

本科生毕业论文

|  |  |
| --- | --- |
| **题目**： | **在图数据库中实现事务**  **英文题目（Time New Roman，三号，加粗）** |
|  |  |

姓 名： 张 三

学 号：

院 系： 信息科学技术学院

本科专业：

指导导师： 李 四

二○二一 年 六 月

北京大学本科毕业论文导师评阅表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学生姓名 |  | 学生学号 |  | 论文成绩 |  |
| 学院（系） |  | | | 学生所在专业 |  |
| 导师姓名 |  | 导师单位/  所在研究所 |  | 导师职称 |  |
| 论文题目  （中、英文） | |  | | | |
| 导师评语  （包含对论文的性质、难度、分量、综合训练等是否符合培养目标的目的等评价）  导师签名：  年 月 日 | | | | | |

**版权声明**

任何收存和保管本论文各种版本的单位和个人，未经本论文作者同意，不得将本论文转借他人，亦不得随意复制、抄录、拍照或以任何方式传播。否则，引起有碍作者著作权之问题，将可能承担法律责任。

# 摘要

小四号，宋体/Time new roman，1.5倍行距

**关键词：**3-5个

# Abstract

5号，Time new roman，1.5倍行距

**Key Words：**3-5个

**全文目录**

[摘要 9](#_Toc476314826)

[Abstract 10](#_Toc476314827)

[第一章 绪论（二号、黑体） 12](#_Toc476314828)

[1. 第一级（三号、宋体/Time new roman、加粗） 12](#_Toc476314829)

[1.1 第二级（四号，宋体/Time new roman，加粗） 12](#_Toc476314830)

[1.1.1 第三级（小四，宋体/Time new roman，加粗） 12](#_Toc476314831)

[参考文献 14](#_Toc476314832)

[本科期间的主要工作和成果 15](#_Toc476314833)

[致谢 15](#_Toc476314834)

# 第一章 绪论（二号、黑体）

正文，小四号，中文宋体，英文Time new roman，1.5倍行距

一般建议三级目录

## 第一级（三号、宋体/Time new roman、加粗）

### 第二级（四号，宋体/Time new roman，加粗）

### 第三级（小四，宋体/Time new roman，加粗）

**图表格式**



图1 （五号，宋体/Time new roman）

表1 （五号，宋体/Time new roman）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表头 |  | |  |  | |
|  |  |  |  |  |
| 内容，（五号，宋体/Time new roman） |  |  |  |  |  |
|  |  |  |  |  |  |

(表格使用如上所示的三线表)

# 第一章 绪论

### 研究背景与意义

### 研究现状

### 研究内容

### 论文结构

# 第二章 相关工作

### 图数据库系统

### 多版本并发控制

### 数据库事务

### 分布式ID生成算法

# 第三章 多版本并发控制实现

### 整体架构

本文基于分布式开源图数据库Nebula Graph进行了事务的实现，在其基础上主要修改了Graph Engine层的解析器以实现事务的开始（BeginTx）和提交（CommitTx）操作的解析，在Storage层增加了Transaction Manager用于管理事务，Index Map用于键的多版本管理，以及AOF Manager用于将操作持久化和断电恢复；为了生成唯一的事务ID，本文还修改了Storage服务与Meta服务的接口，并实现了基于SnowFlake算法生成的唯一ID。

对于MVCC的四个隔离等级，本文对读已提交（RC）、可重复读（RR）进行了实现。读未提交（RU）的隔离级别会导致读到其他事务未提交的修改，在现实中并不常用，等同于未进行任何事务隔离时的访问，无序进行修改；串行化（Serialization）则要求所有事务的实现都是串行的，是最高的隔离级别，在本文实现中只需修改Transaction Manager，在当前事务未完成前，阻塞新事务的注册即可，不需要多版本存储实现，因此也是比较简单的。

Nebula Graph是一个属性图系统，因此其多版本存储可以拆分为两个部分，分别是顶点连接的出边和入边的多版本关系，以及顶点或边与对应的属性之间的多版本关系。对于前者，只有边的插入和删除操作会影响其结果，因此选择差分版本形式存储相应的边操作序列，在读取时则按照顺序依次执行差分操作进行复原，不涉及与底层KV存储的交互，相对简单。对于后者，涉及到同一个顶点或边的键对应的属性不同，需要建立对同一键的多版本存储和索引机制来与底层KV存储交互，相对复杂。

Index Map是一个常驻内存中的哈希键值对，维护了点或边形成的键与各种对应的一系列版本号的关系。

Transaction Manager是事务实现的关键，承担了事务注册、提交、回滚等功能，并维护当前活跃事务集，为垃圾回收等功能提供支持。

由于Index Map是内存中的结构，因此需要对其进行持久化，用来断电重启后恢复，于是此处参考了Redis的实现方式，设计了一个AOF Manager用来写入对Index Map的操作到文件以及恢复时读取，同时还会定时进行重写，合并以及清理过时版本的操作记录，以避免AOF体积无限制增长。

