软件学报ISSN 1000-9825, CODEN RUXUEW E-mail: jos@iscas.ac.cn

*Journal of Software*, [doi: 10.13328/j.cnki.jos.000000] http://www.jos.org.cn

©中国科学院软件研究所版权所有. Tel: +86-10-62562563

分布式系统中的因果一致性综述[[1]](#footnote-1)\*

江雪1, 魏恒峰1,2, 黄宇1

1(南京大学 计算机科学与技术系, 江苏 南京 210023)

2(南京大学 软件学院, 江苏 南京 210023)

通讯作者: 魏恒峰, E-mail: hfwei@nju.edu.cn

摘 要:

关键词: 因果一致性;协议;理论;检测

中图法分类号: TP311

中文引用格式: 江雪,魏恒峰,黄宇.分布式系统中的因果一致性综述.软件学报,2023,32(7). http://www.jos.org.cn/1000-9825/0000.htm

英文引用格式: Jiang X, Wei HF, Huang Y. Survey on causal consistency. Ruan Jian Xue Bao/Journal of Software, 2023 (in Chinese).http://www.jos.org.cn/1000-9825/0000.htm

Survey on Causal Consistency in Distributed Systems

JIANG Xue1, WEI Heng-Feng1,2, HUANG Yu1

1(Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China)

2(School of Software, Nanjing University, Nanjing 210023, China)

**Abstract**:

**Key words**: causal consistency; protocol; theory; testing

# 引言

使用副本的分布式系统

CAP, PACELC, 弱一致性

因果一致性,非事务和事务

规约,理论,协议,检测几个部分的重要性,每个部分大致包括哪些内容

# 规约框架

在本章中我们将介绍针对弱一致复制数据类型的通用规约框架.该框架是通过扩展 Burckhardt 等人[1,2]提出的针对最终一致的复制数据类型的规约框架获得的.

## 关系和序

一个二元关系

# 非事务因果一致性

因果一致性是在分布式系统中被广泛使用的一种弱一致性模型[3,4].因果一致性的规约中最关键的概念是事件之间的发生顺序(happens-before order/hb)[2,5].直观地说,因果一致性确保如果一个事件发生在事件之前,那么所有会话(session)必须在之前观察到.然而,同时并发事件可能被不同的会话以不同的顺序观察到.

相关文献中定义了几种因果一致性的变体,他们之间有细微的差别[2,6,7].在本节中,我们考虑六种变体,称之为Weak Causal Consistency(WCC)、Weak Causal Convergence(WCCv)、Causal Memory(CM)、Causal Memory Convergence(CMv)、Strong Causal Consistency(SCC)和Strong Causal Convergence(SCCv),如表2所定义.请注意,这些变体在相关工作中可能有不同的名称,如表2所总结的.

就可见性()而言,所有因果一致性变体需要来捕捉事件之间发生的先后关系.在仲裁()方面,它们都要求,用以强制要求当解释事件在其操作上下文中的返回值时,可见的所有事件()执行hb顺序.然而,它们可能在以下两个方面有所不同:

* 用于指定其返回值必须被考虑的可见事件子集的函数有多大?请注意,在因果一致性模型中,对于任何事件(因为).因此,候选.
* 仲裁关系对于解决冲突有多强?给定,根据是否是全序,我们区分两种情况.

表 1 因果一致性变体在((框架中的规约

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Consistency Models | Alternative Names |  |  |  |
| WCC ([6]) | CC ([7]) |  |  |  |
| CM ([3,7]) | CC ([6]) |  |
| SCC |  |  |
| WCCv | CCv ([6,7])  CAUSALCONSISTENCY ([2])  CAUSALITY ([8]) |  |  |
| CMv |  |  |
| SCCv |  |  |

## 规约

### Weak Causal Consistency

在[6]中提出的Weak Causal Consistency(WCC)是我们考虑的最弱的因果一致性变体.非正式地讲,一个抽象执行满足WCC,当且仅当每个事件的返回值)能够被其可见事件集的某种序列化所解释(忽略所有可见事件的返回值).更具体地说,在仲裁方面,WCC对收敛性不做额外的限制,因此并发的事件可能被不同的会话以不同的顺序观察到.就而言,WCC允许每个事件忽略其可见事件的所有返回值.

