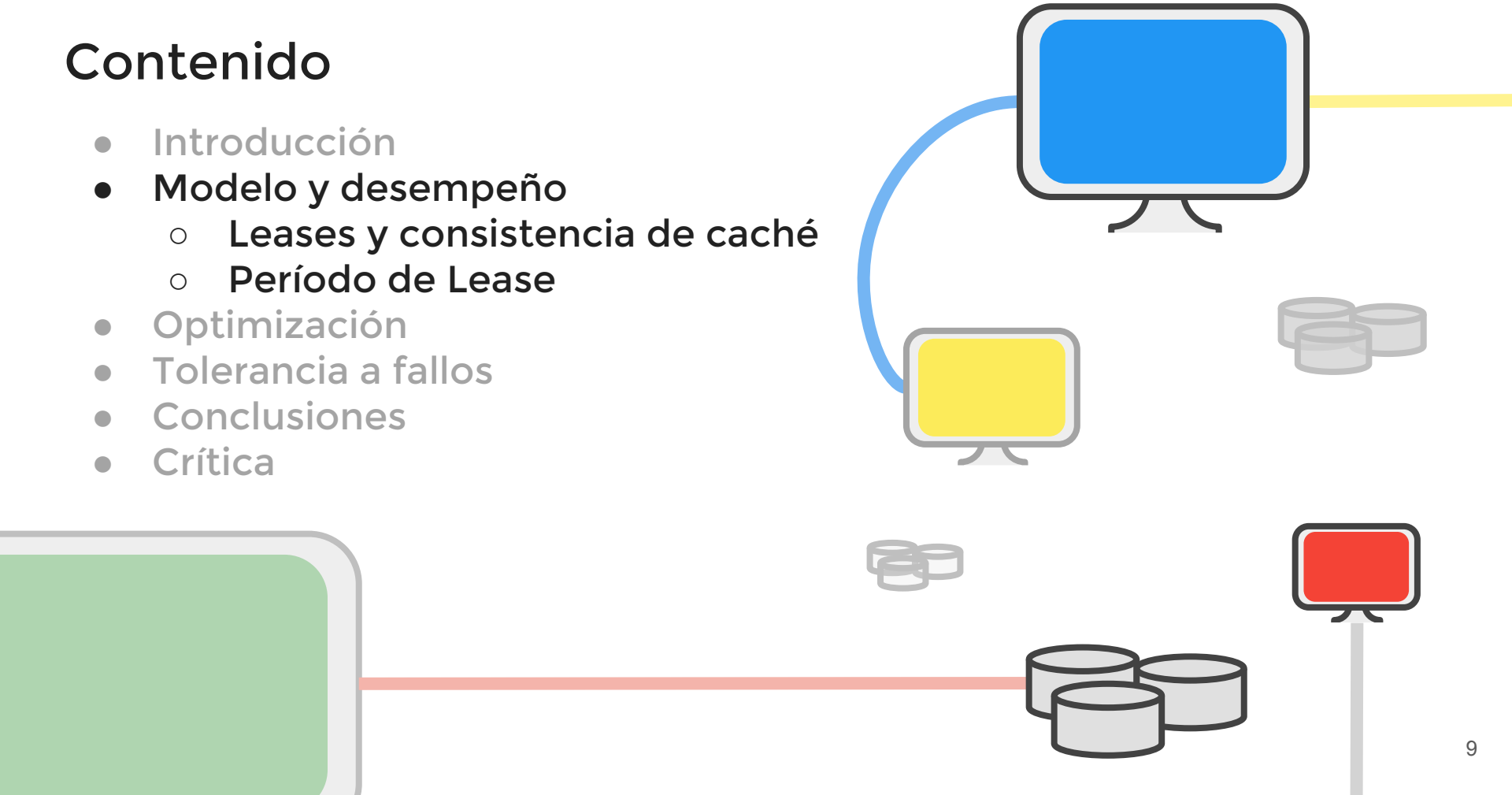
Trabajo relacionado

- Enfoques previos:
  - Asumían un canal de comunicación confiable: no es la norma en sistemas distribuidos, o
  - Verificación de consistencia para cada lectura: desempeño pobre
- ¿Por qué o uno o lo otro?
- No se puede tener todo en esta vida: Teorema CAP para sistemas distribuidos
- Esto asume un ambiente sencillo, no considera escalabilidad, responsividad.



