Apuntes – Recuperación ante Fallas - Logging

Nota:
El siguiente apunte constituye sólo un apoyo para las clases prácticas del tema de recuperación ante fallas. Los algoritmos están esquematizados a alto nivel y no tienen como objetivo una rigurosidad formal. A tal fin se deberá consultar la bibliografía de la materia[1]*. Les agradecemos informar de cualquier error encontrado o sugerencias a bddoc@dc.uba.ar, con tal de ir mejorando este apunte.*

En este apunte cubriremos tres métodos para la recuperación antes fallas: *Undo*, *Redo*, y *Undo/Redo*.

Sección 1: Undo

En esta sección desarrollaremos el método Undo, con sus tres variantes. Los registros de las transacciones siguen el formato <T,x,v> con la siguiente semántica: la transacción *T* actualizó el dato *x* y su valor previo era *v*. Para este método, sólo es necesario mantener para cada dato el valor anterior en cada modificación.

Con respecto al manejo de entradas en el log, la política a seguir es la siguiente:

- Regla Uno: Los registros del tipo <T,x,v> se escriben a disco **antes** que el nuevo valor de *x* se escriba a disco en la base de datos.
- Regla Dos: Los registros <Commit T> se escriben a disco **después** que todos los elementos modificados por T se hayan escrito en disco.

A continuación presentamos los pasos a seguir para recuperarse ante una falla con las tres variantes mencionadas.

Sección 1.1 Undo Sin checkpoints.

Paso Uno:
Dividir las transacciones en completas e incompletas. Las completas son aquellas para las cuales encontramos un <Start T> y un <Commit T>, como también aquellas con <Start T> y <Abort T>; en cambio aquellas para las cuales encontramos solamente el <Start T> son las incompletas.

Paso Dos:
Las transacciones completas son ignoradas. Esto se debe a que si encontramos el <Commit T>, entonces sabemos por la segunda regla que todos los elementos que T modificó ya fueron grabados a disco. Si encontramos un <Abort T>, sus acciones también ya fueron manejadas. En cambio, para las incompletas no podemos asegurar nada. Luego, se recorre el log desde el registro más reciente hasta el principio (de atrás hacia delante) y para cada registro <T,x,v> donde T es una transacción no comiteada, se deshace esa modificación (al dato x se le vuelve a asignar el valor previo a la

[1] Este apunte está basado en el capítulo 17 del libro de García Molina, Ullman y Widom: "DataBase System: The complete book".

modificación almacenado en v). Finalmente, se agrega un <Abort T> para cada transacción incompleta, y se efectúa un *Flush*[2] del Log.

Sección 1.2 Undo con Checkpoint Quiescente.

La técnica anterior tiene la desventaja que ante cada falla se debe recorrer el log en su totalidad. Una optimización se logra introduciendo checkpoints, de manera de ahorrarnos tiempo al recorrer el log. La política para el checkpoint quiescente es la siguiente:

1. Dejar de aceptar nuevas transacciones.
2. Esperar a que todas las transacciones activas o bien comiteen o aborten.
3. Escribir un registro <CKPT> en el log y luego efectuar un flush.
4. Aceptar nuevas transacciones.

Se llama quiescente porque la base de datos está "quieta" mientras se produce el checkpoint. Los pasos a seguir ante una falla son idénticos a la técnica anterior, sin embargo, no debo seguir hasta el principio del log, sino hasta que encuentre un registro <CKPT>. Es seguro detenernos aquí, ya que las transacciones activas al momento del checkpoint ya abortaron o comitearon, y no es necesario efectuar cambios sobre sus acciones.

Sección 1.3 Undo con Checkpoint no-Quiescente.

