基于快照隔离的分布式数据库同步协议研究与实现

王 珏[1]，李立新[1]，张绍月[2]，杨梦梦[1]，付建丹[1]
(1. 解放军信息工程大学 电子技术学院，郑州 450004；2. 中国人民解放军 77626 部队，拉萨 850007)

摘 要：针对分布式数据库系统中副本一致性的问题，结合快照隔离的性能优势和组通信技术的消息定序特性，提出了一种满足单副本可串行化的数据同步协议。首先，形式化定义了快照隔离可串行化的准则，并证明了该规则可以保证单副本可串行化。进而基于组通信系统的消息定序特性，提出了满足单副本可串行化的数据同步协议 SSI-REP。实验表明，与两阶段协议(2PL)相比，SSI-REP 协议提高了系统的性能，降低了事务的系统响应时间；与全局快照隔离算法 GSI 相比，SSI-REP 协议在保证单副本可串行化的前提下，对系统性能的影响甚微。

关键词：数据库同步；快照隔离；单副本可串行化；组通信；SSI-REP
中图分类号：TP311.1 **文献标志码**：A **文章编号**：1001-3695(2012)08-3012-06
doi:10.3969/j.issn.1001-3695.2012.08.054

Research and implementation of distributed database synchronization protocol based on snapshot isolation

WANG Jue[1], LI Li-xin[1], ZHANG Shao-yue[2], YANG Meng-meng[1], FU Jian-dan[1]
(1. *Institute of Electronic Technology, PLA Information Engineering University, Zhengzhou* 450004, *China*; 2. *Unit* 77626 *of the PLA, Lhasa* 850007, *China*)

Abstract: To solve the problem of data consistency in distributed database system, this paper presented a distributed database synchronization protocol providing one-copy-serializability based on the performance of SI and the message ordering feature of group communication. At first, this paper formally defined serializable snapshot isolation (SSI) and then proved the correctness of the algorithm. Then it proposed SSI-REP based on SSI and the feature of group communication. Experimental results show that SSI-REP has clearly better performance than 2PL and nearly the same performance comparing to GSI.
Key words: database synchronization; snapshot isolation(SI); one-copy-serializability; group communication; SSI-REP

0 引言

基于分布式数据库的应用系统在目前人们的生活工作中得到了广泛应用，像银行管理系统、铁路售票系统等。为了提高系统的可靠性和可用性[1]，通常采用数据库同步的方法，即在应用系统中引入多个数据副本，一方面系统可以在本地副本节点操作，减少通信代价，提高系统性能；另一方面系统可以从一个出现故障的副本节点切换到另一个正常运行的节点，不会因为某个节点的故障导致系统的瘫痪，提高了系统的容错能力。如何保证这些数据副本的一致性是数据库同步领域的热门问题。

许多数据库同步的解决方案由于采用两阶段锁(2PL)机制来达到单副本可串行化(one-copy-serializability)，导致效率低、响应慢，降低了系统的可靠性。近期国内外学者在数据库同步技术的研究中，由于系统在性能上有了很大的提高，快照隔离的同步方式成为当前研究的主流和热点。

Lin 等人[2]提出了一种“1-copy-si”的数据库同步模型，在每个数据副本提供快照隔离的同时达到全局的快照隔离，即所有的快照从全局看效果等同于在一个副本中执行。

Elnikety 等人[3]提出了一种 GSI 算法，将单个副本的快照隔离扩展到多副本的数据库同步系统中，同时保证了快照隔离的特性。

然而目前基于快照隔离的数据库同步方案中主要关注的是将快照隔离级别扩展到全局，从而达到全局快照隔离，并没有达到单副本可串行化的标准，快照隔离所产生的异常现象仍然没有消解。为此，本文提出了一种单副本可串行化的数据库同步协议，实现分布式数据库同步下的快照隔离可串行化。

1 相关研究

1.1 快照隔离(SI)

快照隔离[4]是 Berenson 等人在 1995 年提出的一个应用在单个节点数据库系统中的多版本并发控制协议，可以达到不阻塞读操作并且避免了一些操作异常。快照隔离定义了两个特

收稿日期：2012-01-01；**修回日期**：2012-02-05
作者简介：王珏(1987-)，男，江苏徐州人，硕士，主要研究方向为网络应用(wangjue_js@126.com)；李立新(1967-)，男，湖南澧县人，研究员，主要研究方向为网络应用、数据库等；张绍月(1973-)，男，四川乐山人，硕士研究生，主要研究方向为网络应用；杨梦梦(1980-)，女，安徽舒城人，博士研究生，主要研究方向为网络技术、信息安全等；付建丹(1986-)，男，湖北丹江口人，硕士，主要研究方向为网络技术、信息安全.

