上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

学士学位论文

THESIS OF BACHELOR

论文题目 分布式事务性内存的设计与实现

论文作者 魏 星 达

学生学号 5110379011

指导教师 夏虞斌, 邹南海

专 业 软件工程

学 院 软件学院

Submitted in total fulfillment of the requirements for the degree of Bachelor
in

The design and implementation of distributed software transactional memory

XingDa Wei

Advisor
Prof. YuBin Xia,NanHai tZou

Depart of Software Engineering, School of Software
Shanghai Jiao Tong University
Shanghai, P.R.China

上海交通大学

本科生毕业设计（论文）任务书

课题名称：分布式事务并行执行的优化

执行时间：2014 年 12 月 至 2015 年 6 月

教师姓名：夏虞斌 职称：讲师

学生姓名：魏星达 学号：5110379011

专业名称：软件工程

学院(系)：电子信息与电气工程学院

毕业设计（论文）基本内容和要求：

OLTP 是现在很多业务处理的核心，这类业务的事物处理通常要求强一致性。随着业务规模的扩大和，单机已无法完成传统要求，数据库通常将数据分布在多台机器上。然而，传统的并发控制协议如 occ 和 2pc 在分布式事物的执行下效率非常低。对传统并发控制的改进以及提出新的并发控制协议来提升分布式事物执行效率成为一个迫切的问题。

拟通过观测现有事物处理系统的运行，分析传统并发处理协议在分布式环境下的执行，找出影响系统性能的关键。分析 2pc 和 occ 在分布式环境下执行的瓶颈以及限制，寻找提高并行的可能。在保证强一致性的前提下，优化传统协议，或者使用新的协议来提高数据库系统的性能。

在实现上述的前提的前提下，需要阅读更多分布式事物执行的前沿文献，尝试优化新的方法，比如说不同于传统分布式并发控制的协议，优化数据库事务处理在分布式环境下的性能。

毕业设计（论文）进度安排：

序号	毕业设计（论文）各阶段内容	时间安排	备 注
1	阅读相关参考文献和代码，了解已有系统系统的实现方式，撰写开题报告。	2014.12-2015.01	
2	分析现有系统的瓶颈，提出优化的方法。	2015.01-2015.03	
3	使用优化后的方法实现事务处理协议，提高系统的性能和可扩展性。	2015.03-2015.05	
4	毕业设置及工作总结修改和完善，并最终完成毕业论文与毕业答辩。	2015.05-2015.06	

课题信息：

课题性质 ：设计　　论文

课题来源*：国家级　　省部级　　校级　　横向　　预研

项目编号 _____

其他_____

指导教师签名：_____

年　　月　　日

学院（系）意见： 院长（系主任）签名：_____ 年　　月　　日
学生签名：_____ 年　　月　　日

上海交通大学
学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：_____

日　期：_____年_____月_____日

上海交通大学
学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于

保　密 □，在 _____ 年解密后适用本授权书。

不保密 □。

(请在以上方框内打 ✓)

学位论文作者签名：_____　　指导教师签名：_____

日　期：_____年 _____月 _____日　　日　期：_____年 _____月 _____日

分布式事务性内存的设计与实现

摘　要

本文使用新的硬件特性如 RTM 和 RDMA 实现分布式下的事务性内存的抽象并使用其优化 OLTP 系统。OLTP(Online transaction processing) 是目前许多电子商务的核心，需要保证其在数据库上执行事务的原子性和隔离性。随着数据集的增大以及系统扩展性的需求，现在的 OLTP 系统需要在一个分布式的环境下执行。然而，传统的并发控制方法如二阶段锁或者乐观并发控制在分布式环境下执行的效率并不高；乐观并发控制在分布式下网络开销非常大, 而二阶段锁的拿锁操作则存在比较大的开销。新的硬件技术如 RTM 和 RDMA 可以提供高效的单机事务性内存, 以及高效的网络访问；在一台利用 RTM 实现的单机的 OLTP 系统可以比传统的分布式下的 OLTP 系统每台机器的效率快 100 倍。本系统基于 DBX-TC ，一个利用 RTM 实现的单机 OLTP 系统，使用 RDMA 操作保护程序对远端内存的访问，从而给上层程序一个分布式事务性内存的操作的抽象。利用 RTM 来保护程序的本地内存访问，利用 RDMA 的直接读写以及原子 CAS 操作来实现高效的分布式二阶段锁，并且利用特定的只读程序并发控制方法来避免 RTM 执行和 RDMA 操作所存在的由于读过多数据带来的问题。系统执行传统 OLTP 标准 TPC-C 时每台机器的执行效率接近于单机系统的执行效率，同时仍具有接近线性的可扩展性。

关键词： 并发控制，分布式数据库，事务内存

The design and implementation of distributed software transactional memory

ABSTRACT

The system implements a distributed transactional memory for OLTP systems using new hardware features such as RTM and RDMA.OLTP(Online transaction processing) systems requires that the transactions it executes is atomic and isolated from each other,which needs some concurrency control methods to guarantees these properities.With the increasing data sets and the scalability requirements,these systems are more tentative to be in a distributed setting.However,traditional concurrency control methods like 2-phase locking and OCC is less efficient in a distributed setting.New hardware features like RTM and RDMA can be used to implement high-efficiency single machine OLTP system and efficient network communications,and a single machine OLTP system can outperform a state-of-art distributed OLTP system by 100X with the throughput per machine.The system is based on DBX-TC,a RTM based single machine OLTP system , using RDMA to extend the scope of protection to a distributed setting .The system uses a dedicated Read-only method to optimize Read-only transactions which resolves problems which is not that efficient using the same concurrency control method.The system's performance is compatible with DBX-TCin a single machine setting on TPC-C benchmark and can scale near linearly with machine number.

KEY WORDS: Distributed Transactions, Database,Concurrency Control

目 录

插图索引

表格索引

第一章　绪论

1.1　研究背景和意义

OLTP(Online transaction processing) 是现在许多电子商务后台的核心。随着这些业务的数据集扩大和系统需要相应的处理越来越多的请求，系统的容错性，即当系统中的某些机器崩溃时系统仍可正常运行或者可以在非常短的时间内恢复正常，通常这些系统都需要在一个分布式环境下运行。由于现代计算机上有充足的内存，将数据库中所有的数据放入内存中成为了一种可能。由于 OLTP 系统需要事务执行的原子性和隔离性，因此事务执行的时候都需要并发控制的方法来保证事务的并发执行符合隔离线需求。传统的并发控制方法有 2PL(2-phase locking)[**2pl**] 和 OCC(Optimistic concurrency control)[**occ**]，然而 OCC 在分布式执行下会有许多的网络和重试开销 [**mu2014extracting**]，同时 2PL 的上锁操作会带来许多的开销，因此在分布式环境下事务的执行效率下并不高。同时由于通常分布式事务需要服务器保证一致的比如 2 phase commit[**2pc**]，当前技术水平的分布式系统下的事务处理系统如 rococo[**mu2014extracting**] 每台机器的生产量只有单机系统如 silo[**silo**] 的一百分之一。这意味着分布式系统需要 100 倍更多的计算资源才能够达到单机的效率，显然如何高效的在分布式环境下执行事务处理是非常重要的问题。

幸运的是，当今有许多硬件技术可以帮助减少事务并发控制方法的开销。首先,RTM（Restricted transaction memory)[**rtm**] 是 Intel 在 Haswell 中引进的硬件事务性内存处理，使用其来合理的实现事务并发控制可以提高并发程序执行的效率。比如在 OCC 中，事务提交前需要检查其读写集是否与其他并行的事务的读写集有冲突，如果有冲突则需要中断事务的执行。通常，由于程序需要手动记录并检查自己的读写集，因此这仍然会带来很大的开销。同时对于写的集合 OCC 在检查时仍然需要上锁。并且通常 OCC 需要一个全局的序号，这也限制了可扩展性。[**silo**] 使用 RTM 实现的基于 OCC 的系统比如 DBX[**dbx**] 可以比其他基于 OCC 的系统比如 silo 更为存只提供单机高效。然而硬件的事务性保护只限于单机。第二，现有的如远程直接内存访问 RDMA（Remote Direct Memory Access）技术，可以使得请求直接绕过操作系统内核的 TCP/IP 栈，同时不需要远端机器的 CPU 参与处理，这样可以带来更高的生产量并且可以降低请求的延迟。[**farm**] 目前仍然没有基于 RDMA 特定优化的事务处理系统，只有提供键值访问的数据库。[**herd**, **pilaf**]。因此，如何利用这两种技术加速事务在分布式环境下的并发控制成为一个重要的问题。

1.2　国内外研究状况

数据库系统事务处理的并发控制是一个研究很多的领域。传统的并发控制方法有 2PL[**2pl**] 和 OCC[**occ**]。由于这些协议比如 2PL 通常考虑到磁盘的开销 [**2pldisk**]，而现代的计算系统内存已经足够放下所有的数据集 [**farm**]，这使得有许多研究利用内存的特性来优化事务的并发控制。现今国内外有许多分布式下事务系统的并发控制的研究，这些方法一般都是基于上述提到的这两种方法的分布式版本。

Rococo[**mu2014extracting**] 提出了一个在分布式下的新的并发控制方法，具体做法是，事务在执行时

先去搜集与之冲突的其他事务，而当前的事务并不执行缓存起来，当发现与其他事务冲突时记录这些冲突并且返回后供第二阶段执行，在第二阶段执行时对和事务冲突的所有事务进行排序再进行执行。Rococo 可以有效的避免乐观并发控制中因为冲突造成的终止重试，然而它的问题在于，需要执行两轮通信才能确定事务的执行，这样事务执行的开销在分布式下仍然非常大。同时 Rococo 依赖于事务切分来分析出不可排序的事务集合，在这种情况下 Rococo 则退回到 OCC 来保证所有事务的正确执行。

Lynx[lynx] 利用事务链的抽象来将一个事务分成在多台机器上执行的块，并且依赖 Transaction chopping[**chopping**] 的技术来保证只要这些块被原子隔离的执行，并且这些块之间没有 SC 环，那么整个事务都会被原子隔离的执行。这样 lynx 可以不用考虑机器之间事务的同步，提高整体事务执行的效率。然而 lynx 的问题在于，通常事务都比较复杂，很难将一个事务拆成没有 SC 环的这些块。在这种情况下，lynx 则利用传统的方法比如说乐观并发控制来保证事务与其他事务之间的原子隔离性，这在分布式情况并不是很高效。

Calvin[calvin] 优化了 2PL 在分布式环境下二阶段提交 [**2pc**] 的性能。一般分布式事务提交的时候需要二阶段提交来保证所有机器都达成事务是否提交的一致。Calvin 利用一个序列层，将所有当前的事务在一段时间内累计起来，在序列化层记录事务的确定提交，并且当这段时间过后给所有事务排序再以这种顺序去拿锁，这样不需要所有机器去统一提交，因为如果在序列层确定了顺序的化那么整个数据库的状态就是确定的。同时这样可以避免死锁的问题。Calvin 可以带来很高的并行性，但是本质上 calvin 仍然使用了二阶段锁来执行事务，上锁操作在单台机器中仍然会带来开销，这在内存数据库中是一个很大的开销。最后，事先等一段时间事务的积累会带来更大的延迟以及更多的网络开销。

Hstore[hstore] 使用一个全局的协调者来同步所有的跨多个机器的事务，对于只在一台机器执行而的事务则在当台机器作为单机事务直接执行，否则则使用全局的锁来分布式事务执行满足隔离性。Hstore 将每个数据库分区分在一个核上，这样可以减少单机事务之间同步的开销。同时也就是说一台机器可能会有多个分区。这本身并不恰当，因为这样原本一台机器可以执行的事务反而会作为分布式事务执行。同时，使用一个全局的协调者本身不是一个可扩展的设计，当分布式事务变多时性能会非常差。

