基于操作历史图的分布式 Key-Value 数据库一致性检测算法

廖 彬[1] 张 陶[2,3] 李 敏[1] 于 炯[2] 国冰磊[2] 刘 炎[4]

(新疆财经大学统计与数据科学学院 乌鲁木齐 830012)[1]

(新疆大学信息科学与工程学院 乌鲁木齐 830046)[2]

(新疆医科大学医学工程技术学院 乌鲁木齐 830011)[3] (清华大学软件学院 北京 100084)[4]

摘 要 分布式数据库系统的副本机制在提高系统可靠性及性能的同时,导致了多副本数据管理的一致性问题;数据一致性的实现需要一致性协议模型来进行预防,也需要一致性检测算法对非一致数据进行检测。首先,对读写操作记录之间的时序关系、安全一致性及并行一致性原则等概念进行定义;其次,根据操作记录集合中读写操作之间的并行与时序关系,提取出操作记录集合向操作记录图转化的规则,并在此基础上设计了操作记录向历史记录图的转化算法;然后,以历史记录图为输入,设计了违反一致性查找算法,查找并返回图中所有违反安全与并行一致性读操作的集合;最后,基于 Cassandra 进行实验并将读写一致性设置为 ONE,通过 YCSB 产生并行读写压力测试,与同类算法的对比实验验证了所提算法在功能与效率两方面的优越性。

关键词 分布式数据库,Key-Value 数据库,一致性原则,一致性检测,DAG 图

中图法分类号 TP391 **文献标识码** A **DOI** 10.11896/jsjkx.181102097

Consistency Checking Algorithm for Distributed Key-Value Database Based on Operation History Graph

LIAO Bin[1] ZHANG Tao[2,3] LI Min[1] YU Jiong[2] GUO Bing-lei[2] LIU Yan[4]

(College of Statistics and Data Science, Xinjiang University of Finance and Economics, Urumqi 830012, China)[1]

(School of Information Science and Engineering, Xinjiang University, Urumqi 830046, China)[2]

(Department of Medical Engineering and Technology, Xinjiang Medical University, Urumqi 830011, China)[3]

(School of Software, Tsinghua University, Beijing 100084, China)[4]

Abstract The replica mechanism of distributed database system not only improves reliability and performance of the overall system, but also leads to the consistency problem of multi-replica data management mechanism. To keep the consistency of data, a consistency protocol model is needed to avoid data's inconsistency events. Moreover, consistency checking algorithms are also needed to detect inconsistent data. Firstly, the concepts of temporal relations, security consistency, and concurrent consistency between read and write operations are defined. Secondly, according to the parallel and temporal relationship between read and write operations that recorded in the set of operations, the rules of transforming operation record set to operation record graph are extracted, and then the algorithm of transforming operation records into operation record graph is also designed. Then, taking the set of operation record graph as input, a violation operation search algorithm is designed to find the set of inconsistent read operations which have violated security and parallel consistency. Finally, experiments are conducted based on Cassandra and the read-write consistency is set to ONE. YCSB generates parallel read-write stress tests. The comparative experiments with similar algorithms verify the advantages of the proposed algorithm in both function and efficiency.

Keywords Distributed database, Key-Value database, Consistency principle, Consistency check, DAG diagram

1 引言

分布式数据库中,数据的一致性问题一直都是研究的热点。这是因为不一致的数据不能正确全面地表达现实世界客观事物的真实状态,破坏了客观事物的关联关系,进而误导人们得出错误的结论和结果,大大降低了数据的应用价值[1]。分布式数据库系统在采用数据副本复制机制来提高系统可靠性及性能的同时,引入了数据副本管理的一致性问题。如何

到稿日期:2018-11-14 返修日期:2019-03-25 本文受国家自然科学基金项目(61562078,61462079,61862060),新疆天山青年计划项目(2018Q073)资助。

廖 彬(1986—),男,博士,副教授,硕士生导师,主要研究方向为绿色计算、数据挖掘及大数据计算模型等,E-mail:liaobin665@163.com(通信作者);**张 陶**(1988—),女,博士生,主要研究方向为分布式计算、网格计算;**李 敏**(1990—),女,硕士,讲师,主要研究方向为大数据计算;**于 炯**(1964—),男,博士,教授,博士生导师,主要研究方向为网络安全、网格与分布式计算;**国冰磊**(1991—),女,博士生,主要研究方向为绿色计算、数据库系统等;**刘 炎**(1990—),男,硕士生,主要研究方向为大数据计算。

既保证数据的一致性同时又不影响系统运行的性能，是分布式系统需要重点考虑和权衡的问题。

由于大多传统的关系型数据库采用单机架构，事务的ACID 4个性质之间并不存在相互冲突的关系。但在分布式场景下，根据分布式系统的CAP[2]理论，分布式系统不可能同时满足一致性(Consistency)、可用性(Availability)和分区容错性(Partition tolerance)这3个基本需求；在不同的分布式应用场景下，只能根据不同的需求，选择满足CAP中的任意两项原则。系统选择CA放弃P，即加强一致性和可用性，放弃分区容错性，这是传统单机数据库的选择；系统选择AP放弃C，即追求分区容错性与可用性而放弃一致性(一般指强一致性)，这是当前大部分分布式数据库系统(如Base，MongoDB，Couchbase等)的选择；系统选择CP放弃A，即追求一致性与分区容错性，放弃可用性，由于分布式系统中引入副本机制的目的之一就是加强系统的可用性，因此CP不会被采用。

