文章编号:1000-9957(2005)04-0335-05

改进的两阶段提交协议

于 红, 高艳萍, 郭连喜
(大连水产学院 信息工程学院,辽宁大连 116023)

摘要:提出了一种改进的两阶段提交协议,该协议通过增加本地写LOG的次数来减少信息交换量和阻塞次数,增加成功提交的次数和事务的成功率,提高了提交协议的性能。并对该协议中算法的正确性和性能进行了分析。

关键词:事务处理;两阶段提交协议;故障恢复;信息交换
中图分类号:TP311.13 **文献标识码**:A

分布式数据库的事务处理是分布式异构数据库技术研究的热点问题,研究主要包括两方面的内容:分布式事务的并发控制[1-3]和分布式事务的提交协议[4-8]。本文中笔者主要研究分布式事务的提交协议。一个性能良好的分布式提交协议,必须尽量减少信息交换量、阻塞的次数、系统重启的次数、写LOG的次数和不必要的夭折,增加成功提交的次数。信息交换涉及两个以上的站点,两个以上站点之间需要通过网络连接,故操作费用相对较高;系统重启和写LOG只涉及一个站点,而单个站点的操作费用相对较低。因此,笔者提出了一种改进的两阶段提交协议,该协议以增加本地写LOG次数来减少信息交换量和阻塞的次数,并增加成功提交的次数,从而提高系统效率。

1 相关工作分析

目前,有很多关于提交协议的研究,如两阶段提交协议(2PC)、假提交协议(PC2PC)、假夭折协议(PA2PC)、三阶段提交协议(3PC),假两者协议(PE2PC)[7]等。基本的两阶段提交协议中,协调者需要向每一个参与者发送“VOTE”命令,参与者必须向协调者发送投票决定“YES”或“NO”,然后,协调者根据投票结果向参与者发送事务处理的最后决定“COMMIT”或“ABORT”,而参与者必须向协调者返回确认信息“ACK”。如果一个事务涉及n个节点,则信息交换量为4(n-1)次,而且,协调者必须等待参与者最终确认信息。若某个参与者在接到协调者的提交或夭折命令后还没来得及发出确认消息时出现故障,则会导致协调者阻塞。

为了提高基本的两阶段提交的性能,人们做了大量的研究,提出了2PC的各种不同版本[4-7]以及3PC的各种不同版本[8],文献[4]中介绍了PA2PC和PC2PC。PA2PC在事务COMMIT时与基本2PC相同,信息交换量为4(n-1)次,在事务夭折的情况下,由投票为“NO”的参与者独自决定ABORT子事务,信息交换量小于3(n-1)次;PC2PC在事务ABORT时信息交换量为4(n-1),在事务COMMIT时,信息交换量小于3(n-1)次。这两种协议都在一定程度上减少了信息交换量,但仍然存在阻塞的情况,且不能增加成功提交的次数。文献[5]中提出的提交协议,强调全局可串行化,放松了对原子性和隔离性的要求,不能满足事务的ACID特性。文献[6]中介绍了一种两阶段提交协议,该协议提出了保存一个故障站点列表以避免无为的事务提交,减少了夭折的次数;同时提出在协调者站点出现故障时,通过发送“YOU”消息再次选举一个新的协调者,以避免因参与者主机不能接受到协调者的决定而阻塞的情况。但文献[6]没有阐述选举的原则以及在选票比较分散情况下的处理办法。文献[7]中提出的PE2PC协议,在事务树生成过程中尽量将参加者的相应记录

收稿日期:2004-11-15
作者简介:于红(1968-),女,教授。E-mail:yuhong@dlfu.edu.cn

分段写入LOG中。如果在COMMIT启动前已经将所有的参与者信息写入LOG中，就采用PC2PC协议，若不能及时写入，则采用PA2PC协议。PE2PC协议具有PA2PC和PC2PC的双重优点，但由于PA2PC和PC2PC都会出现阻塞现象，而且都没考虑增加成功提交的次数，故PE2PC也不太理想。文献[8]中提出了一种三阶段提交协议，该协议基于CORBA体系结构，能避免产生阻塞现象，但因增加了一个阶段，就增加了交换信息量和系统响应时间，所以，系统效率会相应降低。

