一种应用于银行业务的分布式事务处理框架

A Distributed Transaction Processing Framework For Bank Front End System

吴 军 (中国科学院研究生院 北京 100071)
(中国农业银行软件开发中心 北京 100073)

摘要: 本文在对常用的分布式事务实现方式进行分析的基础上,根据银行前置系统的业务特点,结合了 Saga 模型、柔性事务以及基于知识库的异常规则处理方法等长事务处理方法的优点,提出了一种适用于银行前置系统应用的分布式事务保护模型,并在该模型的基础上设计了一个前置系统分布式事务处理框架。

关键词: 分布式事务 前置机 Saga 柔性事务 长事务 异常规则

1 引言

当前,银行业务系统中的跨系统处理应用越来越多,譬如银行卡的联网工程,国家现代化支付系统,以及各种各样的银行与外单位系统互连的中间业务处理系统等等。由于这些跨系统业务流程的特性,这些应用必须满足分布式事务的要求。也就是说一个完整的业务流程需要经过分布在不同资源环境下的多个处理环节,所有环节都处理成功后,才算一个完整业务流程的成功处理完毕。在银行业务系统中,通常通过前置系统来实现跨系统的信息交换及业务流程控制。为了提高开发效率,并保证稳定运行,有必要在系统设计时就认真研究从前置系统框架层次对分布式事务处理的支持。

2 常用的分布式事务实现方式

2.1 事务管理器及两阶段提交协议

处理分布式事务最常见的处理方式是采用事务管理器以及两阶段提交协议。通过事务管理器这样一个外部实体来保证一个事务的参与者能够跨越多个应用程序或者多个计算机。事务管理器管理着参与事务的所有资源,事务管理器通过与每个资源交换信息协调事务结束。事务管理器通过两阶段提交协议协调事务提交过程,由此来保护分布式事务的原子性和一致性。常见的分布式事务处理模型一般都采用这样的方法来实现,例如 X/Open 分布式事务处理参考模型 DTP 以及对象事务服务模型 OTS[3][5]。

这种方式的优点是比较好的满足了事务的 ACID 特性,但带来的问题是效率低下。另外,如果一个事务的分布层次比较多,事务管理器无法管理所有参与事务的资源时,就无法采用这种方式。在银行前置处理的很多应用中的单个事务都需要跨越了多层次的处理环节,因此并不适合用这种方法来实现分布式事务。

2.2 Saga 模型

长事务是分布式事务的一个子集,特指那些经过多层环节处理,执行时间较长的事务。线性 Saga 是 Garcia – Molina 和 Salem 提出用来解决长时间事务的方法[1]。Saga 模型中,长事务由预先定义执行顺序的子事务集合 T 和对应的补偿子事务集合 CT 组成。子事务集合 T 是子事务序列(T1,T2,—,Tn),补偿事务集合包含集合 T 中的每个子事务对应的补偿子事务(CT1,CT2,—,CTn)。任意子事务 Tk 都具有 ACID 特性。一旦子事务 Tk 完成,通过释放锁定,其操作结果其它事务均可见。只有当(T1,T2,—,Tn)顺序提交后,一个 Saga 事务才成功完成。如果某个子事务 Tk 执行失败,系统将通过执行补偿子事务来撤销 Tk 和前面已经提交的 k – 1 个子事务的操作结果,补偿子事务一般逆序执行,其执行顺序为(T1,T2,…,Tk,CTk,CTk – 1,—,CT2,CT1) 。

Saga 模型放松了对传统事务对隔离度和原子性

的要求，使得事务可以在全部结束之前释放某些占用的资源，这样可以提高资源的利用率，而且有利于对事务进行补偿，整体性能上比传统事务模型有很大的提升。考虑到银行前置机大部分的应用特点，Saga 模型是一种比较适合的模型。

2.3 柔性事务方法

柔性事务（Flexible Transaction）的关键是提供备用执行路径。如果一个子事务被取消，那么提交另外一个子事务并希望它成功执行以完成指定的任务。如果一个柔性事务的主子事务提交或者备用子事务提交，都可看作是柔性事务的提交。在工作流事务中，经常用这种备用路径的方法实现对工作流异常的处理[2]，从而实现对工作流分布式事务的保护。