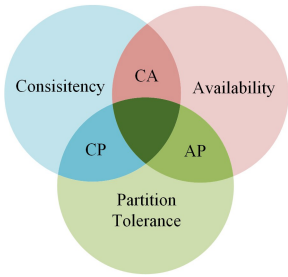

图1 分布式系统的CAP理论

Fig. 1 CAP theory of distributed systems

在CAP的基础上，BASE理论面向高可用高可扩展的分布式数据库系统，在C(一致性)和A(可用性)之间寻求平衡，其核心思想是：虽然无法做到100%的强一致性，但每个系统都可以根据自身应用场景的特点，采用适当的方式来使系统达到最终一致性(最终一致性是指所有的数据副本在经过一段时间的同步之后，最终都能达到一致的状态)。

大多分布式数据库系统都会引入一致性协议模型(如强一致性协议、乐观一致性协议等)来尽量避免不一致问题的出现，因此当前针对分布式数据库中一致性控制问题的相关研究，大多以设计最优的一致性协议模型为出发点，以更好地平衡系统的可用性与一致性之间的矛盾。但是，利用一致性协议模型来保障一致性的方法在本质上是一种预防性策略。而在真实的大规模分布式系统场景下，由于节点、通信网络及软硬件系统等出现故障已经不是小概率事件，太严格的一致性协议在增加系统复杂度的同时降低了系统的性能，并且过于乐观的一致性协议在提升系统性能的同时容易引发不一致现象的频发，导致大量不一致错误的积累，因此，在分布式数据库中，不仅需要一致性协议模型来预防数据一致性问题的出现，还需要引入一致性检测机制来对系统中出现的一致性错误进行及时的检测，为后续一致性错误数据的修复打下基础。当前，已有研究主要集中在一致性协议与模型的研究上，针对一致性检测算法的研究成果较少。本文在Key-Value分布式数据库中，将多副本数据上的操作集合转化为操作历史图，并基于此设计了一致性检测算法。

本文第2节对相关工作进行了介绍；第3节对安全一致性、并行一致性等相关概念进行了定义；第4节介绍了操作记录向历史记录图的转化与违反一致性查找两种算法；第5节通过实验对本文算法的有效性进行了验证；最后总结全文并展望未来。

2 相关工作

数据一致性的实现即需要一致性协议(模型)来保障，也需要一致性检测算法进行辅助，及时地检测出系统中已经产生的不一致问题，能够为后期的一致性数据修复提供帮助。当前，已有研究主要集中在一致性协议或一致性模型的研究上，而一致性协议主要基于一致性算法来实现，一致性算法的主要分类如表1所列。

表1 一致性算法的模型分类

Table 1 Model classification of consensus algorithm

基本模型	代表算法	应用场景
以Leader选举为主	Paxos[3-4]，viewstamp[5]	Zookeeper[6-8]，Chubby等
以弱一致性、因果一致性、顺序一致性为主	未开源公布	应用在Linkedin，Twitter，Facebook等公司内部
以租赁机制为主，分布式锁	理论阶段，还未实现	理论阶段，目前没有纯粹“分布式”锁的实现
以弱一致性为主	Cassandra[9-11]的可配置一致性算法	Cassandra的W，R，N可调节的一致性
以分布式事务为主	两阶段提交	大部分分布式数据库系统都支持，如Hbase[12-13]等
冗余容错及最终一致性	Gossip[14-15]，Phi等	Cassandra等

鉴于分布式数据库系统中一致性检测算法的已有研究成果较少，文献[16]提出通过条件函数依赖(Conditional Functional Dependencies，CFDs)将语义上关联的数据绑定，CFDs-based方法能很好地捕获数据产生的一致性问题。文献[17]同样基于CFDs设计了一种适用于少量数据场景的一致性修复算法，但在大数据或分布式场景下，该算法并不具备较好的适用性。文献[18]设计了Hadoop下的不一致性数据检测算法，根据用户给出的不一致数据、属性权重和CFDs规则，对数据中违反规则的数据项通过修改值的方式，逐步求得其一致性修复结果。以上工作或是基于数据之间的条件函数依赖，或是需要属性权重等额外的元数据信息，在真实的分布式数据场景下，当缺乏这些元数据信息时，将导致这些方法的适用性较差。

文献[19]将关系型数据库中的安全一致性、原子一致性等概念引入到基于Key-Value的分布式数据库系统中，提出基于DAG(Directed Acyclic Graph)[20-22]的一致性检测算法，该算法将每一个作用于Key-Value数据项的操作映射为DAG图中的顶点，并将操作之间的时序关系映射为时间边(Time Edge)、数据边(Data Edge)和混合边(Hybrid Edge)3种，最后通过检测DAG图是否有环来判定数据是否满足一致性原则。由于算法根据DAG是否有环来判断数据的一致性，复杂度较高，因此文献[19]中的算法通常设置在系统离线时执行；并且，判断DAG是否有环，只能简单地得出数据是否违反一致性原则的结论，并不能精确地统计出违反一致性原则的次数。本文借鉴了文献[19]将作用于Key-Value数据项的操作转化为DAG图的思路，改进其基于DAG环检测的一致性检测算法，首先将作用于Key-Value数据项的操作集

