2021年 春季

分布式数据库课程报告

课程教师：卢 暾

目 录

|  |  |  |
| --- | --- | --- |
| 第1部分 | **论文报告与总结** | 第…页 |
| 第2部分 | **项目技术报告** | 第…页 |
| 第3部分 | **课程认识与体会** | 第…页 |

姓名：\_陈 霁 璇\_\_

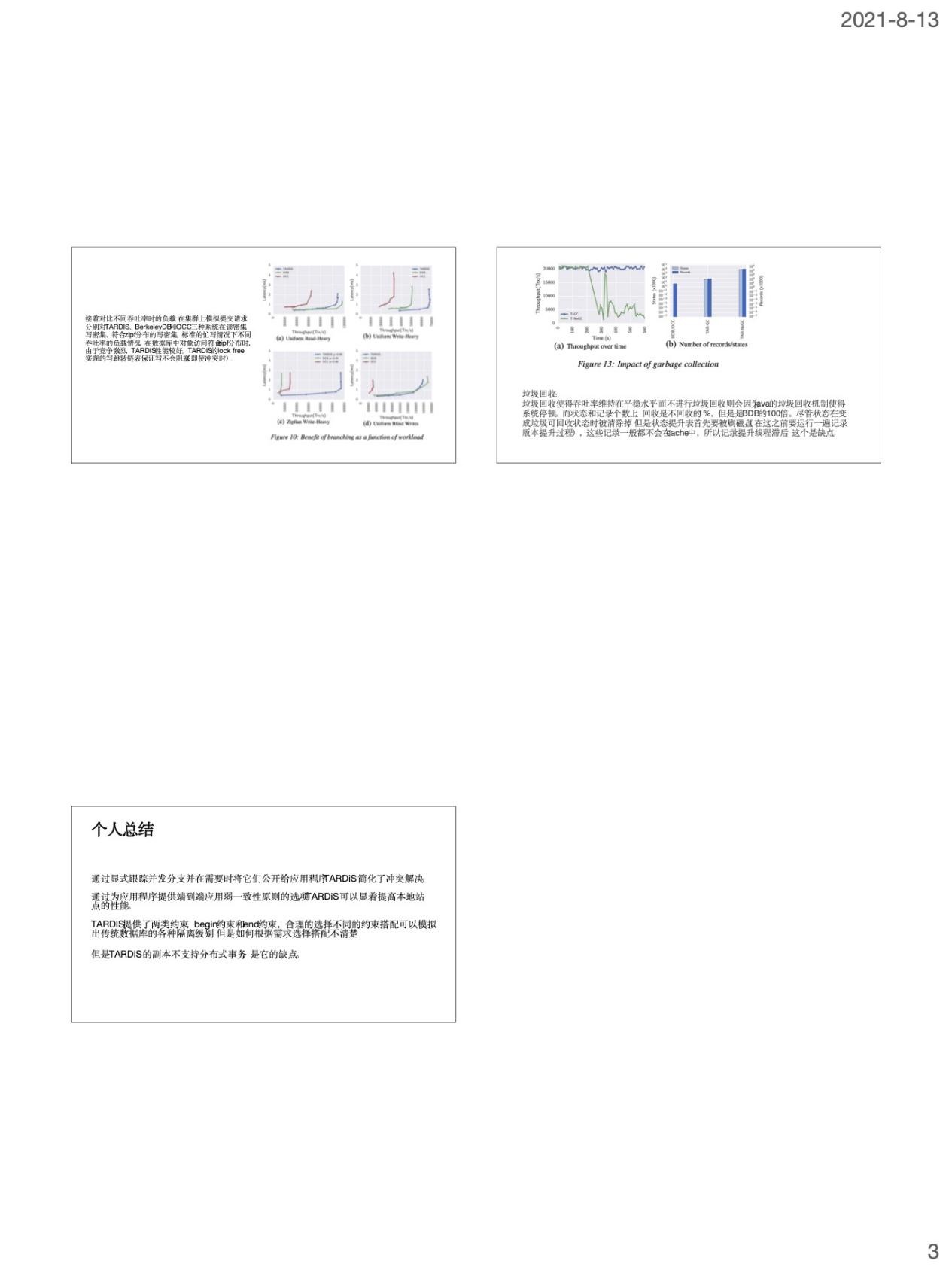
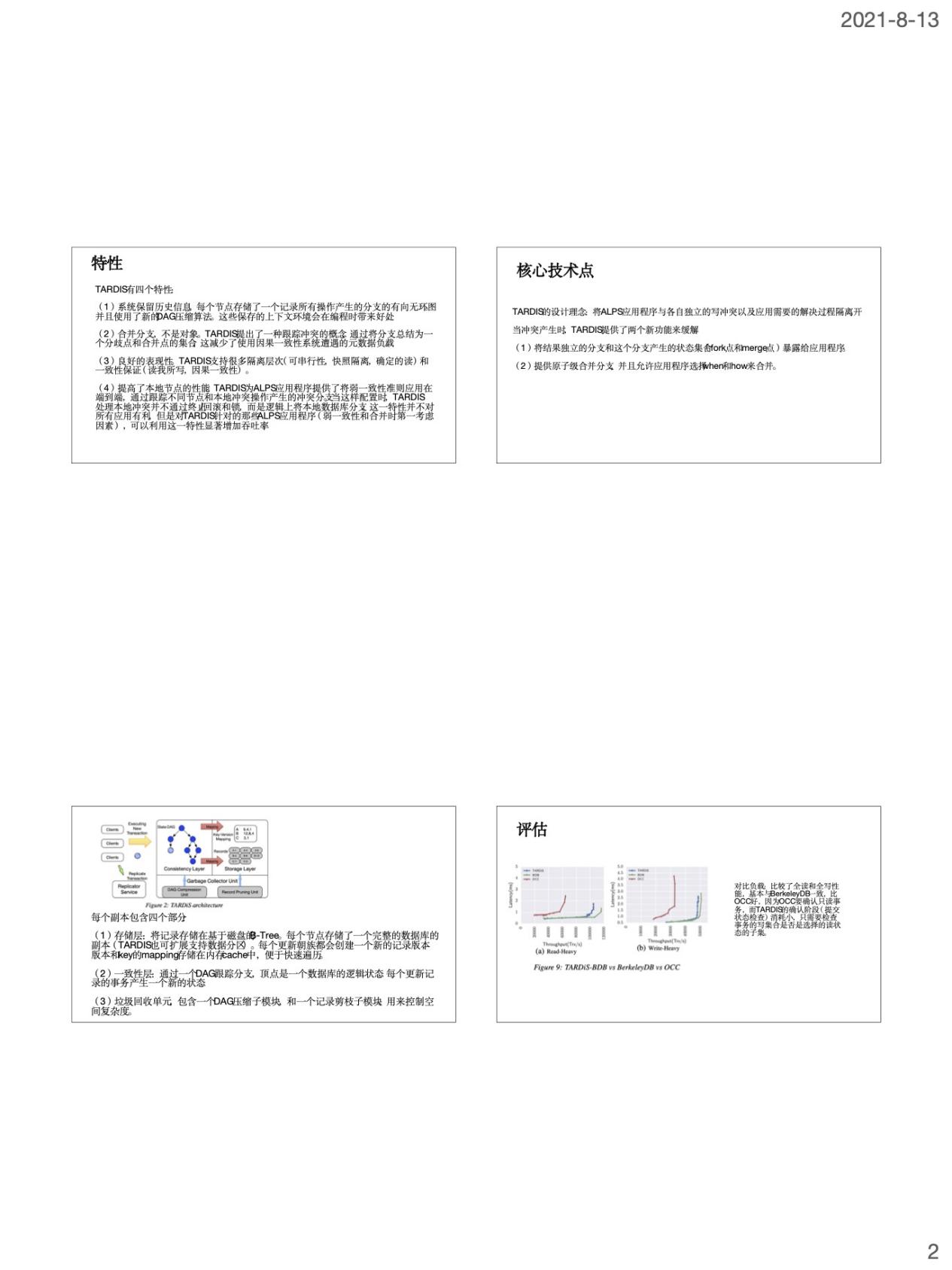
学号：20210240341

导师：\_\_谈子敬 \_

2021年 8月 10 日

**论文报告与总结**

**论文PPT**

****

**论文总结**

**TARDiS:A Branch-and-Merge Approach To Weak Consistency**

前提：我们把一个应用分为两部分 A 和 B ， A 是业务逻辑处理系统， B 是底层的存储系统。TARDIS是B类系统，为具有 ALPS （availability，low latency，partition tolerance，scalability）特性的 A 类系统提供底层支持。