还有一些单机的内存数据库系统 [**silo**, **dbx**, **htm**]，他们关注的是如何在单机下扩展系统的性能。比如说 DBX[dbx] 利用 RTM 来保护乐观并发控制中的提交阶段，这样可以有效的避免传统乐观并发控制中的提交阶段对写的集合上锁并且对读集合的验证过程的开销，而且一般乐观并发控制中需要一个确定的序号，这一般需要一个全局的序号。这会带来扩展性的问题。[**silo**] 虽然 silo[silo] 避免了一个全局的序号，但是仍然会有上锁和验证操作的开销。

Timestamp Order[tso] 是一种不同于 2pl 和 OCC 的并发控制方法，但是由于在磁盘系统不高效所以没有广泛的实现。[**htm**] 然而随着数据库的数据集可以完整的放入内存使得 TSO 的实现变为可能。HTM[htm] 基于 RTM 实现了 TSO，利用 RTM 保护原本开销很大的时间戳检查操作。但是 HTM 仍然是一个单机系统，依赖于事务冲突的重试，在分布式下重试会带来非常大得开销。[**mu2014extracting**]

除了上述工作外，还有一些相关工作比如 Spanner[spanner]，但是 Spanner 更关注于如何在全球范围内可靠的分布式执行事务，为了保证可靠性而采用了 paxos 协议 [**paxos**]，这在分布式下效率非常

低因为分布式情况下的 paxos 需要许多轮数据通信。同时还有一些工作关注于放松事务的隔离性和原子性的限制，比如 Percolator[bhatotia2011large]，只提供了快照隔离机制 [so]，从而牺牲正确性来达到更好的性能；而一些系统如 cops[cops] 只提供了一个键值储存，并没有提供完整的事务保护。同时对于键值之间的一致性也有放松，因为 cops 希望提供的是跨地域的键值访问。FaRM[farm] 利用 RDMA 提供了分布式共享内存的抽象，并且提供了事务执行的描述。然而 FaRM 只是利用了 RDMA 来提高乐观并发控制中收发消息的效率，并且 FaRM 的事务并非针对传统的数据库事务，只是用来分配和访问内存对象。其论文中也没有详细的说明和测试。本质上来说 Farm 的并发控制方就是乐观并发控制，并且使用快速 RDMA 消息优化多轮通信的开销。

1.3 主要研究内容

本文主要考虑如何使用新的硬件特性如 RTM 和 RDMA 加速分布式下事务的并行执行的效率，使分布式下每台机器的效率接近同等配置下单机系统的效率，并具有良好的扩展性。近几年来，有许多研究开始使用硬件性能来提升系统的性能，比如 DBX[dbx] 使用硬件的事务性内存来加速事务并发控制的记录读写集和验证工作，而也有系统如 Pilaf[pilaf] 则使用高效的单方向 RDMA 操作去实现键值数据库。然而，硬件事务性内存无法在分布式下运行，同时没有利用 RDMA 实现的数据库事务处理系统。

本文选用 DBX-TC[dbsstx] 系统作为基准系统，使用了 RDMA 设计并实现了分布式下的事务性内存，并且使用多种优化提高系统的可扩展性。在传统的 OLTP 标准的测试下，系统中每台机器的性能接近原有 DBX-TC 的性能，同时当机器数增加和处理器核数增加时系统仍具有接近线性的可扩展性。

1.4 论文组织结构

本论文的组织结构为，本章介绍了工作的背景和意义以及国内外相关的研究，以及主要的研究内容。第二章介绍了与本项目相关的技术背景，包括使用到的 RTM 技术和 RDMA 技术和本系统基于的 DBX-TC 系统。第三章介绍了 DrTM 的设计与实现，包括如何利用 RDMA 将硬件的 RTM 保护扩展到分布式环境下作为分布式事务内存，以及相关的优化技术。第四章是对系统的一些测试，第五章是总结与展望。

1.5 本章小结

本章主要介绍了项目的背景和研究意义，以及当前各种事务处理系统的现状。接着介绍了本项目的主要研究内容以及论文的组织结构。

第二章　相关技术背景

2.1　RTM

事务性内存是为了方便多线程应用程序的编程而提出的。通常多线程程序为了保证程序的正确性，即并行操作间的隔离性需求，需要上锁来保证程序的正确执行。锁的力度越细，允许的并发度越高但是上锁可能会有开销，同时不方便编程，容易造成人为的错误。[**mpi**] 事务性内存则是一种替代这种基于锁的保护方法，事务性内存不需要应用程序来显示的对程序访问的内存数据进行保护，只需要让程序进入事务执行模式。同时处理器会尝试执行程序，如果遇到冲突则重试。这样可以避免由于用户拿锁所造成的诸如死锁的问题，而且没有上锁的开销。事务性内存通常有软件方法和硬件方法的实现，虽然有软件的事务性内存，但是软件事务性内存通常十分不高效。[**htm**]。由于事务性内存的需要记录程序读写集合并且当冲突时需要回滚，所以通常软件的实现性能很低。Intel's Restricted Transactional Memory(RTM) 是 Intel 在实现的硬件事务性内存。RTM 实现了强原子性 [**strongatomicity**]，即其他非事务性的指令会中断事务性执行程序，如果两者之间有冲突的话。这一点非常重要，在之后的正确性证明中会说明。RTM 使用处理器的第一级缓存来记录事务性程序的读写集合，利用处理器的缓存一致协议来检测执行时遇到的冲突。[**htm**] 所以这样的做法十分高效，因为对于冲突事务的回滚可以直接将下一级缓存中的数据直接写回上一级缓存，同时可以利用硬件高效的记录读写集合。

RTM 提供的调用分别是 XBEGIN,XENDXABORT 和 XTEST。一般如果一个应用程序需要保证一段程序事务性的执行，需要在程序片段的开头调用 XBEGIN，同时使用 XEND 来说明事务执行的终止。这样处理器就可以知道需要事务执行的代码段。比如下代码2.1的例子所示。

代码 2.1 一段使用 RTM 的 C 程序

```
1 void transafer(int account0,int account1,int balance) {
2
3     xbegin();
4     if(account0 - balance < 0)
5         xabort(0x1);
6     account0 -= balance;
7     account1 += balance;
8     xend();
9 }
```

这段程序执行了一个简单的转账程序。通常为了保证并行的执行的更新不被丢失，需要在函数执行前获取一把锁来保证程序执行的隔离线，即用锁保护这段 CS（Critical Section)。这样的话多线程下这种执行还没有单个线程执行有效率，因为逻辑上就是一次一个 CS 执行，然而拿锁还有开销。然而，使用 RTM 的话，只需要将这一段程序放在 XBEGIN 和 XEND 中，处理器会保证这段程序像一条指令一样原子的执行，这样就可以保证隔离性。同时允许同一段程序并行执行，只要没有冲突，

这样会带来比较好的可扩展性。另外，在事务程序执行中，可以调用 XABORT 显示的退出事务性执行。XABORT 接受一个 1 字节的值作为返回值可以给应用程序检查退出的情况，程序会退出到调用 XBEGIN 的地方，并且程序对内存的操作直接忽略。在之后的执行中可以用 XTEST 去判断是否在 RTM 区域中。

尽管 RTM 使用起来非常高效方便，在使用中仍然有许多限制。[**rtmconstraint**] 首先，由于 RTM 使用处理器的缓存来记录事务性程序执行时的读写集合，因此程序不能执行过长，否则当缓存存满时变会强制终止执行。这一般称为 capacity abort。这意味着太大的 RTM 区域将会没有办法执行。其次，通常在 RTM 如果提交不成功则处理器会不断重试这段程序的执行直到成功，所以处理器并不保证事务程序执行的一定成功，这时候就需要显示的退出 RTM 执行并使用常规方诸如加锁的方式保证程序依然能够正确执行。这种退化的方式一般称为 Fallback handler。程序可以使用上文所述的 XTEST 来检查是否到 Fallback Handler。最后，系统调用或者中断会百分之一百中断 RTM 的执行，意味着无法在 RTM 中进行系统调用或者类似的操作。这给能进入 RTM 中的程序的种类带来了限制，或者通常需要采用别的方法来达到原来的系统调用的效果。

2.2 RDMA

Remote Direct Memory Access(RDMA) 是一种新的网络通信方式，由应用程序直接向网卡发送请求并且双方的网卡直接处理请求，拥有高吞吐量，低延迟和高处理器利用率等特性。传统的通信一般是用类似 TPC/IP 协议进行收发网络包来完成，如2–1所示。我们可以看到，一个完整的 TCP/IP 的通信在每台机器需要经过多层调用，其中需要进入内核，这会带来很大的开销，同时每次进入一层都会对网络包进行拷贝，这也会带来额外的不必要的开销，因为数据并没有变化。同时，接受信息的机器同样需要走相同的过程去处理网络包，这样接受信息的机器还需要额外的处理器去处理这些操作。一般这些处理器需要轮询来监听信息，这样还会浪费很多处理器时间，这使得处理器越来越成为计算的瓶颈。[**farm**] 而 RDMA 得操作如2–1所示，应用程序直接将数据发给网卡，再直接由网卡发给对方的网卡进行操作，这样不需要处理器的参与同时不需要昂贵的进入内核以及减少数据来回拷贝的次数，因此 RDMA 相比于原有通信有更高的吞吐量和更低的延迟。

RDMA 提供多种通信模式，通常有一个称为 verbs 的库提供给上层应用程序调用。RDMA 通常有如下接口2.2, 包括对远程内存地址的读写操作和直接利用网卡的接受发送消息。Rdma Read 和 Write 分别对远端机器内存进行读写，而 cas 则对远端机器的内存作类似处理器的比较再替换操作。

代码 2.2 verbs

```
1  RDMA_read(char *local_buf,int size,int remote_off);
2  RDMA_write(char *local_buf,int size,int remote_off);
3  RDMA_atomic_cas(char *local_buf,int size,int remote_off);
```

其中比较需要注意的是，RDMA 提供一些对内存的原子操作，比如说原子的比较替换。但是需要注意的是，不同厂商网卡之间的 RDMA 的原子操作的语义不同，即和哪些操作互相原子的范畴不同，可以使用 verbs 的接口查看当前系统的级别，[**rdma_manual**]，其中有全局模式, 即 RDMA 的原子操作和处理器的原子操作兼容，还有设备模式，即只有网卡之间的 RDMA 操作是相对于对方原子，还有不支持原子操作。不同的模式影响程序的性能，因为 RDMA 的原子操作的开销还是要

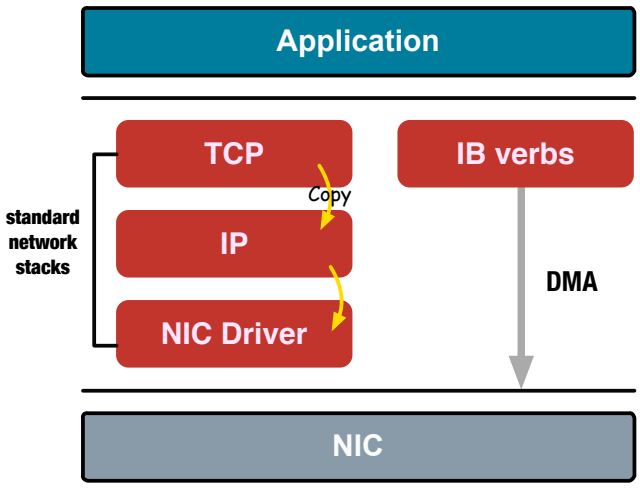


图 2–1 TCP/IP vs RDMA
Fig 2–1 TPC/IP vs RDMA

比处理器的原子操作开销大很多。[1]

