CSC3170 Final Review

Lecture 13: RAID

RAID: Redundant Arrays of Independent Disks

- high capacity and high speed
- high reliability

Mean time between failure: MTBF

Mean time to failure: MTTF

in Poisson distributed arrivals, MTTF = MTBF

So we have to deal with system failure.

The solution is - Redundancy (also called backup)

By - mirroring (or shadowing)

数据分段(striping)-可以提高并行效率

分为：

- Bit-level striping
- Block-level striping

RAID 0: Block striping, non-redundant

RAID 1: Mirrored(镜像备份) disks with or without block striping

Parity blocks: Parity block j stores XOR of bits from block j of each disk. 奇偶性效验

RAID 4: 将所有的parity block放在每行的第一位

RAID 5: 将所有的parity block放在第n行的第n位

RAID 6: 和RAID 5相似，但放了两位error correction blocks (P , Q)

Chapter 14: Transactions

Definition: A transaction is a unit of program execution that accesses and possibly update various data items，不可分隔的一个执行单位

Problem to deal with:

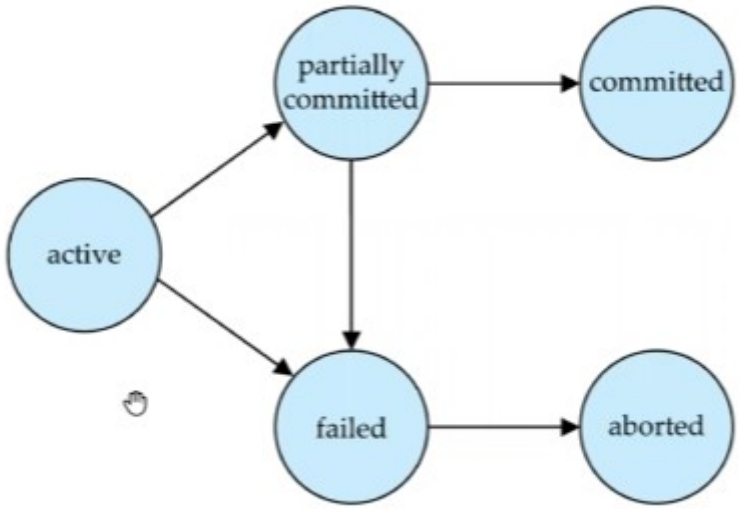
- Failure of various kinds, such as hardware failures and system crashes
- Concurrent execution of multiple transactions，防幻读脏读等

Features: (数据库ACID特性)

- Atomicity-原子性, 要求在执行失败的时候实行数据回滚
- Durability-持久性, permanent change to database after every execution of transactions.，数据永久保存
- Consistency-一致性, 当完成事务的执行后，需要保持一致性
- Isolation-隔离性，保证不出现脏读/幻读等错误

Transaction State:

- Active-initial state
- Partially committed -
- Failed
- Aborted - fail之后回滚到原状态。有两种出路：
    1. 重新执行事务 restart transaction
    2. 终止事务 kill transaction
- Committed



Concurrent Execution （并发执行）：

优点：增加处理器和磁盘应用率，减少反应时间

需要一个并发控制计划(concurrency control schemes), 引入后文提到的事务隔离机制、MVCC多版本隔离机制、锁机制等

三个著名问题:

- The lost Update Problem - 在执行过程中，值被改了(unrepeatable read)
- The Temporary Update problem - 数据需要回滚，但回滚前的数据被人读了 (dirty read 脏读, 读取了未commit的数据，一旦回滚就会导致数据错误)
- The Incorrect Summary Problem -

Unrepeatable Read Problem - 不可重复读

事务(before commit)重复读了一个数据两次，但期间被其他事务改了数据，导致两个数据不一样

Schedule

一系列执行命令（顺序），需要保证事务和事务之间的逻辑关系保持正常，最后commit出去。

Serializability 串行化

表示回到最原始的block by block(series)的执行方式时，数据依然保持等效。

Conflict Serializability 冲突串行化

指调度中出现了write的情况

Conflict Serializable: 指的是可以通过调换顺序，实现"冲突调度(有write出现)"的串行化处理。

Testing for Serializability:

用"优先图"(precedence graph).

