Francisco Torres-Rojas, Mustaque Ahamad y Michel Raynal

Timed Consistency for Shared Distributed Objects

CARLOS MARTÍN FLORES GONZÁLEZ, Carné: 2015183528

Instituto Tecnológico de Costa Rica
Maestría en Computación
Sistemas Operativos Avanzados
Profesor: Francisco Torres Rojas, Ph.D

Ordenamiento y tiempo son dos aspectos diferentes de consistencia de objectos compartidos en sistemas distribuidos. Uno evita conflictos entre operaciones, el otro dice que tan rápido los efectos de una operación son percibidos por el resto del sistema.

Consistency Criteria: La historia global H de un sistema distribuido es un conjunto parcialmente ordenado de todas las operaciones que ocurren en todos los sitios del sistema. H_i es la secuencia de operaciones que son ejecutadas en el sitio i. Si a ocurre antes que b en H_i decimos que a precede a b en orden de programa. Se asume que todas las operaciones en H son **read** o un **write**. A estas operaciones les toma un tiempo finito ejecutarse, por lo tanto, hay un intervalo que va desde el momento cuando un **read** o un **write** "inicia" al momento cuando alguna de estas operaciones "finaliza". Sin embargo, para los propósitos de la *timed consistency*, se asocia un instante a cada operación llamada el tiempo efectivo (*effective time*) de la operación. El *effective time* de una operación corresponde a algún instante entre su inicio y si fin. Si el tiempo efectivo de a es t, se dice que a fue "ejecutado" en el tiempo t. Si D es un conjunto de operaciones, entonces la serialización de D es una secuencia linear S conteniendo exactamente todas las operaciones de D tal que cada operación **read** para un objecto particular retorna el valor que fue escrito por la operación **write** más reciente. Sea $\sim$ una relación de orden parcial arbitraria definida sobre D, decimos que la serialización S "respeta $\sim$" si $\forall a, b \in$ D tal que $a \sim b$, pasa que a precede a b en S. La historia H satisface linealización (**LIN**) si hay una serialización de H que respeta el orden inducido por los tiempos efectivos de las operaciones. Serialización estricta se define sobre historias formadas por transacciones y requere de la existencia de una serialización de H que respete el orden de tiempo real de las transacciones. Normalidad es equivalente a **LIN** cuando operaciones en objectos son unarias, pero es estrictamente más débil que **LIN** cuando las operaciones pueden abarcar varios objectos. Un nivel más débil de consistencia es ofrecido por consistencia secuencial (**SC**). Este modelo, no garantiza que una operación **read** retorna el valor más reciente con respecto al tiempo real sino solo que el resultado de cualquier ejecución sea el mismo como si las operciones en todos los sitios fuera ejecutados en orden secuencial, y las operaciones de cada sitio individual aparecen en esta secuencia en el orden especificado por su programa. La historia H satisface **SC** si hay una serialización de **H** que respete el orden del programa para cada sitio en el sistema. La relación de causalidad "$\rightarrow$" para sistemas de paso de mensajes puede ser modificada para ordernar las operaciones de H. Sea a, b y $c \in$ H, se dice que $a \rightarrow b$ (a precede causalmente a b) si se da una de las siguientes: (i) a y b son ejecutados en el mismo sitio y a es ejecutado antes que b, (ii) b lee el valor de un objeto escrito por a, (iii) $a \rightarrow c$ y $c \rightarrow b$. Dos operaciones distintas a y b son concurrentes is ninguna de estas condiciones se dan entre ellas. Sea H_{i+w} el conjunto de todas las operaciones en H_i más todas las operaciones **write** en H. La historia H satisface consistencia causal (**CC**) si para cada sitio i existe una serialización de H_{i+w} que respete el orden causal "$\rightarrow$". Así, si $a, b, c \in$ H son tales que a escribe un valor **v** en el objeto X, c lee el mismo valor **v** del objeto X, y b escribe el valor **v'** en el objeto X, nunca se da el caso que $a \rightarrow b \rightarrow c$. **CC** require que todas las relaciones relacionadas con causalidad sean vistas en el mismo order por todos los sitios, mientras que diferentes sitios pueden percibir operaciones concurrentes en órdenes diferentes. **CC** es un modelo de consistencia más débil que **SC** pero puede ser implementado eficientemente.