Contenido

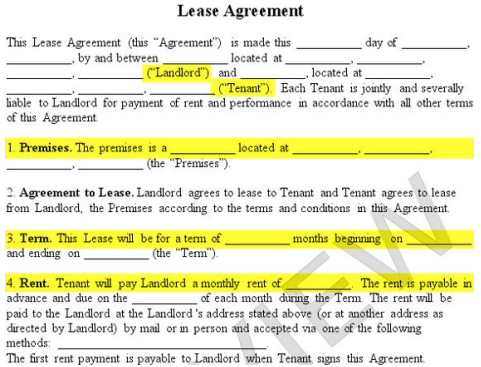
- Introducción
- Modelo y desempeño
  - Leases y consistencia de caché
  - Período de Lease
- Optimización
- Tolerancia a fallos
- Conclusiones
- Crítica



Modelo y desempeño

Leases

- Arrendamiento/Leasing: un contrato que le da a su poseedor derechos específicos sobre cierta propiedad por un período de tiempo.
- Arrendamiento en computación: una promesa hecha por el servidor al cliente de que lo mantendrá actualizado de cambios, así como que le cederá el control de la escritura sobre los datos cubiertos durante un período específico.



Lease Agreement

This Lease Agreement (this "Agreement") is made this _____ day of _____, _____, by and between _____ located at _____, _____, _____, _____ ("Landlord") and _____, located at _____, _____, _____, _____ ("Tenant"). Each Tenant is jointly and severally liable to Landlord for payment of rent and performance in accordance with all other terms of this Agreement.

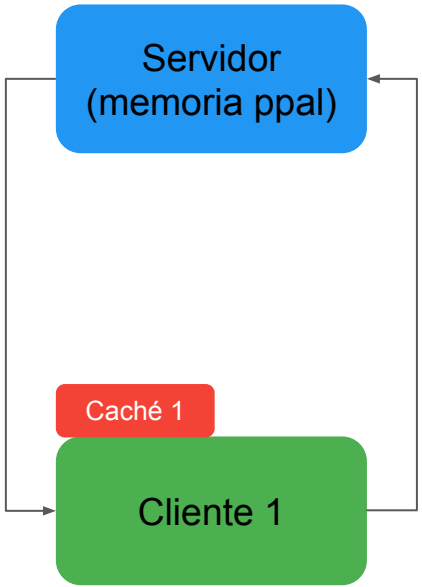
1. **Premises.** The premises is a _____ located at _____, _____, _____, _____ (the "Premises").

2. **Agreement to Lease.** Landlord agrees to lease to Tenant and Tenant agrees to lease from Landlord, the Premises according to the terms and conditions in this Agreement.

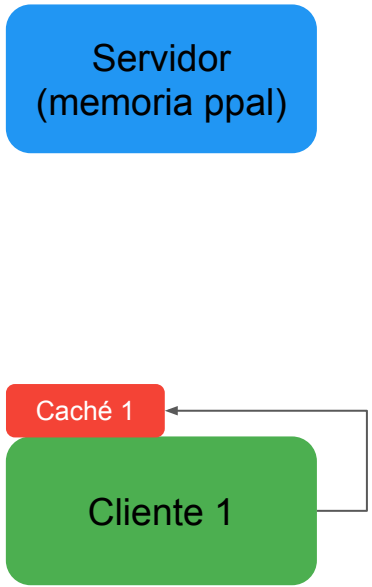
3. **Term.** This Lease will be for a term of _____ months beginning on _____ and ending on _____ (the "Term").

4. **Rent.** Tenant will pay Landlord a monthly rent of _____. The rent is payable in advance and due on the _____ of each month during the Term. The rent will be paid to the Landlord at the Landlord's address stated above (or at another address as directed by Landlord) by mail or in person and accepted via one of the following methods: _____
The first rent payment is payable to Landlord when Tenant signs this Agreement.