其可见事件的所有返回值.

1. (Weak Causal Consistency)

### Causal Memory

Causal Memory(CM)最初是由Ahamad等人[3]提出的针对读/写寄存器(read/write register)的定义.最近,Perrin等人[6]将其扩展到任意的复制数据类型.CM比WCC更强,因为在证明每个事件的返回值)的合理性时, CM不仅考虑到了WCC中对可见的事件集的操作调用,还考虑到同一会话中之前的事件集合中事件的返回值.换句话说,与WCC相比,CM要求每个会话与之前提供的返回值相一致[6].

1. (Causal Memory)

### Strong Causal Consistency

我们通过进一步要求每个会话和其他会话提供的返回值保持一致,来加强CM使其成为 SCC(Strong Causal Consistency).形式上,我们有.

1. (Strong Causal Consistency)

### Weak Causal Convergence

WCCv(Weak Causal Convergence)[2,6]是WCC的收敛版本.它通过对执行中的所有事件施加一个全序加强WCC,这为所有会话提供了一个解决并发事件引起的冲突的统一方法.因此,每个事件的返回值在对可见的事件的集合上进行验证时(忽略他们所有的返回值),这些可见事件要按照共同的全序排序.

1. (Weak Causal Convergence)

### Causal Memory Convergence(CMv)

CMv(Causal Memory Convergence)是CM对应的收敛版本.它要求是一个全序.

1. (Causal Memory Convergence)

### Strong Causal Convergence(SCCv)

SCCv(Strong Causal Convergence)是SCC对应的收敛版本.它要求是一个全序.

1. (Strong Causal Convergence)

## 理论

现有的系统采用不同的技术来实现因果一致性,但它们都有相同的关键机制.因果一致性的实现中最核心的内容就是追踪事件之间的因果依赖关系(Causal Dependencies).无论利用何种机制来进行追踪,都离不开对事件进行标记(本章以下内容称为标识符),通常可以使用以下几种方式.

1. Lamport 时间戳

Lamport时间戳由计算机科学家Leslie Lamport于1978年提出.Lamport时间戳是一种逻辑时钟,它可以被用于确定事件之间的顺序关系.

Lamport时间戳的基本思想是,为每个事件分配一个唯一的标识符,并保证在同一个进程中,事件标识符的值是单调递增的.当事件在一个进程中发生时,它会被分配一个时间戳,该时间戳由该进程的本地计数器值和进程标识符组成.

使用Lamport时间戳,我们可以将事件排序,并确定它们之间的顺序关系.如果事件A的时间戳早于事件B的时间戳,则可以得出结论,事件A在事件B之前发生.但是,Lamport时间戳不能用于确定事件的实际发生时间,因为事件的时间戳只是一个逻辑标记,不具有实际的时间单位.

Lamport时间戳是分布式系统中一种广泛使用的技术,可以用于并发控制、同步、消息传递等方面.它的优点是实现简单,可扩展性好,缺点是无法反映实际时间,也不能用于测量两个事件之间的时间差.

1. 向量时钟(Vector Clock)

向量时钟由Paul M. Mellor-Crummey和Michael L. Scott于1991年提出.向量时钟可以用于解决Lamport时间戳无法解决的一些问题,如处理并发事件的因果关系.

向量时钟的基本思想是,为每个事件分配一个向量时间戳,该向量时间戳由所有进程的本地计数器值组成.每个进程维护一个本地计数器,该计数器在进程执行每个事件时递增,同时将本地计数器值和进程标识符包含在向量时间戳中.

当一个进程向另一个进程发送消息时,它将自己的向量时间戳附加到消息中,并将其发送给接收进程.接收进程接收到消息后,将自己的本地计数器递增,并使用接收到的向量时间戳更新自己的向量时间戳.如果接收到的向量时间戳中有任何进程的计数器值大于自己的计数器值,则将该进程的计数器值复制到自己的计数器中.

使用向量时钟,我们可以比较两个事件的时间戳,并确定它们之间的因果关系.如果事件A的时间戳在事件B的时间戳中的所有进程上都小于或等于事件B的时间戳,则可以得出结论,事件A在事件B之前发生.如果事件A的时间戳与事件B的时间戳中的某些进程计数器值相等,而其他进程的计数器值更大,则无法确定两个事件的因果关系,因此它们是并发事件.