性:快照读(snapshot-read)和快照写(snapshot-write)特性。其中快照读特性指事务 T 读取的快照中包含所有在事务 T 开始之前更新成功的数据(包括 T 自己的更新);快照写特性指更新同一个数据元素的两个并发事务中,只有一个能够成功提交。快照写特性又称为先提交胜原则(first committer wins, FCW),在单个节点的数据库系统中,快照写特性避免了丢失写(lost update)异常的出现。但是由于快照隔离没有实现可串行化,所以快照隔离存在如写异常等缺陷。

例 1　写异常(write skew)

快照隔离可能会造成事务的不可串行化(non-serializable)进而导致数据的不完整性[4,5]。

一个银行客户有一个支票账户 X 和一个储蓄账户 Y,当从支票账户中取款的金额超过了该账户的余额时就会自动从客户的储蓄账户转账过来,约束条件为 $X+Y>=0$。初始化 $X=50$,$Y=50$,假设事务 T_1 和 T_2 操作和顺序如下:

T_1:　$r1(x,50)r1(y,50)$　$w1(x,-20)c1$

T_2:　$r2(x,50)r2(y,50)$　$w2(y,-40)c2$

由例 1 可知,T_1 和 T_2 为并发事务。当 T_1 提交时,计算 $x+y=-20+50=30$;当 T_2 提交时,计算 $x+y=50+(-40)=10$。因此快照隔离条件下,两个事务都符合完整性约束,可以正常提交,然而提交 T_1 和 T_2 后的结果是 $x+y=(-20)+(-40)=-60$,违反了完整性约束,这种现象称为写异常现象。

目前,已经有很多研究工作对快照隔离进行了研究。文献[4]对快照隔离技术进行研究并提出了快照隔离会导致不可串行化的执行结果。文献[6]在文献[7,8]的基础上提出了快照隔离中,任何不可串行化的执行所对应的串行化的图中的环包含两个连续的 *rw* 边,且这两个边对应的事务为并发事务。但多数方法都是基于修改数据库代码来实现的,在商业数据库中很难实现。本文基于 GSI 的思想,增加了对并发事务读冲突的判断,并在分布式环境下采用组通信技术[9,10]的全序广播机制将事务的写操作集合广播到其他副本节点中,在本地节点进行冲突检测,达到单副本可串行化的标准。

1.2　组通信系统(group communication system,GCS)

组通信系统是指将网络环境中的若干个副本节点构成一个组,组内的节点之间的相互通信关系对等,组内的某个成员节点向组内所有成员节点发送/广播信息。

在数据传输中,主要采用了组通信系统中提供的消息定序和可靠交付的特性[11,12]。

其中,消息定序包含先进先出(FIFO)定序、因果定序(causal)和全序服务(total order)。全序服务的特性指在组通信系统中的所有节点上,信息按照相同的次序被交付。也就是说,系统中节点 N_1 和 N_2,接收信息 m 和 m',在 N_1 和 N_2 中,或者先交付 m 后交付 m';或者先交付 m',后交付 m。信息的交付次序在组通信系统中的所有成员节点上是完全一致的。

可靠交付(uniform-reliable delivery)指如果组内广播出一条信息,则该组的所有或者全部正确地收到该消息,或者全部未收到。不容许出现组内有些成员收到而有些未收到的情况,即 all-or-nothing 特性。

由于组通信的全序服务和可靠交付的特性可以保证组内的副本节点接收到消息的顺序一致,国内外很多研究将其特性应用在数据库同步系统中[2,12~16]。本文基于这两个特性,在保证所有副本节点接收到写操作集合顺序一致的前提下,完成快照隔离的可串行化判定规则,实现所有数据副本的一致性。

2　快照隔离的可串行化算法

2.1　基本思想

基于快照隔离的可串行化算法(serializable snapshot isolation,SSI)的实质是在事务开始执行的同时,依据可串行化规则,将其与快照隔离机制相结合,通过所设计的算法机制检测当前事务是否存在不可串行化的操作,使得快照隔离应用在数据库同步系统的同时最终达到单副本可串行化,消除快照隔离的异常。

算法的关键点在于将快照隔离的读写关系以及读写冲突定义出来,并通过三个规则来避免事务运行中可能出现的读写和写写冲突。

2.2　数据模型定义

2.2.1　事务

快照隔离中的事务 T_i 开始对数据项进行读/写操作时,会产生一个开始的时间戳标记,记做 snapshot(T_i),也称为 s_i。下标 i 是用来区分事务的标记。$r_i(x)$ 表示事务 T_i 对数据 x 的读操作,$w_i(x)$ 表示事务 T_i 对数据 x 的写操作。事务提交或结束时分别用 c_i(commit(T_i))和 a_i(abort(T_i))表示。在 Oracle、SQL Server 以及 PostgreSQL 等数据库系统中,事务 T_i 的快照是在第一个读/写操作执行之前创建的。