【此处为Index Map、Transaction Manager以及KV store的交互图。】

### 多版本并发控制协议选择

多版本并发控制协议存在多种选择，大体上分为MVTO、MVOCC、MV2PL以及基于依赖的有向无环图冲突分析。过往实验显示，MVOCC由于使用乐观锁机制，在读写比较大时写入冲突较低，因此性能良好；但在写操作占比逐渐增加的情况下，由于验证阶段不通过导致的事务重启次数较多，性能下降明显。MV2PL是基于悲观锁实现的两阶段提交协议，性能稳定，但现有数据库中使用已不太多。权衡了可能的使用场景以及为了在写操作占比较多时仍能有稳定的事务性能，本文选择了MVTO作为多版本并发控制协议进行了实现。

选择MVTO作为MVCC协议，需要为每个多版本存储增加三个字段：txn-id、read-ts以及begin-ts，每个字段都是int64大小，占据8字节。其中，txn-id记录当前条目的写入者编号，如果没有写入者则该项为0，起到了写锁的作用；read-ts记录距离最近的读者编号，这可以保证“迟到的写者”无法更新新的读者读过的内容；begin-ts表明起始生效时间，当且仅当事务的版本号大于等于begin-ts才表明该版本对该事务可见。

在原始论文中，MVTO还需要end-ts用来表示版本结束时间，但由于实际操作中同一键的多版本都以链表形式连接，前一版本的end-ts等于后一版本的begin-ts，因此只需要存储起始时间，再与后续版本的起始时间进行比较即可找到事务对应的版本。

在传统读写锁实现中，访问共享区域变量需要互斥锁结构，用于保证同时只能有一个写者到达共享区域，但这样的加锁-解锁操作往往对性能损耗较高。因此，本文对写者加锁采用了无锁CAS方式，利用Atomic库提供的原子性比较并交换值的方式，对版本的txn-id进行操作，节省了加互斥锁的开支。但同时，由于无锁CAS的本质是多个写线程争夺写入的权限，同样可能存在重复碰撞-等待-重启的现象，因此可以引入网络传输协议中的随机重启算法，降低访问同一个键的不同写者之间的冲突。

### 属性的多版本存储实现

同一键对应的多版本以链表形式在内存中存储，其中按照存储的顺序、位置以及存储内容等方面又可以分为以下四类：Append-Only存储：O2N、Append-Only存储：N2O、时间遍历存储以及差分存储。Append-Only存储中，每个键指向的位置固定为最新或最旧版本，已存储的版本只会在垃圾清理情况下被移除，因此更新的代价相对较低；时间遍历存储将各键的最新版本保存在主表中，更旧的版本保存在额外的表中，由此可以扩大多版本存储的版本数量，但在每次插入新版本时都需要将当前版本从主版本换出，更新代价相对较高；差分存储链表头部记录一个基准版本，随后是按照顺序排列的版本操作，在更新的内容不多的情况下相对于前三种全量存储方式能明显降低版本存储体积，但当读操作发生时需要遍历版本连依次进行复原，时间代价上升。

本文选用了Append-Only：O2N的方式进行存储，版本头部全量存储较旧的版本，尾部存储较新的版本。由于本文存储的差分版本是键的差分版本，不包含属性值，因此每个版本的大小可控，所以选择了Append-Only方式；实现中使用STL List，List是一个双链表结构，因此可以直接调用tail方法获取最新版本，再通过反向迭代访问到之前的版本。

每一个键对应的多版本都是一个链表，而键与对应的版本链组成了一个键值对，所有的键值对组成了全局结构Index Map。对于属性值的读写请求，会先访问Index Map获取或插入对应的版本，然后拼接出带有版本号的键，再向底层KV存储读取或写入新的键-值。

【此处有版本链示意图】

多版本链中每个版本都记录了该版本的版本号，并额外使用两个字段作为读写锁、一个字段标记起始时间。由于本文实现中属性的多版本是针对Index Map建立的多版本，所以多版本内容与具体属性值无关，而恰好与版本的起始时间相同，所以可以将起始时间与要存储的内容合并，使得每个版本都只包含3个int64大小的字段。

与通行做法一致，本文将删除操作实现为标记删除，即，删除一个版本并不会直接在存储中删掉，而是将对应版本标记为失效，由后续垃圾清理来完成回收。由于起始时间字段实际是创建该版本的事务的版本号，是一个正数，所以标记事务为删除时，直接对这一字段取相反数得到对应的负值表示删除。这一做法可以在节省删除标志位的同时，保留版本的原始时间用来进行事务的回滚。

