基于共享图和部分复制策略的分布式存储因果一致性模型

田俊峰[1,2]，杨万贺[1,2]，庞亚南[1,2]，张俊涛[1,2]
（1. 河北大学网络空间安全与计算机学院，河北 保定 071002；2. 河北省高可信信息系统重点实验室，河北 保定 071002）

摘 要：针对目前因果一致性模型中存在的元数据传播开销大、操作时延、远程更新可见时延高等问题，提出了基于共享图和部分复制策略的分布式存储因果一致性模型。该模型以共享图拓扑结构为基础，每个数据中心存放完整数据集的子集，同时，提出了共享稳定向量与混合逻辑时钟相结合的全局稳定策略，在保证因果关系的前提下，实现数据中心间的数据一致性。理论分析和仿真实验结果表明，与现有模型相比，所提模型在降低操作时延的同时，可有效地权衡远程更新可见性能和元数据开销。

关键词：数据一致性；因果一致性；共享图；部分复制策略；全局稳定策略
中图分类号：TP393
文献标识码：A
doi: 10.11959/j.issn.1000−436x.2020079

Causal consistency model for distributed data store based on shared graph and partial replication strategy

TIAN Junfeng[1,2], YANG Wanhe[1,2], PANG Ya'nan[1,2], ZHANG Juntao[1,2]
1. School of Cyber Security and Computer, Hebei University, Baoding 071002, China
2. Key Laboratory on High Trusted Information System in Hebei Province, Baoding 071002, China

Abstract: In order to solve the problem of metadata propagation overhead, operation delay and remote update visibility latency in the current causal consistency model, a causal consistency model for distributed data stores based on the shared graph and partial replication strategy was proposed. This model was based on the topology of the shared graph, and each data center stored an arbitrary subset of the data. At the same time, the global stabilization strategy combining shared stable vector and hybrid logical clocks was proposed to provide data consistency guarantees on the premise of ensuring causality. The theoretical analysis and experimental results show that the proposed model can effectively balance the remote update visibility and the metadata overhead compared with the existing models while reducing the operation delay.
Key words: data consistency, causal consistency, shared graph, partial replication strategy, global stabilization strategy

1 引言

云存储是当今互联网服务的重要组成部分[1]。为了给用户提供高效率和可扩展的服务，云存储采取分布式数据存储方式，将数据中心部署在不同位置实现大数据存储，数据中心之间的数据一致性成为保证数据可用的关键[2]。

数据一致性一般分为强一致性和弱一致性[3]。在理想情况下，对任何数据的更新在其所有副本中立即可见称为强一致性。强一致性虽然有简单的语义，但是引发了高时延和不允许网络分区现象；多个节点的状态没有充分同步但仍可以处理读写操作称为弱一致性。弱一致性对数据的及时更新没有要求，难以为用户提供最新的数据[4]。因果一致性是介于强一致性和弱一致性中间的协议[5]，既可以有效解决高时延和网络分区等问题，也能保证数据

收稿日期：2020−01−03；**修回日期**：2020−03−24
通信作者：杨万贺，ywh_hbu@163.com
基金项目：国家自然科学基金资助项目（No.6180060654）
Foundation Item: The National Natural Science Foundation of China (No.6180060654)

的及时更新。

目前，基于因果一致性协议的模型成为构建地理复制数据存储中有吸引力的模型[6]。在地理复制数据存储中，分布在不同地理位置的数据中心通过网络时钟协议（NTP, network time protocol）更新当前的节点时钟，不同数据中心利用数据复制策略完成数据同步[7]。Agrawal 等[8]对分布式存储系统中的数据复制策略进行了分类，详细阐述了地理复制模型中的数据一致性方法，数据复制策略分为完全复制策略和部分复制策略。

基于完全复制策略的因果一致性模型要求每个数据中心都存储完整的数据集，本地数据中心写入数据需要向其他所有远程数据中心同步数据[9]。例如，文献[10]模型在完全复制策略的基础上，仅依靠单一标量时间戳实现对因果关系的追踪。文献[11]提出通用稳定向量和混合逻辑时钟相结合的全局稳定策略，提高了模型的吞吐量。文献[12]提出数据中心稳定向量和混合逻辑时钟相结合的全局稳定策略，解决了操作时延问题，降低了数据的更新可见时延。但数据中心稳定向量中的条目包含所有数据中心，增加了元数据开销，数据中心之间的同步开销大。

随着数据中心存储的数据量迅速增长，为降低数据中心之间的同步开销，基于部分复制策略的因果一致性模型得到了广泛应用[13]。基于部分复制策略的因果一致性模型要求每个数据中心存储完整数据集的子集，本地数据中心写入数据仅需向部分远程数据中心同步新数据。文献[14]模型中的每个数据中心都设置序列化器，利用时间戳对操作进行序列化，实现了操作的高并发性，并降低了操作时延。但是，该模型取消了数据中心之间的全局稳定。文献[15]模型利用共享树表示数据中心之间的拓扑关系和部分元数据作为标签，数据中心之间的所有更新操作都通过共享树和标签实现，提高了数据的可见性和系统的吞吐量，但是造成了数据中心之间的元数据传播开销。Xiang 等[16]提出的模型是对数据中心之间的全局稳定方法进行扩展，利用全局稳定时间戳和物理时钟的全局稳定策略实现模型的全局稳定性。该模型降低了元数据开销，提高了系统的吞吐量，但会产生操作时延，从而造成较高的更新可见时延。

为了有效降低操作时延并权衡元数据传播开销和远程更新可见性能，本文提出一种基于共享图和部分复制策略的分布式存储因果一致性模型（CCSGPR, causal consistent model for distributed data stores based on shared graph and partial replication strategy），利用共享图表述数据中心之间的拓扑关系，通过共享稳定向量和混合逻辑时钟相结合的全局稳定策略实现因果一致性。本文主要工作如下。

1) CCSGPR 使用部分复制策略实现分布式数据存储，每个数据中心存放完整数据集的子集，利用共享图表示数据中心之间的邻接关系，元数据通过同步消息和心跳信息传播，仅需在有邻接关系的部分数据中心传播，降低元数据传播开销和数据远程更新可见时延。

2) 提出一种全局稳定策略，该策略使用混合逻辑时钟标记操作的时间戳，可为新数据标记满足因果依赖关系的混合逻辑时间戳，避免写入数据操作时延。同时，在混合逻辑时钟的基础上提出了共享稳定向量，该向量仅由 2 个时间戳构成，这降低了元数据开销，通过选取远程数据中心中时间戳的最小值为远程更新稳定的界限，避免读取数据操作时延。