2.2.2　数据版本

由于快照隔离是多版本的并发控制策略,所以数据库系统(DBMS)需要维护数据项的多个版本。为了更容易理解事务在快照隔离模式下的执行,需要相应地表示数据项的多个版本。对于每一个数据项 x,使用 x_i,x_j…来表示 x 的不同版本,下标用来说明该版本所对应的事务。因此每个写操作可以定义为 $w_i(x_i)$,数据版本号的下标与事务的下标相同。读操作 $r_i(x_j)$ 表示事务 T_i 读取的 x 是 T_i 之前 T_j 更新的版本。

2.2.3　事务历史

定义 1　历史(history)

一个定义在事务集合 $T=\{T_1,\cdots,T_n\}$,T 的执行历史 H 中定义了一种偏序,称为时间优先顺序 $<_H$,历史 H 中的事务集合 T 有以下特性:

a) $H=\cup_{i=1}^{n}T_i$

b) $<_H \supseteq \bigcup_{i=1}^{n} <_i$

c) $W_i(X_i)\in H, R_j(X_i)\in H \Rightarrow W_i(X_i)<_H R_j(X_i)$

d) 任意两个属于 H 的已提交的事务 T_i 和 T_j,或者 $C_i<_H S_j$ 或者 $S_j<_H C_i$。

属性 a)表明历史 H 包含事务集合 T 中所有的事务操作,属性 b)表明历史 H 保持每个事务中操作的顺序。属性 d)表明历史 H 中已经提交的事务是串行的,不存在操作的交叉重叠。

2.2.4　可串行化图(serialization graph)

本文在并发控制中使用可串行化图 SG(H)来表示历史 H 中事务之间的关系,SG 中的节点代表每个事务,图中的边→表示两个事务之间的关系。如表 1 所示,SG(H)中有以下几种关系的边:

a) ww 边：$T_i \xrightarrow{ww} T_j$，表示存在一个数据项 X，事务 T_i 更新生成版本 X_i，事务 T_j 在 T_i 之后对 X 版本进行更新，生成版本 X_j。

b) wr 边：$T_i \xrightarrow{wr} T_j$，表示对数据项 X，事务 T_j 中有一个读操作 $R_j(X_i)$，即读取事务 T_i 所更新的数据版本 X_i。

c) rw 边：$T_i \xrightarrow{rw} T_j$，表示对数据项 X，存在连续的版本 X_k，X_j 且 $X_k < X_j$，事务 T_i 读取 X_k，事务 T_j 更新版本 X_j。

表 1 SG 中事务的关系

名称	事务关系	历史 H 操作的关系
ww 边	$T_i \xrightarrow{ww} T_j$	$C_i <_H S_j$
wr 边	$T_i \xrightarrow{wr} T_j$	$C_i <_H S_j$
rw 边	$T_i \xrightarrow{rw} T_j$	$S_i <_H C_j$

2.3 算法描述

在 SSI 中，每个事务 T_i 开始生成一个快照版本时，使用 snapshot(T_i) 来表示，数据库中所有在 snapshot(T_i) 之前最新提交的数据都包含在快照中。将事务分为只读(read-only)事务和更新(update)事务，即只包含读操作的事务和包含更新操作的事务。事务 T_i 提交的标志 C_i 用 commit(T_i) 表示。具体定义如下：

readset(T_i) 为更新事务 T_i 中所有读操作的集合。

writeset(T_i) 为更新事务 T_i 中所有写操作的集合。

snapshot(T_i) 为事务 T_i 的快照开始的时间。

commit(T_i) 为事务 T_i 提交的时间。

由于快照隔离中，只读事务对其他事务没有影响，所以只需对快照的更新事务之间的关系进行分析来解决事务冲突并发的目的。事务之间的关系分为并发和顺序两种，顺序事务之间互不影响，只有并发事务之间才会产生冲突。

定义 2 写冲突(write-conflict)：

T_j write-conflict T_i 当且仅当

writeset(T_i) ∩ writeset(T_j) ≠ ∅，and snapshot(T_i) < commit(T_j) < commit(T_i)

定义 3 读冲突(read-conflict)：

T_j read-conflict T_i 当且仅当

readset(T_i) ∩ writeset(T_j) ≠ ∅，and snapshot(T_i) < commit(T_j) < commit(T_i)

由定义可知，对将要提交的更新事务 T_i，只有已经提交的更新事务才会影响到 T_i，未提交的事务和只读事务则不会影响到 T_i。当提交当前进行的事务 T_i 时，出现 T_j 影响 T_i，表示如果当前事务 T_i 立即提交，则会出现冲突的情况。

SSI 由三条规则组成：

对于任意的遵循 SSI 规则生成的历史版本 H，有以下三条属性。

规则 1 (SSI 读规则)

$\forall T_i, X_j$ 若 $\exists R_i(X_j) \in H(i \neq j)$，则 H 有以下属性：

1-$W_j(X_j) \in H$ 且 $C_j \in H$；

2-commit(T_j) < snapshot(T_i)；

3-$\forall T_k$，若 $W_k(X_k), C_k \in H(i \neq j \neq k) \Rightarrow$ commit(T_k) < commit(T_j)；或 snapshot(T_i) < commit(T_k)