合转化为图,并基于图设计了一种高效的违反一致性查找算法。对比实验结果表明,本文算法不仅能够返回违反一致性原则操作的集合,而且时间复杂度更低,算法效率更高。

3　基本概念定义及其示例

3.1　相关概念及定义

本文用到的所有符号及其含义如表 2 所列。

表 2　符号及其含义

Table 2　Symbol and its meaning

符号	含义解释
D	数据对象
k	数据对象 D 的 key 值
v	数据对象 D 的 value 值
r	数据的副本系数
L	数据 D 的读写记录
st	一条读写记录 L 的开始时间
et	一条读写记录 L 的结束时间
ot	L 的操作类型,分为 Read(R)和 Write(W)两种
T	一致性算法的执行时间周期
LS	读写操作记录集合
SCP	安全一致性原则(Safety Consistency Principle)
PCP	并行一致性原则(Parallel Consistency Principle)
G	操作记录图 G
V	操作记录图 G 的顶点集合
E	操作记录图 G 有向边的集合
C	操作记录图 G 的顶点值集合
f	图 G 中顶点到{0,1}值域的映射函数
g	图 G 中顶点到{0,1}值域的映射函数
$S_{\neg scp}$	违反安全一致性读操作的集合
$S_{\neg pcp}$	违反并行一致性读操作的集合

定义 1(数据对象 D 及其副本)　在 Key-Value 数据库系统中,设任意数据对象为 D,D 可以用二元组(k,v)表示,其中 k 表示数据对象 D 在系统中的唯一标识符,v 是数据对象的具体值。在大多数 Key-Value 数据库系统中,可以通过 k 得到 v 值。当系统副本系数为 r 时,一致性检测算法需要检测在特定时间段 t 内包括 D 与其副本$\{D_1,D_2,\cdots,D_{r-1}\}$在内的数据一致性。

定义 2(数据对象读写操作记录)　针对数据对象 D 的操作有读(Read)与写(Write)两种,设读写记录 L 用五元组$\{k,v,st,et,ot\}$表示,其中 k 表示数据项 D 的唯一标示,v 是当前读写记录的结果值,而 st 与 et 分别表示操作的开始时间与结束时间,ot 表示操作的类型。特别地,当 ot=Read 时,v 是读操作的返回值,而当 ot=Write 时,v 是写操作的写入值。

由于数据访问的并行性,针对数据对象 D 的副本$\{D_1,D_2,\cdots,D_{r-1}\}$的读写记录都统一记录为对数据项 D 的操作。分析不同读写记录之间的时序关系可以发现,任意两个读写记录 L 与 L'之间存在着顺序与并行两种关系。

定义 3(读写记录之间的顺序关系)　如果任意两个读写记录 L 与 L'之间满足条件:$L.et \leqslant L'.st$,则认为 L 与 L'之间是顺序关系,记为 $L \rightarrow L'$,逻辑上可称 L 是 L'的前驱操作,L'是 L 的后继。在分布式系统中,顺序关系并不会引起数据的不一致现象,研究读写操作记录之间的并行关系,是检测数据出现一致性问题的核心。

定义 4(读写记录之间的并行关系)　当任意两个读写记录 L 与 L'之间任意满足以下两个条件之一:即 $L'.st \leqslant L.st < L'.et$ 或 $L.st \leqslant L'.st < L.et$ 时,则可以判定 L 与 L'之间是并行关系,记为 $L \parallel L'$。

定义 5(直接前驱与直接后继关系)　设在某个一致性检测周期 T 内,针对数据对象 D 的所有 n 条读写操作记录集合为 LS,即 $LS=\{L_1,L_2,L_3,\cdots,L_n\}$,当 LS 集合中任意的两条记录 L 与 L'存在 $L \rightarrow L'$,并且满足条件 $\exists L'' \in LS-\{L,L'\}$,$L \rightarrow L'' \rightarrow L'$时,可以判定 L 是 L'的直接前驱,而 L'是 L 的直接后继,记为 $L \vdash\!\!\rightarrow L'$。

定义 6(安全一致性原则,Safety Consistency Principle)　对于操作集合 LS 中任意的读操作 $L_R \in LS$,如果不存在任意的写操作 $L_W \in LS$,满足并行条件 $L_R \parallel L_W$,那么 L_R 读到的值一定为 L_R 所有前驱操作中某个写操作的写入值 v,并且不存在任意的一个写操作 $L_{W'} \in LS$ 同时为 L_R 的前驱及 L_W 的后继。当存在与 L_R 并行的写操作时,允许 L_R 读到任意值。安全一致性原则的逻辑表达式为:

$$\forall L_R \in LS, \exists L_W \in LS, L_W \parallel L_R \Rightarrow \exists L_{W'} \in LS, L_R.v = L_{W'}.v \wedge \exists L_{W''} \in LS, L_W \rightarrow L_{W''} \rightarrow L_R \quad (1)$$

当 LS 中所有的读操作 L_R 都满足上述原则时,称 LS 满足安全一致性原则,记为:$LS \in SCP$;相反,如果存在任意的 $L_R \in LS$ 不满足上述规则,称 L_R 违反安全一致性原则,记为:$LS \in \neg SCP$。