2　背景介绍

2.1　基于部分复制策略的分布式模型

分布式模型采取多版本键值存储方式，一个键对应一个版本链，版本链包含键的多个版本，每个版本又包括键对应的值和元数据，写（PUT）操作会为键的版本链增加一个新版本，模型定期处理版本链中的旧版本[17]，数据存放在不同的数据中心，用户可以在不同的数据中心完成数据的写入、更新或下载[18]。基于部分复制策略的分布式模型如图 1 所示。由图 1 可知，完整数据集被分为 m 个子集，每个数据中心存储完整数据集的子集；每个数据中心由 n 个分区（服务器）构成，所存储的数据子集根据哈希函数按键分配到 n 个分区。服务器完成写入数据操作后，为对应的键创建一个新版本，新版本仅需同步到部分数据中心，数据中心之间通过点对点 FIFO（first in first out）通道进行通信。

基于部分复制策略的分布式模型包含以下 2 个基本操作。

1) PUT→(k, v)。PUT 操作将值 v 赋给键 k 标识的数据。若键 k 标识的数据存在，创建一个值为 v

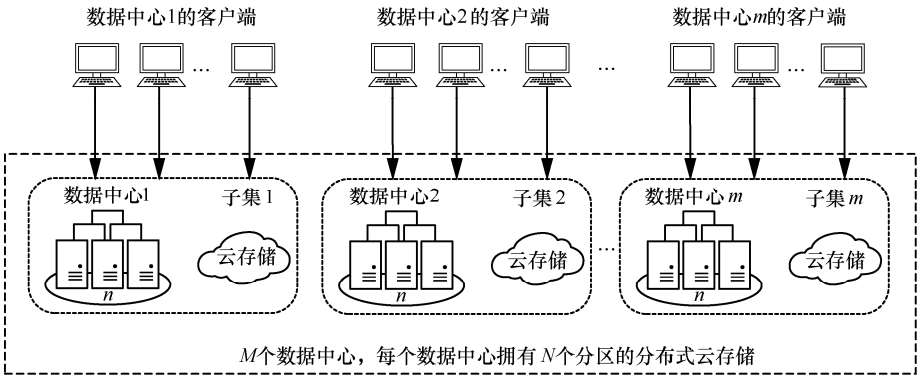


图 1　基于部分复制策略的分布式模型

的新版本；否则，创建一个键为 k 且初始值为 v 的新数据。

2) val←GET（k）。GET 操作返回键 k 标识的数据的值 v，返回的值 v 需满足因果一致性。

2.2　因果一致性

因果一致性根据 2 个操作之间的 happens-before 关系[19]定义，要求服务器返回的值与因果关系定义的顺序一致。对于 2 个任意操作 a_1 和 a_2，若满足以下条件，称 a_1 因果依赖于 a_2，记作 $a_1 \rightarrow a_2$。

1) a_1 和 a_2 是同一线程中的操作，a_1 发生在 a_2 之前。

2) a_1 为写数据操作，a_2 为读数据操作，且 a_2 读 a_1 写的值。

3) 存在其他操作 a_3，其中 $a_1 \rightarrow a_3$，$a_3 \rightarrow a_2$。

定义 1　数据稳定性。若键 k 标识的数据可以被客户端 c 读取，且该数据的依赖项也可被客户端 c 读取，则称该数据具有稳定性。

定义 2　全局稳定性。分布式存储模型划定一个时间戳界限，该时间戳范围内的所有数据都具有稳定性，则称分布式存储模型在该时间戳具有全局稳定性。

定义 3　更新可见时延。键 k 标识的数据在其原始数据中心写入的时刻到该数据在此数据中心具有稳定性的时刻间隔称为更新可见时延。

定义 4　并发操作。存在 2 个操作 a_1 和 a_2，若 a_1 既不因果依赖于 a_2，a_2 也不因果依赖于 a_1，则称 a_1 和 a_2 为并发操作。

定义 5　冲突操作及其处理规则。若 a_1 和 a_2 是对键 k 标识的同一数据进行并发写操作，则称 a_1 和 a_2 为冲突操作。冲突操作造成键 k 标识的同一数据产生 2 个相互冲突的版本，相互冲突的版本按照不同的顺序同步到远程数据中心，这会造成数据中心之间的不一致性问题。基于部分复制策略的分布式模型使用最后更新为准（last-writer-wins）规则[20]处理数据的冲突操作。该规则规定，对于键 k 标识的同一数据的 2 个写入操作，规定时间戳大的操作为后发生的，从而写入的值为时间戳大的操作所确定的值，若 2 个操作的时间戳相等，则通过数据的写入数据中心 ID 来解决冲突。

3　共享图

共享图（shared graph）是无权无向图，由顶点的有穷非空集合和顶点之间边的集合组成，记作 $G^s=(V^s,E^s)$，其中 $V^s=\{1,2,3,\cdots,n\}$ 为顶点集合；E^s 为边的集合，包含实线边和虚线边 2 种，实线边集合记作 $E_1(G^s)$，虚线边集合记作 $E_2(G^s)$。CCSGPR 使用共享图表述分布式结构，顶点 $i \in V^s$ 代表数据中心 i；若数据中心 i 和数据中心 j 存在共享键，则存在实线边 $(i,j) \in E_1(G^s)$，若存在客户端 c 可以访问数据中心 i 和数据中心 j，则存在虚线边 $(i,j) \in E_2(G^s)$。

N_i 表示与数据中心 i 存在实线连接的数据中心集合；P_i 表示与数据中心 i 存在虚线连接的数据中心集合；K_i 表示存储在数据中心 i 的键的集合；$K_{ij}=K_i \cap K_j$ 表示数据中心 i 和数据中心 j 的共享键。数据中心邻接关系的共享图如图 2 所示。由图 2 可知，共享图 G^s 由 i、V_1、V_2、V_3、V_4、V_5 顶点组成，2 个数据中心的共享键标记在图中的实线边上，存在客户端 c 可以访问数据中心 i、V_2、V_3 和 V_5，因此，顶点 i、V_2、V_3 和 V_5 之间存在虚线边，如 $N_i=\{V_1, V_4\}$，$P_i=\{V_2,V_3,V_5\}$，$K_{V_1V_2}=\{k_2\}$，$K_{V_1V_5}=\varnothing$。

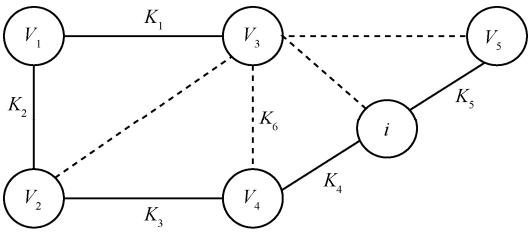


图 2　数据中心邻接关系的共享图

在部分复制数据存储中，每个数据中心只存放完整数据集的任意子集，本文利用共享图表述数据中心之间的邻接关系，取消数据中心之间额外的同步开销，本地数据中心 i 写入新数据仅需向 N_i 集合中其他远程数据中心发送数据同步消息和心跳信息，若数据中心 i 存储的子集不包含待读取的数据，则向 P_i 集合中其他数据中心发送迁移操作。元数据利用数据同步消息和心跳信息实现传播，通过共享图拓扑结构，元数据不需要传播到所有的数据中心，同时，允许不同数据中心的服务器之间直接传播数据同步消息和心跳信息，降低元数据同步开销，以提供高吞吐量并降低远程更新可见时延。