有许多研究表明 RDMA 的单轮操作，即对远端的读写的操作具有最好的性能 [**farm**, **herd**]，虽然会带来使用的限制，比如只有读写操作，但是本文将关注如何利用这些操作来优化系统的性能而不是只将原来系统替换为更快的网络。

2.3 DBX-TC

本系统基于 DBX-TC[**dbsstx**], 是一个单机的利用 RTM 实现的内存事务性数据库。

2.3.1 DBX-TC 架构

DBX-TC 总共有两层架构，分别是数据库层和事务层，如下图所示2–2。

其中数据库层提供了一个并行可访问的键值数据库，其中主要使用的数据结构是可并行访问并且修改的 B+ 树。通常高扩展的并行 b 树非常难以实现，而 DBX-TC 使用一个完整的 RTM 区域来保证每个数据库操作的原子性，这样可以借助硬件特性直接高效的实现并行 b 数。[**dbx**] 然而作为一个事务性数据库，我们知道仅仅保证对数据库的单个操作原子性是不够的，因为一个事务可能会有多个数据库操作。因此 DBX-TC 提供了一个事务层，使用 Transaction chopping 和静态分析来将一个事务切成多个小块 [**lynx**]，随后使用 RTM 区域来保护每个这个区域的原子执行。因为我们在上文2.1中说过，进入 RTM 区域的读写集合不能过大，否则程序会没法成功执行。Transaction chopping 是一种分析事务并把事务切分成更小的事务的方式，其中每个被切出的块称为 piece。将事务切分后，事务访问的内存便没有完整的这么多，因此可以避免 capacity abort。Transaction chopping 依赖于静态分析来确定切分后的执行是否合法，即如果切分后的 piece 当做事务的力度来执行，最后所有事务的执行是否保证序列化。

[1] 在我们的测试机器所采用的是 IBV_ATOMIC_HCA, 即只有 RDMA 操作之间是相互原子的。

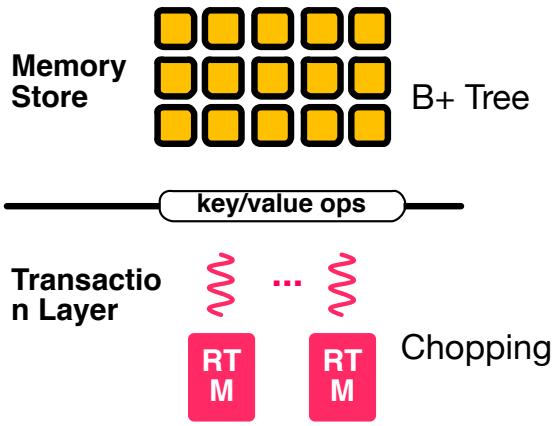


图 2–2 DBX-TC
Fig 2–2 DBX-TCoverview

2.3.2 DBX-TC 的事务切分

一个 chopping 的 SC 图有如下方法构建。SC 图中的所有点为所有事务的 piece，一般每个事务的每个 piece 会作为两个点在图中，用来捕获通类事务间的冲突关系。当 piece 之间有冲突，也就是说两者之间可能操作共同的数据，而其中至少有一个是写时，两个 piece 之间冲突。这时候给冲突的 piece 间加上 C-边。同时，对同一个事务的所有 piece，用 S-边来连接这些 piece，同一个事务间从开始到结束的 piece 间只有一条 S 路径。根据 [**chopping**], 有

定理 2.1. *一个切分 (chopping) 是正确的，即对切分后的 piece 作事务执行后的结果对应于一个所有事务的序列化执行当且仅当这个 chopping 的 SC 图没有一个包括 S-边和 C-边的环。*

DBX-TC 使用多种方法，包括可合并的操作，比如说对两个整数做加法操作并不会带来额外的操作，以及使用多版本数据库来不保护只读的事务，这样只读的事务就可以从 SC 图中去除等等 [**dbsstx**]。对事务预先切分后，这样切分完后的 piece 间就可以放入一个 RTM 区域中，这样由硬件就能够保证 piece 间的执行满足事务性，进而根据2.1以及静态分析得出的 SC 图来保证整个事务层执行事务的正确性。

2.3.3 DBX-TC Fallback handler

由2.1中提到，RTM 的操作不保证能够百分百执行通过, 即程序可能一直在重试 RTM 之中的代码。DBX-TC 提供一个特点 Fallback handler 来保证程序最终的执行。具体的做法如下所示2.3，

代码 2.3 fallback handler

```
1  if(retry_time exceeds times){
2
3  }else{
4      RTM_start(spin_lock);
5  }
```

```
6     if( not XTEST()){
7             //Fallback handler
8         lock(spin_lock);
9         ...
10        unlock(spin_lock0;
11    }
12    RTM_end();
```

即如果 RTM 代码重试的次数超过一个初设的值，那么就不进入 RTM，使用一把粗粒度的锁来保护并发执行。所有 RTM 程序需要显示的读这把锁，这样拿锁的操作会中断其他 RTM 程序的执行，保证 Fallback handler 与其他事务执行之间的隔离性。由于一般进入 Fallback 的次数不会特别多，因此 DBX-TC 只使用粗粒度的锁来保证 Fallback 程序和 RTM 程序间的正确性，同时不会明显的影响效率。这样还有不容易发生编程错误的好处。为了避免一把全局锁而提供更多的并行性，DBX-TC 使用了稍微细粒度的锁，即每个数据库分区共享一把锁，每个事务只获取和检查他所操作的分区的锁。这样同时可以有多个 Fallback handler 在执行，达到更多的并行性。

2.4 本章小结

本章主要介绍了本项目基于的系统 DBX-TC 和使用到的 RTM 和 RDMA 的技术。

第三章　设计与实现

DrTM 的设计基于 DBX-TC，继续使用事务切分来执行原来的事务。只是当遇到远端的数据操作时，DrTM 采用新的方法保护隔离性和原子性。具体的设计与实现包括以下几部分：能够在分布式环境下保证隔离性的并发控制方法，能够使用 RDMA 的单轮操作访问的数据结构以及对并发控制方法的实现细节，包括只读事务的并发控制协议，以及 DrTM 的容错机制。

3.1　设计

3.1.1　DrTM 并发控制方法

DrTM 分别将 DBX-TC 的数据库层和事务控制层进行扩展。DrTM 的数据层需要提供分布式的访问，特别的，这些数据需要使用 RDMA 单轮操作访问，因此数据结构必须对 RDMA 单轮操作友好，和传统的 RDMA 键值数据库一样 [**farm**, **herd**]。而 DrTM 的事务层依旧基于 DBX-TC 的切分方法，但是利用二阶段锁来保护分布式操作之间的隔离性。DrTM 的具体架构图如下所示：3–1。

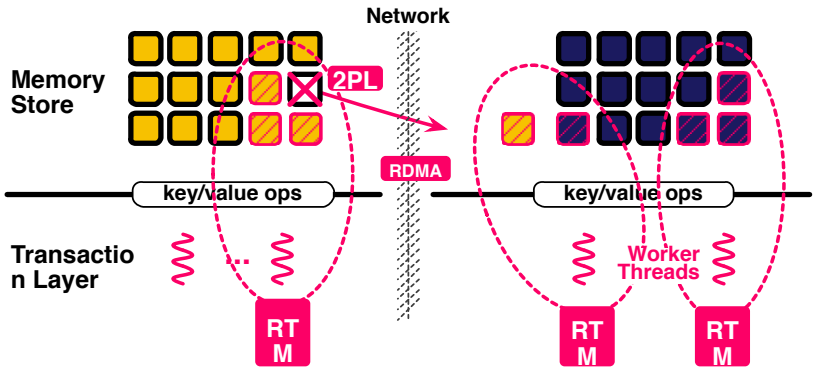


图 3–1 DRTM-arc

Fig 3–1 DRTM overview

DrTM 的对数据层的假设是存在一个可以供并行访问的键值数据库。其中对单个数据的并行访问的每个操作如读写和插入互相原子，并且相互隔离。DrTM 的事务层制基于 DBX-TC 的事务层，保证多个这种数据操作的的整体原子性和隔离性。由前一章的2.3可知，DBX-TC 的事务层基于事务切分，随后使用硬件特性 RTM 来保护每个切分的块的原子执行。DrTM 依旧基于事务切分，但是由于此时 DrTM 中得每个块可能包含对远端机器的数据的访问，因此不能简单将这个块由 RTM 去保护，这时候 DrTM 使用二阶段锁来保护远端操作。由于观察到，RTM 硬件实现的是乐观并发控制，而乐观并发控制的方法如下所示 [**occ**]，

1. 事务执行的时候，对其读的数据记录到读集合中，对其写的数据记录到写集合中，同时不将写的数据写回数据库中，在本地缓存起来。

2. 在事务提交的阶段，先将所有写集合中的数据上锁，同时检查所有的读集合中的数据是否被修改。如果有一个失败则重试，如果成功则将数据写回数据库再释放锁，提交成功。

当 DrTM 通过在进入 RTM 前利用二阶段锁的方法来提前将远端读写集合中的数据锁上，这样在2的检查步骤中就没有必要将远端的数据上锁或者进行检查，因为写的数据已经上锁而读的数据不会被修改，只要所有的事务在执行时检查自己的数据是否被上锁并且不进行修改而中断。这时候，在 RTM 的区域中，可以显示的检查数据是否被上锁，这样可以为了确保远端的操作可以中断 RTM 区域中的代码，保证硬件提供的语意，即当和 RTM 区域内的读写集合与其他事务或者非事务的操作有冲突时 RTM 会被中断。因为硬件需要读写集合有冲突时才会中断，所以原本单机的操作一定需要检查是否被远端上锁，这样就足够保证硬件语义。具体的正确性，即远端操作确实能够中断 RTM 将在下文说明。

DrTM 基于的观察是一般数据库事务执行中大部分操作的数据都在本地而不在远端比如说 TPCC，在一个好的划分下，80% 的事务 [**hstore**] 都能够在一台机器完成。通常将事务放在由包含其数据集最多的机器去执行，这样可以尽可能的使用快速的单机方法如 RTM 来保证事务的隔离性。同时，DrTM 开始的二阶段锁操作不会带来许多的开销，大部分的事务操作都会在单机使用高效的硬件特性去完成。当事务遇到一个被上锁的数据，说明这个数据被一个分布式的事务访问，那么此时本地事务变回被中断，这样起到了优先执行分布式事务的特性。因为在分布式环境下的保护方式使用二阶段锁，相对于本地操作来说开销要大许多。

3.1.2 DrTM 数据层

DrTM 的数据层需要重新设计以满足远端操作的访问，更具体的是事务的远端数据操作使用 RDMA 单轮操作来完成，这样可以最大限度减少操作延迟和提高处理器的利用率以及利用网卡的吞吐量。因为 DrTM 的目标在于使用 RDMA 单轮来进行远端数据的读写操作，特别的需要使用 RDMA 的单轮的读，写以及原子操作才可以达到不需要对方的处理器参与，这样最大化的增加处理器的处理效率和利用网卡的低延迟。尽管 RDMA 提供收发消息的接口，但是这些接口需要远方机器的处理器处理 [**rdma_manual**]，一般仍然需要使用轮询的方式，比如说类似 FaRM[**farm**]，这样就退化到了原来收发消息包得处理方式，所以对处理器的利用率来说并没有单轮的操作高。DrTM 希望能够只用 RDMA 的单轮操作来处理所有的数据操作。

