分布式短事务系统中提交协议的研究

付艳艳 陈 驰 冯登国
(中国科学院软件研究所 北京 100190)

摘 要 分布式短事务系统具有事务简短、请求频繁的特点。现有的分布式提交协议多面向复杂的长事务应用场景,难以满足短事务处理的需求。针对分布式短事务系统的特点,提出一种新型的分布式事务提交方法。该方法在事务参与者之间建立互用日志,并通过询问协议进行交互,获得事务进展情况,从而减少了强制写日志次数和通信次数,优化了故障处理过程。与其他分布式提交协议和系统比较,本协议可以有效地提高事务效率,并具有更高的日志可用性。

关键词 分布式,短事务,提交协议,日志备份
中图法分类号 TP311.11 **文献标识码** A

Research on Commit Method in Distributed Short Transaction Systems

FU Yan-yan CHEN Chi FENG Deng-guo
(Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

Abstract Transactions in distributed short transaction system are usually short and asked frequently. Most of the existing distributed commit protocols are designed for the complex long transaction system, therefore they can not meet the needs of short transactions. In this paper, an improved schema for distributed short transaction systems was proposed. By backuping logs and querying among participants, we can efficiently reduce the number of log-written, communications during the transaction and optimizing the failure processing. Compared with other distributed commit protocols and systems, this protocol can effectively improve the efficiency of services, and has a very high log availability.

Keywords Distribute, Short transaction, Commit protocol, Log backup

1 引言

随着信息化的不断深入和用户数量的急剧增长,分布式数据库系统越来越多地进入银行、电信、工业控制、电子商务等行业。在这些行业的应用中,分布式短事务是最为典型的事务处理模式。分布式短事务具有以下两个特点:第一,事务请求相对简短,每个事务单元通常由单条命令构成;第二,事务的发生频率很高[11]。据调查,在典型的C2C电子商务应用中,每秒钟内到达分布式数据库系统的事务高达200个以上。应用场景的新特点为分布式系统事务处理方法带来了新的挑战,这主要体现在以下3个方面。

首先,分布式短事务的复杂度不高,在单个节点上执行所消耗的时间代价较低。但由于需要协调访问多个不同节点上的数据,通信代价成为影响分布式系统响应效率的主要因素之一。

其次,事务到达频繁的特点将导致系统日志的频繁写入,大量的磁盘IO使磁盘成为系统效能的瓶颈。以目前主流磁盘寻道时间8ms的标准计算,当事务到达频率超过125次/s时,将引起磁盘响应延迟,影响分布式系统的实时性和可用性。

此外,分布式事务需要访问位于多个不同物理节点上的数据,任何一个节点上事务的故障都会导致其他相关节点的事务回滚或重做。若多个事务间存在数据依赖,那么事务回滚将导致级联回滚的情况,这不仅严重降低了系统效率,还可能导致数据不一致的情况。

传统的两阶段提交协议面向长事务进行设计。由于长事务在单个节点上的执行时间较长,在协议设计中往往忽略通信的代价;由于事务发生频率较低,对日志写入的次数也较少进行优化。此外,对于站点失效和一般的网络延迟故障不予区分,短时的网络延迟故障也会导致事务的不必要回滚和重做,甚至是级联回滚,这些都会对分布式系统造成额外的效能损失。在分布式短事务应用场景下,如何提高事务处理系统的响应效率,已成为近期的研究热点。

本文针对分布式短事务的特点,提出了一种改进的事务提交协议,它通过减少日志强制写操作和通信的次数,有效提高了短事务的处理效率,同时避免了不必要的回滚,保证了数据的一致性。

2 相关工作

为了保证分布式事务的ACID属性,研究人员于1978年提出了基本两阶段提交协议[1,2]。基本两阶段提交协议分为投票阶段和决定阶段。在投票阶段,协调器节点向所有的参与者节点发送事务命令并询问是否可以提交事务。参与者节

到稿日期:2010-09-06 返修日期:2010-12-09 本文受863重点项目(2007AA120404,2007AA120405),中国科学院知识创新工程领域前沿项目(ISCAS2009-DR13),国家自然科学基金项目(61003228)资助。