2 改进的两阶段提交协议

2.1 算法基本思想

笔者提出的改进的两阶段提交协议的基本思想是：协调者在发出投票决定之前先检查目标站点的状态，若目标站点正常，则发送投票决定，若站点有故障，先尽量恢复，如果不能恢复，就终止该事务；在发送投票决定之后，启动计时器计时，若在指定的时间内收到所有参与者的投票结果，则根据投票结果做下一步决定。若所有参与者的投票结果均为“YES”，则向所有参与者发送“COMMIT”，否则，向投票结果为“YES”的参与者发送“ABORT”，并将操作的每一步写入LOG。参与者在接收到协调者的“VOTE”命令后，检查本地资源情况，申请进行事务提交所需资源，如果可以满足，则发回“YES”，否则，参与者根据本地资源情况和事务的时间限制进行反复申请。期间，如果申请到了必要的资源，则向协调者发回“YES”，并启动计时器等待协调者的最后决定；若在指定的时间内，收到协调者的最后决定，则按照协调者的命令进行最终的事务处理，否则，向其他参与者询问协调者的最终决定；若有参与者回答“COMMIT”，则提交该事务，否则，ABORT该事务。同样，参与者也要将操作的每一步都记入本地的LOG中，以便在出现故障时进行恢复。由于笔者提出的提交协议中用超时检测技术来检测故障，故不会出现阻塞现象，参与者在本地多次进行RETRY，可以减少由于临时问题而导致的事务夭折，减少不必要的信息交换。

2.2 算法描述

2.2.1 协调者算法

```
CS1：检查目标站点的状态
      IF 某些目标站点处于故障状态 THEN
        IF 站点故障 THEN
            RETRY
        ELSE
            ABORT
        END IF
        IF RETRY FAILED THEN
            ABORT
        END IF
      END IF
    //投票阶段
CS2：在LOG中写入“Trans Begin”
      SENT “VOTE” to all Cohorts
      开始计时
      DO UNTIL received response from all Cohorts
        IF one response is “NO” THEN GOTO CS4
          ON time - out GOTO CS4
        WAIT
      END DO
    //决策阶段
        IF All response is “YES” THEN
CS3：在LOG中写入“Trans COMMIT”
          SENT “COMMIT” to all Cohorts
        ELSE
CS4：在LOG中写入“Trans ABORT”
          SENT “ABORT” to the Cohorts whose vote result is “YES”
        END IF
```

2.2.2 参与者算法

```
    //等待来自协调者的 VOTE 消息
PS1:DO UNTIL received "VOTE"from the coordinator
      WAIT
    END DO
    //根据本地的资源情况确定 VOTE 结果
    IF 本地资源能够满足事务要求 THEN
PS2:向协调者发送"YES"
      开始计时
      DO UNTIL received final decision from the coordinator
        WAIT
        ON time - out GOTO PS22
      END DO
      IF COMMIT THEN
    PS21:在本地 LOG 中写入"COMMIT"
          COMMIT 本地事务
      ELSE
    PS22:IF time - out THEN
          向其他参与者发送"INQ"消息,询问协调者的最终决定
          启动计时器
            IF 在时限内收到"COMMIT" THEN
              在本地 LOG 中写入"COMMIT"
              COMMIT 本地事务
            ELSE
              在本地 LOG 中写入"ABORT"
              ABORT 本地事务
            END IF
          ELSE
            在本地 LOG 中写入"ABORT"
            ABORT 本地事务
          END IF
      END IF
    ELSE
    //本地资源不能满足事务需求
PS3:根据本地资源情况以及"RETRY"次数决定是否重新申请
        IF 可以重新申请 THEN
          重新申请本地资源
          GOTO PS2
        ELSE
          向协调者发送"NO"
          在本地 LOG 中写入"ABORT"
          ABORT 本地事务
        END IF
    END IF
```

2.2.3 故障恢复算法　分布式数据库的故障主要有以下几大类:

1)机器故障(站点故障)　这种故障可以通过重启后恢复来解决,在本文中笔者主要考虑的是这种故障。

2)报文丢失　可采用面向连接的协议来解决这一问题,不需要在算法中考虑。

3)本地实例故障　是由本地资源暂时不能满足要求或者临时异常情况引起的,这种故障可以通过 RETRY 来解决,这在上述算法中已经体现。

4)网络故障　分两种情况,一是出现网络分割,另一种是未出现网络分割。对于前者,可通过超时检测采用重启后恢复来处理,对于后者,可通过选用合适的路由和算法来保证站点间的连接。

本文中笔者主要讨论对站点故障进行重启的恢复处理算法。无论对于参与者还是协调者,在不同的位置出现故障的处理方法是不同的,图 1 给出了可能出现故障的位置示意图。

协调者在任何一个位置出现故障,都会因超时而独立 ABORT 本地子事务,而协调者在重启后,查找事务列表,找到未终止的事务,便根据 LOG 的内容来 ABORT 该事务,所以,整个分布式事务都处于一

个一致的 ABORT 状态。

若参与者在位置 1 或位置 2 发生故障时,协调者会因有参与者超时而做出 ABORT 的决定,并向所有准备提交的参与者发出"ABORT",这样,不能提交的参与者已经独立 ABORT 子事务。发生故障的参与者在恢复时,会因自己已经在最后决定做出之前出现故障而判断出全局事务已经 ABORT,故进行本地 ABORT,这样,全局事务都处于 ABORT 状态。

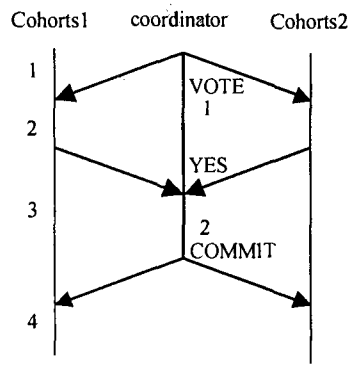

图 1 故障位置示意图

Fig. 1 Instance of failure position

若参与者在位置 3 发生故障时,如果该参与者做出的投票结果为"NO",它已经 ABORT 本地子事务,协调者也会因为投票结果不全是"YES"而决定 ABORT 全局事务,整个事务的结果依然是一致的;如果该参与者做出的投票结果为"YES",则全局的结果可能是"COMMIT"或"ABORT",这时,它不能独自做出"COMMIT"还是"ABORT"的恢复决定,必须向其它参与者或协调者询问最终的决定,根据获取的信息做最后的恢复决定,恢复的结果一定会与全局结果一致。

若参与者在位置 4 发生故障时,由于它已经收到协调者的最后决定并把该决定写在本地 LOG 中,所以,恢复后可以通过查本地 LOG 独立进行恢复,恢复的结果也一定与全局事务的结果一致。

3 算法的正确性和性能分析

3.1 算法的正确性

提交协议的正确性由该提交协议所处理的事务的正确性来衡量。若事务是正确的,则说明相应的提交协议是正确的,而只要事务满足 ACID 特性,则说明事务是正确的。

3.1.1 *原子性* 由于协调者的协调作用,整个分布式事务要么全部提交,要么全部夭折,也就是要么全做,要么全不做,即满足原子性。

3.1.2 *一致性* 任何一个参与者执行 ABORT 命令,均会在自身夭折后向协调者发送"ABORT"消息,协调者会因收到参与者发来的"ABORT"消息而决定夭折整个事务;相反,任何一个"COMMIT"命令均是在协调者收到所有的参与者发来"COMMIT"命令后才发出的,所以,所有的参与者均提交事务,整个分布式事务处于一致的提交状态,事务的执行结果是使数据库从一个一致的状态到达另一个一致的状态,故该提交协议满足一致性。

3.1.3 *隔离性* 任何一个事务的完全完成是以协调者发出"COMMIT"命令后每一个参与者完成"COMMIT"动作为标志,必须在一个事务处理完之后才处理下一个事务,故事务之间是隔离的,满足隔离性。