定义 7(并行一致性,Parallel Consistency Principle)　并行一致性是指,对于操作集合 LS 中任意的读操作 $L_R \in LS$,如果不存在任意的写操作 $L_W \in LS$ 满足并行条件 $L_R \parallel L_W$,那么 L_R 读到的值一定为 L_R 所有前驱操作中某个写操作的写入值 v,并且不存在任意的一个写操作 $L_{W'} \in LS$ 同时为 L_R 的前驱及 L_W 的后继。而当存在与 L_R 并行的写操作时,L_R 需读到与其并行的这些写操作的写入值。其逻辑表达式为:

$$\forall L_R \in LS, \exists L_W \in LS, L_W \parallel L_R \Rightarrow \exists L_{W'} \in LS, L_R.v = L_{W'}.v \wedge \exists L_{W''} \in LS, L_W \rightarrow L_{W''} \rightarrow L_R \wedge L_R.v = L_W.v \in \{L_W, L_W \parallel L_R\} \quad (2)$$

当 LS 中所有的读操作 L_R 都满足上述原则时,称 LS 满足并行一致性原则,记为:$LS \in PCP$;相反,如果存在任意的 $L_R \in LS$ 不满足上述规则,称 L_R 违反并行一致性原则,记为:$LS \in \neg PCP$。

根据安全一致性及并行一致性的定义可以看出,并行一致性的规则强于安全一致性。即对于任意的 $L_R \in LS$,当 L_R 满足并行一致性原则时,L_R 一定满足安全一致性原则;相反,当 L_R 满足安全一致性原则时,L_R 不一定满足并行一致性原则。

3.2　违反一致性原则示例

设数据项 D 与其 3 个副本$\{D_1,D_2,D_3\}$的数据访问记录列表如表 3 所列。

表 3　数据访问记录列表示例

Table 3　Example of data access record list

编号	key	记录项
1	D_1	$L_1=\{D_1,v20,1,2,R\}$
2	D_3	$L_2=\{D_3,v10,3,4,W\}$
3	D_2	$L_3=\{D_2,v20,5,7,R\}$
4	D_3	$L_4=\{D_3,v10,7,9,R\}$
5	D	$L_5=\{D,v50,8,14,W\}$
6	D_1	$L_6=\{D_1,v50,9,11,R\}$
7	D_2	$L_7=\{D_2,v20,11,12,R\}$
8	D_1	$L_8=\{D_1,v50,13,15,R\}$
9	D	$L_9=\{D,v50,16,17,R\}$
10	D_2	$L_{10}=\{D_2,v10,18,20,R\}$

将表 3 中的数据访问记录中的时序关系信息抽取出来，得到图 2 所示的各访问记录之间的时序关系。

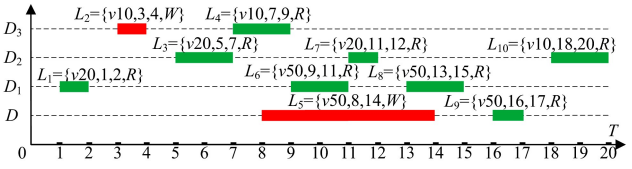

图 2 数据访问记录的时序关系

Fig. 2 Timing relationship for data access records

根据安全一致性及并行一致性的定义，图 2 中的数据操作记录违反安全一致性与并行一致性的结果。违反一致性操作结果如表 4 所列。

表 4 违反一致性操作结果

Table 4 Results of violating consistency operations

记录项	安全一致性	并行一致性
$L_1=\{D_1,v20,1,2,R\}$	√	√
$L_2=\{D_3,v10,3,4,W\}$	—	—
$L_3=\{D_2,v20,5,7,R\}$	×	×
$L_4=\{D_3,v10,7,9,R\}$	√	√
$L_5=\{D,v50,8,14,W\}$	—	—
$L_6=\{D_1,v50,9,11,R\}$	√	√
$L_7=\{D_2,v20,11,12,R\}$	√	×
$L_8=\{D_1,v50,13,15,R\}$	√	×
$L_9=\{D,v50,16,17,R\}$	√	√
$L_{10}=\{D2,v10,18,20,R\}$	×	×

注：√表示满足相关一致性定义，×表示不满足，—表示缺省项

表 4 中，读操作 $L_3=\{D_2,v20,5,7,R\}$并没有与其他写操作并行，L_3 应该读到 L_2 的写入值 $v10$，但 L_3 读到的却是 $v20$，根据定义 6 与定义 7，L_3 违反安全与并行一致性原则；同理，L_{10} 也违反安全与并行一致性原则。L_4，L_6，L_7 与 L_8 同时并行于写操作 L_5，其中 L_4 读到的是 L_2 的写入值 $v10$，而 L_6 读到的是写操作 L_5 的写入值，L_4 与 L_6 都满足安全与并行一致性的定义。根据安全一致性原则的定义，L_7 与 L_8 满足安全一致性原则，但不满足并行一致性原则。

4 操作记录图与一致性检测算法

4.1 基于操作集合 *LS* 构建操作记录图

定义 8(操作记录图，Execution Log Graph) 操作记录图是由操作记录 LS 转化而来，属于 DAG(有向无环图)类型。操作记录图可用一个三元组表示 $G=(V,E,C)$，其中 V 是操作记录图 G 顶点的集合，E 是顶点之间有向边的集合，C 是顶点值模型。其中，顶点的值模型用三元组 $C=(L_i,f,g)$表示，其中 L_i 表示一条操作记录，f 与 g 都是顶点到{0,1}值域上的映射。操作记录集合 LS 向操作记录图的转化遵循以下 4 条规则。