综上，本文的版本结构如下：

【此处是版本的struct代码，struct Version {int64 txn-id; int64 read-ts; int64 begin-ts;};】

txn-id为int64类型的写锁，read-ts为int64类型的读锁，begin-ts同时代表版本号以及该版本的起始时间。

【此处是Index Map的代码，unordered\_map <string, List<struct Version> >】

### 点-边关系的多版本存储实现

点-边关系的多版本存储实现采用差分版本存储，将对边的插入以及删除两种操作记录按照事务版本顺序记录为条目。采用差分版本存储可以有效降低图中超级顶点在多版本存储中的存储量膨胀问题。

在Nebula Graph设计中，顶点由TagID和VertexID唯一确定，边由SrcID、DstID、EdgeType以及Rank唯一确定（不支持多入多出边），因此顶点和边本身是不支持修改的。由于边的编码中已经包含了SrcID和DstID，故一条边对应的出顶点和入顶点也是一旦建立就不会改变的。所以点-边关系的多版本只需考虑一个顶点对应的出入边在不同时间的情况，因此可以只记录边的插入和删除操作。

点-边关系的版本结构如下：

【此处是点边关系的struct代码，struct VEVersion { int64 txn-id; int64 read-ts; int64 begin-ts; vector<Edge> insert-edges; vector<Edge> delete-edges};】

Txn-id、read-ts以及begin-ts的含义与上一小节中含义相同，insert-edges数组记录本次事务中插入的边，delete-edges数组记录本次事务中删除的边。

【此处是VE Map的代码，unordered\_map <string, List<struct VEVersion> > OutEdges，unordered\_map <string, List<struct VEVersion> > InEdges】

读写操作与上一小节中对属性的多版本实现相似，对同一顶点的出边的并发写入会被禁止，对同一顶点的入边的并发写入也会被禁止，而允许对同一顶点的入边和出边同时写入——因为这不会导致对同一条边的并发修改。

### 垃圾回收算法

多版本存储会随着事务的增多产生更多的过时版本，因此需要进行垃圾回收。主流的GC操作分为Tuple-Level GC以及Transaction-Level GC，前者又分为后台线程回收以及事务内合作回收。后台线程回收机制下，系统会定时启动GC线程，按照一定的算法进行标记-回收，把过时的以及被删除的版本删除，缩短版本链的实际长度；事务内合作回收机制只适用于传统Append Only：O2N存储方式，在从头遍历版本链时可以同时回收靠前的失效版本，此方法可以避免GC线程标记回收时的“Stop the World”现象，但如果一个版本链一直没有被任何后续事务访问，则它的失效版本不会被回收，所以此方法需要与定时后台GC相结合；事务级别GC同样采用后台线程扫描，但以事务为级别进行回收，会将一个失效事务生成的所有版本都一同回收（一个事务被认为失效当且仅当其产生的所有版本都不再对活跃事务集中的事务可见），在关系型数据库中由于局部性原理，同一事务产生的版本通常存储在相邻位置，因此可以提高回收效率，缺点在于此法需要维护事务的读写集合，带来额外的开销。

本文采用了Tuple-Level后台线程回收机制，定时且并行地扫描Index Map和VE Map中的每条版本链，回收过期和被删除的版本。考虑到各版本的活跃版本链相对均匀，本文实现中使用了粗粒度的锁，对整个Map加锁，当且仅当所有键对应的版本链都回收完成时才解开锁，允许后续事务的访问。

### 操作流程

# 第四章 持久化设计

### AOF系统设计

### 持久化写入

### 重启恢复

# 参考文献

正文，五号，中文宋体，英文Time new roman，1倍行距

1. 期刊 作者. 论文名. 刊名, 出版年份, 卷号(期号): 起始-截止页
2. 专著 作者. 书名. 版本(第一版不写). 出版城市: 出版社, 出版年份: 起始-截止页
3. 论文集 论文作者. 论文题目//编者. 论文集名: 其他题名信息. 出版城市(或者会议城市): 出版者, 出版年: 引文起始-截止页码
4. 学位论文 作者. 学位论文题名. 城市: 论文保存单位, 年份
5. 网络文献 作者. 题名[文献类型标志/文献载体标志]. 出版地: 出版者, 出版年(更新日期)[引用日期]. 获取和访问路径

\*注意: 作者姓前名后, 超过3名作者列前3名, 后加“, 等”; 英文姓名, 姓前名后, 姓首字母大写, 名缩写; 文献的项目要完整, 各项的顺序和标点要和格式要求一致; 未公开发表的论文、报告不列入正式文献, 如有必要可在正文当页下加注。英文文献格式同上。参考文献在正文中按出现顺序用[1], [2]......在右上角标注, 放在“参考文献”中时, 用[1], [2], ...顺序标注。

# 致谢

正文，小四号，中文宋体，英文Time new roman，1.5倍行距