付艳艳(1986—),女,博士生,主要研究方向为分布式事务处理、隐私保护,E-mail:fuyy@is.iscas.ac.cn;陈 驰(1978—),男,博士,助理研究员,主要研究方向为网络与系统安全;冯登国(1965—),男,教授,博士生导师,主要研究方向为网络与信息系统安全、密码理论与技术。

点根据自身情况回答 yes 或者 no。在决定阶段，协调者可以决定提交事务(当所有参与者都回答 yes)，或者夭折事务(当有参与者回答 no 或者超时未回复)。如果决定提交，则协调者发送 commit 消息给所有的参与者。如果决定夭折事务，则协调者发送 abort 消息给投票 yes 的参与者，协调者的这一决定称为最终决定(final decision)。其中，每一轮应答都需要协调器节点和参与者节点强制写入日志。

由于基本两阶段提交协议中日志强制操作的次数和信息交换的轮数较多，因此它不适用于对事务效率要求较高的场景。为了解决这个问题，研究人员提出了多种改进协议，包括假提交协议(Presume Commit Protocol，PrC)[5]、假夭折协议(Presume Abort Protocol，PrA)[8]、协同日志协议(Coordinator Log Protocol，CL)[4]、提前准备协议(Early Prepare Protocol，EP)、隐含同意协议(Implicit Yes Vote Protocol，IYV)等。PrC，PrA，EP，IYV 协议分别从减少日志操作数量和信息交换数量方面对事务处理进行了优化，但并不能完全解决分布式短事务系统的效率问题。CL 协议虽然尽量避免了事务执行过程中的强制日志操作，但日志的安全性很难得到保证。文献[6,7]对这些协议正常运行和故障处理的性能进行了比较。

考虑到磁盘 IO 对事务效率的影响[9]，文献[11]中提出了一种新的分布式并行数据库日志机制来提高事务效率。但系统出现故障，日志丢失的情况是少量的。因此，进行日志的大量备份是不必要的，也浪费了网络带宽和系统资源。文献[14]中采用了实时日志模式，同时利用非易失性高速存储设备的优势，尽可能降低事务运行时日志的代价，但它对基础设施具有一定的要求。

在分布式短事务系统中，由于事务请求频繁，数据更新迅速，很容易出现多个事务连续操作相同数据的情况，从而产生事务依赖。此时，如果在本地已提交的事务需要回滚，就必然影响到其他相关事务。因此，应尽量避免因单纯的网络故障导致的事务盲目回滚。文献[12]研究了网格环境下的事务模型、并发控制以及故障恢复等问题，充分讨论了分布式事务系统中网络环境易变的特点，分析了网络环境对事务处理的影响。文献[17-19]中针对目前并行数据库、移动数据库等的新特点分别提出了基于超时和一致性检查的新提交方案，它们对处理网络拥堵导致的事务故障具有一定的借鉴意义。

3 分布式短事务提交协议

3.1 系统模型

在分布式事务系统中，通常将发起事务的节点作为协调器节点，负责协调事务处理、决策等。事务的参与者节点持有本次事务操作的具体数据、资源等。如图 1 所示，系统中可能存在多个协调器节点；同一个协调器节点发起的不同事务也可能有不同的参与者。在分布式短事务系统中，一个事务单元一般可能由 1～2 个步骤构成。

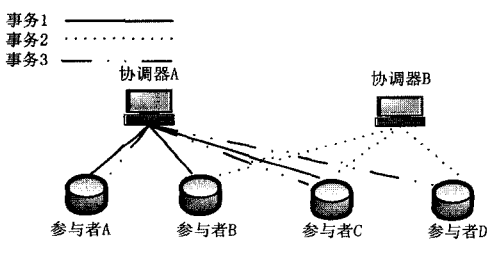

图 1 系统模型

3.2 原理

由 3.1 节的系统模型中可以看出，分布式短事务系统的事务处理以协调器节点为中心。各个参与者节点只与协调器节点通信，而参与者节点之间并无信息交换。但在分布式系统中，参与同一事务步骤的参与者间的操作和日志具有极大的相似性；各个参与者节点接收到的事务提交或者回滚消息也是相同的。如果利用参与者间的相似性，在参与者之间进行日志互用和事务进展情况查询，就能够很好地缓解日志丢失和网络故障带来的不良影响。