规则 2 (SSI 无写冲突规则)

$\forall T_i, T_j$ 若 $C_i, C_j \in H$，则 H 有以下属性：

4-若 writeset(T_i) ∩ writeset(T_j) ≠ ∅ ⇒ commit(T_i) < snapshot(T_j)；或 commit(T_j) < snapshot(T_i)

规则 3 (SSI 无读冲突规则)

$\forall T_i, T_j$ 若 $C_i, C_j \in H$，则 H 有以下属性：

5-若 readset(T_i) ∩ writeset(T_j) ≠ ∅ ⇒ commit(T_i) < snapshot(T_j)；或 commit(T_i) < commit(T_j)

规则 1 表明，在 H 中，如果存在 $r_i(x_j)$，则代表生成版本 x_j 的更新事务 T_j 也在 H 中，且 $w_j(x_j)$ 和 c_j 都在 H 中，commit(T_j) < snapshot(T_i)。任一个对 X 操作的事务 T_k，要么 T_k 在 T_j 前提交，要么 T_k 在 T_i 开始之后提交。规则 1 确保了每个事务读取的是已经提交事务的版本，也就是说，事务读取的是当前数据库中已提交数据的快照，在分布式系统中，可以是其他任何副本中的事务修改后的快照版本。

规则 2 和 3 限定了历史版本 H 中哪些并发事务可以提交。在规则 2 中，当检测到一个更新事务对事务 T_i 存在写冲突时(write-conflict)，阻止 T_i 的提交。在规则 3 中，当检测到一个更新事务对事务 T_i 存在读冲突时(read- conflict)，则阻止 T_i 的提交。

2.4 正确性证明

引理 1 若历史 h 满足规则 1 ~ 3，则 h 是单副本可串行化的。

证明 假设历史 h 满足规则 1 ~ 3。要证明 H 是单副本可串行化的，即需要证明历史 H 所对应的串行化图 SG(H) 中不存在环，证明分 a)b) 两个步骤。

a) 令事务 T_i, T_j 属于历史 H，存在操作 p 属于 T_i，q 属于 T_j 且操作的数据对象相同，若 SG(H) 中存在边 $p \rightarrow q$，则必须满足 commit(T_i) < commit(T_j)。下面详细证明 $p \rightarrow q$ 的三种情况：

(a) $T_i \xrightarrow{ww} T_j$。假设数据 X，操作 $p = W_i(X_i)$，$q = W_j(X_j)$，且在 X 的版本中，X_i 先于 X_j。因为 $X_i \in$ writeset(T_i) ∩ writeset(T_j)，所以事务 T_i 和 T_j 的写事务集合的交集不为空，由规则 2 可知，commit(T_j) < snapshot(T_i) 或者 commit(T_i) < snapshot(T_j)。若情况为 commit(T_j) < snapshot(T_i)，则事务 T_j 的所有数据版本都在事务 T_i 的数据版本之前，即 X_j 先于 X_i，与前提 X_i 先于 X_j 冲突，所以为后者，即 commit(T_i) < snapshot(T_j) < commit(T_j)。

(b) $T_i \xrightarrow{wr} T_j$。假设数据 X，操作 $p = W_i(X_i)$，$q = R_j(X_i)$。由规则 1 可知 commit(T_i) < snapshot(T_j)。因为 snapshot(T_j) < commit(T_j)，所以事务 T_i 的所有执行操作先于事务 T_j，即 commit(T_i) < snapshot(T_j) < commit(T_j)，可得出 commit(T_i) < commit(T_j)。

(c) $T_i \xrightarrow{rw} T_j$。假设数据 X，操作 $p = W_i(X_k)$，$q = W_j(X_j)$，在 X 的版本中，X_k 先于 X_j。因为 $X_i \in$ readset(T_i) ∩ writeset(T_j)，所以事务 T_i 和 T_j 的读事务集合的交集不为空，由规则 3 可知，commit(T_j) < snapshot(T_i) 或者 commit(T_i) < commit(T_j)。若为前者，由规则 1 可知，commit(T_j) < commit(T_k)，即事务 T_j 的版本 X_j 先于事务 T_k 的版本 X_k，与前提冲突，因此为后者，即 commit(T_i) < commit(T_j)。

b) 由 a) 中(a) ~ (c) 可得，SG(H) 中不存在环。

假设 SG(H) 中存在环，如图 1 所示，则存在事务 T_1、T_2、T_n 有如下关系：

$$T_1 \rightarrow T_2, T_2 \rightarrow T_n, T_n \rightarrow T_1$$

由(a) ~ (c)的证明可知

$$T_1 \rightarrow T_2 \Rightarrow \text{commit}(T_1) < \text{commit}(T_2);$$
$$T_2 \rightarrow T_n \Rightarrow \text{commit}(T_2) < \text{commit}(T_n);$$
$$T_n \rightarrow T_1 \Rightarrow \text{commit}(T_n) < \text{commit}(T_1);$$