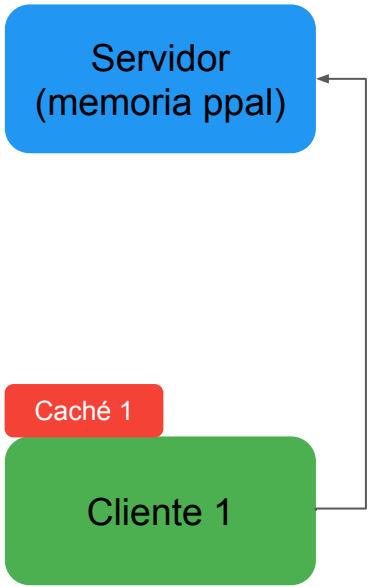
Cómo se usa *Leases* para lecturas



Dato y Lease solicitado al servidor y concedido de 10 segundos



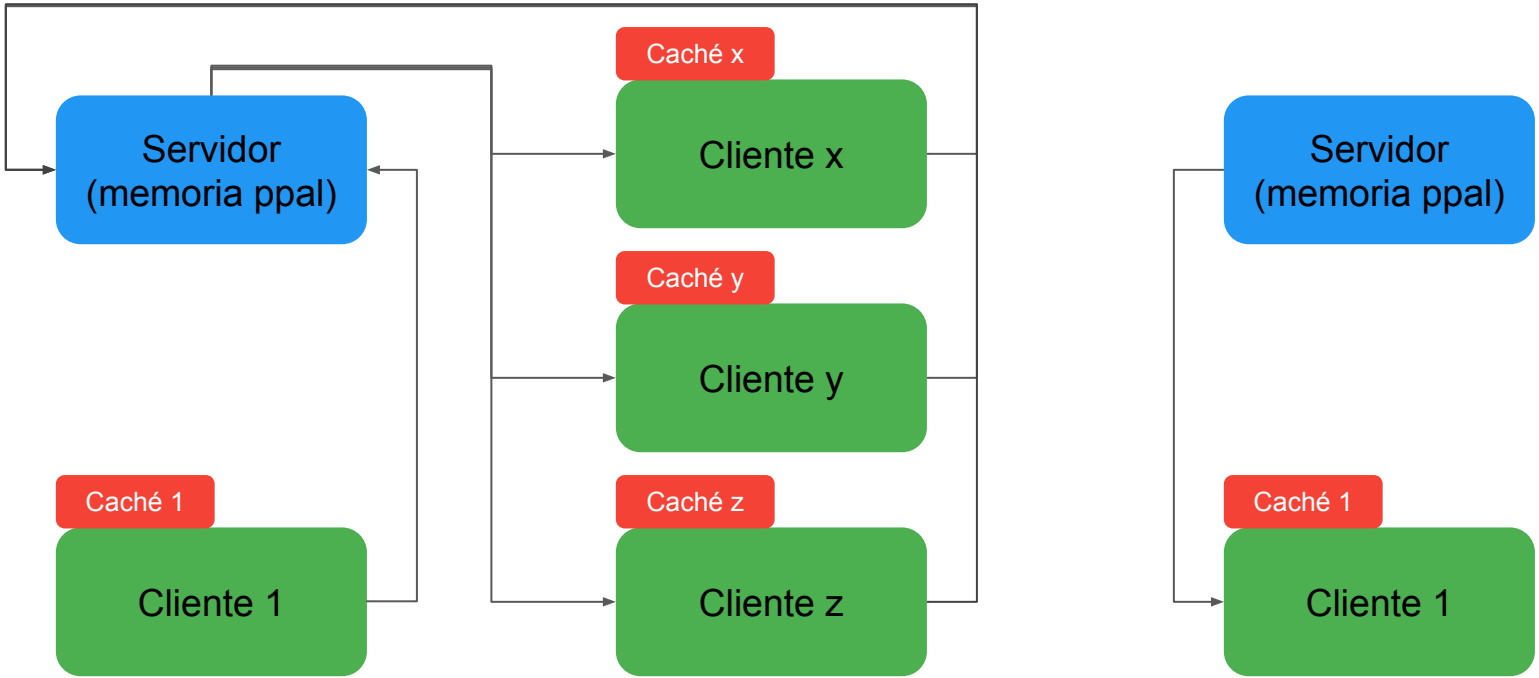
Después de 5 segundos cliente lee del archivo sin consultar al servidor



Después de 10 segundos cliente tiene que extender su lease y modificar si es necesario