3.1.4 *永久性* 任何提交操作都会在每一个节点的数据库中进行物理操作,而这些物理操作对数据库的影响是永久的,故具有永久性。

3.2 性能分析

1)通常,失败的操作中 60% 以上是因本地资源不足引起的,在参与者提交事务过程中,当系统资源不足时,通过增加本地 RETRY 操作,可避免不必要的 ABORT,增加成功提交的次数。

2)系统中若没有时限,当一个站点出现故障后,其它站点就陷入了无限的等待中,从而引起阻塞现象,影响整个系统的性能。引入超时检测技术后,通过时间来控制进程之间的同步,避免了阻塞现象,并将系统效率提高 30%。

3)在影响系统效率的因素中,写 LOG 是本地操作,只涉及一个站点,而信息交换是网络操作,涉及多个站点,二者相比,本地操作的费用要比网络操作的费用低一个数量级。从算法中可以看出,参与者本地写 LOG 的次数最多为 3 次,协调者写 LOG 的次数最多为 2 次,即分别增加 1 次和 2 次,所以,通过增加本地写 LOG 次数,来减少消息量,提高系统效率,实现了本地独立恢复。

4 结束语

影响提交协议的因素很多,在一个提交协议中不可能使所有的指标都达到最优,在研究提交协议时必须有所取舍,优先考虑对性能影响较大的指标,本文中笔者提出的提交协议就是以牺牲一些次要指标优化另一些重要指标来提高系统的整体效率。

本协议的不足之处,是没考虑在参与者发出的投票决定为"YES"后,等待协调者的最后决策时出现网络分割,即当该参与者站点成为孤立站点时,全局事务的一致性问题。下一步的工作是在此基础上如何进一步完善该算法,以弥补这些不足。

参考文献:

[1] INDRAJIT R, MANCINI L V, JAJODIA S, et al. ASEP: A secure and flexible commit protocol for MLS distributed database systems[J]. IEEE Transaction on Knowledge and Data Engineering, 2000,12(6):880-899.

[2] THOMASIAN A. Distributed optimistic concurrency control method for high-performance transaction processing[J]. IEEE Transaction on Knowledge and Data Engineering, 1998, 10(1):173-189.

[3] ZHANG A, NODINE M, BHARGAVA B. Global scheduling for flexible transactions in heterogeneous distributed database systems[J]. IEEE Transaction on Knowledge and Data Engineering, 2001,13(3):439-450.

[4] MOHAN C, LINDSAY B, OBERMARCK R. Transaction management in the R* distributed management system[J]. ACM Transaction On Database System, 1986,11(4):378-396.

[5] CHUNG S M, PYEONG S M. Multidatabase transaction management scheme supporting multiple subtransactions of a global transaction at a site[J]. Informatics and Computer Science, 1997, (3~4):241-266.

[6] BYUN T Y, MOON S C. Nonblocking two-phase COMMIT protocol to avoid unnecessary transaction ABORT for distributed systems[J]. Journal of Systems Architecture, 1997,43(1~5):245-254.

[7] ATTALURI G K, SALEM K. The presumed-either two-phase commit protocol[J]. IEEE Transaction on Knowledge and Data Engineering, 2002, 14(5):119-1196.

[8] KOMMAREDDY M, WONG J. Non-blocking distributed transaction processing system[J]. The Journal of Systems and Software, 2000,54: 65-67.

Improved two-phase commit protocol

YU Hong, GAO Yan-ping, GUO Lian-xi

(School of Information Engineering, Dalian Fisheries Univ., Dalian 116023, China)

Abstract: To improve the performance of the commit protocol, the information exchange, the times of blocking, the times of the system restart and the times of log writing must be decreased and the unnecessary abort must be avoided. Because the system restart and log writing affects local sites, while information exchange and blocking concerns multiple sites, the latter has much more effect on the performance of the system than the former. In this paper, an improved two-phase commit protocol is presented. This protocol can decrease the times of information exchange and the time of blocking, and increase the times of successful commit, though the time of log writing will increase. In the last section of this paper, the performance of the protocol is analyzed.

Key words: transaction processing; two-phase commit protocol; failure recovery; information exchange