ETH Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Dr. G. Alonso *C. Hagen, B. Kemme*

AR 1: Transaktionsverwaltung

Praktische Übung 3 Besprechung: 15. Mai

Transaktions- und Sperrverwaltung

Einführung

Das Ziel dieser Aufgabe ist es, Einblick in den Aufbau und die Arbeitsweise einer Sperrverwaltung zu erhalten. Sie erhalten ein Programmpaket, das auf sehr einfache Weise die Module Transaktionsverwaltung und Sperrverwaltung (Lock-Management) eines Datenbanksystems implementiert. Datenverwaltung, Log-Management und Recovery-Komponente sind nicht berücksichtigt. Ihre Aufgabe ist es, einige wichtige Funktionen zur Realisierung von 2-Phase-Locking (2-Phasen-Sperren) in die Sperrverwaltung des Systems einzubauen.

Die Implementation benützt das Simulationstool CSIM. CSIM ermöglicht es, ein System als eine Menge von Prozessen zu simulieren, die über CSIM-Strukturen miteinander kommunizieren können. Durch die Verwendung von "simulierter Zeit" kann ein dynamisches Verhalten modeliert werden. Wir werden nur sehr wenige Funktionen von CSIM ausnützen, so dass seine Verwendung keine Verständnisprobleme darstellen sollte. Im Verzeichnis `/home/ar0/pub/csim_docs` befindet sich umfangreiche Dokumentation zu CSIM, die einfach mit `more 'dateiname'` aufgerufen werden kann.

Die Programme sind in C++ implementiert und befinden sich im Verzeichnis `/home/ar0/pub/p3`. Wir benützen nur sehr einfache Klassenkonstrukte und Methodenaufrufe, so dass der Code für C-Kenner, die eine grundlegende Vorstellung von Objektorientierung haben, leicht verständlich sein sollte. In dem Header-File (`*.h`) eines Moduls werden die Klassen des Moduls und ihre Methoden deklariert, während die entsprechende C++-Datei (`*.cpp`) die Methoden implementiert. Einfache Methoden sind bereits im Header-File implementiert.

Systemablauf und Statistikinformation

In der Datei `dbs.cpp` wird die eigentliche Systemkonfiguration definiert und einige zusätzliche Funktionen implementiert.

Beim Start des Systems werden zuerst Eingabeparameter aus der Datei `parameter_file` gelesen. Somit kann das Verhalten des Systems unter verschiedenen Bedingungen getestet werden. Insbesondere spielen die Anzahl und der Typ der Operationen einer Transaktion

und die Anzahl der Datenobjekte in der Datenbank eine grosse Rolle. Die Zwischenankunftszeit (Inter-Arrival-Time) und die Dauer der einzelnen Operationen bestimmen unter anderen, wie viele Transaktionen sich zu einem Zeitpunkt gleichzeitig im System befinden. Eine Beschreibung des Deadlock-Zeitintervalls findet sich weiter unter.

Im eigentlichen Kern der Hauptroutine werden entsprechend der Zwischenankunftszeit neue Transaktionen gestartet. Dabei wird jeweils ein neuer Prozess (simuliert durch CSIM) kreiert, der eine neue Transaktion generiert, sie ausführt und sich dann wieder beendet. (In "richtigen" Datenbanksystemen werden häufig bei Systemstart eine Anzahl von Prozessen kreiert, die dann jeweils eine Transaktion nach der anderen ausführen.)

Während der gesamten Simulation werden Statistikdaten mit Hilfe von CSIM-Funktionen gesammelt, die am Ende ausgegeben werden.

Transaktionsverwaltung

Die Transaktionsverwaltung ist in den Dateien `transactMgr.h` und `transactMgr.cpp` implementiert. Eine Transaktion hat einen Identifikator und besteht aus mehreren Lese- oder Schreiboperationen auf Datenobjekten. Bei der Generierung einer Transaktion werden die Anzahl der Operationen, der Typ jeder Operation (schreibend oder lesend) und die Datenobjekte, auf die die Operationen zugreifen, festgelegt.

Eine Transaktion wird ausgeführt, in dem ihre Operationen der Reihe nach durchgeführt werden. Die eigentliche Operation auf einem Datenobjekt wird durch Verstreichen von simulierter Zeit implementiert. Ohne Lock-Management beenden Transaktionen immer erfolgreich.