4　全局稳定策略

目前，基于部分复制策略的因果一致性模型取消了传统的依赖项检查信息，通过 NTP 实现服务器的同步，并利用全局稳定时间戳（GST, global stable timestamp）和物理时钟相结合的全局稳定策略实现模型的全局稳定性（见 2.2 节的定义 2）。

较目前的因果一致性模型，CCSGPR 提出了一种共享稳定向量和混合逻辑时钟相结合的全局稳定策略。首先，CCSGPR 用混合逻辑时钟（HLC, hybrid logical clock）代替物理时钟来跟踪时间的进展，完成时间戳的更新。混合逻辑时钟结合了物理时钟和逻辑时钟[21]，其时间戳 t 由一个物理组件 t.p 和一个逻辑组件 t.l 组成，记作<t.p, t.l>，其中 t.p 是操作发生的物理时钟值，t.l 是追踪因果关系的计数器。因此，混合逻辑时钟充分发挥了两者的优势，既包含物理时钟自发递增的优点，也包含逻辑时钟易追踪因果关系的优点。在分布式模型中，由于距离差异等原因会造成节点之间存在一定的时钟偏差，存在 2 个 PUT 操作 e_1 和 e_2，若 e_1“happens-before” e_2，则 e_2 因果依赖 e_1，受时钟偏差的影响，e_2 的时间戳小于 e_1 的时间戳。为保证上述 PUT 操作满足因果性，使用物理时钟的因果一致性模型发生操作阻塞，等待 e_2 的物理时钟超过 e_1 的时间戳，造成 PUT 操作时延。CCSGPR 中 e_2 不需要阻塞等待，直接完成操作并为 e_2 标记 $t_{e_1}.p<t_{e_2}.p$ 或 $t_{e_1}.p=t_{e_2}.p \wedge t_{e_1}.l<t_{e_2}.l$（其更新算法详见 5.2 节）的混合逻辑时间戳，在保证因果性的前提下，有效地避免了写入操作时延。

其次，记某数据的写入时间戳为 t，因果一致性模型全局稳定的界限为 T，规定仅当 $t<T$，该数据可读（见 2.2 节的定义 1）。CCSGPR 使用共享稳定向量（SSV, shared stable vector）代替全局稳定时间戳来划定模型全局稳定的界限。共享稳定向量由 2 个元素组成，记作 SSV=$\{t_1,t_2\}$，其中 t_1 代表本地数据中心稳定的界限，t_2 代表远程数据中心稳定的界限。其 2 个时间戳的选取原则为：t_1 选取本地数据中心 i 中最新数据的时间戳，t_2 选取共享图 N_i 集合中所有远程数据中心时间戳的最小值，即规定本地数据中心更新的数据立即可见，而远程数据中心更新的数据需等待 N_i 集合中所有远程数据中心同步完成更新。HB_{xy} 代表数据中心 x 向数据中心 y 发送的心跳消息值，共享稳定向量中的 t_2 为一组心跳值中的最小值，即 $SSV_i(t_2)=\min\limits_{g\in N_i} HB_{gi}$。CCSGPR 中数据中心之间的心跳同步如图 3 所示。由图 3 可知，本地数据中心最新数据的时间戳为 a，接收远程数据中心 1 的心跳消息时间戳为 b，接收远程数据中心 2 的心跳消息时间戳为 c，接收远程数据中心 3 的心跳消息时间戳为 d，则 SSV=$\{a, \min(b, c, d)\}$。一个数据中心内的分区互相定期交换其分区向量（PV, partition vector），作为该数据中心更新 SSV 的依据，详见 5.2 节。

CCSGPR 在共享图拓扑结构的基础上，利用混合逻辑时钟有效避免了写操作时延，同时，共享稳定向量仅包含 2 个时间戳，其根据选取最小值的原则，避免了读取操作时延，降低了元数据开销和数据中心之间的同步开销。

5　CCSGPR 协议

CCSGPR 协议是在分布式存储中稳定运行的通信协议，在操作满足因果一致性要求的前提下，为用户提供安全快速的写入、查询和存储服务。

5.1　元数据

元数据是描述数据属性的信息，用来支持历史数据、资源查找、文件记录、指示存储位置等功能，元数据开销和效率是衡量系统性能的重要指标，如

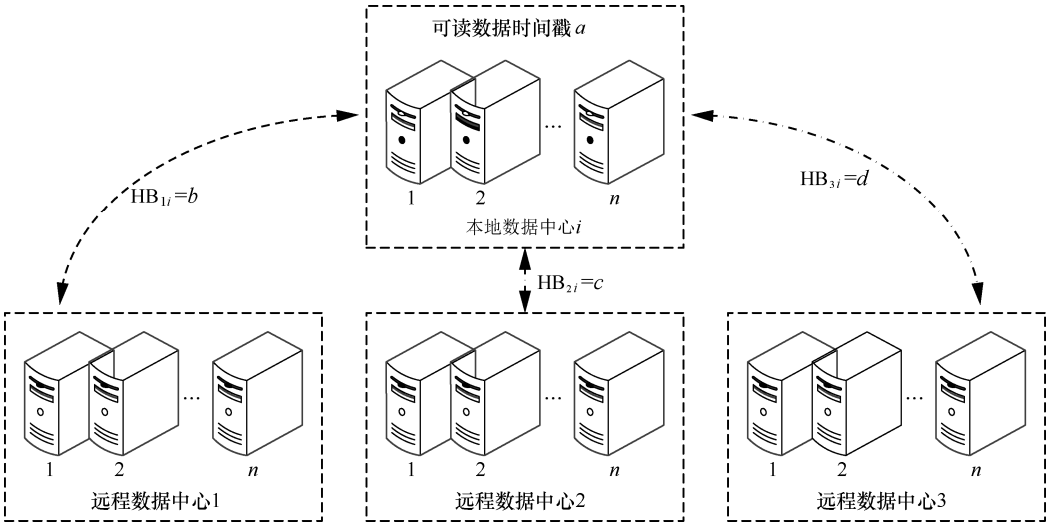


图 3 CCSGPR 中数据中心之间的心跳同步

Facebook 中，元数据比数据本身大，大型元数据会增加通信和存储开销[22]。分布式模型中，采取多版本键值存储方式，用 d 表示元组，该元组包括<k, v, ut, sr>，其中，k 是标识键唯一的 ID；v 是键对应的值；ut 是键的更新时间，即该数据的写入时间戳；sr 是 d 的源副本，即创建 d 的数据中心 ID。