1. 论文解决了什么问题，从 business problem 到 technical problem 。

Business problem ：很多 A 系统的数据存储都是依赖BerkeleyDB或其他现有的数据库系统的，这些数据库系统将A系统与数据冲突隔离开，B系统需要自己实现全局弱一致性的保证措施和解决冲突的机制，使得B系统的实现比较复杂，而且由于B系统要解决冲突，吞吐率也会受到限制。反应到应用层面就是不能有效的支持竞争增多的情况。

Technical problem ： B 类系统中有两个比较难解决的问题：（ 1 ）写写冲突，很多系统采用一套固定的解决策略，如保存最后写成功的值。但是不能完全符合 A 系统的真实想法，而且没有给A系统解决冲突的充分信息。而 A 系统在不同的应用场景下可能有不同的需要。（ 2 ）缺乏跨对象语义，无法解决两个对象间的语义冲突问题（ A （人）和 B （人）对同一个对象 a 的不同副本进行了修改，这两个事务完成后， C （人）和 D （人）分别根据 a 的这两个不同副本的两种语义对其他对象 c 和 d 进行了修改，这时 c 和 d 就有了语义冲突，但是正常合并时只能检查出 a 的冲突并进行冲突解决，不能检查出 c 和 d 有了冲突，这样就会出现对象间的语义冲突）

2. 论文的问题是不是成立的？有哪些地方是 solid 的的，有哪些地方其实应用场景是存在疑问的。

         论文的问题是成立的

solid的地方：

（1）现有的大多数B系统没有保留用以解决冲突的上下文（在解决冲突后就扔掉了）。而且B系统无法根据实际需求来处理冲突，只能根据系统逻辑判断。

         （ 2 ） B 类系统在遇到事务之间冲突时，需要先解决冲突，使数据保持一致状态，再进行下一批事务的执行。这样就会等待。吞吐率会受到影响，如果不解决冲突，直接在不同的分支写数据可以大大提升吞吐率。

    存在疑问的地方：

(1）TARDIS提供了两类约束，begin约束和end约束，合理的选择不同的约束搭配可以

模拟出传统数据库的各种隔离级别。但是如何根据需求选择搭配不清楚。

    （2） TARDIS 解决的是为 A 类系统提供解决冲突所需信息的问题。如果这个冲突指的是多节点间副本的一致性，那么这个冲突由谁感知？状态的 DAG 存在哪个节点？如果这个冲突指的单节点的并发操作时产生的冲突，那么 TARDIS 解决就是单节点的并发控制问题。这样的话如何解决多节点间的数据不一致问题？

    （3） TARDIS 针对的是数据在各个节点间全备份的情况，针对不同节点间保存数据的不同部分（如 hdfs ）如何检测冲突并创建分支？

3. 如果我沿着这个大方向研究下去，我会对这个问题做一些什么样的改变，或者衍生出一些什么样的新问题，问题和原有问题差异在哪。

我理解TARDIS是解决单个节点高并发情况下如何解决写数据冲突的问题，那么如何解决不同节点间数据冲突的问题？比如A节点写a为1，B节点写a为2。这样如何感知冲突并在不同节点间维护状态DAG图？

4. 现有的技术是什么，为什么不能很好地解决这个问题。

其他的分支系统也有的暴露了分支，如git，Olive文件系统。但是他们暴露的分支太明确了，信息太多，增加了合并的复杂度，这恰是ALPS应用程序希望避免的。TARDIS提供的分支概念则比较简单，容易进行合并。

还有一类因果一致系统，将写冲突直接暴露给应用。其实是将单个对象的多个值暴露给应用。这类系统没有提供跟踪冲突或分支的功能，将多个值暴露给应用增加了复杂度。在TARDIS中，开发人员只在合并模式需要是才处理一个对象的多个值，便于掌控。

在拜占庭问题上，SUNDR和FAUST开发了分支的一致性和线性来避免客户端看到错误客户端的不同的值。Depot系统扩展了SUNDR的模型来支持加入分支，但是Depot没有暴露分支的抽象概念，也不支持跨对象原子合并。Sporc系统支持原子合并分支，但是只针对有限的情况（协同文本应用），并且使用操作变换来解决冲突。