得出 $\text{commit}(T_1) < \text{commit}(T_1)$,所以假设不成立,即 SG($H$)不存在环。

综上所述,满足规则 1 ~ 3 的历史版本 H 的串行化图 SG(H)中不存在环,因此满足单副本可串行化。

证毕。

引理 2　对于满足规则 1 ~ 3 的历史 H 中,只读事务不会产生环,即只读事务可以直接提交。

证明　假设事务 T_i, T_j, T_k,没有对只读事务 T_i 进行判断,导致历史 H 的对应的串行化图 SG(H)中产生环,如图 2 所示。

因为 T_i 为只读事务,由表 1 可知,事务 T_i 和 T_j 的关系只能为 $T_i \xrightarrow{rw} T_j$,同理事务 T_k 和 T_i 的关系只能为 $T_k \xrightarrow{wr} T_i$,因此事务 T_j 和 T_k 都为更新事务,即都满足规则 1 ~ 3。

由表 1 中事务关系所对应的操作关系可知,$T_i \xrightarrow{rw} T_j \Rightarrow \text{snapshot}(T_i) < \text{commit}(T_j)$;$T_k \xrightarrow{wr} T_i \Rightarrow \text{commit}(T_k) < \text{snapshot}(T_i)$;所以,$\text{commit}(T_k) < \text{commit}(T_j)$。

由于图中事务 T_k 和 T_i 的关系为 $T_j \rightarrow T_k$,由引理 1 的证明知:$T_j \rightarrow T_k \Rightarrow \text{commit}(T_j) < \text{commit}(T_k)$,与($i$)矛盾,因此假设不成立,所以只读事务在满足规则 1 ~ 3 的历史中不存在环。

证毕。

由引理 2 可以得知,即使引入了读冲突的判断,只读事务也不会产生环,即只读事务仍然可以直接提交。这一特性表明,保证单副本可串行化的条件只是增加了对更新事务中的读操作集合的读冲突判断,对系统性能并不会产生明显的影响。

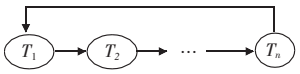

图 1　SG(H)中的环

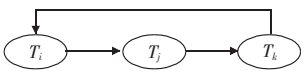

图 2　只读事务产生的环

3　SSI-REP 数据库同步协议

由于分布式数据库系统中数据副本物理位置的分散,SSI 规则中并发事务的判断需要包含所有数据副本中的事务。且 SSI 规则中的读冲突规则需要判断读操作集合,实际应用系统中的读操作数据量庞大,传播读操作会导致系统通信消耗加剧,影响系统的性能。本文基于组通信的可靠定序的特性,设计了 SSI-REP 数据库同步协议,只将更新事务的写操作集合广播到所有副本节点中,所有判断操作均在本地完成,完成 SSI 算法的同时,提高了系统的性能。

3.1　协议描述

当用户发起事务 T 的提交请求时,获取到用户提交的事务,开始对事务进行分析并执行本地副本读取所有副本更新的流程,将同步流程主要分为本地、冲突检测、发送、执行四个阶段。

3.1.1　本地阶段

本地阶段中截获用户提交的请求,通过对请求类型的判断进行不同的操作,最后将事务 T 提取为读操作集合和写操作集合 readset(T)和 writeset(T),若事务 T 为只读事务,则可以直接在本地执行并将结果返回给用户;若为更新事务,进入本地冲突检测。

3.1.2　冲突检测阶段

在每个数据副本中使用队列 $Q_{l,w}$ 来存放所有数据副本的已经通过本地冲突检测阶段的写操作集合,使用队列 $Q_{l,r}$ 来存放本地更新事务中读操作集合。通过本地阶段进入冲突检测阶段时,进行快照隔离可串行化规则 2 和 3 的判定,即判定队列 $Q_{l,w}$ 是否存在读冲突和写冲突。如果在本地的队列中发现冲突,则中止该事务并返回给用户;如果不存在冲突,则将事务的写操作集合通过组通信系统全序广播到所有副本节点中。

3.1.3　发送和执行阶段

通过组通信系统接收到事务 T 的写操作集合,由于组通信的全序特性,所有副本中接收到的写操作集合顺序一致,所以所有副本的判定结果一致,即若存在冲突则所有副本中的判定都为冲突,若不存在冲突则所有副本中的判定都不冲突。

首先,将其与本地中的写操作集合队列进行写冲突的判断,如果不存在冲突且该副本为远程副本,则将写操作集合加入本地的队列中;如果为本地副本,则在本地进行读冲突(SSI 规则 3)的判断,若不存在冲突,则表示事务满足 SSI 串行化规则,通过组通信系统广播事务 T 的执行信息(T,commit);如果存在读冲突,则表明该事务不满足串行化规则,通过组通信系统广播事务 T 的中止信息(T,abort)。