(1)操作记录图 G 中的顶点 v_i 与操作记录集合 LS 中的记录 L_i 一一映射。

(2)操作记录图 G 中的有向边与操作记录之间的直接后续关系(定义 5)一一映射。

(3)映射函数 f 到{0,1}值域的映射规则是：当存在与 L_i 并行的操作记录，且这些操作记录与 L_i 类型不同时，$f=1$，否则 $f=0$。

(4)映射函数 g 到{0,1}值域的映射规则是：当任意的并行的读操作记录 L_R 和写操作记录 L_W 满足条件：$L_W \parallel L_R \wedge L_W.v=L_R.v$ 时，$g=1$，否则 $g=0$。

根据以上规则，操作记录 LS 向历史记录图 G 的转化算法如算法 1 所示。

算法 1 操作记录 LS 向历史记录图 G 的转化算法

```
INPUT:操作历史记录集合 LS={L1,L2,L3,…,Ln}
OUTPUT:历史记录图 G
1. LS.orderByTimeASC//将 LS 集合中的记录按照时间排序
2.    G←(V,E,C)//定义并初始化图 G
3.    V←List{vi|i∈[0,|LS|−1]}//初始化图 G 的顶点集合 V
4.    E←List{ei|i∈[0,|LS|−1]}//初始化图 G 的边的集合 V
5.    Len←|LS|//LS 集合的大小
6.    f(vi)←0 //初始化映射函数 f
7.    g(vi)←0 //初始化映射函数 g
8.      for i=0 to Len−1 do
9.        for j=i++ to Len−1 do
10.         if L[j].st<L[i].et and L[j].ot≠L[i].ot//检测并行
11.           f(vi)←1
12.           f(vj)←1
13.           if L[j].v==L[i].v
14.             g(vi)←1
15.             g(vj)←1
16.           end if
17.         end if
18.         if L[j].st≥L[i].et and L[i]↛L[j]
19.           eij←edge(L[i],L[j])
20.           E←E+eij
21.         end if
22.       end for
23.     end for
24. return G
```

算法 1 的第 1 行将 LS 集合中所有的记录按照时间进行升序排序。第 3 行在排序的基础上初始化图的顶点，满足规则 1，将记录图 G 中的顶点 v_i 与操作记录集合 LS 中的记录 L_i 一一映射。第 10－12 行满足规则 3，即当存在与 L_i 并行的操作记录，且这些操作记录与 L_i 类型不同时，$f=1$。第 13－15 行满足规则 4，即当任意的并行的读操作记录 L_R 和写操作记录 L_W 满足条件 $L_W \parallel L_R \wedge L_W.v=L_R.v$ 时，$g=1$。第18－20 行满足规则 2，即将图 G 中的有向边与操作记录之间的直接后续关系进行一一映射。

算法 1 的时间复杂度分析如下：在算法 1 的第 1 行的排序操作中，如果采用时间复杂度最优的快速排序或堆排序，则时间复杂度为 $O(n*\log n)$，其中 n 为 LS 集合中操作的个数。第 2－7 行的初始化时间复杂度都为 $O(n)$。第 8－10 行为两层 for 循环嵌套 if 判断，时间复杂度由 if 判断真值次数决定，第 10 行为检测 LS 集合中所有并行的操作，设最大并行数为 m；对于内层 for 循环以及 if 判断，对于 LS 集合中任意的操作 L_i，当与之最大并行度为 m 时，每次 if 判断中的代码最多会被操作 m 次；加之外层 for 循环，因此第 8－10 行代码的复杂度为 $O(n*m)$。同样，18－20 行代码的复杂度为 $O(n*m)$。因此，算法 1 的时间复杂度为 $O(n*\log n+5n+2n*m)\approx O(n(\log n+m))$。

4.2 基于 *G* 寻找违反一致性原则的操作记录

寻找历史记录图 G 中所有违反一致性原则的读操作列

表,需要将图 G 作为输入,算法操作后,分别将所有违反安全一致性与并行一致性的读操作的记录放入到集合 S_{scp} 与 S_{pcp} 中,并将最终结果 S_{scp} 与 S_{pcp} 返回。该算法的核心思想是,对于任意的读操作 L_R(或图 G 中任意的顶点类型为读操作的顶点 v),可根据其前驱写操作中最晚结束的时间作为索引,找到与其并行的所有写操作,如果找到某些操作的值等于该读操作 L_R 的值,则根据定义 6 与定义 7,该操作一定满足安全一致性与并行一致性要求。如果该读操作 L_R 的并行度为 0,则违反了安全一致性与并行一致性。如果存在和 L_R 并行的写操作,根据定义 6,满足安全一致性原则;如果不存在与其读写结果相关的写操作,则说明违反了并行一致性原则。违反一致性查找算法如算法 2 所示。

算法 2　违反一致性查找算法

```
INPUT:历史记录图 G=(V,E,C)
OUTPUT:违反安全一致性读操作的集合 Scp,违反并行一致性读操作的集合 Spcp
1. 定义并初始化 Scp 与 Spcp
2.    V←getVertex(G)
3.    V1←V.orderByTimeAsc()//将节点进行 Asc 排序
4.    Len←|V1|
5.    Index[Len-1]={0,1,2,…,Len-1}//定义并初始化数组 Index
6.    for i=0 to Len-1 do
7.       if vi.ot==R//如果该节点所对应的操作是读
8.       V[Index[i]]←V1[j]//V[Index[i]]与 V1[j]对应
9.       lt←getLastTime(V1[j]∈{v|(v‖V1[j])})//最早结束时间
10.         findV←0//当前节点的前驱是否存在相关的写操作
11.         for k=0 to j do
12.           if V1[k].et>lt
13.              if V1[k].v==V[i].v and V1[k].ot==R
14.                 findV←1
15.              end if
16.              lt←getMax(lt,V1[k].st)
17.           end if
18.        if f(vi)==0 and findV==0
19.           Scp←Scp+vi
20.           Spcp←Spcp+vi
21.        else if g(vi)==0 and findV==0
22.           Scp←Scp+vi
23.        end if
24.    for each V[j]∈getVertex(G) do
25.       if V[Index[i]].et>V[Index[j]].et
26.          Index[j]>Index[i]
27.       end if
28.    end for
29. end for
30. return Scp and Spcp
```