5. 作者采取的方法是什么。和 baseline 或 pre-art 相比，理论上为什么会好？实验是怎么做的？实验结果是否成立（比如实验集群的规模过小…）？

    作者认为每一个副本应该完整地记录描述一个冲突是如何产生的所有信息，但是只在上层应用需要地时候再暴露出去。而且将上层应用与底层冲突隔离不是一个好的方式。

TARDIS是一个为弱一致性系统提供支持的存储系统B，为A系统提供了很好的解决端到端冲突的方法。

每个节点都保存了全部数据副本。每个副本包含四个部分：

（1）存储层次：将记录存储在基于磁盘的B-Tree。每个节点存储了一个完整的数据库的副本。每个更新操作都会创建一个新的记录版本，版本和key（记录）的mapping存储在内存cache中，便于快速遍历。

（2）一致性层次：通过一个DAG跟踪分支，顶点是一个数据库的逻辑状态。每个更新记录的事务产生一个新的状态。

（3）垃圾回收单元，包含一个DAG压缩子模块，和一个记录剪枝子模块。用来控制空间复杂度。

（4）复制服务器：传播提交的事务，将远程的事务正确应用。

TARDIS在内存中维护了一个状态DAG。在遇到写冲突时会建立新的状态分支，每个写操作都会产生一个新的状态，并且用一些压缩算法来保证一定的空间复杂度。同时将具体数据按key-version set存储为map结构，每个记录都对应一个按版本新旧顺序连接的链表，存储多个版本的数据，这样可以快速找到一个分支上的所有的数据的值。

为A系统提供合并分支需要的所有信息（如冲突时的分支点，某一条分支路径上的所有冲突对象等）。使得A系统可以自己定义分支合并规则。简化了A系统的复杂度。

在冲突时建立分支而不是等待冲突解决使得在写数据竞争激烈时本地吞吐率比其他系统要大很多，理论上是正确的。

实验做法：

（1）首先对比了实现ALPS系统的复杂度。分别基于TARDIS和BerkeleyDB实现了几个CRDTs（一致可复制数据类型）和Retwis（Twitter的简化版），对比了他们的代码行数和吞吐率。TARDIS的代码是BerkelyDB的一半到三分之一。复杂度大大降低。吞吐率提高了2到8倍。

（2）对比负载：比较了全读和全写性能，基本与BerkeleyDB一致，比OCC好，因为OCC要确认只读事务，而TARDIS的确认阶段（提交状态检查）消耗小，只需要检查事务的写集合是否是选择的读状态的子集。

（3）接着对比不同吞吐率时的负载，在集群上模拟提交请求。分别对TARDIS、BerkeleyDB和OCC三种系统在读密集、写密集、符合zipf分布的写密集、标准的忙写情况下不同吞吐率的负载情况。在数据库中对象访问符合zipf分布时，由于竞争激烈，TARDIS性能较好。TARDIS的lock free实现的写跳转链表保证写不会阻塞（即使冲突时）。

在zipf分布时，由于竞争激励（加锁），BDB的读写比读密集或写密集时慢十倍，TARDIS只慢了一点。OCC表现最好，因为OCC保证了（1）至少一个事务总会被提交（2）读不会阻塞写。但是OCC的高召回率和复杂的提交过程使其吞吐率是TARDIS的1/5。

在一致的写数据时，由于锁发生的几率小，TARDIS的垃圾回收速度不能跟上状态的产生。增加的内存使TARDIS的性能比BDB差10%。

实验结果是成立的是对比实验，对比的内容比较有代表性而且比较大众。

6. 作者为什么会想出这样的方法？

    因为作者在做存储系统尝试解决并发产生的冲突时总觉得少了一些信息，而这些信息只有上层应用才能获取到，这些信息就是实际需求。所以为什么不把冲突保存下来留给上层应用去解决呢？就像git一样，人工处理冲突。

7. 作者在解决这个问题中是不是尝试建立了一些假设或者约束去简化问题，或者尝试把读者从一个问题引导到另一个问题。

    在开头举的例子是两个人，对一个网页的不同副本进行修改，之后又来了两个人，在前面的改动的基础上继续改动，由此引发的单对象多值冲突和多对象间语义冲突。但是后面解决的问题都是单节点的冲突。并没涉及多节点间冲突如何检测和解决。