Protección

Cómo se usa *Leases* para escrituras



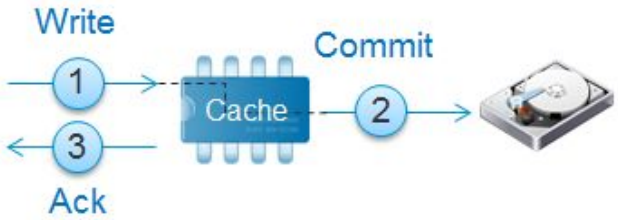
Lease solicitado por clliente 1

Servidor pide permiso a los otros arrendatarios vigentes (sobre el mismo archivo)

Servidor da el lease a cliente 1

Consideraciones

- Sólo se consideran cachés write-through (i.e. se escribe a memoria principal y caché simultáneamente)
- Esto permite no tener escrituras perdidas de algún caché
- El costo de este tipo de política de escritura es alto
- Con un manejo especial de archivos temporales se puede aminorar ese costo
- Las operaciones de lectura y escritura se extienden más allá del contenido del archivo: re-apertura y modificación de nombre de archivo



¿De qué tamaño tiene que ser el período del arrendamiento?

- Artículo propone períodos cortos:
  1. Minimizan el retraso ocasionado ante falla del servidor o clientes en una escritura: servidor se tiene que esperar “poco” ó si servidor falla sólo retrasa un poco las escrituras
  2. Minimizan la falsa-compartición (aparente conflicto cuando no lo hay): overhead de la llamada al arrendador ante escrituras
  3. Reducen los requerimientos de almacenamiento en el servidor: poco overhead de llevar un registro de los arrendamientos que ha concedido a cada cliente ¿?

  100 leases por cliente -> 1 KB por cliente. Se puede reducir para archivos que se comparten mucho.

¿De qué tamaño tiene que ser el período del arrendamiento?

- ¿Por qué no poner un período de 0 entonces?
- ¡Se pierde el sentido de *Leases*!
- Períodos largos:
  - Sólo para archivos que se acceden repetidamente y que se comparten poco para escritura
  - Caso de uso: Andrew File System pasó de tener un período de 0, a un período arbitrariamente grande
- ¿Cómo determinar analíticamente el período de arrendamiento?

Período de arrendamiento: supuestos

- Trade-off entre minimizar el overhead de extender el arrendamiento y minimizar la falsa-compartición
- El overhead de almacenamiento no se considera
- La tasa de fallo se asume que no tiene un efecto considerable en el tiempo de respuesta
- La extensión del arrendamiento se asume a demanda

Modelo

- Parámetros

Symbol	Description
N	number of clients (caches)
R	rate of reads for each client
W	rate of writes for each client
S	number of caches in which the file is shared
m_{prop}	propagation delay for a message
m_{proc}	time to process a message (send or receive)
ϵ	allowance for uncertainty in clocks
t_S	lease term (at server)

- Lecturas~Poisson(R)
- Escrituras~Poisson(W)
- Compartición en la escritura

Modelo: períodos

- Sea m_{prop} tiempo de propagación de mensaje
- Sea m_{proc} tiempo de procesamiento de mensaje (envío/recepción)
- Período efectivo en caché:

$$t_c = \max(0, t_s - (m_{prop} + 2m_{proc}) - \varepsilon)$$

Tiempo de recepción de mensaje por canal **unicast**

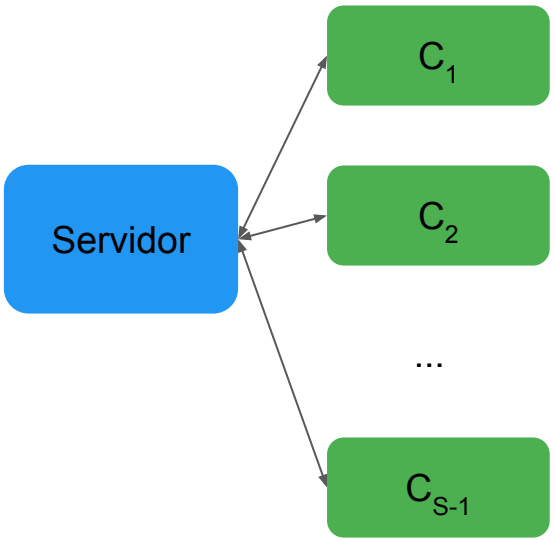
Sesgo del reloj (físico)

i.e. para que el caché reciba el arrendamiento

Modelo: períodos

- Tiempo para obtener la aprobación de los otros arrendadores (S-1), S>1:
  - $t_a = 2m_{prop} + (S + 2)m_{proc}$