Timed Consistency. Ni en **SC** ni en **CC**, el tiempo real es explícitamente capturado. En **SC**, operaciones puede aparecer fuer de orden en relación con sus tiempos efectivos. En **CC**, cada sitio puede ver operciones de escritura concurrente en diferente orden. Por otro lado, **LIN** requere que las operaciones sean observadas en un orden que respete su ordenamiento de tiempo real. *Timed consistency* requiere que si el tiempo efectivo de un **write** es t, el valor escrito por esta operación tiene que estar visible a todos los sitios en el sistema distribuido en el tiempo $t + \Delta$, donde Δ es un parámetro de ejecución. Se puede ver que cuando Δ es 0, *timed consistency* se convierte en **LIN**[1]. *Reading on Time:* en modelos basados en tiempo, el conjunto de los valores que un **read** puede retorna está restringuido por la cantidad de tiempo que ha transcurrido

[1] *Timed Consistency* es una generalización (o un debilitamiento) de **LIN**

a partir de los **write** que le preceden. Un **read** ocurre a tiempo (*on time*) si no devuelve datos obsoletos cuando hay valores más recientes que han estado disponibles por más de Δ unidades de tiempo. Esta definición depende las propiedades del reloj usando para assignar marcas de tiempo (*timestamps*) a las operaciones en ejecución. Primero, se asume relojes físicos perfectamente sincronizados en donde $T(a)$ es el instante en tiempo real correspondiente al tiempo efectivo de la operación a. **Definición 1:** Sea $\mathtt{D} \subseteq \mathtt{H}$ un conjunto de operaciones y S una serialización de $\mathtt{D}$. Sea $w, r \in \mathtt{D}$ tal que w escribe un valor dentro de un objeto $\mathtt{X}$ que luego es leído por r (w es la operación **write** más cercana dentro del objeto $\mathtt{X}$ que aparece a la izquierda de r en la serialización S). Se define el conjunto $\mathtt{W}_r$ asociado con r como: $\mathtt{W}_r = \{w' \in \mathtt{D} : (w'$ escribe un valor dentro del objeto $\mathtt{X}) \wedge (T(w) < T(w') < (T(r) - \Delta))\}$. Se dice que la operación r ocurre o lee a tiempo en la serialización S si $\mathtt{W}_r = \emptyset$. S es *timed* si cada operación **read** en S occurre a tiempo. *Approximately-syncronized real-time clocks:* En relojes de tiempo real aproximadamente sincronizados, resincronizaciones periódicas garantizan que no hay dos relojes difieran en más de ε unidades de tiempo. Típicamente también se garantiza que la diferencia entre cualquier reloj y el tiempo "real"[2] nunca es mayor que $\varepsilon/2$. De esta forma, si el tiempo efectivo de una operación a fue reportada por un sitio en particular como $T(a)$, entonces, desde el punto de vista del servidor de tiempo, este tiempo efectivo corresponde a algún instance en el intervalo $[T(a) - \varepsilon/2, T(a) + \varepsilon/2]$. Se dice que $\forall a, b \in \mathtt{H}$, a definitivamente ocurrió antes de b si $T(a) + \varepsilon/2 < T(b) - \varepsilon/2$, o, de forma equivalente si $T(a) + \varepsilon < T(b)$. Si ni a ni b definitivamente ocurrieron antes que el otro, se se dice que sus marcas de tiempo son no comparales o concurrentes. Sea $w, r \in \mathtt{D} \subseteq \mathtt{H}$ tal que w escribe un valor dentro del objecto $\mathtt{X}$ que luego es leído por r. Dejar que $w' \in \mathtt{D}$ actualice también el valor del objeto $\mathtt{X}$. Si $T(w')$ definitivamente ocurrió antses que $T(w)$, o si $(T(r) - \Delta)$ ocurrió definitivamente antes que $T(w')$, entonces el claro que w' no afecta el hecho que r ocurra a tiempo. Ahora, si ambos $T(w')$ y $T(w)$ son concurrentes, o si $(T(r) - \Delta)$ y $T(w)$ son concurrentes, entonces no hay evidencia que w' sea más reciente que w, o de w' ocurriendo en más de Δ unidades de tiempo real antes que r respectivamente. En estos casos, aun se puede argumentar que r ocurre a tiempo. **Definición 2:** sea $\mathtt{D} \subseteq \mathtt{H}$ un conjunto de operaciones y S una serialización de $\mathtt{D}$. Sea $w, r \in \mathtt{D}$ como se presenta en la definición 1. Se asume un conjunto $\mathtt{W}_r$, asociado con r como: $\mathtt{W}_r = \{w' \in \mathtt{D} : (w'$ escribe un valor dentro del objeto $\mathtt{X}) \wedge (T(w) + \varepsilon < T(w')) \wedge (T((w')) \wedge (T(w') + \varepsilon < T(r) - \Delta)\}$. Se dice que una operación r ocurre o lee a tiempo en la serialización S si $\mathtt{W}_r = \emptyset$. S es *timed* si cada operación de **read** en S ocurre a tiempo. *TSC and TCC:* se combinan los requereimientos de los modelos de consistencia bien conocidos como **SC** y **CC** con requerimientos de lectura a tiempo. **Definición 3:** la historia $\mathtt{H}$ safisface *timed serial consistency* (**TSC**) si hay una ***timed*** *serialization* S de $\mathtt{H}$ que respete el orden del programa para cada sition en el sistema. **Definición 4:** la historia $\mathtt{H}$ satisface *timed causal consistency* (**TCC**) si para cada sitio i hay una ***timed*** *serialization* S_i de $\mathtt{H}_{i+w}$ que respete el orden causal "$\rightarrow$".