箭头从

write A -> write A

read A -> write A

write A - > read A

若优先图有环(cycle)，则说明是不可冲突串行化的(not conflict serializability)

若无环(acyclic)，则说明conflict serializability

Cycle-detection algorithm: take order n^2 time, where n is the number of vertices in the graph

可以用拓扑排序（topological sorting）来实现串行化顺序(serializability order)

Recoverable Schedules

目标要设计出可恢复调度(Recoverable schedule)

保证某个事务若读了另外一个事务时，在另外一个事务commit前不要提前进行commit，防止另外一个事务发生回滚(rollback)。

Dirty read 脏读

在设计成Recoverable Schedule的前提下，若某个事物发生错误（failure），有可能会发生级联回滚(cascading rollback，多个事务读取到失效事务的数据，导致大量回滚,效率下降)。这时候这些读取到回滚数据的事务叫做脏读(dirty read)

Cascadeless Schedules

我们要避免级联回滚，关键要想办法让上文提到的第一个事务commit后，第二个事务再读取关联数据，这样就可以避免发生级联回滚。Note that every cascadeless schedule is also recoverable.

Weak levels of consistency

有时候，数据库就需要特意被设计成“脏读”的模式，即一次读取某一瞬间的数据，所以我们就不需要或只需要使consistency保持在一个较低的水平。

Transaction Definition In SQL

在SQL中，事务有两种结局：

1. commit work
2. Rollback work

Type of Violation

- Dirty read- 脏读（读到了一个uncommitted data，可能会因回滚而丢失）
- Nonrepeatable read - 不可重复读（在两次读(read)之间更新了数据，导致两次读到的不一样）
- Phantoms-幻读（指的是在事务执行过程中插入了一条数据，这条数据在开始的时候没用，而在结束的时候出现了）

Isolation Levels：

- Read UNCOMMITTED - 最低等级，会发生全部问题
- Read COMMITTED - 解决了Dirty read脏读问题
- Repeatable Read(RR) - 解决了脏读和不可重复读（Nonrepeatable Read）
- Serializable - 最高等级，不会幻读/脏读/不可重复读

Chapter 15: Concurrency control

Lock-Based Protocols锁协议

分为

- 排他锁(exclusive, X锁)，数据可以被(锁主人)读取和写入。command: lock-X
- 共享锁(shared, S锁)，数据只能被(锁主人)读取。command: lock-S

Lock requests are made to concurrency-control manager. Transaction can proceed only after request is granted.

每次read()都要给并发控制系统(concurrency-control manager)加S锁。每次write()则要给并发控制系统(concurrency-control manager)加X锁。

如果有人给某个item加了X锁，那么其他人就无法再给它加任何锁，所以得名“排他锁”。共享锁则相反，任何人都可以再叠加共享(S)锁在上面 *（当然不能叠排他锁，因为会“排他”）*

Deadlock 死锁

两个排它锁撞在了一起，会使进程陷入死循环 - 这时候需要进行回滚(rollback)

Starvation 锁饿死

有可能存在一个事务序列，其中每个事务都申请对某数据项加S 锁，且每个事务在授权加锁后一小段时内释放封锁，此时若另有一个事务T1 欲在该数据项上加X 锁， 则将永远轮不上封锁的机会。这种现象称为“ 饿死”（starvation)。
可以用下列方式授权加锁来避免事务饿死。
当事务T2 中请对数据项Q 加S 锁时，授权加锁的条件是：
(1)不存在在数据项Q 上持有X 锁的其他事务；
(2)不存在等待对数据项Q加锁且先于T2 申请加锁的事务。

Two-Phase Locking Protocol

Phase 1: Growing Phase，事务正在上锁，可以获得S,X锁，也可以把S锁升级为X锁

Phase 2： Shrinking Phase，事务正在解锁，可以解锁S,X锁，也可以把X锁降级为S锁

Lock point：最终获得锁的位置

两种策略

- Strict two-phase locking，限制X锁，可以保证不发生脏读
- Rigorous two-phase locking，限制S+X锁，一般数据库采用的策略，保证不发生不可重复读+脏读。