客户端 c 维持 SSV_c，SSV_c 仅由 2 个时间戳组成。客户端 c 还维持依赖向量（DV, dependency vector）DV_c，存储远程数据中心的依赖项，以保证后续操作满足因果一致性。

服务器 p_n^m 为数据中心 m 中的第 n 个分区，服务器 p_n^m 中配备单调递增的物理时钟，记作 $clock_n^m$。服务器 p_n^m 中的 DV 与客户端 c 中的 DV_c 类似，存储依赖项；SSV_n^m 为服务器存储的全局稳定向量；PV_n^m 为分区向量，$PV_n^m[i]$为从数据中心 i 中的服务器接收到的最新同步消息/心跳消息的时间戳，其中 $i \neq m$，$PV_n^m[m]$为服务器 p_n^m 写入数据版本的最大时间戳。

5.2 操作

CCSGPR 协议中包含 6 个基本操作：GET 操作、PUT 操作、数据同步操作、更新共享稳定向量操作、心跳机制操作和迁移操作。

5.2.1 GET 操作

客户端 c 发起读取数据操作，存储该数据的服务器 p_n^m 响应操作，如算法 1 所示。其具体流程如下，客户端 c 发起<GET k, SSV_c>请求，其中 k 为待读取数据的键，服务器 p_n^m 接收到请求后，为确保安装最新快照使用 SSV_c 更新 SSV_n^m。服务器 p_n^m 选择键 k 的版本链中最新数据的版本，其中版本是本地数据中心写入或远程数据中心写入且更新时间戳小于 SSV_n^m 中的时间戳。服务器 p_n^m 把最新 SSV_n^m、键 k 对应版本的值 v、源副本 sr 和键 k 对应版本的更新时间戳 ut 以 GETREPLY 消息格式返回客户端 c。客户端 c 接收 GETREPLY 消息后，通过 maxDV 算法用更新时间戳 ut 和源副本 sr 更新 DV_c，用 SSV_n^m 更新 SSV_c。

算法 1 数据中心 m 中客户端 c 的操作

//GET 操作

1) function GET（key k）

2) send<GETREQ k, SSV_c> to server //发送读数据请求

3) receive<GETREPLY v, DV, SSV_n^m> //响应读数据请求

4) $SSV_c \leftarrow \max(SSV_c, SSV_n^m)$ //更新客户端 c 中的共享稳定向量

5) $DV_c \leftarrow \text{maxDV}(DV_c, \{<\text{sr, ut}>\})$ //更新依赖向量

//PUT 操作

6) function PUT（key k, value v）

7) send <PUTREQ k, v, DV_c> to server //发送写数据请求

8) receive<PUTREPLY ut, sr> //响应写数据请求

9) $DV_c \leftarrow \text{maxDV}(DV_c, \{<\text{sr, ut}>\})$ //更新依

赖向量算法

10) maxDV（dependency vector DV_1, dependency vector DV_2）

11) for each <*I*, *h*> ∈DV_2

12) if ∃ <*I*, *h'*> ∈DV_1 //DV_1存在条目<*I*, *h*>

13) DV_1←DV_1 – <*I*, *h'*>

14) DV_1← <*I*, max(*h*, *h'*) > //更新 DV_1

15) else //DV_1不存在条目<*I*, *h'*>

16) DV_1←DV_1∪{<*I*, *h*>}//DV_1中添加条目<*I*, *h'*>

17) end if

18) return DV_1

//迁移操作

19) function migrate（key *k*）to P_m

20) send migrate<*m*, SSV_c > to server from P_m

//向共享图 P_m集合中的数据中心发送迁移消息

21) receive<migratereply>

5.2.2 PUT 操作

客户端 *c* 发起写数据操作，服务器 p_n^m 响应写操作，如算法 2 所示。其具体流程为，客户端 *c* 发起<PUT *k*, *v*, DV_c>请求，其中 *k* 为待写入数据的键，*v* 为键对应的值，DV_c 为依赖向量，记录客户端 *c* 的依赖项。服务器 p_n^m 接收写数据请求后，用 DV_c 值更新SSV_n^m，使用 update clock 算法（算法 3 中的 update clock on PUT 算法）更新$PV_n^m[m]$，保证 PUT 操作满足因果关系。服务器 p_n^m 为数据 *k* 的版本链中创建一个新版本，该版本的键为 *k*，键对应的值为 *v*，键对应的更新时间戳 ut 为$PV_n^m[m]$，键对应的源副本 sr 为该数据中心的 ID。服务器 p_n^m 把最新SSV_n^m、键 *k* 对应的更新时间戳 ut 和源副本 sr 以 PUTREPLY 消息格式返回客户端 *c*。同理，客户端 *c* 接收 PUTREPLY 消息后，通过 maxDV 算法用更新时间戳 ut 和源副本 sr 更新 DV_c，用SSV_n^m更新SSV_c。

算法 2 服务器 p_n^m 响应客户端 *c* 的请求操作

//响应 GET 请求

1) Upon receive<GETREQ *k*, SSV_c> from *c* do

2) SSV_n^m ←max（SSV_c, SSV_n^m）

3) obtain latest version *d* from version chain of key *k*，the version with highest value <ut, sr> //选取键 *k* 的版本链中最新版本

d.sr = *m*，or any <*i*,*h*> ∈ *d*.DV, *h* ≤$SSV_n^m[i]$

4) DV ← maxDV（DV, {<sr, ut>}）

5) send<GETREPLY *d*.*v*, DV, SSV_n^m >to client

//向客户端返回读操作请求消息

//响应 PUT 请求

6)Upon receive <PUTREQ *k*, *v*, DV>from *c* do

7) SSV_n^m ←maxDV（SSV_n^m, DV）

8) dt←max value in {DV.values∪{$SSV_n^m[m]$}}

//标记时间戳 dt

9) Creat new item *d* //创建新的数据项 *d*

10) *d*.*k*←*k*; *d*.*v*←*v*; *d*.ut←*dt*; *d*.sr←*m*; *d*. DV← DV

11) insert *d* in the version chain of *k* //将新版本插入键 *k* 的版本链中

12) send <PUTREPLY *d*.ut, *m*> to *c* //向客户端返回写数据操作请求消息

13) for each server p_n^i, *i*∈N_m，*i*≠*m* do

14) send<Replicate *d*> to p_n^i //向共享图 N_m集合中的数据中心发送数据同步消息

//Replicate 算法

15)Upon receive <Replicate *d*>from p_n^i do //响应数据同步消息

16) Creat new item *d*

17) *d*.*k*←*k*; *d*.*v*←*v*; *d*.ut← $PV_n^m[m]$; *d*.sr ← *m*

18) insert *d* in the version chain of *k*

19) $PV_n^m[i]$←*d*.ut //用 ut 更新$PV_n^m[i]$

//响应迁移操作

20) Upon receive migrate<*m*, SSV_c>from p_n^i do, *i*∈P_m

21) SSV_n^m ←max（SSV_c, SSV_n^m）

22) send <migatereply>

5.2.3 数据同步操作

分布式存储系统由不同节点（数据中心）协同为用户提供存储、查询服务的平台，所以服务器 p_n^i 完成 PUT 操作后，其他数据中心要同步最新数据，即本地数据中心 *i* 写入新数据后，需向 N_i集合中的数据中心发送同步数据信息。服务器间的数据同步如图 4 所示。由图 4 可知，数据中心 *i* 中的服务器 p_n^i 发送 Replicate 消息到其他数据中心（算法 2 中的 Replicate 算法），N_i集合中的数据中心接收 Replicate