Applications of Timed Consistency: A pesar de la utilidad y la importancia de modelos de consistencia como **SC** y **CC**, su definición formal y su implementación práctica permite ejecuciones que ignoran la ocurrencia en tiempo real de ciertos eventos. En muchos casos eso podría considerarse una ventaja que permite implementaciones eficientes y convenientes. Por ejemplo, **CC** se ajusta bien a aplicaciones móviles y tiene la habilidad de manejar desconexiones sin problemas. Los modelos de *timed consistency* son útiles en aplicaciones en donde la observancia de paso del tiempo real y sus efectos es parte de la denifición de una definición de correctitud/exactitud. En la misma forma que ocurre en la naturaliza, se puede permitir el paso de un periódo finito antes que los efectos de una operación san conocidos por todos los sitios en el sistema. Este periódo es precisamente el parámetro Δ.

Implementación: *Lifetime Based Consistency Protocol:* esta técnica provee consistencia a través de objetos diferentes pero relacionados. Se asume una arquitectura donde cada objeto tiene un conjunto de sitios-servidor que proveen almacenamiento a largo plazo para el objeto y donde sitios-cliente ponen en caché objetos antes de accederlos. Intentos falldos de caché (*cache misses*) son manejados por cada sitio-servidor, que tiene una copia del objeto solicitado o puede obtenerlo. Sea el conjunto C_i denotar el caché del sitio i, en donde se almacena copias de objetos que han sido accesados recientemente. Si un *cache miss* ocurre cuando se accede el objeto X, algún servidor provee una copia de su versión actual de X. Una vez que esta copia es almacenada en C_i, se denota como X_i. El tiempo de inicio de X_i, denotado como X_i^α, es el momento cuando el valor X_i fue escrito. El último momento en que se conoce el valor almacenado en X_i se conoce como su tiempo de finalización y se denota con X_i^ω. El intervalo $[X_i^\alpha, W_i^\omega]$ es conocido como el tiempo de vida (*lifetime*) del valor almacenado en X_i. Los valores X_i y Y_i (en el caché C_i) son mutuamente consistenctes si $\text{máx}(X_i^\alpha, Y_i^\alpha) \leq \text{mín}(X_i^\omega, Y_i^\omega)$ - sus tiempos de vida se sobreponen y por lo tanto, coexisten en algún instance . C_i es consistente si el tiempo máximo de inicio de cualquier valor de objeto en C_i es menor o igual que el tiempo mínimo de finalización de cualquier valor de objecto en C_i - cada par de objetos en C_i es mutuamente consistente. En general, el tiempo de vida de valores de objetos arbitrarios no se conoce. Cuando el sitio i actualiza la versión del objeto X_i en el tiempo t, la marca de tiempo t se asigna a ambos X_i^α y X_i^ω. Se tiene que descubrir conforme se avanza que ninguna copia $X_j (i \neq j)$ ha sido sobreescrita y usa esta información para avanzar X_i^ω. Una variable de la marca de tiempo local llamada $\mathbf{Context}_i$ es asociada con C_i. Su valor inicial es 0, y es actualizada con las siguiente reglas: 1. Cuando una copia del objeto X se introduce en C_i (convirtiendose en X_i): $\mathbf{Context}_i \coloneqq \text{máx}(X_i^\alpha, \mathbf{Context}_i)$. 2. Cuando la copia del objeto X_i se actualiza en el tiempo t: $\mathbf{Context}_i \coloneqq X_i^\alpha \coloneqq t$. $\mathbf{Context}_i$ mantiene el último tiempo de inicio de cualquier valor de objeto que está o ha sido almacenado en C_i. Cuando