Si bien la sección anterior presenta una mejora, tiene una desventaja importante debido a que es necesario "parar" la base de datos para efectuar el checkpoint. Para solucionar esta situación, surge esta extensión denominada checkpoint no quiescente. Bajo este mecanismo, la política al introducir un checkpoint es la siguiente:

1. Escribir un registro <Start CKPT(T_1,T_2,…,T_k)> en el log, y efectuar un flush. T_1,T_2,…,T_k son las transacciones activas (aquellas con <START T> y sin <Commit T>) al momento de introducir el checkpoint.
2. Esperar a que todas las transacciones T_1,T_2,…,T_k terminen (ya sea abortando o comiteando).
3. Escribir el registro <End CKPT> en el log y efectuar un flush.

Las acciones de recuperación son las siguientes. Nuevamente, el objetivo es deshacer las acciones de las transacciones incompletas, y escribir un <Abort T> para cada una de ellas. Empezamos desde el último registro hacia atrás, y tenemos dos opciones para ver donde finalizar:

- Si encontramos un <End CKPT>, sabemos por la segunda condición que todas las transacciones que estaban activas al momento del Start CKPT terminaron. Luego, no es necesario controlar registros anteriores al Start CKPT. Por lo tanto, finalizo las acciones una vez encontrado el Start CKPT.
- En cambio, si no encontramos el registro <End CKPT>, existe al menos una transacción activa al momento del Start CKPT todavía no comiteó, por lo que debemos deshacer todas sus acciones, hasta encontrar su registro <Start T>.

[2] Flush es la operación que graba todo el log a disco.

Entonces en este caso debemos retroceder hasta el <Start T> más antiguo de las transacciones pertenecientes a $T_1,T_2,\ldots,T_k$ que no haya comiteado.

Sección 2: Redo

Para este mecanismo solo veremos la versión sin checkpoints, y la versión con checkpoints no quiescente.

La técnica Redo es muy similar a la técnica Undo, pero con otro enfoque. Los registros de las transacciones mantienen el formato <T,x,v> , pero con un cambio importante en la semántica. El registro debe leerse como: la transacción *T* actualizó el dato *x* y su valor pasa ahora a ser *v*. Para este método, sólo es necesario mantener para cada dato su nuevo valor en cada modificación, en vez de mantener el valor anterior.

Con respecto al manejo de entradas en el log, la política a seguir es la siguiente:

- Regla Uno: Los registros del tipo <T,x,v> se escriben a disco **antes** que el nuevo valor de *x* se escriba a disco en la base de datos.
- Regla Dos: Los registros <Commit T> se escriben a disco **antes** que todos los elementos modificados por T se hayan escrito en disco.

Es decir, que bajo este enfoque, primero se escribe el <Commit T> en el log en disco, y luego sus modificaciones son volcadas a la base de datos. Esto trae cambios en las estrategias de recuperación como veremos a continuación.

Sección 2.1 Redo Sin checkpoints.

Paso Uno:
Dividir las transacciones en completas e incompletas, según el criterio definido anteriormente.

Paso Dos:
Las transacciones incompletas las podemos ignorar por completo, ya que estamos seguros por la regla dos que sus datos nunca llegaron a disco. De la misma manera, aquellas transacciones completas debido a que se encontró un <Abort T> pueden ser ignoradas porque sus acciones ya fueron manejadas previamente.

En cambio, para aquellas con <Start T> y <Commit T>, no estamos seguros del todo que sus cambios se hayan registrado en la base de datos. Luego, el objetivo de este enfoque se basa en rehacer todas las acciones de las transacciones comiteadas. Por lo tanto, se recorre el log desde el registro más antiguo hasta el final y para cada registro <T,x,v> donde T es una transacción de este tipo, se rehace esa modificación (al dato x se asignar nuevamente el valor almacenado en v). Finalmente, se agrega un <Abort T> para cada transacción incompleta, y se efectúa un *Flush* del Log.

Sección 2.2 Redo con Checkpoint no-Quiescente.