8. 作者的方法有什么缺陷或者可改进的空间。

    在数据存储类型上有可能进行优化，在多节点冲突时的分支状态产生。

9. 如果我来仅仅从方法层面做接下来的工作，我会选择什么方向，为什么。

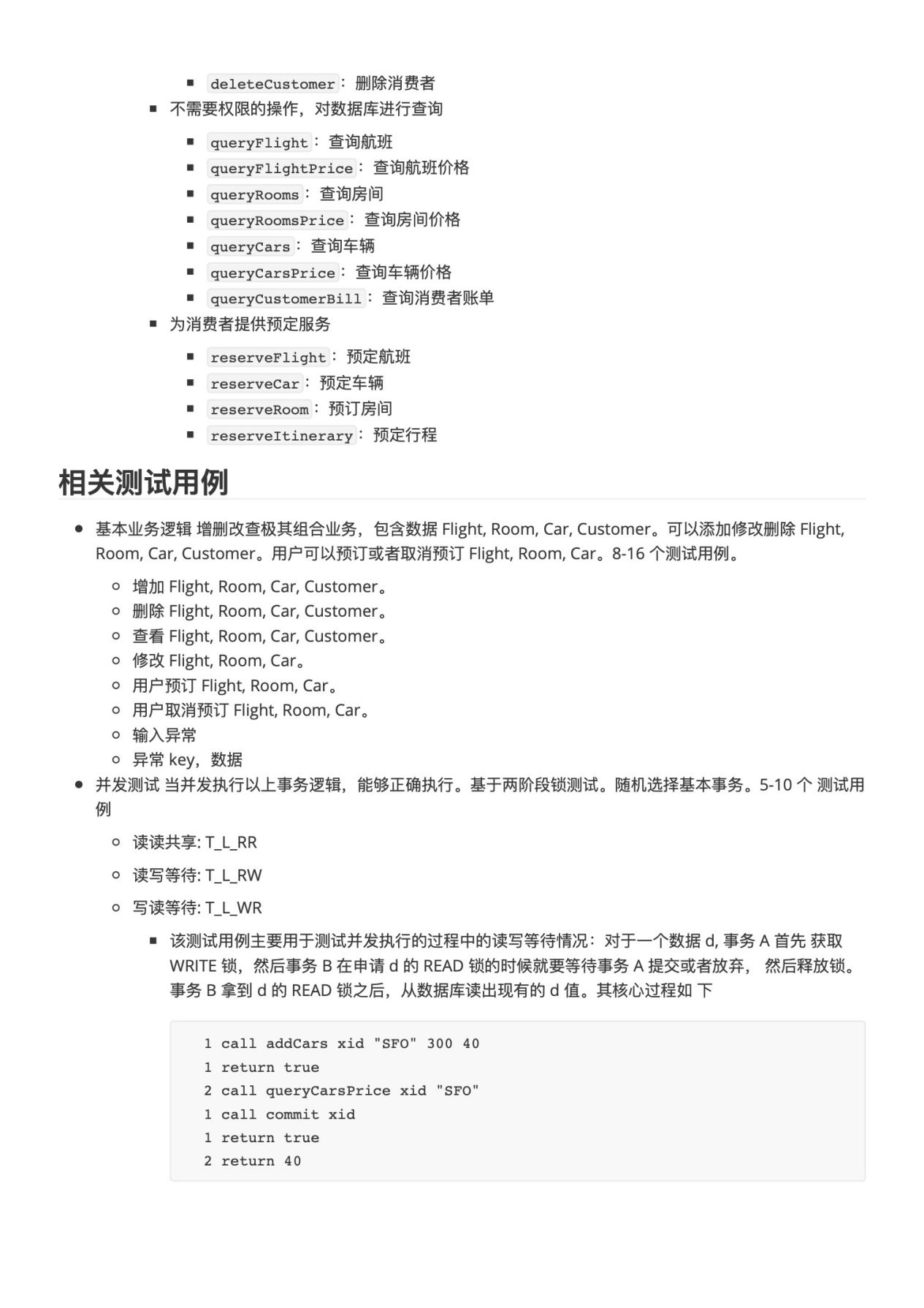
    本文方法是记录系统演进状态并为上层应用提供简单的获得所需信息的接口，思想是存储系统不解决冲突，交给上层应用利用专业知识合并冲突分支，类似git。