例如，在大型连锁超市中，协调器节点 A 要求参与者节点 A，C，D 返回本地库存中牙膏的数量。各参与者节点均在本地数据库中执行查询牙膏数量的操作，各参与者节点的日志也具有极大的相似性。通过协调器节点的资源映射，可以实现日志的互用。本文设计了一种新的日志结构，即在日志中记录本操作的所有参与者。这样，即使本地日志因故障或其他原因丢失，通过访问其他参与者节点和协调器节点，日志记录也是可恢复的。

在网络故障的情况下，可能有某些参与者节点没有及时接收到协调器节点发送的事务提交消息，从而导致事务的超时失败，进而产生不必要的回滚。而各个参与者节点接到的事务最终决定是相同的，因此可以在参与者节点之间发起询问，确定事务的进行状态，这样就可以大大减少网络故障对系统的不良影响。

3.3 日志标记

为了提高事务效率，避免写入日志带来的时间消耗，可将事务日志暂时存储在本地内存中。当事务日志积累到一定量，或者最旧的事务日志保存到了一定的时间，即可将所有日志批量写入静态存储。为了保证日志的可用性，实现不同参与者间的日志互用，需要在日志中严格区分不同的事务、同一事务的不同步骤，并分别记录每一步骤的参与者及事务命令。

如果本地事务日志遗失，本节点需要向其他所有节点查询相关的日志。遗失日志的事务提交时间必然晚于未遗失日志的事务的提交时间。因此，可以限定某一个时间点“timestamp-Δt”，以向其他所有参与者询问在此时间点之后的相关事务日志。其中 timestamp 为本地已提交的日志中最晚的时戳，Δt 为事务执行所需的最长时间。

根据以上需求，设计了一种新的日志结构。协调器节点日志结构为〈ctid，cid，Plist，op，i，timestamp〉，参与者节点的日志结构为〈tid，cid，ctid，Plist，op，vote，timestamp〉。其中：

cid 为协调器 ID，用来区分不同协调器发起的事务；

ctid 为协调器事务 ID，用来区分同一协调器发起的不同事务；

tid 为参与者节点本地事务 ID；

Plist 为参与者 list，用来确定本步操作的所有参与者；

TimeStamp 为协调器节点的当前时戳，用来确定日志查询的时间点；

OP 为事务单步请求内容，用来记录事务操作内容；

i 表明当前操作为事务的第几步请求，用来区别事务的多个步骤。如果 i 为 0，代表此步骤执行事务提交或夭折决定。

Vote 表明参与者投票结果，值为 yes 或 no。当 op 为 commit 或者 abort 时，vote 必须为 yes，表明必须执行协调器节点的最终决定；

所有在 Plist 中的参与者节点，均可利用此事务日志进行恢复。具体恢复方法在 3.6 节故障恢复准则中描述。

3.4 消息结构

为了帮助参与者节点确定日志中需要填写的各项，需重新定义参与者节点和协调器节点间交换消息的结构。

协调器节点发送到参与者节点的消息格式为〈ctid，cid，Plist，op，i，timestamp〉。协调器节点在每次发送的消息中附加此事务本地日志的最新时戳，以备故障时进行日志检索。参与者节点发送到协调器节点的消息结构为〈cid，ctid，Pid，i，vote，timestamp〉，其中 Pid 为本参与者 ID。参与者节点将每次事务执行结果以 vote 的形式反馈给协调器节点。