然后,所有节点通过组通信系统接收到事务的提交/中止(commit/abort)信息,并根据相应的信息执行提交/中止操作,最后将事务移出本地队列。

令 $R^L \in R, R = \{R_i, \cdots, R_n\}$ 为组内所有节点副本的集合;$Q_{l,r}$ 表示本地中更新事务的读操作集合的队列,$Q_{l,w}$ 表示所有副本中的等待提交的写操作集合,协议的工作流程如图 3 所示,协议具体步骤如下:

a)本地阶段(R^l 接收到用户发起事务 T_i 的操作)

(a)若为开始操作,初始化事务 T_i,等待下一个操作;

(b)若为读写操作(select, update, insert, delete),提交操作,并等待结果;

(c)若为中止(abort)操作,中止事务 T_i 并返回;

(d)若为提交(commit)操作,将事务 T_i 提取为读操作集合和写操作集合 $T_i.\text{WS} = \text{writeset}(T_i)$,$T_i.\text{RS} = \text{readset}(T_i)$;

(e)若 T_i 为只读型事务(即 $\text{writeset}(T_i) = \varnothing$),将 T_i 在 R^l 上提交并结束;否则进入冲突检测阶段。

b)冲突检测阶段

(a)在本地副本 R^l 中检测 SSI 中的规则 2 和 3,判断是否存在写冲突($\exists T_j \in Q_{l,w} \wedge T_i.\text{WS} \cap T_j.\text{WS} \neq \varnothing$)和读冲突($\exists T_j \in Q_{l,w} \wedge T_i.\text{RS} \cap T_j.\text{WS} \neq \varnothing$);

(b)若存在冲突则中止事务并返回,否则将事务 T_i 的读操作集合和写操作集合分别加入队列 $Q_{l,r}$ 和 $Q_{l,w}$ 中,并进入发送阶段。

c)发送阶段

(a)将 $\text{writeset}(T_i)$ 通过组通信系统广播到其他组成员节点;

(b)组成员节点接收到 $\text{writeset}(T_i)$,进行冲突检测。

d)执行阶段

(a)远程副本节点接收到事务 T_i 的更新操作且不存在冲突时,将其加入本地队列;

(b)若本地副本接收到 T_i 的更新操作,第二次进行读冲突

和写冲突规则的判断,若不存在冲突则全局广播(T_i,commit),否则全局广播(T_i,abort);

(c)若接收(T_i,commit),远程副本执行 T_i,本地副本执行 T_i 并返回用户结果;若接收到(T_i,abort),将 T_i 移出队列。

SSI-REP 协议中,由于要达到 SSI 串行化算法的标准需要进行并发事务之间读操作和写操作的判断。在实际应用中,由于读操作集合较大,若采用写冲突广播写操作集合的方式,广播读操作集合进行读冲突判断,会增加系统的通信消耗,影响系统的性能。因此,协议中通过在本地存放读操作集合,并在本地进行读冲突的判断,将判断结果通过组通信系统广播。这样虽然多了一次广播操作,但是由于广播的只是事务标志和提交信息,而且判定都是在本地执行,相对广播读操作集合减少了系统的通信消耗,提高了系统的性能。

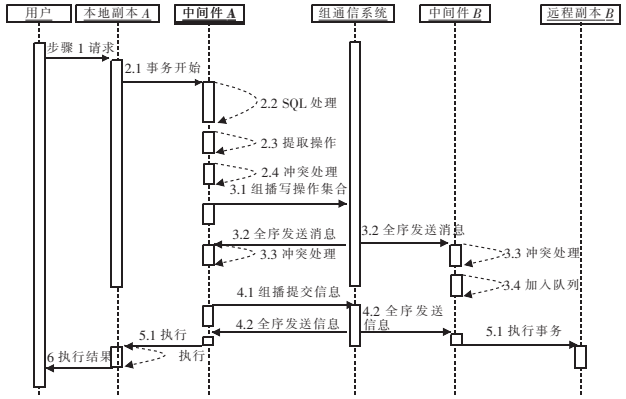

图 3 SSI-REP 协议流程

3.2 实例分析

通过一个实例来详细描述 SSI-REP 协议的工作流程。假设副本 R_A,$R_B \in R$,事务集合 $T=\{T_1=w_1(x);\ T_2=w_2(y);T_3=w_3(x);T_4=r_4(x),r_4(y);T_5=r_5(x),r_5(y)\}$,$T_1$ 和 T_4 通过中间件连接副本 R_A、T_2、T_3、T_5 通过中间件连接副本 R_B。如图所示,使用灰色颜色的方块图来表示远程事务,队列 Q 表示等待提交的事务。图中事务在中间件和数据库上执行的时间顺序从左至右依次递增,竖线表示事件在中间件和数据库中执行的依赖关系(因果)。为方便理解和阅读,暂不标记事务的上标,即使用 T_1 而不是 T_1^A 标记事务 T_1。