La política para los Start CKPT y End CKPT son las siguientes:

- Escribir un registro <Start CKPT(T_1,T_2,…,T_k)> en el log, y efectuar un flush. T_1,T_2,…,T_k son las transacciones activas (aquellas con <START T> y sin <Commit T>) al momento de introducir el checkpoint.
- Esperar a que todas las modificaciones realizadas por transacciones ya comiteadas al momento de introducir el Start CKPT sean escritas a disco.
- Escribir el registro <End CKPT> en el log y efectuar un flush.

(*Notar que no es necesario esperar que las transacciones activas terminen para introducir el End CKPT.*)

Las acciones de recuperación son las siguientes. Como siempre, el objetivo es rehacer las acciones de las transacciones comiteadas (aquellas con <Start T> y <Commit T>), y escribir un <Abort T> para cada una de las incompletas. Aquellas con <Start T> y <Abort T> son ignoradas. La cuestión es desde donde comenzamos. Tenemos dos opciones:

- Si encontramos un <End CKPT>, podemos ignorar todas aquellas transacciones que comitearon previamente al último registro Start CKPT, ya que por la segunda condición, sus datos ya fueron escritos a disco y no es necesario rehacerlos. Luego, solo nos concentramos en las transacciones activas al momento del checkpoint y las que comenzaron después. Por lo tanto, debemos comenzar desde el start más antiguo de las transacciones a rehacer (observar que si se trata de una de las transacciones pertenecientes a T_1,T_2,…,T_k puede ser necesario retroceder más allá del Start CKPT).
- Si en cambio, si no encontramos un registro <End CKPT> la situación es más complicada. Como no sabemos con certeza si las modificaciones de las transacciones ya comiteadas al momento del Start CKPT fueron escritas a disco, debemos retroceder hasta su comienzo, para rehacer de ahí en más. Para esto, se ubica el registro <END CKPT> anterior y su correspondiente <Start CKPT(S_1,S_2,…,S_k)> y rehacer todas las transacciones comiteadas que comenzaron a partir de ahí o pertenecen S_i. Es decir, si una o más transacciones de las S_i comiteó, deberá empezar desde el start más antiguo de ellas.

Sección 3: Undo/Redo.

En esta sección desarrollaremos el método Undo/Redo. Este mecanismo es el más flexible de los tres, con el costo de mantener un ítem más en los registros del log. Los registros de las transacciones siguen el formato <T,x,v,w> con la siguiente semántica: la transacción *T* actualizó el dato *x* , su valor previo era *v* y su valor nuevo es *w*. Como vemos, es necesario mantener tanto el valor anterior como el nuevo valor en cada modificación.

Con respecto al manejo de entradas en el log, la política a seguir es la siguiente:

- Regla Uno: Los registros del tipo <T,x,v,w> se escriben a disco **antes** que el nuevo valor de *x* se escriba a disco en la base de datos.

La flexibilidad que nos brinda el mecanismo en cuanto al momento de escribir el registro commit de cada transacción da la posibilidad de encontrarnos con transacciones comiteadas cuyo datos todavía no fueron escritos a disco como también transacciones no comiteadas cuyos datos si fueron escritos a disco. Por lo tanto, el objetivo es deshacer las transacciones incompletas y rehacer aquellas con <Start T> y <Commit T> (como siempre, aquellas con <Start T> y <Abort T> son ignoradas). En otras palabras, debemos **combinar los mecanismos de Undo y Redo**, en el siguiente orden:

Primero, deshacer las transacciones incompletas desde el último registro hasta el principio (esta situación puede variar si se introducen checkpoints, como vimos para el caso undo), y luego rehacer las transacciones comiteadas desde el principio (también puede variar si se introducen checkpoints, como vimos para el caso redo) hasta el final.

Veamos en detalle cada alternativa:

Sección 3.1 Undo/Redo Sin checkpoints.

Aquí los pasos a seguir son los descriptos previamente. Es decir, debemos primero, deshacer las transacciones no comiteadas desde el último registro hasta el principio y luego rehacer las transacciones comiteadas desde el principio hasta el final. Finalmente se agrega un registro abort para cada transacción no comiteada, y se hace un flush del log.