3.5 故障处理及恢复

3.5.1 正常状况下的事务处理流程

不考虑故障情况并且日志缓存未满时，事务的基本处理流程如图 2 所示。

1)协调器节点根据事务进行情况写日志缓存，并向参与者节点发送事务请求消息。

2)参与者根据事务请求完成情况写日志缓存，并向协调器节点发送反馈消息。

3)协调器节点接收参与者节点发送的反馈消息，分析投票结果。

如果所有参与者节点投票均为“yes”且事务尚未完成，则回到步骤 1)。否则进入步骤 4)。

4)协调器节点给出事务最终决定，写入日志缓存，并向参与者节点发送消息。

5)所有投票为“yes”的参与者节点执行最终决定，并写入日志缓存。

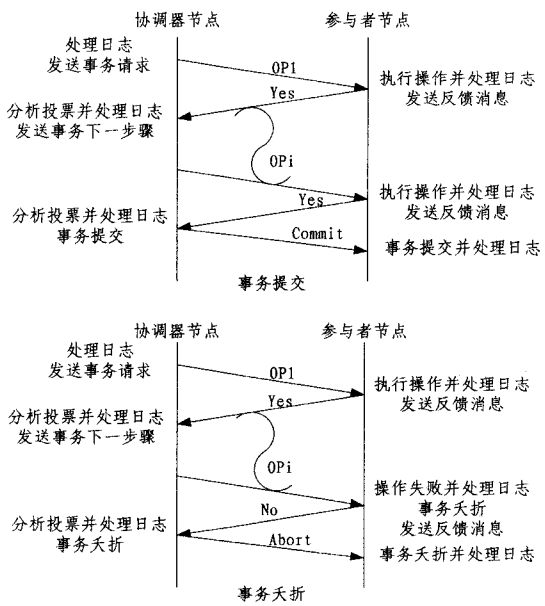

图 2 事务流程

3.5.2 事务通信故障处理

系统故障共可分为两类：通信故障和节点失效(site failure)。现有的分布式提交协议中，并未区分这两类故障。为了避免由于通信故障造成的事务盲目回滚或夭折，在本协议中，等待超时的节点可从其他参与者节点获取事务的确切进展情况，进而决定是否回滚、夭折或提交事务。

事务处理过程中出现超时时，参与者节点和协调器节点可分别按照以下步骤进行处理。

(1)本地事务超时未得到回复

参与者节点：

1)向协调器节点发起询问。询问的具体方式为将此事务最新的本地反馈消息重发。

2)参与者节点依据自己持有的本事务参与者 list 发起询问。询问的具体方式为将此事务最新的本地反馈消息发送给其他参与者。

3)如果接到 1)或 2)的反馈消息是事务的最终决定，则执行该决定。

4)如果接到 1)，2)的反馈消息是事务继续，则询问者须开始另一轮询问。

5)如果 1)、2)中发起的询问均未得到回应，则询问者回滚事务并通知其他参与者。

协调器节点：

超时未得到参与者节点回复，即认为该参与者节点投票结果为 no，按照 3.5.1 节中事务处理流程继续事务。

(2)接到来自参与者节点的询问

参与者节点：

查找本地与询问信息中相同事务的日志。如果已接到最终决定，则反馈最终决定；如果本地事务夭折，则反馈事务夭折。如果事务最新时戳比 timestamp 更新，则反馈事务仍继续；否则，不予回应。

协调器节点：

查找本地与询问信息中相同事务的日志，根据事务进行情况回复。

3.5.3 站点失效的故障恢复

参与者节点进行故障恢复时，需要依靠协调器节点进行日志分析、资源映射等。两类节点的故障恢复方法并不相同，下面分别进行说明。

1)协调器节点

在协调器节点重启之后，首先查找本地最后的事务结束日志的时戳 timestamp，然后询问所有的参与者节点在 timestamp$-\Delta t$ 之后的本协调器发起的事务记录。

各参与者节点返回指定时间之后的所有相关日志，协调器节点按照返回的日志里的协调器事务 ID 逐个进行处理。对于日志记录完整的事务，协调器节点将日志整理后写入本地日志即可。日志记录不完整的情况可分为 3 种：

①部分日志表明某事务已结束。如果协调器节点接到一个或以上参与者节点回复的事务结束消息(包括提交或回滚)，协调器节点通过日志解析模块整理接收到的日志，以恢复本地日志，并重新通知所有参与者该事务结束即可。

②日志表明某事务仍在等待。如果所有参与者节点均在等待回复事务，则此时协调器节点整理接收到的日志以恢复本地日志，并根据事务的执行情况决定事务提交或者回滚。

③如果有参与者尚未从参与者节点失效中恢复，不具有全部的日志，或者无法取得联系，则认为事务回滚，协调器节点整理接收到的日志以恢复本地日志，写入回滚日志并在与参与者节点恢复联系后向参与者节点发送决定。