    Tiempo en mandar un mensaje multicast y recibir S-1 respuestas
  - $t_a << t_s$ (mseg vs segs)



Modelo: costos por consistencia

S>1, $t_s \neq 0$

- Carga en el servidor (mensajes totales):

$$\underbrace{\frac{2NR}{1+Rt_C}}_{\text{Mensajes lectura}} + \underbrace{NSW}_{\text{Mensajes escritura}}$$

- Retraso promedio por cada lectura o escritura:

$$\frac{1}{R+W}\left(\underbrace{\frac{2R(m_{prop}+2m_{proc})}{1+Rt_C}}_{\text{Lectura}} + \underbrace{Wt_a}_{\text{Escritura}}\right)$$

- Si además $t_a \ll t_s$, $t_C > \frac{1}{R(\alpha-1)}$ y $\alpha > 1$ donde $\alpha = \frac{2R}{SW}$ mide la razón de lecturas a escrituras (factor de beneficio), entonces

$$\underbrace{2NR}_{t=0} > \frac{2NR}{1+Rt_C} + NSW$$

- Valores más grandes de α y R tienen mejor rendimiento para períodos de arrendamiento chicos

Modelo: costos

$t_s=0$

- Si t es muy chico, pero $t_s \neq 0$ entonces $t_c = 0$ i.e. escrituras penalizadas pero lecturas no se benefician
  - Mejor $t_s=0$
- Entonces carga en el servidor $2NR$

Modelo: caso muchos archivos

- La carga es aditiva, R y W también
- El caché puede hacer un lote con sus solicitudes de extensiones
- La tasa absoluta de de lecturas aumenta y α también aumenta, entonces el beneficio por arrendamiento es mayor
- La carga en el servidor es proporcional al total de mensajes enviados y recibidos
- Al tiempo de respuesta a nivel aplicación se le deben sumar otros elementos

Desempeño esperado

Sistema operativo distribuido V

- Se recolectaron una traza de tráfico de acceso a archivos y parámetros:

rate of reads	R	0.864 /sec
rate of writes	W	0.039 /sec
message propagation time	m_{prop}	1.0 msec
message processing time	m_{proc}	0.25 msec
allowance for clocks	ϵ	100 msec

Sólo un usuario (no hay escrituras en archivos compartidos)

- La mayor parte del beneficio de tener $t_s \neq 0$ se gana a los pocos segundos
- En $t_s = 0$ la carga por consistencia es el 30% del tráfico total

Se simularon varios niveles de compartición

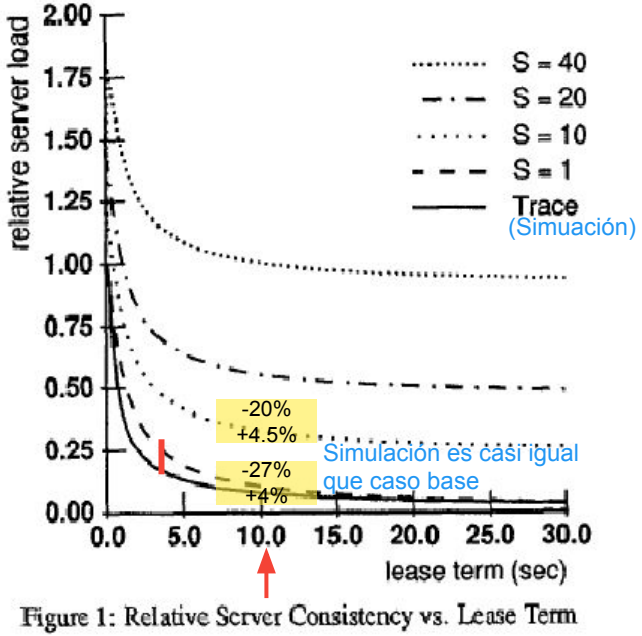


Figure 1: Relative Server Consistency vs. Lease Term

Desempeño esperado

Sistema operativo distribuido V

- Como hay pocos writes, las curvas para diferentes niveles de S son casi iguales
- La mayor parte del beneficio se gana a los pocos segundos
- La monotonicidad de la curva se debe a los altos tiempos de acceso a archivos
- Se esperan resultados parecidos en otros sistemas, salvo en la tasa lecturas/escrituras:
  - Archivos temporales se manejan aparte
  - Traza incluye carga programa y acceso a info de archivos
  - Operaciones de directorio (abrir y cerrar)