2.4 基于知识库的异常规则处理方法

Mark Klein 和 C. Dellarocas 提出了该方法，先对异常进行分类，定义每一类别的特征，建立起知识库，每个异常有一个"异常探测"处理模板，来捕获异常，通过自顶向下的启发式搜索找到异常的原因，再采用相应的过程去处理。

除了上述的这些方法，还有其它一些基于事后补偿方式的分布式事务实现方式，它们都是基于异常发生后，从如何补偿的角度进行研究的。这方面的研究突出表现在工作流事务的研究中。每种方法都侧重于某一方面，有的侧重于补偿，有的侧重于重试。

3 面向银行前置系统的分布式事务保护模型

根据银行前置机的应用特点，我们提出了一种面向银行前置机应用的分布式事务处理模式，该模式在 Saga 模型的基础上进行扩展，结合了基于知识库的异常处理规则的优点，能够更灵活地适应银行业务的需要。

3.1 模型描述

假定事务 T 由 n 个子事务顺序序列 T1，T2，---，Tn 组成，T 中的每个子事务都有对应的补偿子事务，分别是（CT1，CT2，---，CTn）。任意子事务 Tk 都具有 ACID 特性。Tk 在执行完毕后都有输出结果 RTk，如果出现异常，可以通过 Tk 的返回值 RTk 知道异常的原因。f（RTk）是根据 RTk 匹配异常处理规则知识库的处理函数，根据 f（RTk）的处理结果决定后续的处理序列。

在正常情况下，事务 T 的各个子事务顺序执行，直到全部执行完毕。假设在执行到 Tk 时出现异常，则根据 f（RTk）的处理结果进行调整，一般有如下几种情况：

T1，T2，---，Tk，f（RTk）

CTk－1，CTk－2，---，CT1	回退处理
Tk，Tk＋1，---，Tn	重复处理
Tk'，Tk＋1，---，Tn	替代处理
Tk＋1，Tk＋2，---，Tn	忽略处理
Abort	中断处理

3.2 本模型与其它常用模型的关系

首先，本模型包含了基于知识库的异常规则处理方法。其中的异常处理函数 f（RTk）就是依据这种方法的思想来实现的。一般在设计一个完整事务时，需要分析最常见的异常情况，并根据实际需要配置好异常规则知识库，从而提高系统的整体可用性。

其次，本模型包含了 Saga 模型的处理方式，是 Saga 模型的扩展模型。从上面的回退流程可以看出，回退处理事实上就是 Saga 模型的处理方式，这也是本模型的最基本的处理方式。如果异常处理函数 f（RTk）匹配不到后续处理方式，则默认为回退方式。

最后，本模型包含了柔性事务的处理方式。替代处理就是从柔性事务的处理方式中吸取的经验。柔性事务一般而言更适用于工作流事务的特点，用于 OLTP 事务时，设计复杂度太高，而银行前置机的大量应用属于 OLTP 事务，因此一般情况下，在银行前置机里不会经常使用真正的替代处理方式。仔细分析可以发现重复处理和忽略处理事实上可以看作是两种特殊情况下的替代处理流程，一种是 Tk' = Tk 时的替代处理就是重复处理；另一种是当 Tk' = 空操作时，替代处理就是忽略处理。而这两种处理，尤其重复处理是银行前置机应用中常用的处理方式。一般需要通过 f（RTk）来控制重复的次数，以免造成无休止的循环处理。

3.3 本模型的适用性分析

本模型和 Saga 模型一样，放松了对隔离性和原子性的要求，从理论上会带来一定的风险。从隔离度上来说，由于在每个子事务完成后就释放了锁定的资源，因此有可能造成中间结果被别人访问。从原子性角度考虑，由于异常发生后，补偿事务也有可能发生异常，因此需要考虑其它特殊方式来解决事务不完整的异

常。但是由于银行业务的特点,这些风险都是在可控的范围之内。相比通过多阶段提交方式来实现分布式事务所带来的系统性能低下的后果,是一种合理的选择。

从放松隔离性带来的影响来分析,一般采用合理设计业务处理流程的方法来规避风险。在设计银行交易时一般采用"先借后贷"的顺序来实现,保证即使有其它交易访问了执行过程中的数据结果,也不会造成经济损失。例如借记卡取现业务,一定是先扣减借记卡帐户余额成功后,受理行才会将现金付给持卡人。反之,则可能造成借记卡透支的风险。