在此过程中，如果有参与者节点被通知某事务回滚，但是

此参与者自身已经遗失了事务相关的记录，则需再次请求协调器节点事务的相关日志。协调器节点根据自身存储的日志进行解析，返回对应参与者节点所需的日志。

2)如果参与者节点失效，则在参与者节点重启之后，首先查找本地静态日志的最后时戳，然后询问协调器节点在此时间之后的事务情况。协调器节点对本地日志记录进行解析，返回对应参与者节点所需的日志即可。

4 日志可用性分析

通常情况下，事务日志不会马上写入静态存储器，而是在不同的参与者内存中存储一段时间。根据文献[3]知，设一台主机出现故障的概率为常数 λ，主机的平均无故障时间(MTTF)为 T。主机间出现故障的概率是相互独立的，假设主机数为 n，则在时间 0 到 t 内，所有主机均出现故障的概率为

$$p=(1-e^{-\lambda t})^n$$

$\lambda=1/T$，所以，上式等价于

$$p=(1-e^{(-\frac{t}{T})})^n$$

假设单台主机的无故障时间为 100h，事务缓存时间最长为 5min，则在此时间内所有主机均出现故障，日志无法获得的概率为

$$p=(0.00083298620754168 61)^n\approx(1\times10^{-3})^n$$

即使事务只有一个参与者节点，事务操作内容也会由参与者节点和协调器节点共同持有。也就是说，在最差情况下，日志丢失的概率约为 10^{-6}，可以满足日志可用性的要求。而且，如果在 5min 内日志缓存区满，就会将日志写入硬盘，日志保存在内存中的时间被缩短，遗失的概率更低。尤其对于协调器节点，日志增长迅速。通过合理控制日志缓存区的大小，可以有效提高日志的可用性。

5 性能分析

本协议与其他分布式提交协议相比，主要的优势体现在日志强制记录次数的减少、信息交换数量的减少和事务故障处理的优化。下面针对这 3 个方面分别进行分析。

5.1 日志强制记录次数

对于有 n 个参与者节点和 1 个协调器节点的分布式系统来说，按照本协议的设计方案，在事务进行中无需强制日志记录。而按照传统 2PC 的方案，在一次事务进行中需要 $2n$ 次强制日志记录。由此可以明显看出，本协议可以大大减少日志操作次数，提高了系统效率。

如果不考虑向内存中缓存日志所需的时间，假设磁盘寻道时间为 t，需要寻道的概率为 p，日志大小为 L，日志缓存区大小为 M，磁盘写入速度为 K，事务到达频率为 100 次/s，则传统 2pc 和本提交协议在 1s 钟内需花费在日志记录上的时间可具体表示为

$$T_{2pc}=(pt+L/K)*100$$

$$T=100*(L/K+ptL/M)$$

假设 p 约为 50%，磁盘寻道时间为 8ms，事务到达频率为 100 次/s，且仅包含一条 SQL 请求，参与者节点数量为 2，在这种情况下的日志丢失概率与可节省的时间均与 M/L 的大小相关。图 3、图 4 分别分析了 M/L 的大小与日志可用性和效率的关系。

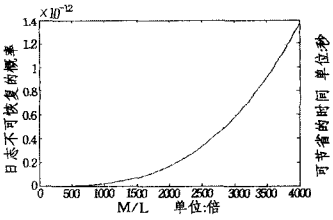

图 3 M/L 的大小与日志丢失概率的关系

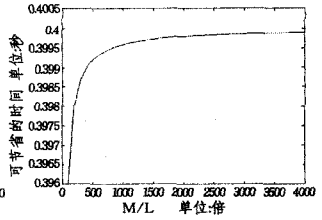

图 4 M/L 的大小与节省时间的关系

由图 3 可以看到，日志的可用性在 $M/L<4000$ 时都是极高的。当参与者的节点数量增加时，日志的可用性还会进一步提高。协议使用者完全可以只从节省系统时间和可用的系统资源分配的角度来考虑 M/L 的大小。

5.2 信息交换数量

在本协议中，协调器节点发出事务最终决定之后，即默认参与者节点会接收到该决定并执行，因此减少了一般协议中最后一轮的参与者回复信息。如果参与者节点没有收到最终决定，必将等待超时，参与者可通过发起超时询问来获得最终决定。