由图4 可知,T_1、T_2、T_3 为并发事务,假设三个事务中 T_1 最先发起 commit 请求,中间件接收到请求后从事务 T_1 中提取写操作和读操作。T_1 的写操作集合 writeset(T_1) = {$w(x)$},经过本地冲突检测,Q^A 中不存在并发事务会与事务 T_1 产生读冲突和写冲突,将 T_1 的写操作集合广播。副本 R_A,R_B 接收到该集合后加入到队列 Q^A 和 Q^B 中。事务 T_2 发起提交请求时,副本 R_B 的队列 Q^B 中为 T_1,与其不发生冲突,因此将 T_2 的写操作集合广播,加入队列 Q^A 和 Q^B 中。事务 T_3 发起提交请求时,由于并发事务 T_1 已经通过检测,并且与 T_3 产生写冲突,因此事务 T_3 将被中止。

在 R^A 中开始对收到广播的事务 T_1 时进行第二次冲突检测,此次是检测队列中是否存在 T_1 在本地检测之后,广播之前接收到的并发事务与其产生冲突,例子中 Q^A 中不存在与 T_1 冲突的事务,因此广播事务 T_1 的提交信息(T_1,commit)。图中 C_1 之前的空白时间为中间件广播并收到(T_1,commit)的时间。在副本 A 中,提交事务 T_2 之前,只读事务 T_4 开始执行,由于快照的读特性,事务 T_4 开始时只有事务 T_1 成功执行,因此读取的版本为 x_1 和 y_0。

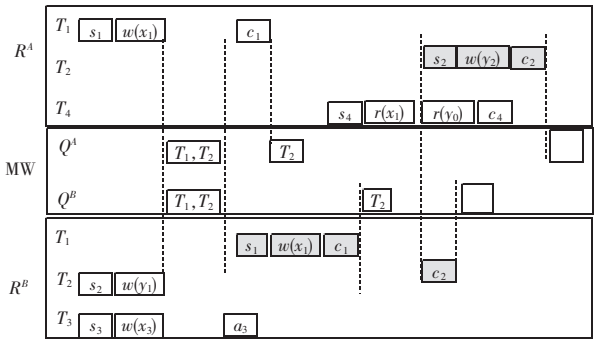

图 4 SSI-REP 执行实例

4 性能评估

与文献[3]中基于 GSI 算法的数据库同步协议相比,SSI-REP 数据库同步协议在分布式环境下基于组通信系统实现,增加了对并发事务读冲突的判断,从而实现单副本可串行化的准则。由于读冲突的判断需要获得并发事务的读操作,方法之一是将更新事务的读操作和写操作全部广播到所有副本节点中进行冲突检测;但是实际应用中的数据库操作中读操作的信息量过于庞大,广播读操作会增加通信消耗,影响系统性能,因此采用在本地进行读冲突检测,并通过广播检测结果的方法来实现 SSI-REP 协议。本文在实验中对 GSI、SSI-REP、2PL 三种协议进行性能测试并比较分析。

4.1 测试环境

本文实现了一个基于组通信系统 Spread[17] 的数据库同步中间件,部署在每台组成员节点上。模拟环境由四台配置相同的 PC 作为数据副本的节点组成员相互通信:CPU 采用 Intel 的 Pentium4 处理器 2.93 GHz 主频,2 GB 内存,100 GB 硬盘。

前端采用在两个副本的 PC 机上模拟两个用户同时向组成员的工作节点发送更新事务请求,即模拟并发事务的产生。

测试使用表名 table_group,表字段中一个 integer 类型作主键,一个 50 Byte 的 char 类型,一个 integer 型的随机数作为数据。更新事务请求以 update 类型提交。

4.2 性能测试

测试模拟了不同副本节点中的用户同时连续提交 50、100、200、300、400、500 个事务的请求,更新事务的比重分为 100%、50% 两种情况,观察了系统完成事务所需的时间和各节点副本达到数据一致性所需要的时间。

由图 5 可以看出,当为 100% 的更新事务时,不存在只读事务,SSI-REP 与 GSI 相比,多了一次事务提交信息的广播,性能上随着事务量的增大要略微下降,但与 2PL 相比,性能显著提高。

图 6 表明,当提交的事务中的只读事务所占比例变大时,SSI-REP 和 GSI 的系统响应时间越来越接近,且相对 2PL 都在性能上有了显著的提升。因此,实际应用中,SSI-REP 的性能略微弱于 GSI,但是对比原有策略(2PL),性能随着事务量的增加而明显提高。

图 7 显示了在 50% 的更新事务中,随着并发事务的增加而产生的事务冲突导致事务中止的现象。SSI-REP 和 GSI 都

对写写冲突进行了判断并中止,SSI-REP 为达到单副本可串行化同时对读写冲突进行判断。实验结果表明,由读写冲突引发的额外的中止情况占全部中止事务的很少一部分,即 SSI-REP 中增加的读写冲突的判定对事务中止的比率的影响很少。