消息后，服务器 p_n^m 将数据的新版本增加到该数据的版本链中，更新其 PV 中服务器 p_n^i 对应的条目，即设置 $\mathrm{PV}_n^m[i]$=ut，完成服务器间的数据同步。

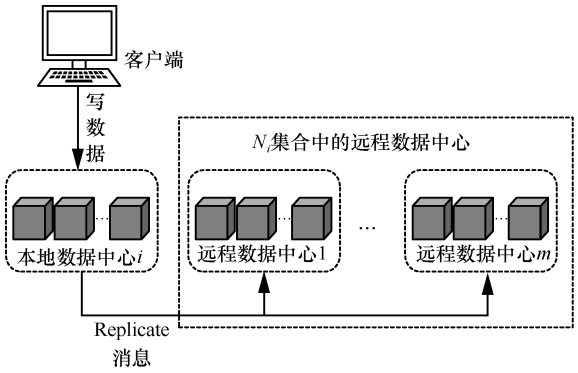


图 4 服务器间的数据同步

5.2.4 更新共享稳定向量操作

一个数据中心内的服务器定期更新全局稳定向量 SSV，规定每过 θ 时间间隔（θ 的具体时间在仿真实验中设置），服务器之间彼此共享其分区向量 PV，并计算 SSV 为所有服务器的分区向量 PV 中的最小值（算法 3 的更新共享稳定向量算法），从而确保该数据中心数据稳定界限内的数据在所有服务器内可读。

5.2.5 心跳机制操作

服务器 p_n^m 的时钟管理和心跳机制操作如算法 3 所示。若数据中心 m 中的服务器 p_n^m 未接收客户端 c 的更新数据请求，则不会向 N_m 集合中的数据中心发送 Replicate 消息，其分区向量 PV 和全局稳定向量 SSV 对应的值不能完成更新，从而后续操作无法满足因果一致性。为了避免这种情况发生，规定数据中心 m 中的服务器 p_n^m 超过Δt 的时间内（Δt 的具体时间在仿真实验中设置）未接收客户端 c 的更新数据请求，则将 $\mathrm{PV}_n^m[m]$时间戳广播到 N_m 集合中的其他远程数据中心，远程数据中心中的服务器 p_n^i 接收广播消息后，设置 $\mathrm{PV}_n^i[m]=\mathrm{PV}_n^m[m]$。

算法 3 服务器 p_n^m 的时钟管理和心跳机制操作

//PUT 更新时钟算法

1)function update clock on PUT (dt)

2) dt ← max{DV_c} //DV 中的最大元素标记 dt

3) $\max p \leftarrow \max\{\mathrm{PV}_n^m[m].p,\ \mathrm{clock}_n^m,\ \mathrm{dt}.p\}$ //记录最新的物理时间戳

4) if ($\max p == \mathrm{PV}_n^m[m].p == \mathrm{DV}_c[m].p$) then

5) $l=\max\{\mathrm{PV}_n^m[m].l,\ \mathrm{DV}_c[m].l\}+1$

//记录因果序

6) else if($\max p == \mathrm{PV}_n^m[m].p$) then $l=\mathrm{PV}_n^m[m].l+1$

7) else if($\max p == \mathrm{DV}_c[m].p$) then $l=\mathrm{DV}_c[m].l+1$

8) else $l=0$

9) end if

10) $\mathrm{PV}_n^m[m].p \leftarrow \max p$；$\mathrm{PV}_n^m[m].l \leftarrow 1$ //设置 $\mathrm{PV}_n^m[m]$

11) end function

//更新共享稳定向量算法

12)function update clock on SSV

13) $\mathrm{SSV}_n^m \leftarrow \min \mathrm{PV}_j^m,\ j \in \{1,2,\cdots,n\}$

//选取 PV 中的最小值更新 SSV

//心跳机制

14)upon every Δt time do

15) if there has not been any replicate message in the past Δt time //若Δt 时间内未接收数据同步请求，则按照心跳机制更新最新状态

16) update clock on Heartbeat()

17) for each server p_n^j，$j \in N_i$，$j \neq m$ do

18) send<Heartbeat $\mathrm{PV}_n^m[m]$> to p_n^j //向 N_i 集合中的数据中心 j 发送心跳消息

//心跳更新时钟算法

19)function update clock on Heartbeat

20) if（$\mathrm{clock}_n^m > \mathrm{PV}_n^m[m].p$）then

21) $\mathrm{PV}_n^m[m].p \leftarrow \mathrm{clock}_n^m$；$\mathrm{PV}_n^m[m].l \leftarrow 0$//设置 $\mathrm{PV}_n^m[m]$时间戳

22) end if

23)end function

//响应心跳信息

24)upon receiving <Heartbeat $\mathrm{PV}_n^m[j]$> from p_n^i do

25) $\mathrm{PV}_n^m[j] \leftarrow \mathrm{PV}_n^i[j]$ //响应心跳消息，更新 $\mathrm{PV}_n^m[j]$

5.2.6 迁移操作

如算法 2 所示，客户端 c 发起读取数据操作，数据中心 m 作为响应客户端 c 操作的本地数据中心，若数据中心 m 存储的数据子集不包含待读取的数据，则根据共享图表示的数据中心之间的邻接关

系，向 P_m 集合中其他数据中心发送迁移操作，完成读取数据操作。其具体流程为，数据中心 m 的服务器 p_n^m 向 P_m 集合中的数据中心发送迁移操作请求，包括数据中心的 ID 和 SSV_n^m；数据中心 i 中的服务器 p_n^i 响应迁移操作请求，更新其共享稳定向量，完成迁移操作。