然而，使用单轮的操作有一些问题。首先，RDMA 给所有的程序提供的是一个地址空间的抽象，因此存储数据的数据结构都必须自身可以解释，即客户端程序可以通过读到的一些元数据访问数据结构中的特定项。传统的数据结构诸如哈希表和 B 树都可以满足这个特性，因此不需要设计额外的数据结构。第二，访问找到特定数据项地址的 RDMA 操作次数不能特别多，因为数据结构诸如哈希表需要通过多次访问数据结构的元数据来获取确切数据的位置。否则，对于一个数据项的访问会使用过多次的 RDMA 访问，这样的延迟可能比发送消息再进行读取会更慢，这样便会降低整体的性能。由于第二项特性，DBX-TC 中 B+ 树便不适合用 RDMA 进行实现，因为一次 B 树的读取通常会访问多层节点。所以在 DRTM 提供给远端访问的数据层只使用哈希表来存储远端的数据结构。对于许多基准测试平台如 TPCC[**tpcc**], 对于远端的操作访问大部分是键值的访问而不是范围请求，因此使用哈希表作远端访问可以满足大部分需求。为了应对多种数据查询模式 [**data_access**]，对于单机的范围查询请求 DrTM 复用 DBX-TC 的 B 树结构来提供这种访问。

特别的，DrTM 采用传统的链式列表来处理冲突的哈希表实现。这种做法比较简单所以比较适合用 RDMA 去实现。这种哈希表通过一个哈希函数计算一个数据的键来得出这个数据的位置，同时将所有由哈希函数计算出得相同位置的键（即发生冲突的情况）使用链表来储存所有这些数据。这种还有一个好处是，只要每张表使用哈希函数被所有机器知道，那么可以直接通过在本地计算出特定键在远方机器中的链表的偏移量，这样就可以直接读取链表的信息来获取数据。如果数据在链表的表头，则只需要使用一次 RDMA 操作便可以读取到数据，这是十分高效的。这样数据结构下对每个数据的操作量和链表的长度有关，所以在这种实现下为了减少 RDMA 操作的次数就需要减少平均的链表长度。

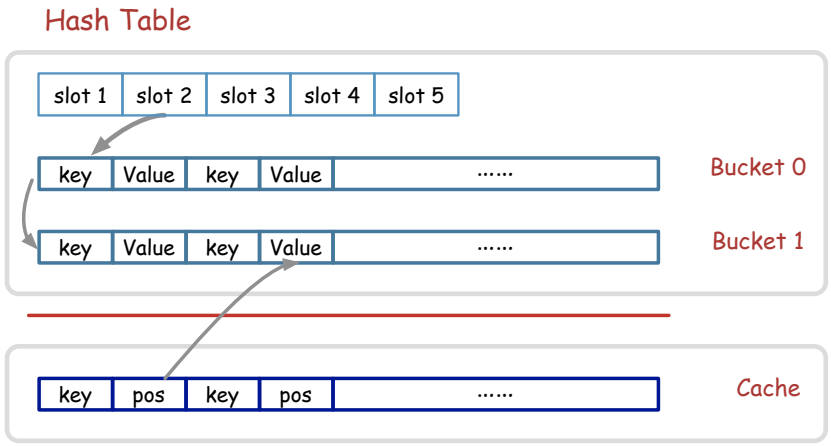


图 3–2 DRTM-data

Fig 3–2 DRTM hashtable

如上图所示3–2，为了减少每条链的平均长度，DrTM 扩大了哈希结构中链表的每项，这样每个链表项可以储存多个键值对。这样可以有效减少链表长度，即需要的 RDMA 操作的次数是链的长度除以每个链表存储的数据量的大小。这样，当表的键的分布比较均匀的时候，大部分的对数据结构的访问操作都可以使用 1-2 次的 RDMA 操作访问到，这样即使使用多次 RDMA 操作访问数据结构不会带来非常大的开销，同时能很好的提高处理器的利用效率。不过根据 [**farm**] 链表每项不能设置过大，因为 RDMA 对比较大的数据的读操作比读小数据要慢很多。不过由于目前冲突不是非常严重，因此不需要特别大的项，这里这个问题便不作考虑。另外，DrTM 的哈希表中存储的是数据在内存中的位置，或者说是偏移量，而 DrTM 保证数据的位置不会被挪动，即数据不会被挪动。这样，DrTM 可以在每个节点利用一个缓存存储读到的数据的位置，这样之后访问访问过的数据只需要一次操作便可以得到数据位置。每个数据的的具体内容包括键和数据的状态，表明数据是否被上锁，以及数据的内容。缓存则使用随机去除一个数据位置来存放新的位置。不同的替换策略并不是本文的重点，而且即使发生缓存缺失也并不会带来非常大得影响，因为正常访问的 RDMA 次数也不会特别多，因此这里不详细讨论缓存。

正如前文所述，对于需要使用 B 树来实现范围查找的操作，DrTM 继续服用 DBX-TC 数据层的 B+ 树。然而这些数据结构无法使用 RDMA 访问，幸运的是对于主流的测试基准来说远端的操作并

不需要方位访问，这样上文所叙述的哈希结构即可以满足要求。

3.1.3 DrTM 事务层

DrTM 的事务层实现了3.1.1中提到的并发控制方法，提供特定的方法供应用程序调用以保证程序运行的远端操作隔离性的上锁操作。应用程序必须在事务的开始和结束通过调用 DrTM 的事务层的特定接口来保证这次事务的执行满足隔离线和原子性的要求。同时，对于事务执行中的每个数据的访问，也调用相应的 DrTM 事务层接口，而这些接口，如下图所示的 [remote|local_read，[remote|local]_write 操作，这些操作是对上文叙述数据层接口的封装。具体给应用程序调用的各种接口如下图所示，3–3

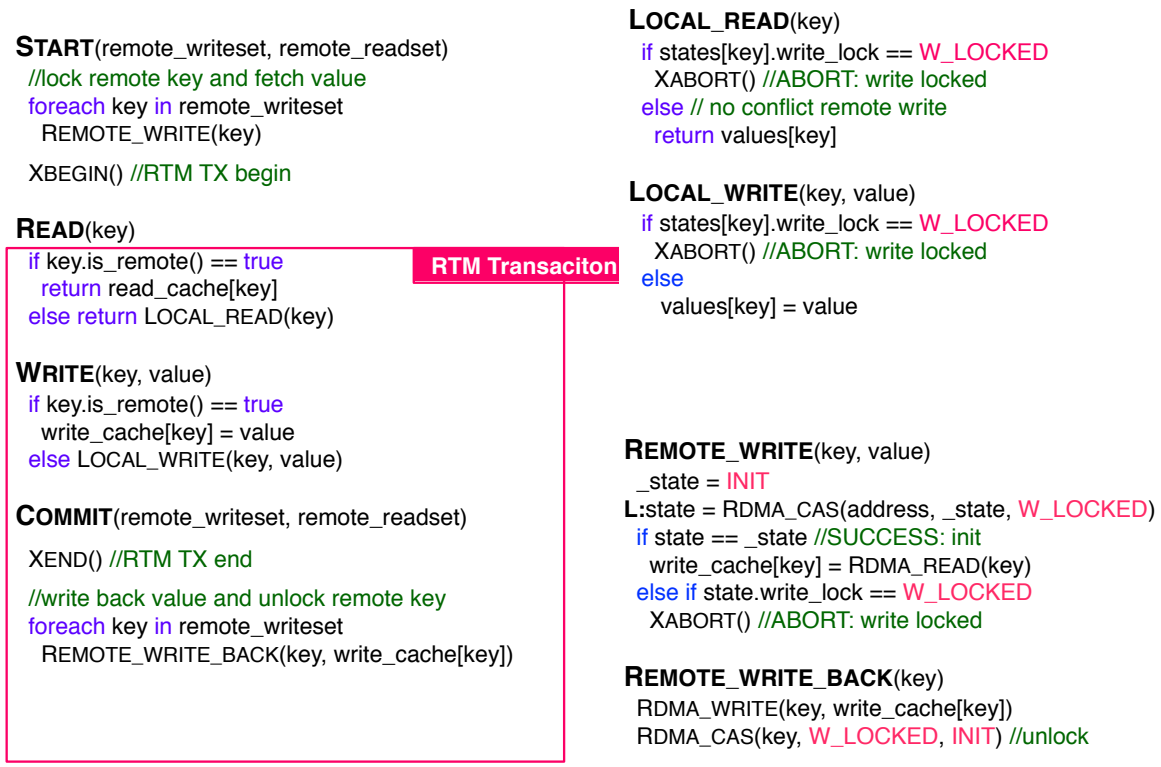


图 3–3 DRTM-code

Fig 3–3 DRTM 流程

在应用程序开始前，应用程序必须将事务的远端的读写集合，即将读和写的远端数据的键告诉 DrTM 事务层。应用程序需要显示的将自己的远端读写集合在 Start 前添加到 DrTM 事务层的读写集合中。这种做法会给可支持的应用程序的种类造成限制，因为当应用程序的读写集合不可知时便无法像往常一样工作。然而，类似的工作如 calvin[calvin] 也假设应用程序预先知道其读写集合，同时对

于流行的基准平台如 tpcc[tpcc] 都会预先知道每个事务的读写集合。因此，DrTM 的事务层假定应用程序知道其读写集合。DrTM 提供接口使得应用程序可以将读写集合告诉 DrTM，在上图中限于篇幅则不做说明。

随后，在开始阶段，DrTM 首先调用 start 调用，开始事务执行。DrTM 首先会将所有远端的读写集合全部使用 Remote_write 进行上锁并将数据读回本地缓存。其中上锁按照一个定义好的排序，比如说按照表的顺序和表中键的字典序。这样可以避免死锁 [**deadlock**] 的问题。DrTM 的上锁操作使用 RDMA_CAS 来实现上锁操作。数据结构中每一个键值对都有一个 state 表示其是否上锁。state 是个 64 位的内存值，因为 RDMA 最大的原子操作单位是 64 位 [**rdma_manual**]，然而这对我们来说足够了。state 是 0 表示不上锁，反之则为上锁。上锁操作会用原子 CAS(compare and swap) 将其变为自身的表示，如果失败则将一直重试，直到成功。这里的执行对应于 2PL 的方法，由于 DrTM 假定知道所有程序的读写集合，因此如前文所述可以采用对所有的远端数据按照一定的顺序依次上锁，这样便可以避免二阶段锁中的死锁问题。放锁操作则同样为一次 RDMA_CAS，然而放锁操作保证会一次成功。

在将所有的远端数据上完锁并且读到一个已经提交的版本到本地后，DrTM 事务层将开始事务的执行，通过调用硬件的 XBEGIN 方法进入 RTM 区域来保护本地的数据的访问。同时，对于本地的数据读写则和往常一样进行正常读写，唯一的区别在于需要显示的检查数据的状态是否处于上锁状态，如图3–3中的 local_read 和 local_write 所示。其中，所有的本地操作将会首先检查数据是否上锁，如果是则直接调用 xabort 退出 RTM 的执行; 随后才会对数据进行读写。这样可以保证如果事务在 RTM 执行中被远端的操作访问或者有本地的冲突操作，RTM 会被正确的中断, 因为数据和他的锁已经进入硬件的读写集中。[1]这里有一个优化是，如果发现操作的数据已经处于上锁状态，则在重试时直接进入 Fallback 而不是像原本一样再多次尝试执行几次 RTM 直到重试次数到了一定的数量。这是一个简单的优化，因为当数据被远端机器上锁时，由于 RDMA 操作比单机操作还是要慢很多，因此这时候硬件的重试并没有特别大的作用。所以直接进入类似 DBX-TC 的 Fallback 中来作一次慢的处理。DrTM 的 Fallback 的实现在下文的3.2.1中会具体的说明。对于远端的数据，则直接对通过 start 阶段读回的缓存，在缓存上面进行读写操作。

最后，当事务执行完成后，所有的远端数据的本地缓存会使用一次 RDMA_WRITE 写回，同时使用 RDMA 原子 CAS 操作将远端数据的状态置为不上锁，当所有远端数据都被写回并解锁之后，事务执行完成，事务执行的结果变得可见。这样应用程序的事务执行完成。