Wird das Lock-Management eingebunden, wird für jede Operation zuerst eine Sperre auf dem entsprechenden Datenobjekt angefordert. Kann die Sperre gewährt werden, wird die Operation ausgeführt. Ansonsten wird die Transaktion vorzeitig abgebrochen. Am Ende der Transaktion, ob erfolgreich beendet oder abgebrochen, werden alle gewährten Sperren freigegeben.

Lock-Management

Die gesamte Spezifikation und der Grossteil der Implementation der Sperrverwaltung ist in `lockMgr.h` und `lockMgr.cpp` vorgegeben. Die wichtigste Datenstruktur des Lock-Management (Sperrverwaltung) ist eine Hash-Tabelle, in der die einzelnen Locks verwaltet werden. Abbildung 1 zeigt den prinzipiellen Aufbau der implementierten Tabelle. Sie besteht aus einem Array von Listen, wobei jedes Listenelement ein Eintrag für ein Datenobjekt ist. In welcher Liste sich der Eintrag für ein Datenobjekt befindet, kann mittels einer Hash-Funktion berechnet werden. Ein Eintrag besitzt eine Liste für gewährte Sperren und eine Liste für wartende Sperren. Es ist auch möglich sowohl gewährte als auch wartende Sperren in einer Liste zu verwalten.

In einem richtigen Datenbanksystem würde allgemein benötigte Information z.B. in einem "shared-memory"-Segment abgelegt werden, auf das von jedem Datenbankprozess aus zugegriffen werden kann. Die Sperrverwaltung wäre in einem solchen System auch für die Parallelitätskontrolle (z.B. durch Verwendung von *Semaphoren*) zuständig. Eine Parallelitätskontrolle ist in unserem Beispiel nicht notwendig, da CSIM-Prozesse nur

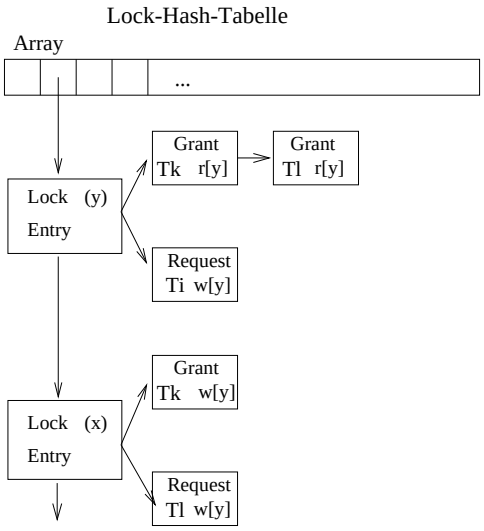

Abbildung 1: Hash-Tabelle einer Sperrverwaltung

pseudoparallel (non-preemptive) ablaufen. In unserem auf CSIM basierenden System sind gemeinsame Informationen in globalen Variablen gespeichert.

Wenn die Sperrverwaltung eine Lock-Anforderung erhält, sucht sie den entsprechenden Eintrag und versucht, die Sperre zu gewähren. Ist dies möglich, kann der Transaktion eine positive Antwort mitgeteilt werden. Ansonsten muss die Transaktion warten, bis die Sperre freigegeben wird. Trifft dies innerhalb des Deadlock-Zeitintervalls ein und wird die Sperre gewährt, dann wird der Transaktion eine positive Antwort mitgeteilt. Ansonsten erhält sie eine negative Antwort.

Bei der Freigabe einer Sperre wird überprüft, ob weitere Sperranforderungen existieren, und die erste Schreib- oder alle Leseanforderungen gewährt.

Damit die Hash-Tabelle so klein wie möglich ist und einen effizienten Zugriff erlaubt, werden Einträge von Datenobjekten gelöscht, wenn keine Sperren vorliegen und erst wieder bei einer neuen Sperranforderung angelegt.