在算法 2 第 1 行的排序操作中,如果采用时间复杂度最优的快速排序或堆排序,则时间复杂度为 $O(n*\log n)$;第 5 行初始化数组时间复杂度为 $O(n)$;第 11 行开始的内层 for 循环由最大并行参数 m 决定,第 6 行开始的外层 for 循环的基本复杂度为 $O(n)$,叠加两层嵌套 for 循环的时间复杂度为 $O(n*m)$。因此,与算法 1 一样,算法 2 的时间复杂度为 $O(n(\log n+m))$。

5　实验及结果分析

5.1　实验环境及配置

本实验中,集群节点规模数为 8,其中节点操作系统的版本是 Ubuntu16.04,CUP 版本是 Intel i7 6700 K,内存 16 GHz,SSD 大小为 250 GB,实验采用的 Key-Value 数据库是版本 3.11 的 Cassandra,在 Key-Value 数据库中,Value 支持多数据类型(如数字、字符串 String、链表 list、集合 set 等),将系统中副本系数设置为 3,测试工具方面采用 Yahoo Cloud Serving Benchmark(YCSB)来产生对 Cassandra 的并行读写操作,详细配置如表 5 所列。

表 5　实验环境描述

Table 5　Description of experimental environment

项目	描述
操作系统	Ubuntu16.04
Java 版本	1.6 for Linux
节点 CUP	Intel i7 6700K LGA 1151 4GHz 8M
节点内存	Kingston DDR4 2400 8G×2
节点硬盘	SEAGATE 2TB 7200 64M SATA3 ST2000DM001
SSD	SAMSUNG 850 EVO 250G M.2 SSD
显卡配置	NVIDIA GeForce GTX 1080 Founders Edition 1607/1733 MHz 8GB 256 bit
网卡信息	Realtek RTL8168/8111 PCI-E Gigabit Ethernet NIC-100Mbps
Cassandra 版本	Version 3.11
Replication_Factor	3
YCSB 版本	0.10.0

Cassandra 中,数据一致性表示数据行在各 Replicas 节点上更新和同步的程度。为了适应不同用户的个性化需求,针对任何的读写操作,用户(客户端)都可以根据运行时间与数据一致性(准确性)的要求来配置一致性需求。常用的读写一致性的配置主要有 QUORUM,LOCAL_QUORUM,ONE 及 ALL,其含义如表 6 所列。

表 6　Cassandra 数据一致性配置项的对比

Table 6　Comparison of Cassandra data consistency configuration item

配置项	写操作一致性	读操作一致性
QUORUM	至少成功写入 $R/2+1$ 个节点	在 $R/2+1$ 个复制节点返回数据后,返回具有最新时间戳的记录给客户端
LOCAL_QUORUM	至少在 coordinator node 所在的当前 DC 成功写入 $R/2+1$ 个复制节点	在 coordinator node 所在的 $R/2+1$ 个复制节点返回数据后,返回具有最新时间戳的记录给客户端
ONE	至少成功写入一个复制节点	从最近的复制节点返回结果(由 snitch 决定)
ALL	成功写入集群中的每个复制节点(R)	在集群中的每个复制节点返回数据后,返回具有最新时间戳的记录给客户端

本实验中采用的编程语言是 Java,由于实验需要,将 Cassandra 的读写一致性都设置为 ONE,其中 Java 写入 Cassandra 配置的代码为:

```
QueryOptions queryOptions=new QueryOptions();
queryOptions.setConsistencyLevel(ConsistencyLevel.ONE);
```

5.2　算法功能实现对比

为了验证本文算法的正确性及运行效率,我们将本文算法与文献[19]提出的算法进行对比。两种算法图的构建及相关基本运算操作都基于 Java 的图论算法包 JGraphT 的 1.3.0

版本，JGraphT 中提供了创建图、修改图、删除图、顶点和边的管理、图的遍历、连通性检查等功能。为了更好地模拟真实的应用环境，本文利用 YCSB 产生工作负载，设置读/写操作的比例服从[0.4，0.6]区间的随机分布，每个读写操作 K-V 中的 value 值取 100 KB 到 150 KB 之间的随机值，每次读写操作 K-V 对的数量取[100，200]之间的随机数；另外，通过调整客户端请求的线程数量，可以模拟出不同并行用户数。

设置不同的并行参数 m，让 m 以 2 的指数形式递增；平均每次检测长度取[800，1 000]之间的随机值（即设置 LS 集合的大小为 800 到 1000 的随机值），设置检测跟踪的 K-V 值数量为 100 时，两种算法最终对安全一致性与并行一致性的检测结果如表 7 所列。需要注意的是，文献[19]中的算法是通过判断 DAG 是否有环，来判断数据是否违反一致性原则，并不能精确地统计出违反一致性原则的次数。因此，实验过程中，本文将文献[19]中的算法生成的 DAG 图作为输入，根据本文定义 6 与定义 7 中的规则，来进行违反一致性规则的查找，扩展后与本文算法的实验结果进行对比。