6　仿真实验与结果分析

6.1　对比分析

本节对现有的因果一致性模型进行对比分析，表 1 为各因果一致性模型的比较。文献[10]模型和文献[11]模型基于完全复制策略，较文献[10]模型，文献[11]模型使用混合逻辑时钟与通用稳定向量相结合的全局稳定策略，降低了操作时延，但增加了元数据开销。文献[10]模型和文献[16]模型都使用物理时钟与全局稳定时间戳相结合的全局稳定策略，较文献[10]模型，文献[16]模型使用部分复制策略，有效降低了远程更新可见时延，但操作时延仍然较高。CCSGPR 基于部分复制策略，使用混合逻辑时钟与共享稳定向量相结合的全局稳定策略，在保证元数据开销小的基础上，有效降低了操作时延，定量对比结果详见以下实验内容。

6.2　仿真实验设置

CCSGPR 使用 Java 实现，采用 Berkeley DB 进行键值数据的存储和检索。Berkeley DB 数据库是以 key-value 为结构的嵌入式数据库，既有关系型数据库中的完整 ACID（Atomicity, Consistency, Isolation, Durability）事务语义支持，也提供 NoSQL 中简单的数据库编程接口。为验证本文所提方法的有效性，在数据一致性 HBU-Cluster 平台上实现分布式键值存储仿真实验，并与经典的基于完全复制策略文献[11]模型和基于部分复制策略文献[16]模型进行实验对比。数据一致性 HBU-Cluster 平台是项目组为因果一致性测试设计的仿真平台，该平台为分布式键值存储管理框架，使用 Google 的 Protocol Buffer 将数据因果一致性协议结构化，并集成 Yahoo 的 YCSB（Yahoo! Cloud Serving Benchmark）基准测试模块作为性能测试工具。仿真实验中服务器的配置是 Windows10 x64，Intel Core i5-4590，3.3 GHz，16.00 GB 内存，256 GB 固态磁盘存储。根据现有模型的性能测试，仿真实验采取传统的测试方法从模型吞吐量、操作响应时间和远程更新可见时延 3 个方面进行定性对比。

根据实际基于 Internet 的应用设置，在仿真实验中设置默认参数如下：数据中心数目为 4，分区数目为 6，读写比例为 3:1。客户端依据 uniform 记录选择策略访问分区数据，服务器之间采取 NTP 同步方法，每次实验前都同步时钟。CCSGPR 中的混合逻辑时间戳使用 64 位编码，其中 48 位设置为物理部分，16 位设置为逻辑部分，通过这些设置，混合时间戳可以编码多达 2^{16} 个逻辑事件，可以跟踪到微秒的物理时间，有效避免了由于逻辑部分已经达到最大值，但又必须增加其逻辑部分以保证因果一致性导致分区发生的阻塞等待。规定分区之间每 5 ms 交换分区向量 PV，若分区在 1 ms 内未接收更新数据操作和同步数据操作，则该分区接收心跳信息。

6.3　吞吐量

吞吐量是单位时间内模型更新键值的总量，是衡量模型性能的一项重要指标。本节实验采取分别增加数据中心和分区数目的传统实验方式评估模型的吞吐量性能。

首先，分析数据中心数对模型吞吐量的影响，控制分区变量，设置每个数据中心的分区数为 6，客户端向本地数据中心发送读写请求，数据中心之间定期同步数据。图 5(a)描述了数据中心数 2~12 所对应吞吐量的变化情况，随着数据中心数的增加，文献[11]模型的吞吐量变化明显，而文献[16]模型和 CCSGPR

表 1　　**不同因果一致性模型的比较**

模型	时钟	复制策略	向量	操作时延
文献[10]	物理时钟	完全复制策略	全局稳定时间戳(1)	高
文献[11]	混合逻辑时钟	完全复制策略	通用稳定向量(m)	较低
文献[16]	物理时钟	部分复制策略	全局稳定时间戳(1)	较高
CCSGPR	混合逻辑时钟	部分复制策略	共享稳定向量(2)	低

注：向量（）为全局稳定策略中向量包含元素的个数，其中 m 为数据中心数。

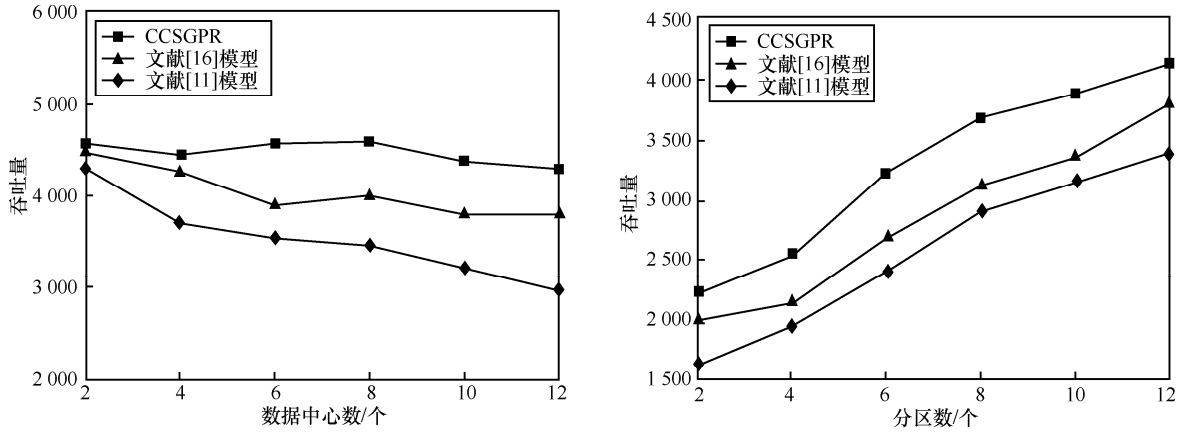
模型的吞吐量变化较小。其原因是文献[11]模型基于完全复制策略，数据中心增加导致数据同步开销增大，从而降低了模型的吞吐量。文献[16]模型和 CCSGPR 基于部分复制策略，仅模型中的部分数据中心同步数据，模型的吞吐量变化不大。当数据中心数为 8 时，CCSGPR 比文献[11]模型提高了 23.47%的吞吐量性能，比文献[16]模型提高了 14.43%。与文献[11]模型相比，文献[16]模型和 CCSGPR 使用部分复制策略，提高了模型的吞吐量。与文献[16]模型相比，CCSGPR 的全局稳定策略中使用混合逻辑时钟避免了操作时延，实现了更高的吞吐量。

其次，分析分区数对模型吞吐量的影响，控制数据中心数变量，设置数据中心数为 4，改变数据中心的分区数量。图 5(b)描述了分区数 2~32 对应吞吐量的变化情况，随着分区数的增加，模型的吞吐量呈上升趋势，吞吐量性能提高了 18.93%。与文献[11]模型和文献[16]模型比，CCSGPR 利用共享图表述数据中心之间的邻接关系，其分区向量 PV 仅存储具有邻接关系的部分数据中心元数据，元数据通过同步消息传播，降低了元数据开销，实现了吞吐量的提升。