3.1.4 DrTM 方法的正确性

从3.1.1已经说到，DrTM 将单机硬件的 RTM 提供的保护扩展到了分布式环境，其并发控制方法的本质是对单机硬件实现的的乐观并发控制方法的扩展。因此如果分布式方法需要保证硬件的语义，即如果有一个和执行的硬件事务的读写冲突发生，则硬件事务会被正确的中断，则 DrTM 的方法将会保证正确性。因为 DrTM 将乐观并发控制的写的上锁操作提前，因此在硬件事务提交时可以不用对写的集合进行上锁。同时，提前对读的集合上锁可以保证在硬件提交时可以不用检查。故 DrTM 只用保证当有操作和 RTM 区域内执行的逻辑有冲突时 RTM 会正确的被中断。

在单机的环境下，本地的冲突操作会终端 RTM 这一点是由硬件来保证。但是在有远端操作的

[1]至于 RDMA 操作是否会中断 RTM 将在下文详细说明。

情况下，硬件事务是否会被远端的 RDMA 操作中断是一个需要考虑的问题。幸运的是，由于 RDMA 的操作一般是由对方网卡的 DMA(Direct Memory Access) 来实现，而 DMA 在现代的计算机上比如说 Intel 的机器上有缓存一致的特性 [**farm**]，所以 RDMA 操作可以中断执行正在执行的硬件事务，如果有冲突的话；下面我将具体说明。

由于在我们的系统中，网卡的原子级别为 IBV_ATOMIC_HCA, 即只有网卡的原子指令间相互原子。[**rdma_manual**] 这意味着，对于本地的操作则无法使用处理器的原子指令。这带来的问题是没有办法使用处理器的原子操作来实现锁，而这样会给性能带来一定影响，具体会在下一章描述。虽然这里我说明的是 IBV_ATOMIC_HCA 情况下的正确性，然而相似的说明对其他模式仍然适用，只要网卡提供原子操作。因为这一级别属于网卡提供的原子操作中的最弱的级别。

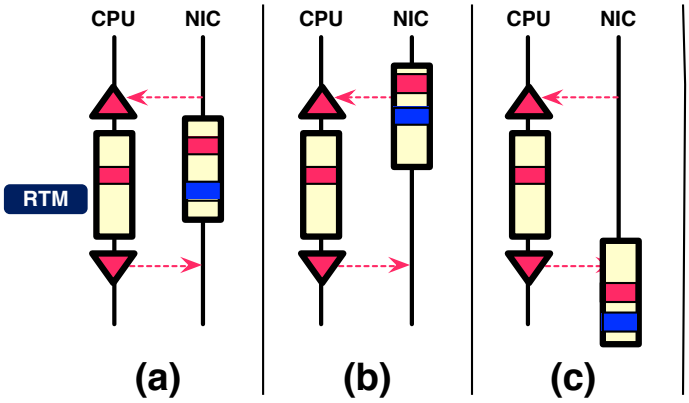


图 3–4 DRTM-correctness

Fig 3–4 DRTM 正确性，红色块代表读操作而蓝色块代表写操作

DMA 写操作会由一次 PCI 写操作实现 [**herd**]，如果该操作是缓存一致的，那么这个写操作将会中断 RTM。这里我讲基于这个假设，即每个 PCI 读或者写操作会中断 RTM，因为在我们的系统中 DMA 的写操作是缓存一致的。RDMA 的原子操作总共有两部分组成，第一个是读操作，将会读取地址的值并且和提供的值比较。如果比较结果相同，则将新值写回，否则则将旧值写回。

如3–4所示。左侧的箭头内的区域表示一个 RTM 区域。其中红色块代表对一个数据的读或者写操作。而另一边代表网卡对该数据的操作。右侧的红色块代表网卡的第一次读操作，而蓝色则代表第一次写操作。由于所有 RDMA 的读写前都会对数据上锁，因此当上锁成功后所有在 RTM 区域内的执行都可以不用考虑，因为 RTM 内的程序会自动检查并且如果发现数据处于上锁则中断执行。

3.1.3

当上锁操作发生时，总共有三种执行流程的相交情况。

1. 如3–4(a) 所示。这种情况 RTM 内的检查操作发生在网卡操作之间。但是 RTM 仍然会被安全的中断，因为第一个 RTM 读操作已经将数据读到了处理器的读集合中，而之后分 RDMA 写操作在不在另一台机器的 RTM 区间内可以讨论两种情况。如果在 RTM 区间内，则第二个网卡的写操作会中断 RTM 的执行。如果 RTM 没有被中断，则说明这个写操作没有发生。此时程序的正确性仍然得到保证，即硬件的语义仍然会被满足，因为远端的写操作会发生在上锁操作

之后，也就是会发生在这个事务执行完成之后。

2. 如 (b) 所示，此时写操作发生在 RTM 区中而 RTM 中的读操作还没有开始。这种情况对应于读一个已经上锁的数据，根据3.1.3中描述的事务层的方法，程序会显示的调用 xabort 来退出 RTM 执行，这样保证了硬件的语义。
3. 如 (c) 所示。此时的情况对应于上锁操作发生在 RTM 执行之后，因此不会发生中断，依旧保证硬件的语义。

综上所述，DrTM 的方法和硬件的特性保证了即使存在远端 RDMA 操作的情况下，硬件事务执行的语义仍然被保证，这样则保证了 DrTM 的基于乐观并发控制的方法的正确性。所以 DrTM 的设计能够满足隔离线和原子性的事务要求。

3.2 实现

这一节具体讲述 DrTM 的实现细节，以及实现中遇到的问题和解决方法，包括 DrTM 的 Fallback handler，即当 RTM 重试次数过多时选用的常规方法的实现，以及 DrTM 只读事务的遇到的问题和解决方法及其实现以及 DrTM 如何实现持久性 (Durability) 的要求，使 DrTM 支持完整的数据库事务的特性。

3.2.1 DrTM Fallback handler

由2.3可知，由于 RTM 不保证事务的成功执行，即事务可能一直处于重试阶段，使用 RTM 的应用程序需要一定的方法保证程序的最终执行。所以，DBX-TC 使用了 Fallback-handler，即当程序的重试次数超过一定限制之后，将会退出 RTM 的执行。而 Fallback handler 需要保证在这一阶段的执行仍然满足隔离线和原子性的需求。这一阶段，DBX-TC 将使用一些粗粒度的锁来保护 Fallback-handler 的执行，而 RTM 的事务将会显示的检查这些锁是否被获取。这样可以保证所有的事务无论在 Fallback 还是在正常执行都是隔离的。由于进入 Fallback Handler 的几率非常少，所以使用一把粗粒度的锁不会带来显著的性能影响。

首先，对于本地的读写操作，由于 RTM 实现的是强原子性 [**strongatomicity**]，所以 Fallback 中的本地读写会中断其他的 RTM，但是仍然需要保证其整体的原子性。DBX-TC 采用粗粒度的锁来实现这一目的。然而，在 DrTM 中不能采取和 DBX-TC 相同的 Fallback handler 策略。由于 DrTM 中的数据存在远端的访问，所以如果采取 DBX-TC 的方法，本地的粗粒度锁没有办法保证数据在执行 Fallback handler 中不被远端的事务访问。幸运的是，在 DrTM 中每个数据有一个状态位来表示其是否被远端上锁。因此，DrTM 的 Fallback handler 复用了这些数据结构，在执行 Fallback handler 时使用二阶段锁来保护所有的事务的读写集。这样便可以保证事务的隔离性。

有两点需要注意，第一，简单的上锁和放锁操作需要使用原子交换覆盖操作，因为 DrTM 的远端上锁操作是使用其实现。然而，根据网卡的不同原子性3.1.4，不同的本地上锁操作需要采用。由于 DrTM 测试的系统的网卡的原子性，DrTM 必须使用 RDMA_CAS 来给本地锁上锁，这带来了性能的影响。如果使用处理器的 CAS 将会带来性能提升，在下一章的测试部分会说明，然而这依赖于网卡的实现。然而同样由于 Fallback 次数不多，因此性能的影响不会特别大。

第二，当进入 Fallback 的时候，事务可能会持有远端数据的锁，简单的拿本地的锁会造成死锁，因为预先拿的远端锁不一定和本地锁满足一个预先定义好的排序。DrTM 采用了一种简单的做法，

即进入 Fallback 时先释放掉所有远端的锁，随后将所有的远端的锁和本地的锁按照一定统一的顺序上锁，即先前提到的统一的全序关系。由于远端的锁通常不会特别多而且进入 Fallback 的次数非常少，因此这种做法不会带来很大的性能影响同时可以避免复杂的预防死锁的协议。

3.2.2 DrTM 只读事务

只读的事务一般会带来很多冲突，通常会带来很多的 SC 环使得事务无法有效的切分 [**mu2014extracting**, **dbsstx**]。然而由于只读事务会访问许多数据又无法将其完整的放在一个 RTM 中保护，以为这样会带来很多 capacity abort。[**dbx**] 所以一般数据库系统会采用一个特定的只读事务的协议来序列化只读的事务，这样只读的事务便可以不用和读写事务一起放在 SC 图中分析。[**mu2014extracting**, **cops**, **dbx**]

DBX-TC 使用的是多版本数据库，[**silo**]，即数据库中的数据存的是多个版本。DBX-TC 有一个全局的时间窗，当时间窗过了一个之后下次修改操作将创建新版本。而只读的事务会读一个一致版本的数据，即所有的数据的版本都小于一个固定的时间窗，虽然可能是一个比较旧的的数据。这样，只读的事务不用被完整的 RTM 保护，这样就避免了只读事务访问数据过多导致的 RTM 容量问题。

然而，在 DrTM 中，这样的设计并不是非常恰当。由于 DrTM 的数据库需要提供 RDMA 的访问，那么多版本的数据库会带来问题。首先访问每次的写入操作会增加，因为可能要创建新的版本，同时垃圾回收也有问题。因为一般垃圾回收与确定新版本数据的创建依赖于一个全局时间 [**dbx**, **silo**]，而这在分布式下不能很有效的操作，因为一般比较难同步分布式下的时间。而且额外的建立版本的操作和垃圾回收，即回收那些不会被访问到的数据的版本会带来开销。而且尤其是垃圾回收操作，因为 RDMA 新分配一块内存作远端访问需要很大的开销，故多版本的构建对 RDMA 的内存压力也会非常大。最后，由于上文中提到的 DrTM 不会移动数据的位置这样可以带来很多好处，比如说可以缓存数据的位置并且可以减少事务中对访问过的数据的 RDMA 操作的次数，所以 DrTM 没有采用 DBX-TC 的只读事务的保护方法。

DrTM 的只读并发控制方法和先前的方法类似，采用不同的策略来对待远端的读操作和本地的读操作。一种简单的策略是对所有的读操作都进行上锁，这样可以借助于先前的 RTM 中的检查的策略来保证隔离线。然而，由于通常只读操作会读取过多数据而且每次 RDMA 的上锁操作相比原本的操作要慢很多，因此这种实现的性能非常糟糕。

DrTM 采用类似先前的混合的协议的策略，对于远端的读操作，DrTM 采用之前的二阶段锁的方式，首先将远端的数据上锁并将数据读回。随后，DrTM 使用如下图所示的方法3–5，首先执行一次尝试执行事务逻辑，对于每一个数据的读操作，如果是远端的读操作，则直接返回远端读回的操作；对于本地的读操作，DrTM 使用一个 RTM 来保护一个数据的读，这样保证能够读到一个提交过的数据，无论是远端的还是本地的数据。随后，当第一次尝试执行完成后，DrTM 再次执行一遍事务逻辑，使用同样的方法保证读到一个提交过的数据。当两次执行返回的结果相同时，则这个只读事务的执行是符合序列化要求的。[**mu2014extracting**] 因为如果有一个其他冲突的事务修改了这个事务的数据，则在第二次检查中可以检查出不同。由于大部分操作仍然是本地的，假设大部分的数据访问在本地，而本地的 RTM 操作只需要保护一次数据的读写，因此避免了 RTM 的容量问题。