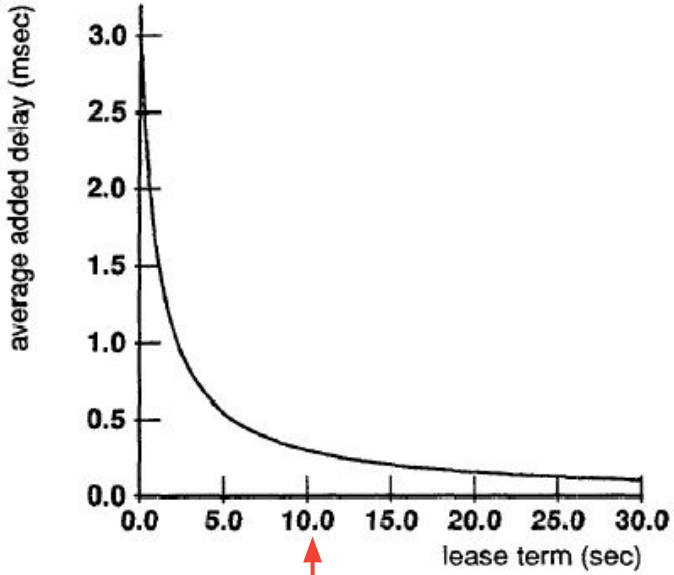


Figure 2: Delay due to consistency.

(to reads/writes)

Aplicabilidad a “futuros” sistemas distribuidos

- Sistemas en zonas de red más grandes:
  - Mayor retraso en comunicaciones: sí afecta
  - Más clientes y servidores: no afecta (S igual)
- Tiempo se vuelve más crítico: extensiones e invalidaciones también
- Procesadores más rápidos: más lecturas/escrituras (rodilla más baja)
- Leases aumenta la razón cliente/servidor (reduciendo el overhead por consistencia), reduciendo el costo de estos sistemas

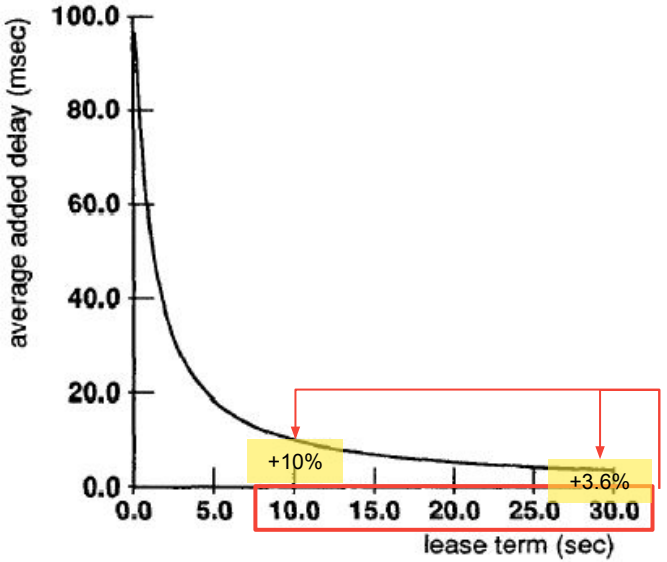


Figure 3: Added delay with 100 ms round-trip time.

Optimización

Manejo del arrendamiento en el servidor

Impacto en términos de carga y retraso

- Servidor puede:
  - Controla el período del arrendamiento
  - Puede esperar a expiración en vez de pedir aprobación
- Cliente puede:
  - Solicitar extensión de arrendamiento
  - Renunciar al arrendamiento
  - Aprobar una escritura

Manejo del arrendamiento en el servidor

Impacto en términos de carga y retraso

- Servidor podría:
  - Optimizar el manejo de archivos instalados (comandos, encabezados y librerías del sistema): muy compartidos, muy leídos y poco escritos:
    - Menos leases para esos archivos: uno por directorio.
    - Extensión periódica de leases para esos archivos para todos:
      - No se necesita llevar registro de los otros arrendadores
      - Elimina retraso en el caché para reads (leases no expiran)
    - Eliminar el lease a un archivo que se modifica:
      - Evita comunicarse
      - No se espera mayor retraso porque el servidor casi siempre tiene que esperar

Manejo del arrendamiento en el servidor

Impacto en términos de carga y retraso

- Servidor podría:
  - Poner un período en función al acceso y al tiempo de propagación al cliente:
    - muy escrito&compartido -> t=0
    - cliente lejano -> t++
    - mejor: elegir el período por cliente/archivo con un análisis
- Cliente podría:
  - Solicitar extensión de arrendamiento: si lo hace antes entonces +carga, -retraso. Clientes inactivos.

