DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Segundo Trabajo Práctico

6 de Julio de 2011 — Bases de Datos

Integrante	LU	Correo electrónico
Bianchi, Mariano	92/08	bianchi-mariano@hotmail.com
Brusco, Pablo	527/08	pablo.brusco@gmail.com
Allekotte, Kevin	490/08	kevinalle@gmail.com
Dondero, Julián	696/07	juliandondero@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

1. Aclaraciones

1.1. Algoritmo getAvailableSteps

El algoritmo en general no trajo mayores dificultades. La idea que aplicamos fue ir leyendo el log de arriba hacia abajo y para cada entrada encontrada, se procedía a sacar o ingresar una nueva opción del conjunto de posibles pasos que se iba a terminar devolviendo.

Ni bien comienza el algoritmo, se agrega como posibles acciones que cualquier transacción disponible pueda hacer un start. Luego se lee el log de arriba hacia abajo, y con cada lectura de una entrada del log hacíamos lo siguiente:

- En caso de leer un "start t", se elimina esa opción del conjunto de opciones disponibles y se agrega al conjunto la posibilidad de hacer un "abort t" y un "commit t".
- En caso de leer un "abort t" o un "commit t", se elminan ambas opciones del conjunto.[1]
- En caso de leer un "startckpt" se agrega al conjunto la chance de hacer un "endckpt". Si ya existía antes la opción de ingresar un "endckpt" (por ejemplo, si hay más de un "starckpt" abierto), no se agrega otro "endckpt", de manera que sólo haya uno por vez.
- En caso de leer un "endckpt", se elmina el "endckpt" que existía como opción en el conjunto de pasos disponibles.
- En cualquier otro caso (por ejemplo, el de los updates) no se realizaba ninguna acción

Además, durante este algoritmo, se iban acumulando las transacciones que quedaban activas. De esta manera, al finalizar de recorrer el log, si habían quedado transacciones activas, se agregaba al conjunto de opciones la opción de que cada una de estas transacciones haga un "update". Finalmente, utilizando este mismo conjunto de transacciones activas, se agrega la opción de hacer un "starckpt".

1.2. Algoritmo recoveryFromCrash

La idea principal del algoritmo fue recorrer la lista de Logs Record desde el final hacia el comienzo, buscando "endchkpt" y "startchkpt", y al mismo tiempo guardando los registros de "commit" y "updates" que se encontraban. Estos últimos, solo de las transacciones que tenian "commit" (estos sí o sí eran encontrados anteriormente en el recorrido).
El recorrido finalizaba de dos maneras posibles:

- La primera consiste en encontrar un "startchkp", habiendo encontrado un "endchkpt" anteriormente. Así solamente se rehacen las transacciones que hicieron un "commit" despues del "startckpt" que sabemos finalizó. Sin embargo, necesitábamos seguir recorriendo hacia atrás, buscando los updates de las transacciones activas en el momento de dicho "startckpt". Para esto, una vez llegado al primer "startckpt" con su respectivo "endckpt", guardamos en un conjunto las transacciones que hicieron "commit" y

[1] En un comienzo pensamos que había que agregar la opción de que dicha transacción pueda hacer nuevamente un "start" pero en una consulta por mail nos aclararon que no se debía agregar nuevamente esta opción.

que estaban en la lista de transacciones activas en dicho "startckpt". Luego continuamos recorriendo acumulando los "update" de estas, y sacandolas de dicho conjunto a medida que veíamos sus respectivos "start". Este segundo recorrido finaliza al vaciar totalmente este conjunto que es al momento de llegar al "start" de la transacción que empezó primera, que "commiteo" despues del "startckpt" y que está en la lista de transacciones activas. Al finalizar esta segunda etapa del recorrido, teníamos guardados todos los "update" que había que rehacer, en orden inverso, es decir desde los más nuevos a los más antiguos. Tambien teníamos las transacciones que hicieron "commit", y las que abortaron. Teníamos que hacer un "abort" por todas las transacciones incompletas, estas las obtuvimos restando de todas las transacciones, las abortadas y las commiteadas al momento del checkpoint.

- La segunda se produce cuando no hay un "endckpt" registrado, con lo cual se leerá todo el Log entero. También tendremos la información guardada de la misma manera que en el item anterior.

Con toda esta información armamos luego la lista de acciones para devolver el Recovery Result, abortando el conjunto de transacciones que debian abortar y recorriendo inversamente la lista de "updates" que había que rehacer. Finalmente, se realiza un "flush" del log solamente si el conjunto de las transacciones incpompletas, es decir las que debían abortarse, tenía almenos un elemento.

1.3. Test

Realizamos test manuales con el emulador, y 8 test de Unidad con JUnit, cubriendo los casos que nos parecieron pertinentes. Realizamos instancias con "startckpt" y sus respectivos "endckpt" como tambien instancias de solo "startckpt". Transacciones que commitearon antes y despues de los checkpoints y que abortaron o quedaron incompletas. Tratamos de cubrir lo más posible las diversas situaciones en una caída de una base de datos. Todos estos rest se encuentran en la carpeta "alumnos" junto a la carpeta "catedra" en la seccion de Test.