正常情况下，整个事务过程的信息交换的数量为 $2n\_op+n$，其中 n_op 是事务总共进行的操作数。例如，假设事务请求总共为 k 步，每次请求大约有 60% 的参与者节点参与此事务，则 $n\_op=k*60\%*n$。对于通常情况下的只包含一个请求的事务，$n\_op=n$，此时的信息交换数量为 $3n$。同传统 2PC 只在提交过程就需要 $4n$ 条信息的交换量相比仍具有一定优势。同其他分布处理事务协议相比，本协议也具有明显优势。表 1 给出各协议性能比较。

表 1 各协议性能比较

	信息数量 Commit/abort	强制写次数 Commit/abort	通信步骤 Commit/abort
2PC	4n	2n+1	4
PrA	4n/3n	2n+1/n	4/3
PrC	3n/4n	n+2/2n+1	3/4
EP	2N_op+n/2N_op+2n	n+2/2n+1	由事务步骤决定
CL	3n	1	3
IYV	2N_op+2n/2N_op+n	2n+1/2n	由事务步骤决定
本协议	2N_op+n	0	由事务步骤决定

5.3 故障处理的优化

如果事务执行过程中发生了图 5 所示的状况，则参与者节点与协调器节点的网络连接不畅，传统的提交协议方案无法避免事务的失败。最坏情况则是在本节点等待超时的时间内，其他节点已经分别接受了其他不同的事务并执行。如果此事务决定回滚，有可能会对其他各个节点已接受的事务产生影响，导致级联回滚的发生。而在本协议中，根据 3.5.2 节故障处理准则，参与者在等待超时后，可利用参与者名单在参与者之间发起询问。这对网络容易出现暂时性拥堵的情况，具有更好的容错性。

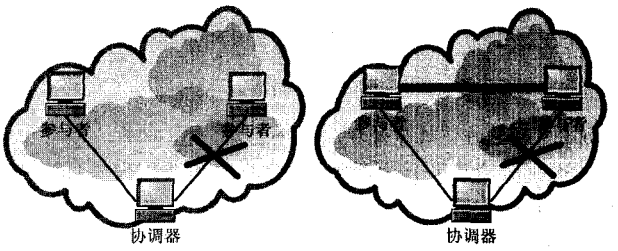

图 5 故障处理的优化

(下转第 41 页)

路由器，并接近于软件路由器的最佳性能。

参 考 文 献

[1] Bavier A, et al. In VINI veritas: realistic and controlled network experimentation[C]// Proceedings of SIGCOMM '06. 2006

[2] Bhatia S, et al. Trellis: A platform for building flexible, fast virtual networks on commodity hardware [C] // Proceedings of ROADS 2008/CoNEXT 2008. 2008

[3] Druschel P, Peterson L L. Fbufs: a high-bandwidth cross-domain transfer facility[C]// Proceedings of SOSP '93. 1993

[4] Click Modular Router[EB/OL]. http://read.cs.ucla.edu/click/

[5] Argyraki K, et al. Can software routers scale? [C]// Proceedings of PRESTO '08. 2008

[6] Egi N, et al. Towards high performance virtual routers on commodity hardware[C]// Proceedings of CONEXT '08. 2008

[7] Intel 64 and IA-32 architectures software developer's manual volume 2B: Instruction set reference, N-Z. Sep. 2009

[8] Soltesz S, et al. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors[C]// EuroSys '07. Mar. 2007

[9] Corbet J, Rubini A, Kroah-Hartman G. Linux Device Drivers (3rd Edition)[M]. O'Reilly Media, Inc., 2005

[10] OpenVZ[EB/OL]. http://www.openvz.org/

[11] Chinni S, Hiremane R. Virtual machine device queues[M]. Intel white paper, 2007

[12] Liao Y, Yin D, Gao L. PdP: Parallelizing data plane in virtual network substrate[C]// Proceedings of SIGCOMM VISA 2009 Workshop. Aug. 2009

(上接第 32 页)

6 进一步改进

6.1 日志的可用性

本协议通过不同参与者间互相备份日志提供了相当高的日志可用性。但是，系统的故障是不可避免的，尤其是协调器节点出现故障时，会对整个系统造成较大影响。考虑到这种情况，可以参考文献[10]中的方案，为协调器节点单独建立一个容错系统，这样日志的可用性可以得到进一步提升。