最后，本地一次数据的 RTM 也需要提供一个 Fallback 的过程。这里的 Fallback 非常简单，如果遇到被上锁则获取数据的锁，而其他情况则简单重试。原因和 Fallback handler 里遇到有上锁数据的

```
START_RO(readset)
  //lock key and fetch value
  foreach key in remote readset
    REMOTE_READ(key, end_time)

L1:
   if key is remote:
     READ_RO(key)
   else:
     READ_LOCAL(key)
L2:
   …//same with L!

if result of L1 ==  result of L2:
   commit
else:
   goto L1

READ_RO(key)
  return read_cache[key]

READ_LOCAL(key)
  rtm:
```

图 3–5 DRTM-RO

Fig 3–5 DRTM 只读操作

情况一样，因为如果读到一个处于锁状态的数据，那么重试只会浪费时间，故此时退化为对该数据上锁。

在这个制度事务完成后，对所有的本地在 Fallback 中拿的锁的和远端的锁进行放锁操作，提交这个事务。

3.2.3 DrTM 容错和持久性

DrTM 通过定期将数据写到备份机器，而备份机器定期将数据刷回固态硬盘中来保证执行完事务的数据被保存下来。这样如果有机器崩溃了的话这台机器的数据可以由集群中的其他机器来恢复。Drtm 的数据恢复可以借助类似 Ramcloud[ramcloud] 的方法，可以使用集群中的所有机器来恢复奔溃机器的数据，这样可以做到非常少的宕机时间。

类似于 Ramcloud[ramcloud]，DrTM 依赖于某种当机器崩溃时可以把内存数据写回固态硬盘的硬件模块 [**dimm**]，有两点原因。第一，类似 Ramcloud，这样每次写日志的操作可以不用等待写回磁盘的时间，极高的提升性能。第二，对于 DrTM 来说，必须需要这样的模块才能保证有效的容错。因为当 Drtm 的本机内存使用 RTM 提交之后，提交的数据可能会被远端的机器访问。这样，当本机奔溃后由于其数据会被其他机器读到，这样当这台机器崩溃后，其他机器依赖于没有被稳定记录的数据的事务都必须被中断，这样会极大的影响性能并且会非常难以实现。故类似 Ramcloud，Drtm 假设机器中有一块内存区域，记录在这块内存区域中的数据当机器因某些原因关机时仍然会被储存进磁盘。

这种日志区域无法变得大到将所有的数据库内容全部包括，因为这样在经济上十分不划算。[**ramcloud**] 所以一般会使用一块预先划分好的内存使用 DIMM 模块。DrTM 的内存日志的结构如下

图所示3–6，假设预先划分好一块内存供日志使用。日志包括三个区域，其中第一个区域记录每个线程执行事务的时候产生的新数据。这时候需要对原有的数据结构增加新的结构，即他的序列号。这样，每次对一个数据进行修改时，必须将其中的一个数据的序列号修改为比原来的大。这样，当恢复一台机器的数据而发现一个数据有多个版本时，则可以选取其中最大的一个版本来进行恢复。第二个区域记录了所有的远端机器的数据。第三个区域则供远端的机器使用 RDMA 进行写日志。除了第一个日志区域外其余的日志都需要被定期的写回磁盘，这样使得大部分时候所有的写日志操作都可以不写磁盘，最大化的提升效率。第一个日志区域不用写磁盘，因为当一个事务将其日志写回到远端时候，这个日志就可以释放了。第二个日志给每个机器的数据一个备份区域，如果一个机器的区域写满的话，则直接将这个机器的数据刷回磁盘。由于着这个操作并不常发生，因为写远端数据的频率并不高，由于有多个备份。第三个区域的的释放将在本节下文详细说明。

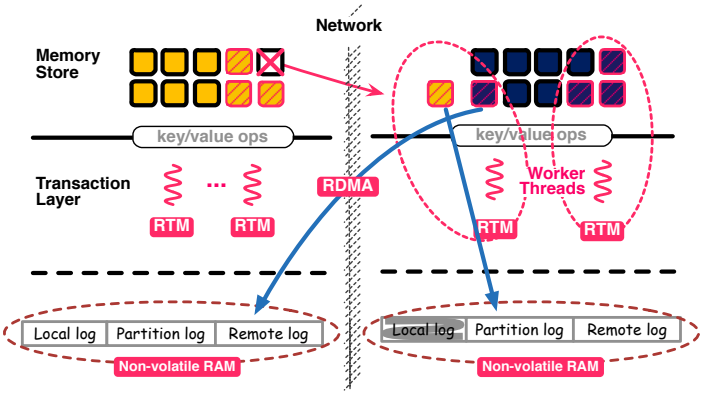


图 3–6 DRTM-LOG

Fig 3–6 DRTM log

为了使得事务的执行变得可以容错以及持久，DrTM 的事务层需要进行适当的修改。特别的，在开始的阶段，start 方法调用将给调用事务预先分配好日志的空间，而这些空间就在日志内存区域中的第一个区域。同时需要记录事务持有哪些数据的锁，这样可以在恢复时对数据进行解锁保证和这台机器不想管的事务可以正常执行。当事务执行完成后，在写回之前，事务将所有的远端的数据写到本地日志中的第二个区域，这样本地机器作为访问的其他机器数据的备份；最后在写回阶段的最后，将所有更改的本地数据及其序列号写到远端备份的机器中，使用一次单轮的 RDMA 写操作。每个机器会记住自己在备份机器中的日志的大小和位置，这样就可以使用一次操作来完成。

然而，尽管使用单轮的 RDMA 写操作虽然最为高效，可以有效避免写备份的开销，但是这样给日志区域回收带来了问题。如果当每台机器检测到日志已满再发消息等远方清空则会带来比较大得开销。Drtm 采用两个缓存空间来避免这个问题。每台机器有一个线程一直监听是否有缓存区已满。如果是则将其刷回。每台机器写缓存区时，如果发现缓存区已满，则使用另一个缓存区，并且将这一个缓存区标记为已满。标记包括两步，第一步将本地的标记为不可用，第二步使用 RDMA 写操作去告诉远方该区域已满，这样对方的监听线程便可以去清空。这样，大部分刷磁盘的时间基本被隐藏，因为只有两个缓存都变满时才会去等待缓存被清空，这样带来的开销变不会特别大。当监听线程完成操作之后同样使用一个 RDMA 写操作去告知对方缓存区已经变得可用，这两个操作基本

可以并行起来执行。

综上，当一台机器出现故障时，其他机器可以帮助恢复这台机器的数据，被恢复的机器的数据分布在备份的机器上，自己的缓存区的第一部分和所有机器中为他人备份的数据。可以利用[**ramcloud**] 的做法利用集群的所有机器来恢复，数据的最终版本则为所有数据中序列号最大的那个。如果需要容忍多台机器故障则可以在写备份的过程中写多台机器。由于数据会被定时刷回磁盘，因此以上过程可以保证事务的持久性。

3.3 本章小结

本章主要介绍了 DrTM 的设计与实现，设计中需要达到的目的和解决方案，和各种问题的方法。

第四章　测试

本章将说明 DrTM 的测试结果，并显示 DrTM 可以达到预期要求。特别的，DrTM 的测试将关注以下问题:

- DrTM 能够纵向扩展。即当集群内机器性能变好，比如说每台机器拥有更多的核时，即有更多的工作现场时性能能够相应的接近线性增长。
- DrTM 能够横向扩展，即当集群内机器数量变多时，且每台机器的性能一致时，DrTM 的性能能够获得线性的增长。
- DrTM 系统在单机下的性能，即 DrTM 的每台机器的性能相比 DBX-TC 的性能会不会带来很大的降低。
- DrTM 的容错对整体性能带来的影响不会特别大。

4.1　测试环境 - TPCC

所有的实验在一个 6 台机器的本地集群中进行。每台机器拥有两个 10 核的 Intel Xeon E5-2650 v3 处理器，并且拥有 64GB 内存。每个核拥有一个私有的 32KB L1 缓存和一个私有的 265KB 的 L2 缓存，同时每个处理器上的 10 各核共享一个 24MB 的 L3 缓存。

每台机器装备了一个 ConnectX-3 MCX353A 56Gbps Infiniband via PCIe 3.0x8 网卡，同时 6 台机器连接在一个 Mellanox IS5025 40Gbps Infiniband 交换机。每台机器运行 Ununtu 14.04，使用 Mellanox OFED v2.3-2.0.5 软件栈。

4.2　测试简介

这里所有的测试使用 TPCC[tpcc] 平台作测试基准。TPCC 是一个流行的 OLTP 事务处理基准。Tpcc 模拟了一个订单系统，包括了一些只读的事务的和一些读写频繁的事务。TPCC 包含复杂的表的依赖关系，同时每个事务可能会跨越多个表。具体的表的关系图如下图所示：4–1

Tpcc 的数据模型主要基于一些仓库，每个仓库有附属信息，包括用户，商品，和在这个仓库完成的订单。因此，Tpcc 一般可以使用仓库来将数据库进行划分，如4–1的继承关系。特别的，每个仓库的数据，包括其用户，商品和订单信息会被划分到一台机器中。从下面的详细描述中我们可以看到大部分的 TPCC 事务都可以在单机完成，但是仍然会有一定数量的分布式事务 (基本在 10% 左右)。具体的来说，TPCC 共包含 5 个事务，分别是 New-Order，Payment，Order-Status，Delivery 和 Stock-level。具体的逻辑如下所示:

- **New-Order** 模拟了一个用户下订单的过程。用户在其所在的仓库订购 5-15 个物品，其中每个物品有 1% 的概率在其他仓库供应。用户将订单添加到对于仓库的数据库中，同时将每个物品的供应仓库的物品数量减少。

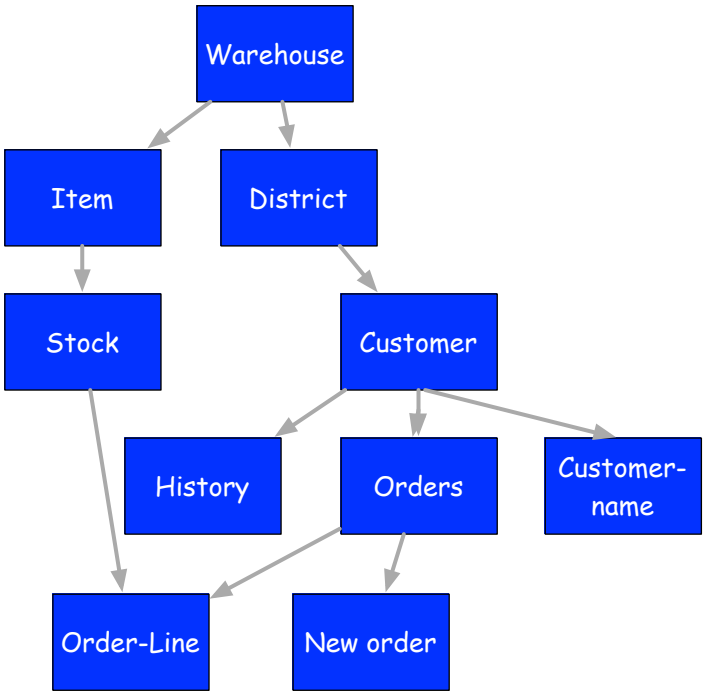


图 4–1 Tpcc - schema
Fig 4–1

- **Payment** 模拟了一个用户支付订单的过程。用户在某个仓库进行支付，更改用户的收支信息并且更改用户进行支付的仓库的收支信息。其中用户有 85% 的几率不在其信息所在的仓库进行支付。
- **Order-Status** 模拟了一个用户查看他最后的一组订单的信息的过程，是一个只读事务。其中这个仓库中这个用户最大的订单号的订单被获取，同时其对应的 Order-Line 也被获取。
- **Delivery** 模拟了系统处理 10 个订单的过程。一个仓库会选取一个区域，选取最旧的订单并且处理所有和这个订单相关的新订单，并更新相应的客户信息。
- **Stock-level** 模拟了一个检查商品库存情况的事务。这是一个只读事务，每个仓库检查最新订单相关的商品信息，如果发现如果某个商品的数量低于一定水平，则将这个商品号返回。