如果已经有锁了，则可以直接读/写

Lock manager：

首先事务向manager发送加锁请求，然后manager回传是否批准。

实现方式：lock table - in memory data-structure（hash table）

会有队列，waiting to be granted

Deadlock

指的是一个set的事务都在等着对方释放彼此的锁。这叫死锁。

Wait-die scheme：

young transaction rollback. Old wait. 通过时间戳实现Young和Old的区别。

是一种非剥夺策略，老的事务等待新的事务释放资源，即若A比B老，则等待B执行结束，否则A卷回(roll-back),一段时间后会以原先的时间戳继续申请。老的才有资格等，年轻的全部卷回。

等待死亡的特点是不剥夺，但只有老的有资格等待。

新的回滚后，再次申请有可能die again，在这个过程中die for many times, until old complete.

Wound-wait scheme:

是一种剥夺策略，如果A比B年轻，A才等待，A比B老，则杀死B，B回滚。换句话说，老事务不等待"你"，直接杀死"你"，抢占资源，小孩子才等。

伤害等待的特点是老的不等待，直接把你干掉，新的才去等待。

两种策略下，事务回滚后都会以原时间戳重新提起请求（并有可能进入等待）

Timeout-based schemas

每次回滚时，隔一段固定的时间才执行第二次请求。但可能会发送锁饿死。

Wait-for graph：用链图表示各个事务间的等待关系，若其中有环，则说明存在死锁。

当检测到死锁的时候，系统就会挑一个victim进行回滚，期望打破死锁环。

两种策略：1、把整个事务都弃用（abort）

2、将部分事务标记为victim，这个事务回滚到最开始的状态（级联）。但有可能会发生锁饿死（starvation），这需要命令oldest transaction never chosen as victim。

Multiple granularity 多种粒度

fine granularity - 细粒度：高并发性(high concurrency)，高锁定开销(high overhead)

Coarse granularity - 粗粒度：低并发性（low concurrency)，

levels:

database > area > file > record (行)

一般在表的粒度上分为1、表级锁，2、行级锁。

Intention lock modes 意向锁

当我们向一张表加入表级锁的时候，这时候我们必须去表中每一行去遍历，看看对应的行是否已经用到对应的锁，这时候如果数据库中的数据海量的话，想要完成这个认为的难度就非常的大，难道没有一个好的方法?

意向锁在原有的锁(X锁和S锁)引入了新的锁(IX和IS)锁
意向锁是个表级锁

获得锁：(获得对象锁)
IX - **intention-exclusive**--表示 如果有事务想获得数据对象X锁之前，先获得表IX锁
IS --**intention-shared**-- 表示如果事务想要获得数据对象S锁之前，先获得表IS锁

SIX --**shared and intension-exclusive**--表示当前已经有S锁，但意向升级为X锁

加锁：（加入表级锁）
　如果事务对表加X锁，看看有没有其他的事务对表加锁（S/X/IS/IX）如果 有的话，加锁失败

细节：

　任意 IS/IX 锁之间都是兼容的，因为它们只是表示想要对表加锁，而不是真正加锁；
　S 锁只与 S 锁和 IS 锁兼容，也就是说事务 T 想要对数据行加 S 锁，其它事务可以已经获得对表或者表中的行的 S 锁。

Intention lock are put on all the ancestors of a node before that node. 当向下深潜的时候，路过的node都会被加上意向锁。

意向锁 allows a higher level node to be locked in S or X mode without having to check all descendent nodes. 不用遍历了!!

锁的相容矩阵 compatibility matrix

三种意向锁：IS,IX,SIX,其中SIX可以表示为S+IX，即有S的性质又有IX的性质，属于(S) and (IX)。S锁不能和IX/X/SIX相容，X锁不能和一切锁和意向锁相容。母节点加上某种锁之后，子节点也会被隐性加上该种锁。

事务T遵循的对多粒度树结点Q的加锁规则：
必须遵循锁相容矩阵；
粒度树的根必须首先被加锁，锁的类型不限；
当Q的父节点被T加上IX或IS时，Q才可以被T加上S或IS型锁。
当Q的父节点被T加上IX或者SIX，Q才可以被T加X、SIX或IS锁。

(可以看出IX锁是万能的，下辖所有类型的锁)

当T还没有释放任何锁，T才可以对结点加锁
当Q的所有后裔结点都不被T加锁，T才可以解锁Q。

（总而言之，要想从此过，留下意向锁）

（所以这就是two-phase的由来）

（涉及到并发控制了~父节点被S/X/SIX卡住的话，意向锁IX/X是走不通的哦）

time-stamp order = serializability order.