表 7 不同并行参数下两种算法的 SCP 与 PCP 的数量对比

Table 7 Comparison of number of SCP and PCP of two algorithms with different parallel parameters

m 值	本文算法		文献[19]的算法	
	SCP	PCP	SCP	PCP
1	1	1	1	1
2	3	3	3	3
4	5	6	5	6
8	12	14	12	14
16	43	48	43	48
32	68	77	68	77
64	98	99	98	99
128	100	100	100	100
256	100	100	100	100

对比表 7 中的实验结果可以发现，当两种算法的并行参数相同时，找出的违反 SCP 与 PCP 一致性原则的操作数相同，证明了两种算法功能上的一致性。需要注意的是，由于 LS 集合的大小为 800 到 1 000 的随机值，但两种算法实验中的 LS 集合大小需要相同，因此 LS 集合只随机生成一次，并作为两种算法共同的输入。另外，随着参数 m 值的不断上升，本文与文献[19]中的算法违反 SCP 与 PCP 的概率都在不断增大，当并行度达到某个极端值（大于 64 的值）时，被跟踪的 100 个 K-V 值数据的操作记录历史都违反了一致性原则。出现以上现象的原因是，在设置 Cassandra 的读写一致性为 ONE 后，由于 ONE 进行写操作时的一致性要求并不严格（至少成功写入一个复制节点），随着并行参数 m 值的增大，大量的并行读操作读到旧数据、脏数据或非一致数据的概率也在增加，因此，并行参数 m 增大时，违反一致性原则的操作历史数量也在增加。

图 3 给出了相同并行数下，违反安全一致性（SCP）与并行一致性（PCP）数量的对比。可以发现，并行参数 m 相同时，违反并行一致性操作历史记录的数量可能大于违反安全一致性操作历史记录的数量。根据定义不难发现，由于满足并行一致性的同时也满足安全一致性，并行一致性的检查条件相比安全一致性更为严格，因此违反并行一致性的操作历史记录数量是大于或等于违反安全一致性数量的。

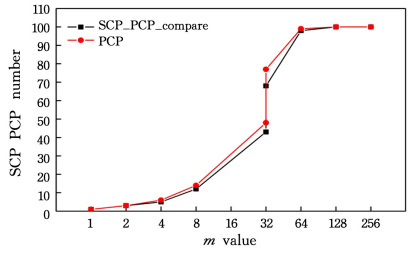

图 3 SCP 与 PCP 的实验结果对比

Fig. 3 Comparison of experimental results on SCP and PCP

5.3 算法性能对比

为了对比两种算法在不同条件下的性能差异，我们将两种算法在不同检测长度条件下进行了对比，操作记录 LS 集合的大小从 500 开始，每次递增 500，直到 10 000 时结束。两种算法的运行时间对比结果如图 4 所示。

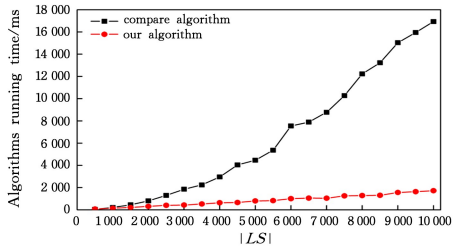

图 4 不同检测长度下两种算法的运行时间对比

Fig. 4 Comparison of running time of two algorithms with different test lengths

从图 4 可以看出，在检测长度（操作集合的大小）小于 1 500 时，两种算法的操作时间差距不大，都能控制在 1 000 ms 之内，但随着检测长度的不断增加，文献[19]提出的算法的增长率明显比本文算法高。特别地，当检测长度为 10000 时，文献[19]的算法的运行时间长达 17 s，而本文算法只用了 1.9 s，这是因为本文算法的时间复杂度为 $O(n(\log n+m))$，而文献[19]中算法的时间复杂度为 $O(n^2)$。

由于算法的复杂度为 $O(n(\log n+m))$。为了测试并行参数 m 对运行时间的影响，设置了操作记录 LS 集合的大小从 500 开始，每次递增 500，直到 10 000 结束，共 20 组测试集。然后设置并行参数 m 从 1 开始，以 2 为指数的形式增长到 64。其实验结果如图 5 所示。

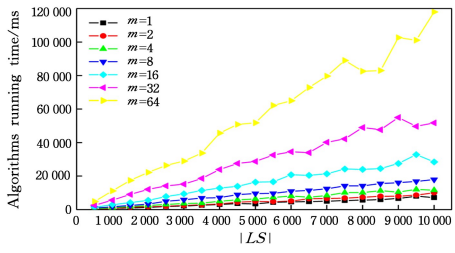

图 5 并行参数 m 对运行时间的影响

Fig. 5 Effect of parallel parameter m on running time

从图 5 可以发现，当 LS 集合大小固定时，随着并行参数 m 值的增大，运行时间延长。另外，当并行参数 m 值固定时，运行时间的增长率基本上固定，运行时间与 LS 集合大小呈

线性关系。在实际的应用场景中,并行度 $m=32$ 或 $m=64$ 已经是比较极端的并行情况,通过图 5 可以看出本文提出的算法在性能上能够满足分布式 Key-Value 数据库系统的性能要求。