Tpcc 有 45% 的事务执行的是 New -Order 事务，，43% 的 Payment，其余的事务比例都是 4%。New-Order 而之中有 10% 的概率是分布式的事务。[1]因此 Tpcc 包含了不少分布式的事务。如果不详细说明，则所有的测试都是在标准的工作比例下进行。

[1]这里计算并不精确，但是可以通过调整访问每个远端物品的概率来达到 10 % 的分布式事务概率

4.3 扩展性测试

首先显示的是 DrTM 在机器数增加时能够获得性能的提升。4–2每台机器中运行 8 个工作线程，和以往的系统一样 [**silo**] 每台机器的仓库数和线程数相同。通过设置有 10% 的 New-Order 事务会访问分布式的数据。根据 [**tpcc**] 测试的结果将只汇报 New-Order 的执行吞吐量，而所有的测试都是按照标准的事务比例来运行，所以数据只记录 New-Order 的数量。如果不显示的说明，所有的测试都不考虑容错，即不使用 DrTM 的容错模式。容错模式给性能带来的影响在本章的后面章节会说明。

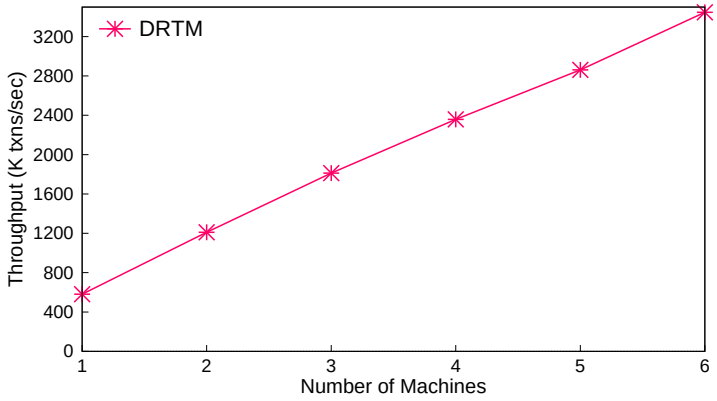


图 4–2 Tpcc - 机器增加时的吞吐量

Fig 4–2

可以看出，随着机器数的增加，DrTM 的性能能够线性的进行扩展。这是由于当分布式的事务并不是特别多 10% 时，利用快速的网络，大部分事务的逻辑都高效的在单机执行。每台机器的吞吐量都有非常高，接近单机情况下的水平。

下面的测试显示的是 DrTM 在每台机器数线程增加时的扩展性，即向上扩展性。每个测试都是在 6 台机器进行，每个机器的使用相同的线程数。同机器数的测试一样，每台机器的仓库数和线程数相同。测试通过增加每台机器的工作线程数来作测试。

如上图的测试结果所示，4–3，随着线程数的增加，事务执行的吞吐量依旧能够获得良好的扩展。同时，不同于 Hstore[**hstore**] 中只有特定线程能够执行特定仓库的事务，DrTM 中的每个线程都可以执行事务，从而当请求不均衡的时候仍然能够有效的利用机器的资源。虽然这可能会因为冲突变得严重会带来的额外开销，但是仍然能够尽可能的利用机器的资源。因为即使在一个仓库中可能带来的冲突也不会特别大。

下面的测试显示4–4的是 DrTM 在单机下的性能和 DBX-TC 的对比。可以看出 DrTM 在随着线程数变化的情况下和 DBX-TC 的性能相比并没有带来太多降低。甚至在线程数比较低的时候 DrTM 的性能更加好。这是因为在线程数低的时候基本不会进入 Fallback handler，而 DrTM 因为需要在 RTM 区域中操作更多的数据，因此刻意减少了程序中使用数据来减少 capacity abort。和 DBX-TC 性能的降低主要来自于分布式的冲突和在单机下 DrTM 的 RDMA 原子操作用来实现的锁。因为原本在 Fallback 中会有拿非常少的锁，然而在 DrTM 中需要拿细粒度的锁，虽然可能带来更多的并行性，但是所使用的锁的开销会特别大，因为每次拿锁操作都是需要网卡来完成，由于网卡原子级别的限制。通过使用处理器的原子操作可以使得本地在 Fallback 中的上锁操作更加高效，但是这一般需要

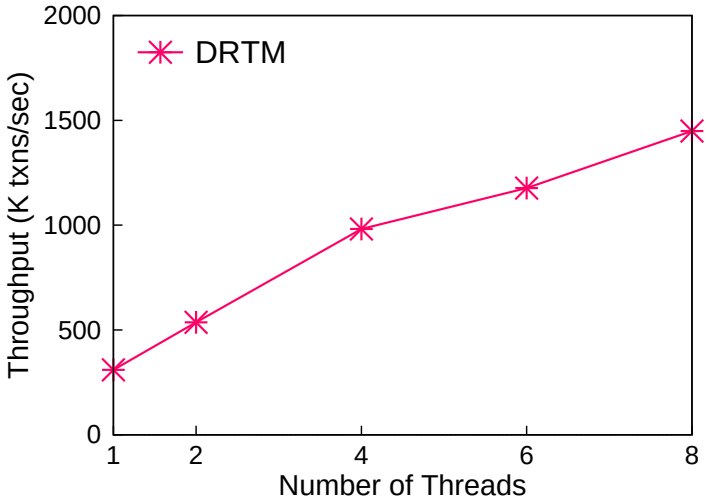


图 4–3 Tpcc - 线程增加时吞吐量

Fig 4–3

网卡的全局原子模式支持2.2。我们的系统使用的网卡原子级别并不是全局原子级别，所以本地上锁操作不能使用处理器的原子操作。所以这里只测试使用 RDMA 原子操作实现 Fallback 的上锁操作。虽然也可以使用处理器指令来作测试，并且很明显会带来性能的提升，但是这样在目前的测试平台并不是正确的实现，因此在这里不作考虑。

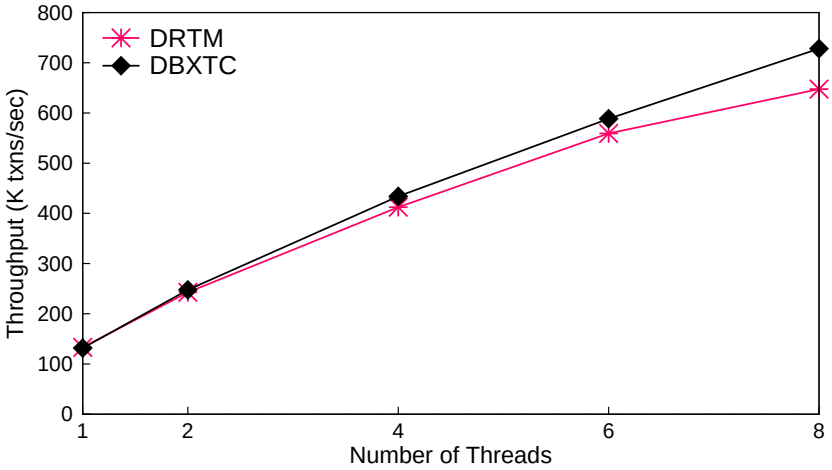


图 4–4 DRTM vs DBXTC

Fig 4–4

最后显示的是 DrTM 在提供容错的情况下性能受到的影响。开启 DrTM 的容错模式后 DrTM 将开启本地的记录日志以及在事务提交时将数据写到远端的备份机器，如3.2.3所描述的。这里主要会带来两个影响。第一个是对 RTM 容量限制的影响，由于每个事务的执行包含了更多的内存访问（日志的内存），而且通常日志的大小和写的数据的大小成正相关，所以事务的受到硬件影响的被打断

次数会增加。在试验中，总共会带来 23% 的更多的 capacity abort。这显然会带来更多的影响，不仅仅是重试会带来开销，还有更多可能进入 Fallback handler，即慢的路径。所以显然进入 Fallback 的次数越多显然会影响整体的性能。第二个影响是每次事务都会有一次 RDMA 的写操作，这意味着原本只用在单机执行的事务也会有分布式的操作。然而，由于 RDMA 的单轮写操作的开销并不大，而且这次写操作不会影响正常事务的执行，即这次写操作不会使其他事务无法操作该事务操作的数据，因此这带来的整体性能影响并不是特别大。最后需要说明的是，由于使用了双缓存区的策略，因此可以基本不用考虑写固态硬盘的开销，因为基本不需要等待，所以对 DrTM 的容错和持久性来说，对整体的吞吐量影响并不大，正如下表所显示。4–1

	Throughput (new-order/sec/machine)	**Latency** (ms)
`No log`	261,679	0.0137
`Logging`	217,742	0.0167

表 4–1 Drtm logging 的影响

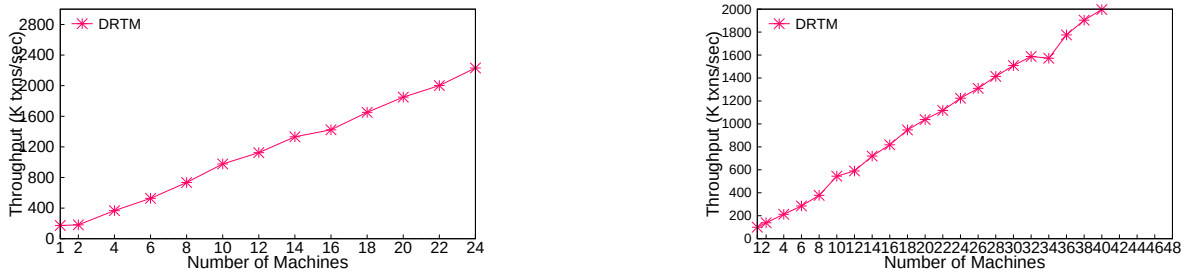
Table 4–1 Drtm logging's impact

容错的测试在 6 台机器上进行，每台机器运行 8 个线程，表中记录了每台机器平均的吞吐量和事务的延迟。可以看出，即使加上容错操作，DrTM 的平均吞吐量也只有基准没有容错的 83%，意味着 DrTM 的容错带来的开销处于一个可以接受的范围。同时，可以看到 DrTM 的平均延迟只比没有容错的情况下高 0.003ms，大概只有 3 微秒，这大致符合一次 RDMA 单轮写操作的延迟。所以可以看出 DrTM 在容错的情况下不会带来特别大的开销，即使原本单机的事务会存在远端的操作。

最后，由于我们的系统中只有 6 台机器，因此没有办法做当机器数扩展到更高数量时的吞吐量，即进一步衡量 DrTM 的横向可扩展性。为了模拟这种情况，我们使用一台机器来模拟多台机器，即每台机器运行一些分开的数据库。为了避免一台机器模拟的数个机器之间带来不必要的冲突，我们给每个机器上模拟的机器使用特别的处理器绑定策略，即将每个模拟的机器绑定到尽可能远的内存处理器节点上。这样模拟的机器间不会出现因为共享一个 socket 导致的缓存容量不足，这样可能会带来更多的 RTM 冲突从而导致的模拟的不精确。