    方法上的改进主要针对数据分区存储时跨节点的冲突检测和状态分支。以及更优秀的垃圾回收机制来减少因保存状态DAG图和多个数据库版本带来的空间消耗。

    思想上没有什么问题。这个思想挺好，甩锅。

**项目技术报告**

项目由我一个人完成



**课程思考体会**

分布式系统是由一组通过网络进行通信、为了完成共同的任务而协调工作的计算机节点组成的系统。分布式系统的出现是为了用廉价的、普通的机器完成单个计算机无法完成的计算、存储任务。其目的是利用更多的机器，处理更多的数据。

　　分布式数据库系统是在冀中是数据库系统的基础上发展来的，比较分布式数据库系统与集中式数据库系统，可以发现分布是数据库系统具有下列优点：

　　（1）更适合分布式的管理与控制。分布式数据库系统的结构更适合具有地理分布特性的组织或机构使用，允许分布在不同区域、不同级别的各个部门对其自身的数据实行局部控制。例如：实现全局数据在本地录入、查询、维护，这时由于计算机资源靠近用户，可以降低通信代价，提高响应速度，而涉及其他场地数据库中的数据只是少量的，从而可以大大减少网络上的信息传输量；同时，局部数据的安全性也可以做得更好。

　　（2）具有灵活的体系结构。集中式数据库系统强调的是集中式控制，物理数据库是存放在一个场地上的，由一个DBMS集中管理。多个用户只可以通过近程或远程终端在多用户操作系统支持下运行该DBMS来共享集中是数据库中的数据。而分布式数据库系统的场地局部DBMS的自治性，使得大部分的局部事务管理和控制都能就地解决，只有在涉及其他场地的数据时才需要通过网络作为全局事务来管理。分布式DBMS可以设计成具有不同程度的自治性，从具有充分的场地自治到几乎是完全集中式的控制。

　　（3）系统经济，可靠性高，可用性好。与一个大型计算机支持一个大型的冀中是数据库在加一些进程和远程终端相比，由超级微型计算机或超级小型计算机支持的分布式数据库系统往往具有更高的性价比和实施灵活性。分布式系统比集中式系统具有更高的可靠性和更好的可用性。如由于数据分布在多个场地并有许多复制数据，在个别场地或个别通信链路发生故障时，不致于导致整个系统的崩溃，而且系统的局部故障不会引起全局失控。

　　（4）在一定条件下响应速度加快。如果存取的数据在本地数据库中，那末就可以由用户所在的计算机来执行，速度就快。

　　（5）可扩展性好，易于集成现有系统，也易于扩充。

　　对于一个企业或组织，可以采用分布式数据库技术在以建立的若干数据库的基础上开发全局应用，对原有的局部数据库系统作某些改动，形成一个分布式系统。这比重建一个大型数据库系统要简单，既省时间，又省财力、物力。也可以通过增加场地数的办法，迅速扩充已有的分布式数据库系统。

　　分布式数据库系统的缺点

　　（1）通信开销较大，故障率高。例如，在网络通信传输速度不高时，系统的响应速度慢，与通信县官的因素往往导致系统故障，同时系统本身的复杂性也容易导致较高的故障率。当故障发生后系统恢复也比较复杂，可靠性有待提高。

　　（2）数据的存取结构复杂。一般来说，在分布时数据库中存取数据，比在集中时数据库中存取数据更复杂，开销更大。

　　（3）数据的安全性和保密性较难控制。在具有高度场地自治的分布时数据库中，不同场地的局部数据库管理员可以采用不同的安全措施，但是无法保证全局数据都是安全的。安全性问题式分布式系统固有的问题。因为分布式系统式通过通信网络来实现分布控制的，而通信网络本身却在保护数据的安全性和保密性方面存在弱点，数据很容易被窃取。

　　分布式数据库系统的优点与缺点：分布式数据库的设计、场地划分及数据在不同场地的分配比较复杂。数据的划分及分配对系统的性能、响应速度及可用性等具有极大的影响。不同场地的通信速度与局部数据库系统的存取部件的存取速度相比，是非常慢的。通信系统有较高的延迟，在CPU上处理通信信息的代价很高。分布式数据库系统中要注意解决分布式数据库的设计、查询处理和优化、事务管理及并发控制和目录管理等问题。