Sección 3.2 Undo/Redo con checkpoints no quiescente.

La política para los Start CKPT y End CKPT es la siguiente:

- Escribir un registro <Start CKPT (T_1, T_2,…, T_k)> en el log, y efectuar un flush. T_1,T_2,…,T_k son las transacciones activas (aquellas con <START T> y sin <Commit T>) al momento de introducir el checkpoint.
- Esperar a que todas las modificaciones realizadas por transacciones (comiteadas o no) al momento de introducir el Start CKPT sean escritas a disco.
- Escribir el registro <End CKPT> en el log y efectuar un flush.

(*Notar que no es necesario esperar que las transacciones activas terminen para introducir el End CKPT.*)

El objetivo para la recuperación es el mismo: deshacer primero las transacciones no comiteadas y rehacer aquellas comiteadas.

Más allá de que encontremos un registro <End CKPT> o no, para deshacer las transacciones incompletas deberemos retroceder hasta el start más antiguo de ellas. Con las transacciones a rehacer en cambio, la situación puede cambiar. Si encontramos un registro <End CKPT> sólo será necesario rehacer las acciones efectuadas desde el

registro Start CKPT en adelante. Si no encontramos el <End CKPT>, para saber desde dónde comenzar utilizamos el mecanismo usado para el mismo caso en la estrategia Redo (Sección 2.2). Como siempre se agrega el registro abort para cada transacción incompleta y se hace un flush del log.

Sección 4: Ejemplos

En esta sección presentamos ejemplos que ilustran los mecanismos desarrollados anteriormente.

Sección 4.1: Ejemplo Undo sin checkpoints
Supongamos el siguiente estado del log:

# Paso	Registro
1	<Start T>
2	<T,A,8>
3	<T,A,16>
4	**<T,B,8>**
5	**<Commit T>**

Situación 1: Acontece un crash y el último registro del log es: <T,B,8> (Paso 4)
La única transacción involucrada es T. Como no encontramos el registro commit para T, la clasificamos como no comiteada. Luego, todas sus acciones deben deshacerse, desde el paso 4 (el último anterior al crash) hasta el principio del log. Luego, obtenemos las siguientes acciones al aplicar el mecanismo Undo.
***Transacciones a deshacer**: T.*
***Cambios en la base de datos**:*
B:=8 (deshaciendo el paso 4)
A:=16(deshaciendo el paso 3)
A:=8(deshaciendo el paso 2)

Llegamos al final y no hay más acciones a deshacer. Valores finales:

Item	Valor Final
A	8
B	8

(Notar que es importante deshacer desde el último registro hacia el principio. Si hubiéramos registrado el log desde el principio hasta el final, el valor final de A hubiera sido 16, lo cual es incorrecto.)

***Cambios en el log**: Se* agrega el registro <Abort T> y luego Flush Log.

Situación 2: Acontece un crash y el último registro del log es: <Commit T> (Paso 5)
Como T es una transacción comiteada, ignoramos todas sus acciones. En este caso, es la única transacción, por lo que no se efectúan cambios en la recuperación.

Sección 4.2: Ejemplo Undo con checkpoints quiescentes.
Supongamos el siguiente estado del log:

# Paso	Registro
1	<Start T1>
2	<T1,A,5>
3	<Start T2>
4	<T2,B,10>

Si decidimos en este momento introducir un CKPT, se dejan de aceptar nuevas transacciones, se espera que terminen T1 y T2 y se introduce el registro CKPT. Una posible continuación es la siguiente:

# Paso	Registro
1	<Start T1>
2	<T1,A,5>
3	<Start T2>
4	<T2,B,10>
5	<T2,C,15>
6	<T1,D,20>
7	<Commit T1>
8	<Abort T2>
9	CKPT
10	<Start T3>
11	<T3,E,25>
12	<T3,F,30>