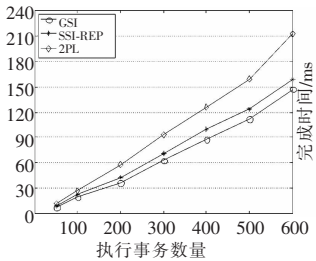

图 5　100%更新事务

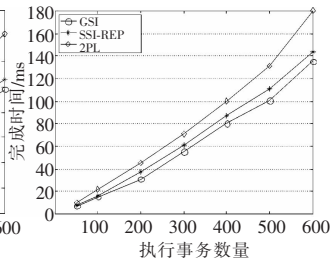

图 6　50%更新事务

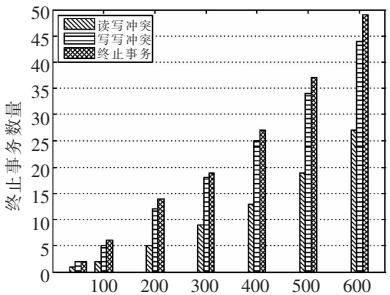

图 7　50%更新事务下的中止事务

6　结束语

维护分布式数据系统中的各节点副本一致性是数据库同步的核心问题,许多同步方案通过牺牲系统的性能来达到数据一致性。本文采用快照隔离技术实现分布式数据库的同步,并基于当前研究提出快照隔离可串行化算法 SSI,设计了分布式环境下的数据库同步协议 SSI-REP。实验结果表明,在 SSI-REP 数据库同步协议下,系统在维护节点副本一致性和保证快照隔离可串行的同时仍然有较好的性能。

参考文献:

[1] GRAY J, HELLAND P, O'NEIL P,*et al*. The dangers of replication and a solution[C]//Proc of SIGMOD. New York:ACM Press,1996:173-182.

[2] LIN Yin, KEMME B, PATINO-MARTINEZ M, *et al*. Middleware based data replication providing snapshot isolation[C]// Proc of SIGMOD. New York:ACM Press,2005:419-430.

[3] ELNIKETY S, ZWAENEPOEL W, PEDONE F. Database replication using generalized snapshot isolation[C]//Proc of the 24th IEEE SRDS. Washington DC:IEEE Computer Society,2005:73-84.

[4] BERENSON H, BERNSTEIN P, GRAY J,*et al*. A critique of ANSI SQL isolation levels[C]// Proc of ACM SIGMOD. New York:ACM Press,1995:1-10.

[5] LIN Y,KEMME B,ARMENDARIZ J E,*et al*. Snapshot isolation and integrity constraints in replicated databases [J]. ACM Trans on Database System,2009,34(2):1-49.

[6] ALOMARI M, FEKETE A,ROHM U. A robust technique to ensure serializable executions with snapshot isolation DBMS[C]//Proc of IEEE ICDE. Washington DC: IEEE Computer Society, 2009: 341-352.

[7] FEKETE A,LIAROKAPIS D, O'NEIL E,*et al*. Making snapshot isolation serializable[J]. ACM Trans on Database Systems,2005,30(2):492-528.

[8] ADYA A. Weak consistency:a generalized theory and optimistic implementations for distributed transactions[D]. [S.l.]: MIT Lab for Computer Science, 1999.

[9] KEMME B,ALONSO G. A new approach to developing and implementing eager database replication protocols[J]. ACM Trans on Database Systems,2000,25(3):333-379.

[10] KEMME B,PEDONE F,ALONSO G,*et al*. Using optimistic atomic broadcast in transaction processing systems [J]. IEEE Trans on Knowledge Data Engineering,2003,15(4):1018-1032.

[11] ATTIYA H,WELCH J. Distributed computing[M]. [S.l.]:Wiley-Interscience, 2004.

[12] KEMME B,ALONSO G. Database replication:a tale of research across communities[J]. Proc of VLDB,2010,3(1-2):5-12.

[13] WU Shu-ging,KEMME B. Postgres-r(si):combining replica control with concurrency control based on snapshot isolation [C] //Proc of the 21st ICDE. Washington DC:IEEE Computer Society,2005:422-433.

[14] ABDELLATIF T,CECCHET E, LACHAIZE R. Evaluation of a group communication middleware for clustered J2EE application servers [C]//Proc of CoopIS/DOA/ODBASE. Berlin: Springer-Verlag, 2004:1571-1589.

[15] KEMME B, ARMENDARIZ J E, PATINO-MARTINEZ M. Database replication[M]. [S.l.]:Morgan and Claypool, 2010.

[16] 娄燕飞,杨树强,李爱平,等. 一种基于组通信的复制数据库在线恢复策略研究[J]. 计算机研究与发展, 2007, 44(Z3):232-237.

[17] AMIR Y, NITA-ROTARU C, STAN TON J,*et al*. Secure spread:an integrated architecture for secure group communication [J]. IEEE Trans on Dependable and Secure Computing,2005,2(3):248-261.