[2]Mantenido por un servidor de tiempo

una copia del objeto X se introduce en C_i, su valor final no debe ser menor que $\mathbf{Context}_i$, si es necesario, otros sitios-servidor o sitios-cliente con contactados hasta que una versión de X que satisface la condición se encuentra. Además, cualquier objeto $Y_i \in C_i$ tal que $Y_i^{\omega} < \mathbf{Context}_i$ es invalidado. Se ha probado que este protocolo induce **SC** en la ejecución. *TSC implementation:* en el protocolo *lifetime*, el sitio i accede un estado de los objetos que fue consistente en el tiempo $\mathbf{Context}_i$, pero el tiempo actual puede ser mucho más tarde. Al controlar que la diferencia entre el tiempo actual y $\mathbf{Context}_i$ es mejor que o igual a Δ, se induce **TSC**. Los valores de los objetos cuyo tiempo de finalización es mayor/más viejo que Δ unidades de tiempo real en el pasao son invalidados localmente. Esto se lleva a cabo agregando una regla extra para actualizar el $\mathbf{Context}_i$ (Sea t_i el tiempo actual en el sitio i): 3. $\mathbf{Context}_i := \text{máx}(t_i - \Delta, \mathbf{Context}_i)$. Esto trabaja muy bien con objetos cuyos tiempos de finalización son conocidos pero puede llegar a generar invalidaciones innecesarias para objetos arbitrarios cuyos tiempos de vida no son conocidos de forma precisa. *TCC Implementation:* todas las marcas de timpo usadas en el sistema son tomadas de relojes vectoriales o plausibles. Se hacen variates al protocolo *lifetime* para no requerir de relojes físicos y generar menor *overhead* en las comunicaciones que **SC**. $\mathbf{Context}_i$ se actualiza con versiones de las reglas 1 y 2[3] adaptadas a relojes lógicos. Entre otras cosas, esto requiere el cómputo del máximo y el mínimo de dos marcas de tiempo lógicas que fueron tomadas yas sea de relojes vetoriales o plausibles. Cuando una copia de un objeto es traído desde el servidor a C_i, se verifica que su tiempo de finalización no sea causalmente antes de $Context_i$. De ser necesario, otros sitios-servidor o sitios-cliente son contactados hasta que una vesion del objeto que satisfaga esta condición se encuentre. Cualquier objeto $Y_i \in C_i$ cuyo tiempo de finalización es causalmente antes de $\mathbf{Context}_i$ ($Y_i^{\omega} \rightarrow \mathbf{Context}_i$) es invalidado. Copias locales de objetos podrían ser invalidadas cuando un nuevo objeto es traido hacia el caché pero nunca son invalidadas como consecuencia de la actualización de un objeto local poque el tiempos de finalización lógicos de las copias locales se avanzan/adelantan juntos con el tiempo lógico del sitio. Para garantizar que los efectos de una operación de **write** sea observada luego de Δ unidades de tiempo real, se agrega una nueva marca de tiempo tomada del reloj físico: el *checking time* de X_i, denotado como X_i^{β}. Esta marca de tiempo es el último instante de tiempo real en que el valor almacenado en X_i se conoce como válido. Sea t_i el tiempo real del sitio i. Cuando una copia del objeto X se trae desde el servidor s hacia C_i, ahora también se requiere que $X_i^{\beta} \geq t_i - \Delta$. De forma similar, cualquier objeto local $Y_i \in C_i$ tal que $Y_i^{\beta} < t_i - \Delta$ es invalidado o marcado como antiguo. Estas reglas garantizan que el sitio siempre lee a tiempo, porque los valores de los objetos que se sospechan que son antiguos son o validados o reemplazados por versiones nuevas. *TCC using logical clocks:* **TCC** puede ser aproximado en un sistema distribuido cuyos sitios solo comparten un reloj lógico. Se define un mapa ξ del conjunto de marcas de tiempo lógicas al conjunto de números reales. El parámetro Δ no será expresado en unidades de tiempo real, sino como un número real que define la máxima diferencia entre los valores que ξ produce para ciertas marcas de tiempo. **Definición 5:** la función ξ mapea marcas de tiempo tomadas de un reloj lógico al conjunto de números reales. Sea t y u dos marcas de tiempo lógicas, ξ tiene las siguientes propiedades: (1) $t = u \Rightarrow \xi(t) = \xi(u)$, (2) $t \rightarrow u \Rightarrow \xi(t) < \xi(u)$. Informalmente, $\xi(t)$ representa la "cantidad" de actividad global del sistema que es conocido cuanod el evento asociado con la marca de tiempo t es generada. Incluso si las marcas de tiempo t y u son concurrentes, se desea que si en el tiempo u el sistema está conciente de mayor actividad global que en el tiempo t, entonces $\xi(u) > \xi(t)$. **Definición 6:** Sea $\mathtt{D} \subseteq \mathtt{H}$ un conjunto de operaciones y S una serialización de $\mathtt{D}$. Sea $w, r \in \mathtt{D}$ como se presenta en la definición 1. Se define el conjunto $\mathtt{W}_r$, asociado a r como: $\mathtt{W}_r = \{w' \in \mathtt{D}: (w' \text{ escribe un valor dentro del objeto } X \wedge (\xi(L(w)) < \xi(L(w')) < (\xi(L(r)) - \Delta))\}$. Se dice que la operación r ocurre o lee a tiempo en la serialización S, si $\mathtt{W}_r = \emptyset$. S es *timed* si cada operación **read** en S ocurre a tiempo. Por lo tanto, *timed consistency* requer que si una operación **write** se ejecuta en el tiempo lógico t, tiene que estar visible en el sitio i antes de $\xi(t_i) - \xi(t) > \Delta$, donde t, es el tiempo lógido del sitio i

1. ¿CUÁL ES EL PROBLEMA QUE PLANTEA EL *PAPER*?

Los modelos de consistencia como consistencia secuencial y la consistencia causal no consideran el tiempo/momento particular en que una operación es ejecutada para establecer un orden válido enter todas las operaciones de una computación.

2. ¿POR QUÉ EL PROBLEMA ES INTERESANTE O IMPORTANTE?