结束语　分布式数据库系统中,在采用数据副本复制机制来提高系统可靠性及性能的同时,也导致了数据副本管理的一致性问题。数据一致性的实现既需要一致性协议(模型)来保障,也需要一致性检测算法进行辅助。及时地检测出系统中已经产生的不一致问题,能够为后期的一致性数据修复打下基础。当前,已有研究主要集中在一致性协议与模型的研究上,针对一致性检测算法的相关研究较少。本文首先对读写操作记录之间的时序关系、安全一致性及并行一致性原则等概念进行定义;在此基础上,研究了操作记录集合中读写操作之间的并行与时序关系,提取出操作记录集合向操作记录图转化的规则并设计了操作记录向历史记录图的转化算法;以历史记录图为输入,设计了违反一致性查找算法,实现了查找并返回图中所有违反安全与并行一致性读操作的集合;最后,通过与文献[19]的算法的对比,证明了本文算法在功能与效率方面的优越性。下一步工作主要集中在以下 3 个方面:1)改进当前算法的计算效率,即研究当操作记录集合数据量较大时,更为高效的一致性检测算法;2)研究针对操作记录集合 LS 操作的配套算法,实现对操作记录集合规模的控制、操作记录起始点的优化选择以及对非并行操作的删减等功能;3)在一致性检测算法的基础上,研究更加高效的非一致性数据修复算法。

参 考 文 献

[1] LI J Z, WANG H Z, GAO H. State-of-the-Art of Research on Big Data Usability[J]. Journal of Software, 2016, 27(7): 1605-1625.

[2] GILBERT S, LYNCH N. Perspectives on the CAP Theorem [J]. Computer, 2011, 45(2): 30-36.

[3] LAMPORT L. Fast Paxos[J]. Distributed Computing, 2006, 19(2): 79-103.

[4] BENZ S, SOUSA L P D, PEDONE F. Stretching multi-ring Paxos[C]// ACM Symposium on Applied Computing. New York: ACM, 2016: 492-499.

[5] OKI B M, LISKOV B H. Viewstamped Replication: A New Primary Copy Method to Support Highly-Available Distributed Systems[C]// ACM Symposium on Principles of Distributed Computing. New York: ACM, 1988: 8-17.

[6] EL-SANOSI I, EZHILCHELVAN P. Improving ZooKeeper Atomic Broadcast Performance by Coin Tossing[C]// European Workshop on Performance Engineering. Springer, 2017: 249-265.

[7] SONG M, LUO G, HAIHONG E. A Service Discovery System based on Zookeeper with Priority Load Balance Strategy[C]// IEEE International Conference on Network Infrastructure and Digital Content. Piscataway, NJ: IEEE, 2017: 117-119.

[8] JUNQUEIRA F P, REED B C. The life and times of a zookeeper [C]// ACM Symposium on Principles of Distributed Computing. New York: ACM, 2009: 1-4.

[9] BECKER M Y, SEWELL P. Cassandra: flexible trust management, applied to electronic health records[C]// Computer Security Foundations Workshop. Piscataway: IEEE, 2004: 139-154.

[10] MALIK P, MALIK P. Cassandra: a decentralized structured storage system[C]// ACM SIGOPS Operating Systems Review. New York: ACM, 2010: 35-40.

[11] WANG B Q, YU Q, LIU X, et al. Efficient and dynamic data management system for cassandra database[J]. Computer Science, 2016, 43(7): 197-202.

[12] VORA M N. Hadoop-HBase for Large-Scale Data[C]// International Conference on Computer Science and Network Technology. Piscataway, NJ: IEEE, 2012: 601-605.

[13] FENG S C, CAO B, CHAO D W, et al. Hash Synopsis Forest Index Schema Based on Hbase[J]. Journal of Chinese Computer Systems, 2018, 39(1): 100-104.

[14] BOYD S, GHOSH A, PRABHAKAR B, et al. Randomized gossip algorithms[J]. IEEE Transactions on Information Theory, 2006, 52(6): 2508-2530.

[15] HAAS Z, HALPERN J Y, LI L. Gossip-based ad hoc routing [J]. IEEE/ACM Transactions on Networking, 2006, 14(3): 479-491.

[16] BOHANNON P, FAN W, GEERTS F, et al. Conditional Functional Dependencies for Data Cleaning[C]// IEEE, International Conference on Data Engineering. Piscataway, NJ: IEEE, 2007: 746-755.

[17] FAN W, GEERTS F, JIA X. Improving data quality: consistency and accuracy[J]. Pakistan Journal of Biological Sciences, 2007, 7(6): 315-326.

[18] ZHANG A Z, MEN X Y, WANG H Z, et al. Hadoop-Based Inconsistence Detection and Reparation Algorithm for Big Data. Journal of Frontiers of Computer Science and Technology, 2015, 9(9): 1044-1055.

[19] ANDERSON E, LI X, SHAH M A, et al. What consistency does your key-value store actually provide? [C]// Proc of Workshop on Hot Topics in System Dependability. New York: ACM, 2010: 1-16.

[20] COLOMBO D, MAATHUIS M H. Learning high-dimensional directed acyclic graphs with latent and selection variables[J]. Annals of Statistics, 2012, 40(2012): 294-321.

[21] WANG Y, CAO S, GUO H, et al. Communication aware multiple directed acyclic graph scheduling considering cost and fairness[J]. Journal of Computer Applications, 2015, 37(3): 316-321.

[22] BING S, ZHEN Z, BING W, et al. Scene Segmentation with DAG-Recurrent Neural Networks[J]. IEEE Transactions on Pattern Analysis & Machine Intelligence, 2018, 40(6): 1480-1493.