Observar que el registro CKPT recién puede introducirse una vez que terminan T1 y T2. Supongamos ahora que ocurre un crash en este momento. T2 es ignorada debido a que se encuentra un <Abort T2>. Se identifica T3 como la única transacción a deshacer. Recorremos desde el final hasta el último CKPT, en el paso 9.
***Transacciones a deshacer**: T3.*

***Cambios en la base de datos**:*
F:=30 (deshaciendo el paso 12)
E:=25(deshaciendo el paso 11)

***Cambios en el log**: Se* agrega el registro <Abort T3> y luego Flush Log.

Sección 4.3: Ejemplo Undo con checkpoints no quiescentes.
Se tiene un estado de log como el siguiente:

# Paso	Registro
1	<Start T1>
2	<T1,A,5>
3	<Start T2>
4	<T2,B,10>
5	<Start CKPT T1,T2>
6	<T2,C,15>
7	<Start T3>
8	<T1,D,20>
9	<Abort T1>
10	**<T3,E,25>**
11	<Commit T2>
12	<End CKPT>
13	**<T3,F,30>**

En el paso 5 se introduce el registro Start CKPT, recordando las transacciones activas hasta ese momento (T1 y T2). El registro <End CKPT> recién puede ser introducido una vez terminadas estas transacciones (paso 12).

Analicemos las siguientes situaciones para ejemplificar este mecanismo:
Situación 1: Acontece un crash y el último registro del log es: <T3,F,30> (Paso 13).

T1 es ignorada debido al registro Abort del paso 9. La única transacción no comiteada es T3, por lo cual sus acciones deben deshacerse. Como encontramos un registro <End CKPT> (paso 12), caemos en el caso más simple, por lo que sólo debemos retroceder hasta el Start CKPT en el paso 5. (En realidad, como T3 comienza en el paso 7, sólo debemos retroceder hasta ese registro).

***Transacciones a deshacer**: T3.*
***Cambios en la base de datos**:*
F:=30 (deshaciendo el paso 13)
E:=25 (deshaciendo el paso 10)
***Cambios en el log**: Se* agrega el registro <Abort T3> y luego Flush Log.

Situación 2: Acontece un crash y el último registro del log es: <T3,E,25> (Paso 10).
En este caso, las transacciones a deshacer son T2 y T3. Como no encontramos un registro <End CKPT>, no nos alcanza con retroceder hasta el Start CKPT, sino que debemos retroceder hasta el start más antiguo entre T2 y T3, que es el Start de T2 en el paso 3.
***Transacciones a deshacer**: T2 y T3.*
***Cambios en la base de datos**:*
E:=25 (deshaciendo el paso 10)
C:=15 (deshaciendo el paso 6)
B:=10 (deshaciendo el paso 4)
***Cambios en el log**: Se* agregan los registros <Abort T3> y <Abort T2> y luego Flush Log.

Sección 4.4: Ejemplo Redo sin checkpoints

Supongamos el siguiente estado del log:

# Paso	Registro
1	<Start T>
2	<T,A,16>
3	<T,A,32>
4	**<T,B,16>**
5	**<Commit T>**

Situación 1: Acontece un crash y el último registro del log es: <T,B,16> (Paso 4)

La única transacción involucrada es T. Como no encontramos el registro commit para T, la clasificamos como no comiteada. Luego, como solo nos concentramos en la comiteadas, podemos ignorar completamente a T.

Transacciones a rehacer*: -.*
Cambios en la base de datos*:-.*
Cambios en el log*: Se* agrega el registro <Abort T> y luego Flush Log.

Situación 2: Acontece un crash y el último registro del log es: <Commit T> (Paso 5). Ahora T es una transacción comiteada, y sus acciones deben rehacerse, ya que no estamos seguros que sus modificaciones hayan sido escritas en disco.