Timestamp-ordering（**TSO**） protocol 时间戳协议

W-timestamp 代表最后一次write (Q)的时间

R-timestamp 代表最后一次read (Q)的时间

如果事务T将要进行read时，他会检测是否有大于当前时间戳的W-timestamp，若有则说明出现了不可重复（？）读，需要对T进行回滚。

如果事务T将要进行write时，他会检测是否有大于当前时间戳的W-timestamp和R-timestamp，若有则说明出现了不可重复读（？）或无效写（？，存疑），需要对T进行回滚。

transaction with smaller timestamp always points to transaction with larger timestamp, 所以采用时间戳排序协议将不会再有死锁问题。

一个事务一次全部执行，然后随着时间的流逝，下一个随后的事务继续全部完成。如果没有达到预期目标，就需要进行回滚。

Thomas' Write Rule

Obsolete(过时的) write operation may be ignored under certain circumstances. 如果在未来write之前做了一个无效写，其实这个无效写是可以被忽略的。

Lecture 16: Recovery

事务故障：

- Logical errors：
- System errors

系统故障

- Fail-stop assumption: 系统终止，但硬盘里的内容一般不会出问题

磁盘故障

- disk drives use checksums to detect failures

几种存储系统：

Volatile storage: Do not survive system crashes.

Nonvolatile storage - Survive system crashes, like disk, etc. But may still fall.

Stable storage - A mythical form (想象的存储) that survive all kinds of failure, approximated by maintaining multiple copies on distinct nonvolatile media.

Physical block - blocks that on the disk

Buffer blocks - blocks that in main memory.

Log-based recovery mechanisms. 日志, 快照

shadow-copy / shadow-paging: less used alternative

<Ti, X, V1, V2>

Update to private part of main memory do not count as database modification.

Update log record must be written before database item is written.

Log 日志数据都存储在**stable storage**（nonvolatile) 上

- Immediate-modification scheme: allows updates of an **uncommitted** transaction to buffer / disk itself.
- deferred-modification scheme: performs updates to buffer/disk only at the time of transaction commit.

Commit Point

当一个事务的commit log到了stable storage的时候，这个事务就叫做committed.

Undo:

undo的时候，日志上会记录. 将T涉及到的所有更新**全部**撤销。

Redo:

redo: 重做，顾名思义，**全部**重做。

在数据恢复时，日志上所有不带和的事务都需要被undone,所有带和的事务都要被redone.

包括一些没做完的failure，恢复数据时依然会redone -> failure again.（called `repeating history`).
优点：不用手动判断failure与否，放权给系统自动完成

：每次redo/undo 所有事务是个很慢的过程，有什么方法改善吗？用。顾名思义，checkpoint用来保存存档点。

- All updates are stopped while checkpoint operation is in progress.
- Output all log records current residing in **main memory** to **Stable storage**.
- Output all modified blocks to the disk
- Output a log record onto **Stable storage** where L is a list of all transactions active at a time of checkpoint.

所以，在ckp之前commit的事务，都不需要redo。

系统会自动查最后一个checkpoint，三种redo/undo情况：

1. 在checkpoint前start，checkpoint后commit -> redo
2. ckp后start, checkpoint后commit -> redo
3. ckp后start，无commit -> undo

Recovery Algorithm

正常情况下的工作

- Logging,将所有事务的活动都计入日志，以结尾。
- Transaction rollback，将正常事务的rollback计入日志，以结尾。往前回滚时，若遇到<T, X, V1, V2>则db先令 X = V1 然后在日志中写<T, X, V1>，代表将事务复原到V1的样子。继续往前回滚，直到遇到, 则写入，回滚结束。

正常情况下的工作，无论是locking还是rollback，活动都会被记录在log里

不正常情况下的工作，分为两个阶段

1. Redo phase, 首先找到第一个checkpoint，然后**从前往后扫描**。若找到普通的或语句，则直接进行redo。若找到, 则加入undo list. 若找到或, 则在undo list中剔除T1的记录。

2. Undo phase，**从后往前扫描。**若当前< T X V1 V2 >的T位于undo list里，则执行复原操作 X = V1且在日志后加上< T X V1 >, 若读到< T start >且T位于undo list里，则在日志后加入< T abort > 且将T从undo list中删除。当undo list为空的时候停止undo phase。

在undo phase后，代表数据恢复完成，程序可正常运行。

Log record buffer一般存储在main memory中，当 **满了** 或 **Execute log force** operation的时候才被写入stable storage.