6.2 拜占庭失效

本协议假设系统中的节点故障方式为故障-停止模式，并没有考虑到节点出现拜占庭故障的情形。尤其是在参与者节点间发起询问时，事务执行的效率很大程度上取决于参与者节点的回复。如果参与者节点出现拜占庭失效的情况，可能会造成事务的失败，甚至导致数据的不一致性。为了避免这种情形发生，可以对节点进行拜占庭故障检测，根据检测结果，决定参与者节点的回复信息是否可用。文献[13,15]对此进行了讨论。但是，分布式短事务系统对效率的要求较高，应尽量控制信息交换的轮数。

结束语 本算法在保持原有系统架构不变的情况下，通过参与者列表，实现了参与者间日志的互相备份，同其他协议相比，提高了事务提交效率，并提高了在网络暂时故障的情况下事务成功提交的可能性。在本文性能分析部分，将本算法与传统两阶段提交协议进行了对比分析，表明本算法在节省日志操作时间、减少信息交换方面具有更优的性能。同时，本协议能够保证较高的日志可用性，通过调整日志暂存区的大小和日志暂存时间还可以将日志可用性进一步提高。在分布式短事务系统的实际应用中，本协议具有更大的优越性。

参 考 文 献

[1] Gray J. Notes on Data Base Operating Systems[M]. Operating Systems. Heidelberg: Springer, 1978: 393-481

[2] Lampson B. Atomic Transactions[M]. Distributed Systems: Architecture and Implementation. Heidelberg: Springer, 1981: 246-265

[3] Martin L. Software Engineering: Design, Reliability and Management [M]. New York: McGraw-Hill Company, 1983: 571-574

[4] Stamos J, Cristian F. A low-cost atomic commit protocol[C]// The 9th IEEE Symp on Reliable Distributed Systems. Huntsville, AL, 1990

[5] Al-Houmaily Y J, Chrysanthis P K, Levitan S P. An argument in favor of the presumed commit protocol[C]// The 13th Int'l Conf on Data Engineering. Birmingham U K, 1997

[6] Liu M L, Abbadi A D. The Performance of Two-phase Commit Protocols in the Presence of Site Failures Distributed and Parallel Databases [M]. Hingham: Kluwer Academic Publishers, 1998: 157-182

[7] Abdallah M, Guerraoui R. One-phase commit: Does it make sense[C] // International Conference on Parallel and Distributed Systems. CA: IEEE Computer Society, 1998: 182-192

[8] Attaluri G K, Salem K. The presumed-either two-phase commit protocol[J]. IEEE Trans on Knowledge and Data Gineering, 2002(10): 1190-1196

[9] 刘云生，覃飙. 分布式实时事务提交协议[J]. 计算机研究与发展，2002，39(7)：827-832

[10] 李宏亮，金士尧，等. 短事务、强实时双机容错系统的研究[J]. 计算机学报，2003，26(2)：244-249

[11] 邱元杰，刘心松，杨峰. 一种高效的分布式并行数据库日志机制[J]. 计算机研究与发展，2004，141(11)：1942-1948

[12] 李陶深，李卫玲，马新娟. 网格环境下的事务提交机制的研究[J]. 计算机研究与发展，2006，43(suppl)：117-122

[13] Gray J. Consensus on transaction commit[J]. ACM Transactions on Database Systems, 2006, 31(1): 133-160

[14] 肖迎元，刘云生，等. 一种实时动态数据库故障恢复策略[J]. 软件学报，2007，18(10)：2516-2527

[15] Zhao Wen-bo. A byzantine fault tolerant distributed commit protocol[C] // Dependable, Autonomic and Secure Computing. USA: IEEE, 2007: 37-46

[16] Chen Jian-ying, Liu Xin-song, Li Tao, et al. Synchronized update over distributed and parallel database, Cyber-Enabled Distributed Computing and Knowledge Discovery[C] // CyberC '09. 2009

[17] Boso B, Sane S S. DTCOT-Distributed Timeout-based Transaction Commit Protocol for Mobile Database Systems[C]// International Conference and Workshop on Emerging Trends in Technology (ICWET 2010). Mumbai, India, 2010

[18] Agrawal S, Shanker U, Singh A N, et al. SPEEDITY-A Real Time Commit Protocol[J]. International Journal of Computer Applications, 2010(1-3): 74-81