Luego, obtenemos las siguientes acciones al aplicar el mecanismo Redo.
Transacciones a rehacer*: T.*
Cambios en la base de datos*:*
A:=16(rehaciendo el paso 2)
A:=32(rehaciendo el paso 3)
B:=16 (rehaciendo el paso 4)

Llegamos al final y no hay más acciones a deshacer. Valores finales:

Item	Valor Final
A	32
B	16

(Notar que es importante rehacer desde el principio del log hacia el final. De efectuar el camino inverso, el valor final de A hubiera sido 16, lo cual es incorrecto.)

Sección 4.5: Ejemplo Redo con checkpoints no quiescentes.
Supongamos el siguiente estado del log:

# Paso	Registro
1	<Start T1>
2	<T1,A,5>
3	<Start T2>
4	<Commit T1>
5	<T2,B,10>
6	<Start CKPT T2>
7	<T2,C,15>
8	<Start T3>
9	**<T3,D,20>**
10	<End CKPT>
11	**<Commit T2>**
12	**<Commit T3>**

Analicemos las siguientes situaciones:
Situación 1: Sucede un Crash y el último registro es <Commit T2> (Paso 11).
Las transacciones a rehacer son T1 y T2. Sin embargo, puedo ignorar T1 ya que encuentro un <End CKPT> (paso 10). Debo empezar a recorrer entonces desde el start de T2 en el paso 3.
***Transacciones a rehacer**: T2 (ignoro T1).*
***Cambios en la base de datos**:*
B:=10(rehaciendo el paso 5)
C:=15(rehaciendo el paso 7)
***Cambios en el log**: Se* agrega el registro <Abort T3> y luego Flush Log.

Situación 2: Sucede un Crash y el último registro es <Commit T3> (Paso 12).
Nuevamente en este caso encontramos un <End CKPT>, por lo que solo nos debemos concentrar en las transacciones activas al momento del Start CKPT y en las que comenzaron después. En este caso, identificamos a T2 y T3 como las transacciones a rehacer. Debo empezar desde el start más antiguo (el de T2 en el paso 3).
***Transacciones a rehacer**: T2 y T3 (ignoro T1).*
***Cambios en la base de datos**:*
B:=10(rehaciendo el paso 5)
C:=15(rehaciendo el paso 7)
D:=20(rehaciendo el paso 9)
***Cambios en el log**: -*

Situación 3: Sucede un Crash y el último registro es <T3, D, 20> (Paso 9).
En este caso, no encontramos un <End CKPT>, por lo que no estamos seguros que las modificaciones realizadas por transacciones comiteadas al momento del Start CKPT hayan sido escritas a disco. En este caso, T1 se encuentra en esta situación. Debemos retroceder hasta el Start CKPT anterior, y ver las transacciones activas a ese momento. En este caso, no hay un registro de chekpoint anterior, por lo que debemos retroceder hasta el principio del log.

***Transacciones a rehacer**: T1.*
***Cambios en la base de datos**:*
A:=5(rehaciendo el paso 2)
***Cambios en el log**: : Se* agregan los registros <Abort T3> y <Abort T2> y luego Flush Log.

Sección 4.6: Ejemplo Undo/Redo sin checkpoints.
Sea el siguiente estado del log:

# Paso	Registro
1	<Start T1>
2	<T1,A,8,16>
3	<Start T2>
4	<T2,B,8,16>
5	**<Commit T2>**
6	**<Commit T1>**

Analicemos las siguientes situaciones:

Situación 1: Sucede un Crash y el último registro es <Commit T2> (Paso 5).
Identificamos a T2 como una transacción a rehacer, y a T1 como una transacción a deshacer.
***Transacciones a rehacer**: T2.*
***Transacciones a deshacer**: T1.*

***Cambios en la base de datos**:*
A:=8 (deshaciendo el paso 2)
B:=16 (rehaciendo el paso 4)

***Cambios en el log**: : Se* agrega el registro <Abort T1> y luego Flush Log.

Nota: recordar que primero se deshacen las transacciones no comiteadas y luego se rehacen las comiteadas.