从放松原子性带来的影响来分析,本模型的补偿处理可以保证绝大部分事务的一致性。个别异常情况将采用系统自动核对各处理方交易日志,并根据核对结果自动调整的方式来再次补偿。根据应用的不同特点,可以设计不同周期的核对方式。对于特殊情况下的事务不完整情况,需要采用人工干预的方式来最后解决。在设计跨系统应用时,必须仔细考虑核对交易日志处理的方式以及人工处理的方式。

相比 Saga 模型,本模型更加灵活。在实际应用中, Saga 模型显得相对死板,如果所有的异常都必须采用回退的方式来处理,将导致很多不必要的浪费和麻烦。譬如对于银行的汇兑业务系统,由于对交易的实时性要求不如银行卡业务那么严格,银行卡的实时性一般在一分钟以内,而汇兑业务系统的实时性一般在 2 小时以内,但是发送方的操作比较复杂,如果对于一些可控异常,譬如短暂的系统忙碌或通讯异常,就直接全部回退,将造成之前的处理工作全部白费,显然并不是最佳的方式。而本模型采用了基于知识库的异常规则处理方式,可以避免 Saga 模型的这个缺点。

4 基于该模型设计的银行前置系统分布式事务处理框架

在设定好分布式事务处理模型后,我们设计了一个银行前置系统的分布式事务处理框架。该框架结构如图 1 所示。

4.1 数据文件区

数据文件区是由前置系统核心数据文件和应用数据文件组成的,一般依托数据库系统来实现。其中最重要的系统核心文件有:

(1) 交易描述文件:该文件描述了一个交易将由那些原子任务组成,并说明执行顺序。

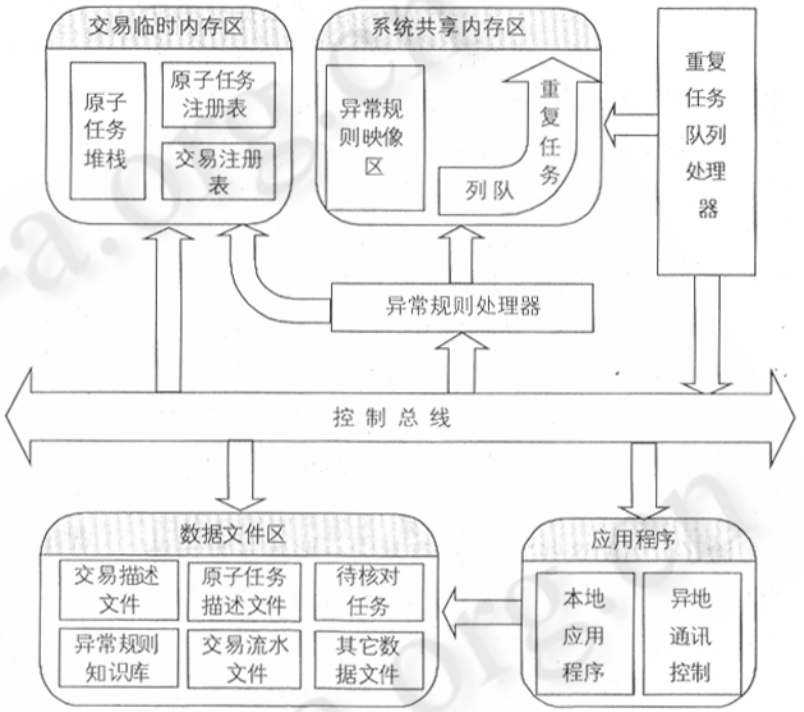

图 1

(2) 原子任务描述文件:该文件描述了各个原子任务的输入输出接口以及调度方式等信息。

(3) 异常规则知识库:该文件描述了一个交易流程中某个原子任务出现异常时,将采取什么样的处理策略。在前置机系统初始化时,可以将知识库内容读取到系统共享内存区的异常规则映像区,以便提高处理性能。

(4) 交易流水文件:该文件存放了所有交易流水信息。

4.2 交易临时内存区

交易临时内存区是总控程序为每个交易在前置系统执行时分配的内存空间,保存了一些交易自身使用的信息,并被总控程序用来控制交易的调度执行和记录交易的过程结果。最后这些信息将被总控程序转存