6.4　操作响应时间

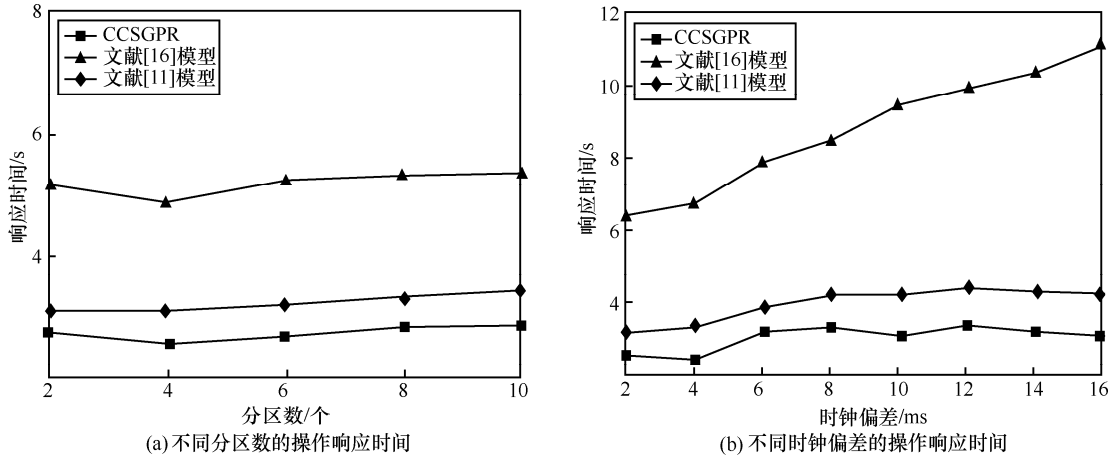
根据传统测试操作响应时间的实验方式，本节实验分两部分来测量模型操作的响应时间，第一部分分区之间不设置时钟偏差，通过改变数据中心分区数的方式观察模型操作响应时间的变化情况；第二部分在一台物理设备上部署 2 个虚拟服务器，以循环方式发送 2 000 个操作请求，人为地控制服务器之间的时钟偏差值，准确分析服务器之间的时钟偏差对操作响应时间的影响。

图 6(a)描述了模型中操作响应时间随分区数的变化情况，图 6(b)描述了模型中操作响应时间随分区之间的时钟偏差值的变化情况。随着分区数和分区之间的时钟偏差值的增加，模型的操作响应时间也会增加，与文献[11]模型和 CCSGPR 相比，文献[16]



(a) 不同数据中心数的吞吐量　　(b) 不同分区数的吞吐量

图 5　不同数据中心数和不同分区数的吞吐量



(a) 不同分区数的操作响应时间　　(b) 不同时钟偏差的操作响应时间

图 6　不同分区数和不同时钟偏差的操作响应时间

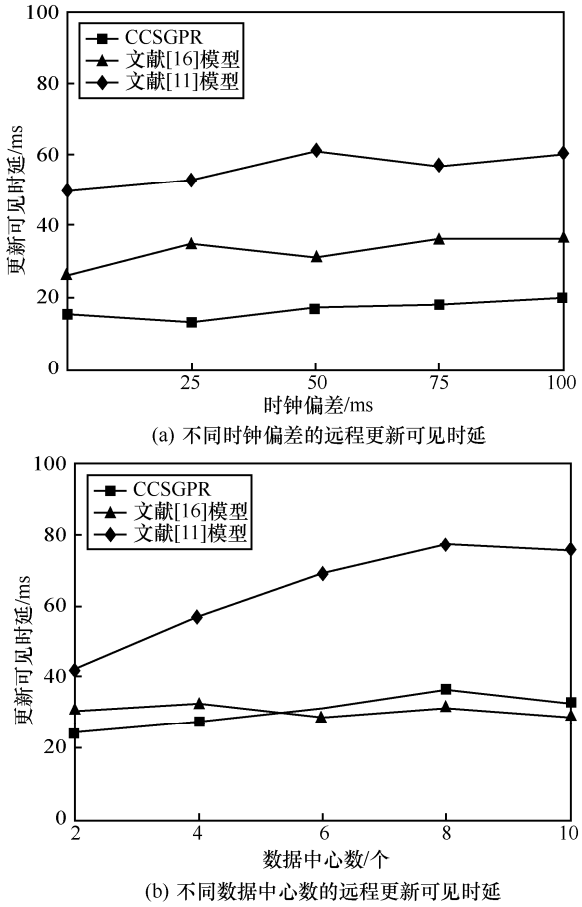
模型的全局稳定策略中使用物理时钟，受分区之间的时钟偏差影响大，产生了较高的操作时延。文献[11]模型和 CCSGPR 的全局稳定策略中使用混合逻辑时钟，当分区之间存在时钟偏差时，可为操作提供符合因果依赖关系的时间戳，有效避免了操作时延。与文献[11]模型比，CCSGPR 的分区向量中仅同步共享图 N_i 集合中的数据中心所对应分区的元数据，受分区总数目变化影响较小，其共享稳定向量规定选取远程数据中心中时间戳的最小值，划定数据可读的下限，降低了 11.72%操作响应时间。

6.5　远程更新可见时延

远程更新可见性时延指的是本地数据中心写入的数据同步到远程数据中心的时间间隔，在衡量模型性能中也占据着重要地位，即使只有几毫秒的时延，也会导致读取错误的过时数据而造成异常。

远程更新可见性时延是由服务器接收的最小时钟值决定，首先通过人为地控制服务器之间的时钟偏差值的方式，分析时钟偏差对远程更新可见时延的影响。其次服务器之间不设置时钟偏差，增加模型中数据中心数目，观察远程更新可见时延的变化情况。

图 7(a)描述了远程更新可见时延随时钟偏差值的变化情况，图 7(b)描述了远程更新可见时延随数据中心数目的变化情况。结果表明，远程更新可见性时延受服务器之间的时钟偏差影响，随时钟偏差值呈线性增长，其节点接收心跳信息值的数量随数据中心数线性增加，远程更新可见时延也会增加。与文献[11]模型比，文献[16]模型和 CCSGPR 基于部分复制策略，远程更新仅在部分数据中心之间同步，降低了远程更新可见时延。较文献[16]模型，CCSGPR 降低了 28.57%的远程更新可见时延。文献[11]模型中使用通用稳定向量 UST，一个数据中心对应向量中的一个时间戳，远程更新规定所有远程数据中心都完成同步才可见，模型的远程更新可见性时延取决于本地数据中心同步到所有远程数据中心的时间，受数据中心数影响较大。文献[16]模型中使用全局稳定时间戳 GST，模型的远程更新可见性时延取决于本地数据中心（即数据写入的数据中心）同步到其时延最大的数据中心的时间，数据中心数的增加会产生更小的全局稳定时间戳，从而造成远程更新可见时延增加。CCSGPR 中使用 SSV，向量中仅包括本地数据中心的时间戳 t_1 和远程数据中心的时间戳 t_2，模型的远程更新可见性时延取决于本地数据中心 i 与共享图 N_i 集合中数据中心同步的时间。由于 CCSGPR 模型其 SSV 中远程数据中心时间戳 t_2 的选取最小值原则，随着数据中心数的增大，CCSGPR 的远程更新可见时延会略高于文献[16]模型。