Situación 2: Sucede un Crash y el último registro es <Commit T1> (Paso 6).
En este caso, ambas transacciones deben rehacerse.

***Transacciones a rehacer**: T1 y T2.*
***Transacciones a deshacer**: -.*

***Cambios en la base de datos**:*
A:=16 (rehaciendo el paso 2)
B:=16 (rehaciendo el paso 4)

***Cambios en el log**: : -.*

Sección 4.7: Ejemplo Undo/Redo con checkpoints no quiescentes.

Supongamos el siguiente estado para el log:

# Paso	Registro
1	<Start T1>
2	<T1,A,4,5>
3	<Start T2>
4	<Start T9>
5	<T9,X,9,90>
6	<Abort T9>
7	<Commit T1>
8	<T2,B,9,10>
9	<Start CKPT T2>
10	<T2,C,14,15>
11	<Start T3>
12	**<T3,D,19,20>**
13	<End CKPT>
14	**<Commit T2>**
15	**<Commit T3>**

Veamos cómo funciona el mecanismo en las siguientes situaciones:

Situación 1: Sucede un Crash y el último registro es <Commit T3> (Paso 15):
Al encontrar un registro Abort para T9, la descartamos. Por otro lado, identificamos a T1, T2 y T3 como transacciones a rehacer, y ninguna transacción para deshacer. Sin embargo, como encontramos un <End CKPT> (paso 13), sabemos que todas las modificaciones anteriores ya fueron escritas a disco (por las reglas del mecanismo). Luego podemos ignorar a T1, y para T2 y T3 sólo debemos concentrarnos en rehacer las acciones del paso 9 en adelante, que es cuando se introdujo el Start CKPT.

***Transacciones a rehacer**: T2 y T3 (ignoro T1).*
***Transacciones a deshacer**: -.*

***Cambios en la base de datos**:*
C:=15 (rehaciendo el paso 10)
D:=20 (rehaciendo el paso 12)

***Cambios en el log**: -.*

Situación 2: Sucede un Crash y el último registro es <Commit T2> (Paso 14):
En primer paso, ignoramos T9 por el registro Abort. Asimismo, identificamos a T1 y T2 como transacciones a rehacer y a T3 como una transacción a deshacer. Para deshacer T3, debemos retroceder hasta su Start en el paso once[3]. Para rehacer T1 y T2, como encontramos un registro End CKPT podemos ignorar T1, y para T2 nos alcanza con empezar a rehacer desde el paso 9 (cuando comenzó el Start CKPT).

[3] Si T3 hubiera estado activa al momento de introducir el Start CKPT, hubiésemos necesitado retroceder hasta su Start, anterior al registro Start CKPT

Transacciones a rehacer*: T2 (ignoro T1).*
Transacciones a deshacer*: T3.*
Cambios en la base de datos*:*
D:=19(deshaciendo el paso 12)
C:=15(rehaciendo el paso 10)
Cambios en el log*: Se* agrega el registro <Abort T3> y luego Flush Log.

Situación 3: Sucede un Crash y el último registro es <T3,D,19,20> (Paso 12).
Identificamos a T2 y T3 como transacciones a deshacer. Como no encontramos un <End CKPT>, debemos rehacer las accciones de T1 también, lo que nos lleva a comenzar desde el principio del log. Para T2 y T3, debemos deshacer hasta el paso 3, el start de T2. Como en los casos anteriores, T9 es ignorada.

Transacciones a rehacer*: T1.*
Transacciones a deshacer*: T2 y T3.*
Cambios en la base de datos*:*
D:=19(deshaciendo el paso 12)
C:=14(deshaciendo el paso 10).
B:=9(deshaciendo el paso 5).
A:=5(rehaciendo el paso 2).

Cambios en el log*: Se* agregan los registros <Abort T3> y <Abort T2> y luego Flush Log.