到交易流水文件，交易执行完毕后即释放。其中主要的数据结构有：

(1) 原子任务堆栈。由控制总线在接受到交易请求后，根据交易描述文件的内容登记到该堆栈内，主要用来控制交易流程中各个原子任务的提交顺序。

(2) 交易注册表。该注册表用来保存该交易的执行状态，以及关键信息，在交易结束前，由控制总线将里面的内容记录到交易流水文件中。

(3) 原子任务注册表。该注册表与交易注册表类似，用来保存每个原子事务的执行状态以及关键信息。

4.3 系统共享内存区

系统共享内存区是前置系统在初始化时分配的一个共享内存空间，用来记载一些公共信息，譬如流水号计数器、系统状态表等等，其中与本框架密切相关的是重复任务队列，该队列用来记录那些需要补偿的原子任务。之所以叫重复任务队列，是因为在该队列的每个节点都是一个补偿原子任务，原则上补偿任务必须执行成功，如果一次没有执行成功，那么可以在一定的调度算法控制下重复执行。

4.4 应用程序区

应用程序区是一系列前置系统的应用程序。既包括在前置系统本地执行的程序，也包括一些异地程序的调度控制方式、通讯适配器等等。

4.5 框架系统构建程序

框架系统构建程序是组成该框架的一些核心程序，这些程序相互配合，完成交易调度的全过程，保证了前置系统分布式事务的实现过程。其中最主要的是如下三个程序：

(1) 控制总线。控制总线是整个前置机系统框架的核心。负责接收客户端请求，根据交易描述文件和原子任务描述文件的配置来调度各个子事务的执行，监控子事务的执行结果，当发生异常时调用异常分析器处理异常，并保证交易最终执行完毕，负责交易日志的记载等处理。

(2) 异常规则处理器。该处理处理器负责根据接收的交易代码码和异常信息码，在异常规则知识库中匹配相应的处理策略，并依据处理策略修正交易注册表和原子任务堆栈。对于需要做回退处理的，将各回退子任务写入重复任务队列。

(3) 重复任务队列处理器。该处理器主要负责对重复任务队列中的原子任务进行调度提交。每个原子任务处理成功后就从队列中被删除，处理不成功的将在一定时间间隔后被重新提交。如果某个原子任务重复提交的次数达到预先设定的限值后还没有执行成功，那么该原子任务也将被删除，然后被记入到待核对文件中，通过应用系统的多方流水核对处理来完成结果修正。特殊情况下，需要通过人工处理来修正。

5 结束语

在充分分析银行前置系统应用特点和各种分布式事务处理方式的基础上，我们提出了这种适用于银行前置系统的分布式事务处理模型，并且在该模型的基础上设计了银行前置系统分布式事务处理的框架。该框架具有高效、灵活、稳定等特点，较好地解决了银行前置系统中分布式事务处理的难题。在实际应用时，需要设计好每个交易包含的子事务执行顺序，尽量减少因为放松了事务的原子性和隔离性所带来的不良影响。另外，为了充分发挥本框架的优越性，异常规则库的配置是关键所在。中国农业银行跨中心汇兑系统就是基于本框架开发的一个大型应用系统，该系统已经在全国范围内投产使用两年多，系统运行稳定，没有出现过需要人工处理的差错。

参考文献

1 H. Garcia - Molina、K. Salem, Sagas In Proc. 1987 SIGMOD International Conference on Management of Data. 1987.5,249 ~259.

2 G. . Alonso, Failure Handling In Large Scale Workflow Management Systems, IBM Research Report RJ9913. November, 1994.

3 齐勇、马莉、赵季中、齐向明、侯迪，分布式事务处理技术及其模型，计算机工程及应用，2001.9。

4 李俊平、李小平，分布式事务处理的一致性控制研究，武汉理工大学学报，2005 年第 5 期。

5 魏茂喜、贺贵明、吴元保，事务管理器的事务恢复处理，微型机与应用，2004 年第四期。

6 张小潘、雷毅、车帅，一种制造企业分布式事务处理系统的节点模型，计算机工程与应用，2002 年第 23 期。