(a) 不同时钟偏差的远程更新可见时延

(b) 不同数据中心数的远程更新可见时延

图 7　不同时钟偏差和不同数据中心数的远程更新可见时延

7　结束语

本文提出了基于共享图和部分复制策略的分布式存储因果一致性模型 CCSGPR，该模型以部分复制策略为前提，利用共享图表示数据中心间的邻接关系。此外，利用混合逻辑时钟标记符合因果关系的时间戳，共享稳定向量选取远程数据中心心跳时间戳的最小值作为数据稳定的下限，以降低操作时延。通过数据一致性平台对 CCSGPR 进行吞吐量、操作响应时间和远程更新可见时延验证，在降低操作响应时间的同时，权衡了远程更新可见性能和元数据开销。但 CCSGPR 未考虑敏感数据恶意篡改等安全问题，实现可信约束下的分布式存储因果一致性模型是未来要做的工作。

参考文献：

[1] DU J Q, ELNIKETY S, ROY A, et al. Orbe: scalable causal consistency using dependency matrices and physical clocks[C]//Proceedings of the 4th Annual Symposium on Cloud Computing. New York: ACM Press, 2013: 11-14.

[2] AKKOORATH D D, TOMSIC A Z, BRAVO M, et al. Cure: strong semantics meets high availability and low latency[C]//2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS). Piscataway: IEEE Press, 2016: 405-414.

[3] MARCOS K, AGUILER A, TERR Y. The many faces of consistency[J]. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2016, 39(1): 3-13.

[4] 朱涛, 郭进伟, 周欢, 等. 分布式数据库中一致性与可用性的关系[J]. 软件学报, 2018, 29(1): 131-149.
ZHU T, GUO J W, ZHOU H, et al. Consistency and availability in distributed database systems[J]. Journal of Software, 2018, 29(1): 131-149.

[5] AHAMAD M, NEIGER G, BURNSI J E, et al. Causal memory: definitions, implementation, and programming[J]. Distributed Computing, 1995, 9(1): 37-49.

[6] SPIROVSKA K, DIDONA D, ZWAENEPOEL W. Optimistic causal consistency for geo-replicated key-value stores[C]//2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). Piscataway: IEEE Press, 2017: 2626-2629.

[7] BAILIS P, VENKATARAMAN S, MICHAEL J, et al. Quantifying eventual consistency with PBS[J]. The VLDB Journal, 2014, 23(2): 279-302.

[8] AGRAWAL D, ABBADI A E, SALEM K. A Taxonomy of partitioned replicated cloud-based database systems[J]. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2015, 38(1): 4-9.

[9] TOMSIC A Z, CRAIN T, SHAPIRO M. PhysiCS-NMSI: efficient consistent snapshots for scalable snapshot isolation[C]//Proceedings of the Workshop on the Principles and Practice of Consistency for Distributed Data. [S.n.: s.l.], 2016: 21.

[10] DU J Q, IORGULESCU C, ROY A, et al. Gentlerain: cheap an scalable causal consistency with physical clocks[C]//Proceedings of the ACM Symposium on Cloud Computing. New York: ACM Press, 2014: 1-13.

[11] DIDONA D, SPIROVSKA K, ZWAENEPOEL W. Okapi: causally consistent geo-replication made faster, cheaper and more available[J]. arXiv Preprint, arXiv: 1702.04263, 2017.

[12] ROOHITAVAF M, DEMIRBAS M, KULKARNI S S. CausalSpartanX: causal consistency and non-blocking read-only transactions[J]. arXiv Preprint, arXiv: 1812.07123, 2018.

[13] SPIROVSKA K, DIDONA D, ZWAENEPOEL W. PaRiS: causally consistent transactions with non-blocking reads and partial replication[J]. arXiv Preprint, arXiv: 1902.09327, 2019.

[14] GUNAWARDHANA C, BRAVO M, RODRIGUES L E. Unobtrusive deferred update stabilization for efficient geo-replication[C]//USENIX Annual Technical Conference. Berkeley: USENIX Association, 2017: 83-95.

[15] BRAVO M, RODRIGUES L, ROY P V. Saturn: a distributed metadata service for causal consistency[C]//European Conference on Computer Systems. New York: ACM Press, 2017: 111-126.

[16] XIANG Z L, VAIDYA N H. Global stabilization for causally consistent partial replication[J]. arXiv Preprint, arXiv: 1803.05575, 2018.

[17] ALMEIDA S, LEITAO J, RODRIGUES L E. ChainReaction: a causal+ consistent datastore based on chain replication[C]// European Conference on Computer Systems. New York: ACM Press, 2013: 85-98.

[18] 刘佩, 蒋梓逸, 曹袖. 一种基于分布式存储系统中多节点修复的节点选择算法[J]. 计算机研究与发展, 2018, 55(7): 1557-1568.
LIU P, JIANG Z Y, CAO X. Node selection algorithm during multi-nodes repair progress in distributed storage system[J]. Journal of Computer Research and Development, 2018, 55(7): 1557-1568.

[19] SPIROVSKA K, DIDONA D, ZWAENEPOEL W. Wren: nonblocking reads in a partitioned transactional causally consistent data store[C]//The 48th International Conference on Dependable Systems and Networks. Piscataway: IEEE Press, 2018: 1-12.

[20] ROOHITAVAF M, AHN J S, KANG W H, et al. Session guarantees with raft and hybrid logical clocks[J]. arXiv Preprint, arXiv:1808.05698, 2018.

[21] ROOHITAVAF M, DEMIRBAS M, KULKARNI S S. CausalSpartan: causal consistency for distributed data stores using hybrid logical clocks[C]//IEEE International Symposium on Reliable Distributed Systems. Piscataway: IEEE Press, 2017:184-193.

[22] DIDONA D, GUERRAOUI R, WANG J J, et al. Causal consistency and latency optimality: friend or foe?[J]. Proceedings of the VLDB Endowment, 2018, 11(11): 1618-1632.

[作者简介]

田俊峰（1965− ），男，河北保定人，博士，河北大学教授、博士生导师，主要研究方向为信息安全与分布式计算。

杨万贺（1996− ），男，河北保定人，河北大学硕士生，主要研究方向为信息安全、数据一致性。

庞亚南（1995− ），女，河北衡水人，河北大学硕士生，主要研究方向为信息安全、数据一致性。

张俊涛（1995− ），男，河北保定人，河北大学硕士生，主要研究方向为信息安全、数据一致性。