在模拟的测试中，每台机器的线程数分别从 2 个调整到 4 个。因为我们的实验环境一台机器最多只有 20 个线程，而每台机器又需要一些额外线程来进行辅助工作，因此假定每台机器有 16 个线程可供工作线程模拟。因此一台机器模拟的机器数可以达到 4，和 2 个，这样最多可以模拟到 48 台机器，这是一个非常高的数量。不使用一个线程的设定是因为这样便无法体现 RTM 带来的多性能的优势，因为 RTM 是为了多线程加速而提出的。

实验测试的结果如4–5所示。可以看到，即使当机器增加到非常多，比如说 48 台时，如4.5(b)所示，DrTM 仍然能够达到接近线性的扩展性。



(a) 每个模拟机器 4 个线程

(b) 每个模拟机器 2 个线程

图 4–5 Drtm 扩展到更多机器

Fig 4–5 The scalability of Drtm on more machines

4.4 本章小结

本章对系统进行了测试，说明了 DrTM 在分布式环境下具有良好的扩展性，同时单机的性能不会带来太多的下降。同时，DrTM 的容错和持久性不会给系统带来非常大的性能影响。

第五章　总结与展望

5.1　总结

本文基本实现了预期目标，实现了一个分布式环境下的事务内存，并且具有良好的性能。尽管国内外对于分布式事务的并发控制有非常多的研究，但是大部分研究都没有考虑到硬件的革新所带来的各项特性。所以关于如何在分布式环境下如何利用新的硬件特性的研究并不多，所以可以参考的解决方案并不多。这同时也是一个好机会，可以深入了解各种新硬件的特性以及如何针对这些硬件特性进行优化。同时可以学习到许多并发控制的方法，结合对硬件特性的知识就可以扩展硬件性能实习一个分布式的事务性内存，进而实现一个数据库系统。

5.2　展望

本项目还有许多限制，还有许多可以完善的地方，可以在未来进行相关的工作。

本项目基于 DBX-TC，由于 DBX-TC 的 RTM 提交后结果即可见，所以必须先行获取分布式值的锁。这使得必须知道事务的读写集合，对支持的事务种类带来了限制。第二，DrTM 对所有分布式读写都采取同样的上锁策略，而是用一些其他策略如 [**war**] 的读的保证可以使得系统对读较多的事务有更好的效率。第三，DrTM 的数据层非常简单，对哈希表没有做过多优化，如 [**farm**] 同时不支持分布式范围查询。这都是可以改进的地方。

此外可以分析更多的事务基准平台，了解这些平台事务的方式，从而设计出更好的事务层和数据层，获得更好的性能。

最后，由于时间有限并没有实现并且测试 DrTM 的数据恢复。如何高效的恢复数据在容错的实现中非常重要，这可以作为之后的工作。

5.3　本章小结

本章对整个项目进行了总结，同时也提出了对于未来工作的展望。

致 谢

首先非常感谢我的指导老师夏虞斌老师陈榕老师以及陈海波老师。各位老师对我的指导十分细致，建议切实有效。各位老师对待工作认真负责，给我产生了很大的影响。从选题开始，导师就给予了很多的帮助，推荐给我很多相关的文献去查阅。在实现过程中，他们也是不断地督促我，同时给予各方面的指导，不仅包括了学业相关的指导，同时也有生活上的关怀。

非常感谢钱昊学长，总是耐心的回答和解决我关于 DBX-TC 的问题。

非常感谢施嘉鑫和陈彦哲两位学长，关于数据库事务方面的很多知识我都是半知半解，在两位学长的耐心教导下我学习到了很多知识，并且成功运用到项目中去。非常感谢他们。

非常感谢洪扬学长和杨朔同学，当我遇到 RDMA 配置的问题和集群机器的问题时他们总能给我提供帮助。

感谢 IPADS 实验室的全体成员，感谢他们在这个项目帮助我解决了很多困难，对我各个方面都有很大的帮助。

THE DESIGN AND IMPLEMENTATION OF DISTRIBUTED SOFTWARE TRANSACTIONAL MEMORY

Database transactions play a major role in e-commerce systems,they typically require the property of ACID,namely atomic consistency isolation and durability.With the increasing size of database datasets and the need to handle more requests the system is more more favorable in a distributed settings.Also with abundant memory available in modern servers the datasets usually can stored entirely in memory.This trend makes the typical concurrency control methods such as 2-phase locking(2PL) or optimistic concurrency control(OCC) which gaurantees isolation execuation of transactions becomes the bottleneck in a distributed main memory systems.Recently there is some new hardware features,such as Intel's restricted transactional memory(RTM) which can boost main memory transaction system's performance on a single machine,and remote direct memory access(RDMA) which can optimize distributed key-value store's performance.However,how to utilise these hardware features to optimize distributed transaction system is still a problem.

Most works focus on optimizing the concurrency control protocol,which mostly affect the system's performance.ACID property requires concurrency control protocl implements serializability,which means the effect of concurrent execuating transactions are the same as they are executed one after another.Traditional methods will allow large concurrency but will incur overheads,i.e,the lock acquirsim overhead in 2PL and the cost of retry and track transaction's read-write sets in OCC.These overhead are amplified in a distributed settings.

Recently there are some works that focus on reducing the overheads of these protocols,for example,there is some work that uses restricted transaction memory to reduce the overhead of optimistic concurrency control in a single machine system,and some works use transaction chopping to allow more concurrency in distributed settings or use reordering based on chopping to eliminate some reties in OCC.However,neither restricted transactional memory can work in a distributed settings nor does the reordering protocol reduce the overhead for transaction to track dependency,and the throughput of most state-of-art distributed transaction system cannot compete with single machine system unless using a lot more computing resources.There is some work which use RDMA one-rounded read-write methods to implement a distributed key-value store,given the fact that one-rounded methods have lower latency , higher throughput and better CPU utilization.However they do not support general propose database transaction.

In this paper,we propose a new system called DrTM which uses a new method to generalize RTM to provide a distributed transactional memory using RDMA.The design and system is based on DBXTC,which is a single machine main memory transactional database system.DBXTC use transaction chopping to chop the transactions into smaller pieces which can be protected using RTM without frequent capacity abort.DrTM extends DBX-TC's concurrency control methods to allow each piece to read and modify remote objects,and

use 2-phase locking to protect these access before each piece enter the RTM.DrTM also added a dedicated hash table which can be accessed using RDMA one round operation efficiently.Given the fact that most transaction's read-write set is in one machine,the new method will offload most of the execution of a transaction to a single machine,which is then utilize the efficient RTM to protect transaction's isolation properity.Also since remote access will abort local transactions,the distributed transactions will be prioritised over local transactions,which also improve performance because the retry of local transaction is fast.

DrTM provides a dedicated hash table for remote data access.DrTM uses a simple form of hash table which resolves collision using chaining.So the access times per item is decided on the list length of each bucket.Because It is desirable to use as little RDMA operations to access the data as possible,otherwise it will be inefficient to access an remote data item.DrTM uses a large bucket size to reduce each chain size.So if the keys are spread evenly,average number of operation to access an item is small.DrTM also do not move the location of an item during all operations so that each node can cache the item's position and then reduce further access operations in one transaction and further access.DrTM reuse DBX-TC's B plus tree for tables which require range queries.Since most range queries do not need remote access,so this is not a problem for application.

DrTM uses a hybrid protocol to protect transaction pieces which may have remote access.Before each piece enters the RTM,DrTM will use RDMA one round atomic compare and swap to lock each items and read their value to local cache.After each piece enter the RTM,it will read and write the cache of the object if the object is remote,otherwise it will execute normal read-write operation except that it needs to explicitly check that if the objects are locked.If the objects are locked ,it will manually abort the execution of RTM.By this way the objects and it's locks are in RTM's read-write set,which means that if there is an RDMA operation conflicts with some the read-write set of this piece,the RTM can be safely aborted.This is guaranteed by the cache coherence of the RDMA operations in our system.After the execution finishes,the remote objects will be written back and unlocked,thus exposing the result of this transaction.DrTM needs to know the read-write sets of each transaction,which is available using static analysis.If a piece enters the fallback path,which means that the RTM restarted time exceed some pre-defined number,DrTM will use normal 2-phase locking to acquire all the locks before start execuating.DrTM uses a canonical order to lock all the objects to avoid deadlocks.

Because the restrictions of transaction chopping and RTM capacity,we need a dedicated read-only protocol to execute read only transactions and thus can remove read-only transactions from chopping graph.DBX-TC uses multi-version objects so that read-only transaction can read a consistency snapshot of old values.However ,it is desirable for DrTM not moving an object's position after each operation and to simplify the data structure,so DrTM use an another method for read-only transaction which is similar to the main method.More specifically,for remote objects,DrTM will first lock and then read back the objects.Then DrTM simulates the effect of RTM by execuating a transaction twice and commit if the results are the same.During each execution,DrTM will use an RTM to read a committed version of objects from the database.Once DrTM encountered a locked objects,it will simply wait to get the lock of that object until the lock of the object is released.The lock is held during the execution so that there is no need to check this value after during retry.If the transaction cannot committed,DrTM will simple retry the execution,but there is no need to lock and read

back the remote objects.

DrTM provides durability and fault-tolerence by writing transaction's modification to backups after the transaction commits.Given the fact that once a piece exit RTM execution the local modification is exposed to other transactions,DrTM relies on DIMM module which can flush the data in RAM to solid state disks after machine failures.Each machine will have a dedicated RAM region to store the logs,and will log the local modification to the local log and release this log until the data is written to backups.Because of economy efficiency we do not put the whole database's ram using DIMM module so we have to use separated logs.Each machine will know it's backup's log offset so that it can use one one-sided RDMA write to write data to backups.Backups will regularly flush it's data to disks to ensure durability.To hide the overhead of writing disks,DrTM use 2 log areas for each node so that while the backup is cleaning one area the transaction can write other buffer.There is a thread on each node which will periodically flush the log data to solid state disk to release log space for future usage.The data loss of a machine is recovered by the local log of this machine and it's backup's logs.We use a sequence number to track the each data's version and ignore the recovery of a data with lower sequence number.

We do a lot of evaluations to show that DrTM can gain a throughput comparable to single machine database systems per node and can scale up or out.We do our test using TPCC benchmark and some small benchmarks such as small bank.TPCC benchmark is a well-known OLTP benchmark which simulates an online trading system.TPCC can be partitioned so that more that 80% of it's transaction can be executed as a local transaction,which means that it will only access the data on one machine.The test setup of the evaluation is on a cluster of 6 machines,each with 20 cores and are equipped with RDMA supported nic and are connected to a RDMA switcher.The results of the system shows that DrTM throughput on a single machine is compatible with DBX-TC with little performance drop,which is due to the slower fallback path.We increase the number of machines when each machine execuation 8 threads and shows that the system can scale out,we further fix the machine number to 6 and add more worker threads and results shows that the system can scale up.Because we only have 6 machine,we further simulates the scenario when there are more machines.We use one machine to run number of DrTM processes which can simulate up to 48 machines where each process run 2 worker threads.We manually bind each worker threads to different cores and different socket to reduce performance drop due to resource conflicts.The results shows that DrTM can also scale up to 48 machines.Finally we do some evaluations to show that DrTM's fault tolerence and durability support will not drop the performance very much.The results show that DrTM with durability support's performance only drop 17%,and increase little latency.This is because one more RDMA write and writing logs in RTM will incur more capacity abort of RTM.

In conclusion,this paper first shows the design and implementation of DrTM, which includes some optimizations to make the system more fast and to scale better.Then we provide a solution for durability which make the system robustness.At last we show the test results of the systems to show that the system fulfills the requirements.