Accesos no locales a los objetos en sistemas distribuidos es caro debido a los costos elevados en la comunicación. Con el fin de aliviar esta ineficiencia y para mejorar la confianza, disponibilidad y escalabilidad del sistema, técnicas como *caching* y replicación son normalmente usadas. Sin embargo, estos métodos conducen a la coexistencia de muchos y posiblemente diferentes versiones del mismo objeto.

En muchos modelos de consistencia, el tiempo real no es explícitamente capturado. Por ejemplo, consistencia secuencial solo requiere que sea posible serializar todas las operaciones ejecutadas en el sistema distribuido. Consistencia causal, un modelo más débil, solo requiere que las operaciones relacionadas con causalidad sean vistas en el mismo orden por todos los sitios del sistema, mientras operaciones concurrentes puede ser percibidas en orden diferente por diferentes sitios. Por otro lado, serialización estricta y linealización requieren que las serializaciones respeten el ordenamiento en tiempo real entre las transacciones y operaciones respectivamente.

[3]Del protocolo *lifetime*

3. ¿QUÉ OTRAS SOLUCIONES SE HAN INTENTADO PARA RESOLVER ESTE PROBLEMA?

La noción de tiempo ha sido explorada también en sistemas de comunicación como *delta causal brodcast*, en sistemas de memoria que tienen un modelo de consistencia temporal y en varios protocolos de caché para la Web.

La simulación de ambientes virtuales interactivos se ha explorado por Singla, donde el concepto de "consistencia delta" es definida como un criterio de corrección para controlar accessos a una memoria compartida.

La noción de causalidad-Δ en redes no confianbles la presenta Baldoni. Este protocolo soporta aplicaciones colaborativas multimedia en tiempo real. Con el fin de entregar un mensaje a un proceso, se verifica que el orden causal de los mensajes se cumple.

4. ¿CUÁL ES LA SOLUCIÓN PROPUESTA POR LOS AUTORES?

Este problema se aborda definiendo modelos de consistencia convenientes para el estado del los objetos distribuidos. Un modelo de consistencia establece una serie de garantías acerca de las relaciones válidas entre las operaciones ejecutadas por los sitios en el sistema, que a su vez limitan los valores posibles que objetos compartidos pueden retornar cuando operaciones de **read** son ejecutadas.

Se propone un modelo de **timed consistency** que define un umbral aceptable máximo de tiempo después de lo cual los efectos de una operación **read** deben ser observados por todos los sitios de un sistema distribuido. *Timed consistency* no solo cumple las necesidades de muchas aplicaciones que deben observar actualizaciones para cambiar dinamicamente objetos en el momento oportuno, sino que también unifica modelos de consistencia existentes como consistencia secuencial y linealización. Se concentra en modelos de *timed serial consistency* (**TSC**) y *timed causal consistency* (**TCC**).

5. ¿QUÉ TAN EXITOSA ES ESTA SOLUCIÓN?

Al combinar los requerimientos de *timed consistency* con criterios con **SC** y **CC** se propone **TSC** y **TCC**, los cuales son muy adecuados para cumplir las necesidades de muchas aplicaciones. Además, este concepto también unifica modelos existentes de consistencia. Se explorar una posible implementación de **TSC** y **TCC** basado en el concepto de tiempos de vida en los valores de los objetos. Finalmente, una posible definición de **TCC** usando solo relojes lógicos fue presentado.

Algunos problemas de esta investigación seran abordados como parte de un trabajo futuro. Con el fin de tener un mejor entendimiento la relación entre el valor de Δ y el costo de lograr un nivel particular de oportunidad (*timeliness*), se estan completando detalladas simulaciones de sustemas basadas en los criterios de consistencia descritos en el artículo. Otras posibles implementaciones de **TSC** y **TCC** tiene que ser consideradas. Para el caso de **TCC** usando solo relojes lógicos, es necesario explorar diferentes mapeos de marcas de tiempo lógicas a números reales, y proveer las con semántica apropiada para la selección del parámetro Δ.