Aufgabe 1: Messungen

Kopieren Sie sich alle Dateien aus `/home/ar0/pub/p3`. Sie erhalten das Executable `dbs` ohne die Einbindung des Lock-Managements, indem Sie `make` aufrufen. Führen Sie einige Systemläufe mit unterschiedlichen Werten für die Transaktionsgrösse, Dauer der Operationen und Zwischenankunftszeit aus und analysieren Sie, welchen Einfluss sie jeweils auf den Durchsatz, die Anzahl parallel laufender Transaktionen und die Antwortzeit haben. Alle Parameter, die eine Zeit darstellen (Simulationszeit insgesamt, Ausführungszeit einer Operation, Zwischenankunftszeit, Deadlock-Timeout-Intervall) sind in einer abstrakten "simulierten" Zeiteinheit definiert. Man könnte sich vorstellen, dass es sich dabei z.B. um Millisekunen oder Sekunden handelt. Bitte beachten Sie, dass die gesamte Simulationszeit im Vergleich zu den anderen Zeitparametern genügend gross ist, damit der Hauptteil der Messungen erfolgt, wenn das System im eingeschwingten Zustand ist und Anfangsabweichungen nicht ins Gewicht fallen.

Aufgabe 2: Implementation

Aufgabe ist es, die Implementation der Sperrverwaltung in `lockMgr.cpp` zu vervollständigen. Implementieren Sie die folgenden Funktionen:

- Die Methoden des Objekts `Lock_Table_Entry` zum Anfordern und Freigeben von Sperren (`request_Lock, release_Lock`).
- Die Methoden des Objekts `Lock_Table` zum Finden und Löschen von Einträgen in der Hash-Tabelle. (`find_Entry, remove_Entry`).

Hinweise:

- Sowohl die Objektklasse `Lock_Table` (Hash-Tabelle) als auch die Objektklasse `Lock_Table_Entry` (Eintrag für ein Datenobjekt) besitzen Funktionen zum Anfordern und Freigeben von Sperren. Dabei ruft die Methode des `Lock_Table` die Methode des entsprechenden `Lock_Table_Entry` auf. Bei einer Sperranforderung gibt letztere sofort einen positiven (Lock kann gewährt werden) oder negativen (Lock muss warten) Bescheid zurück. Die Methode des `Lock_Table` gibt einen positiven Bescheid sofort weiter und blockiert bei einem negativen Bescheid, maximal jedoch für die Dauer des Deadlock-Timeintervalls.
- Beim Freigeben einer Sperre `release_Lock` muss das CSIM-Event `Any_Lock_Granted` gesetzt werden (`Any_Lock_Granted->set()`), falls eine wartende Sperranforerung gewährt wurde. Damit werden Prozesse, die auf eine Sperre warten, aufgeweckt, und können überprüfen, ob ihnen die Sperre gewährt wurde.
- Als Objektklasse für die "Sperre" kann die Objektklasse `Op`, also die Operation einer Transaktion verwendet werden.
- Die Listenverwaltung wird mit Hilfe von `DLList` durchgeführt. Folgende Funktionen auf den Listen sind von Bedeutung:
  - `int List -> empty()`: > 0, falls die Liste `List` leer, sonst 0.
  - `int List -> length()`: Gibt die Anzahl der Elemente in `List` zurück.
  - `void List -> append(element)`: hängt das Element `element` ans Ende von `List`.
  - `void* List -> remove_front()`: Entfernt das erste Element aus der Liste und gibt einen Pointer auf dieses Element zurück.
- Um die Sperrverwaltung wirksam werden zu lassen, müssen Sie in der Datei `globals.h` `LOCK_IMPLEMENTED` definieren.
- Debuggen ist mit CSIM leider sehr schwierig. Falls Sie zum Testen während der Programmausführung Ausgaben auf den Bildschirm schreiben, ist es sinnvoll, diese mit `#ifdef DEBUG` einzuklammern und während dem Testen `DEBUG` in der Datei `globals.h` zu definieren.

Führen Sie nun erneut Messungen durch. Variieren Sie dieses Mal auch die übrigen Eingabeparameter (Update-Rate, Datenbankgrösse, TimeOut). Analysieren Sie, wie jeder einzelne Parameter Einfluss auf die Abbruchrate hat. Wie muss das Deadlock-Timeout Intervall gesetzt werden, um bei einer fixer Vorgabe der anderen Parameter optimale Ergebnisse zu erzielen?