Log force一旦被执行，就会commit all transaction(including the commit record).

如果log被buffer存储的时候，需要遵循以下原则:

- 必须按顺序装入stable storage。
- 必须等到< T commit >之后才写入buffer
- 在一个main memory block被传输到stable storage前，需要保证前序关联数据已完成传输（called： write-ahead logging (WAL)）

Chapter 17 Data Warehouse

面向分析的数据库，叫Data warehouse - 数据仓库。

Business intelligence - BI

Decision support systems - DSS

Descriptive analytics(Past) -> Predictive Analytics(Future) -> Prescriptive Analytics(Decision)

Data warehouse often uses **star schema** or sometimes **snowflake schema**.

ER relationship modeling有时候会有一些缺点：

- Very symmetric
- Cannot tell which table is most **important** or **largest**
- Cannot tell which tables hold static or dynamic business information
- Joining of any tables is possible by user.

关于star schema的定义：

- 由一个中心的fact table和若干各other tables组成，其中fact table与每一个other table互相连接，而每一个other table之和fact table一个进行连接，呈蒲公英状。
- A **star query** is a join between a fact table and a number of lookup tables. Each lookup table is joined to the fact table using a primary-key to foreign-key join, but the lookup tables are not joined to each other.
- 特征：very asymmetric, commonly used in Data Warehouses.

Fact table

- Fact table tends to contain **additive facts**(附加事实)
- fact table have composite keys
- all other tables are dimension tables
- Every combination of key values would give rise to a different record in the fact table.
- Fact table is naturally highly normalized

Dimension Tables

- Dimension table tends to contain textual or non-additive facts
- Dimension tables should not be normalized
- Normalized dimension table destroy the ability to browse

Data Cube view: Each combinations of keys in fact table correspond to one of the small cubes. (dice)

Star Join:

A primary-key to foreign-key join of the dimension tables to a fact table. The fact table normally has a concatenated index on the key columns to facilitate this type of join.

Advantages:

- provide a direct and intuitive mapping between the business entities being analyzed by the end users and the schema design.
- Provides highly optimized performance for typical data warehouse queries.

Snowflake schema (Undesirable):

This is a type of star schema, but more complex. It **normalize** dimensions to eliminate redundancy. The dimension data has been grouped into multiple tables instead of one large table.

Undesirable, because it may lead to more complex queries and reduced query performance.

DSS(Decision support systems) - 决策系统数据 vs. Operational data 操作（生产）数据

一个做分析用，一个做生产用，两个的需求不同，导致设计出来的功能特性也不同。

Data Warehouse的特性(SINT)

- subject-oriented (面向主题，指的是个人或企业在使用数据仓库着重看重的方面)
- integrated （数据集成，指的是需要消除源数据中的异值，保证数据仓库的独立性和一致性）
- non-volatile （稳定性，数据一般用于查询，很少会对数据进行修改）
- time variant （反映历史变化，可以用作数据分析，对未来进行预测）

No update on Data warehouse -> once we need to change, we **refresh**

the data warehouse.

在生成warehouse的时候，要考虑到多数据来源带来的问题，如key conflict, data encoding等。

Data Mart:

A simple form of a data warehouse, focus on a single subject. 一言以概之，一个精简版的data warehouse.

Chapter 18 Data Mining

Overall: Data mining is the process of semi-automatically analyzing large databases to find useful patterns. Similar to Machine Learning, but with a lot of data. Also called **knowledge discovery in databases**

Exam

No 4 NF, MVD, HASHING(DON'T WORRY TOO MUCH, BUT LITTLE), TIME_STAMP,

YES 3NF, BCNF, B-TREE, SQL, DATA_WAREHOUSE, SCHEMAS, LOCKING, ROLLUP, DATA_MINING,