Tolerancia a fallos

Tolerancia a fallos

- Leases garantiza consistencia siempre y cuando las fallas no sean bizantinas:
  - Pérdida de mensajes (incluyendo particiones)
  - Fallas de cliente o servidor (si las escrituras persisten en el servidor)
  - La disponibilidad no se reduce (tiempos de expiración)
- Leases depende de relojes físicos:
  - Reloj rápido en servidor: errores si permiten escritura antes de expiración
  - Reloj lento en cliente: seguir usando arrendamiento ya caduco
  - Al revés: no crea inconsistencias pero genera tráfico

Tolerancia a fallos

- Fallas en relojes son menos comunes que los crashes o fallas de comunicación
- Se detectan rápidamente (sincronización o con marcas de tiempo en los mensajes)
- Los relojes deben estar sincronizados (épsilon << t)
  - Es necesario para otras operaciones del acceso a archivos
  - Se requiere un sesgo acotado

Trabajo relacionado

Trabajo relacionado

- Sistemas previos usaban t=0 o t++++
- Andrew t++++:
  - El servidor debe avisar al cliente cuando datos se reescriben
  - Si la comunicación con un cliente falla el servidor continúa con las actualizaciones:
    - Riesgo del cliente de trabajar con info vieja
    - El cliente se da cuenta hasta que se comunica de nuevo
  - Autores sugieren t----:
    - Las actualizaciones se difieren suficiente tiempo para evitar inconsistencias
    - Otros beneficios de tener t----

Trabajo relacionado

- MFS y Echo file system usan tokens:
  - Se consideran leases de período limitado
  - Soportan cachés tipo non-write-through
- NFS no garantiza consistencia
- CFS no permite escritura compartida
- Xerox DFS usaba candados con timeouts: los clientes no usan el timeout y no se enteran cuando los candados se rompen, lease con t=0
- Mirage usa períodos muy grandes
- Sistemas de nombre de dominio: Lampson, pero no pide aprobación para escrituras ni extensiones. No es tan relevante la consistencia porque usan un administrador.

Conclusiones

Conclusiones

- Modelo analítico y paramétrico: dinámico y adaptativo.
  - Carga en el servidor y retraso como función del período
- En general son mejores los períodos cortos: fallas y falsos.
- Se puede extrapolar a sistemas distribuidos de gran escala.
- Leases garantiza consistencia ante fallas no bizantinas:
  - Pérdida de mensajes (incluyendo particiones)
  - Fallas de cliente o servidor (si las escrituras persisten en el servidor)
  - La disponibilidad no se reduce (tiempos de expiración)
- Leases depende de relojes físicos, requiere sincronización

Conclusiones

- Posiblemente se puede aplicar a multiprocesadores de memoria compartida de gran escala.
- Mecanismo basado en el tiempo: puede usarse en otras aplicaciones como manejo de transacciones distribuidas y protocolos de transporte.

Crítica

Crítica

- El modelo es demasiado simple, paramétrico y está enfocado en relativamente poco nivel de compartición
- El análisis del desempeño es aproximado ya que ignora factores importantes como el retraso por colas
- Experiencia limitada del modelo en ambientes y operación real
- El contexto en el que se desarrolló el artículo no tiene las dificultades de hoy: dudoso si se puede aplicar hoy.
- Escalable ayer no significa escalable hoy.

Crítica

- ¿Qué pasa si se revoca un arrendamiento cuando ya se estaba haciendo una transacción pero no ha sido terminada?
- No se discute cómo se logra la sincronización de relojes ni se hace referencia a otros artículos con este fin
- Dice usar relojes físicos: ¿cuáles? el problema no es trivial (Spanner de Google usa relojes atómicos + GPS + protocolos de sincronización)
- No discute el modelo de consistencia que se usa.

Gracias