与lamport时钟比较

1. Hybrid logical physical clock

Hybrid logical physical clock(HLC)是由Marcin Lis和Roman Dębski在2005年提出的.HLC结合了逻辑时钟和物理时钟的优势,以提供更准确和可比较的时间戳.

HLC的基本思想是将逻辑时钟和物理时钟结合起来,得出一个更准确的时间戳.具体地说,每个进程维护两个计数器：一个物理时钟计数器和一个逻辑时钟计数器.物理时钟计数器递增的速率由物理时钟决定,而逻辑时钟计数器递增的速率由系统时钟决定,通常比物理时钟快得多.

当一个事件发生时,HLC首先使用物理时钟计数器来计算事件的物理时间戳.然后,HLC使用逻辑时钟计数器来计算事件的逻辑时间戳,并将物理时间戳和逻辑时间戳组合成一个Hybrid时间戳.Hybrid时间戳由物理时间戳和逻辑时间戳的加权平均值组成,权重由一个参数控制,通常是一个介于0和1之间的小数.

我们可以通过比较两个事件的HLC时间戳来确定它们之间的因果关系.首先比较两个时间戳的物理时钟部分.较早的物理时间戳被认为在时间上更早.如果两个时间戳的物理时间戳部分相等,那么比较它们的逻辑时钟部分.较大的逻辑时钟值被认为在事件顺序上更晚.如果两个时间戳的物理时间戳和逻辑时间戳都相等,则比较它们的Hybrid时间戳.Hybrid时间戳是物理时间戳和逻辑时间戳的加权平均值.根据加权平均值的结果,较大的Hybrid时间戳被认为在事件顺序上更晚.

1. Version vector…..

有了事件标识符后,我们可以通过日志或是直接利用消息传递来保证事件之间的因果关系.

### 利用日志 (append-only,是否有其他类型的日志)

### 日志是代表系统中事件的一连串记录.日志中的每条记录对应于一个事件,通常会用一个唯一的标识符和一个时间戳表示.利用日志来实现因果一致性的方案是基于日志复制的方案,其基本思想是在系统中维护多个副本,并通过日志记录和复制来保证数据的因果一致性.

### 具体来说,系统中的每个副本都维护一个日志,用于记录所有的写操作.在写入数据时,需要将该操作写入本地日志,成为日志中的一条记录,该记录还包含它与其他记录的因果依赖关系的信息(例如用时间戳表示).副本还需要将该操作的在本地日志中的记录复制到其他副本的日志中.由于每个操作都包含了其与其他操作的因果依赖关系,因此在读取数据时,可以比较该数据项在不同副本上的版本信息,以保证因果一致性.具体来说,读取操作会要求读取特定版本的数据,系统会比较该版本的标识符和副本上对该版本对应数据的已经执行的操作的标识符,只有当该版本的所有因果依赖(Causal Dependencies)都已经执行完成时,才返回数据给客户端.

### 通过利用日志来实现因果一致性,系统可以保证在分布式环境下对数据的操作是有序的,并且不会发生竞争条件.同时,日志的复制也可以保证数据的可靠性,即使某个副本出现故障,系统也可以通过其他副本的日志来进行数据恢复.

### 利用消息传递

利用消息传递来实现因果一致性的方案是基于消息传递模型的方案,其基本思想是在消息中包含操作的因果关系信息,并在接收到消息后,根据这些信息来维护因果关系.

具体来说,对于写操作,可以在消息中包含该操作之前(限定在因果关系上)的操作的标识符,即先决条件(Precondition).在接收到写操作的消息时,系统会检查该操作的先决条件是否已经满足,只有在先决条件满足的情况下才执行该写操作.这样可以保证操作的因果关系是有序的,并且不会发生竞争条件.对于读操作,需要在消息中包含读取数据的版本信息(标识符,通常是一个向量时钟(Vector Clock)或时间戳(Timestamp),用于标识该读操作因果依赖的所有先前操作).在接收到读操作的消息时,系统会比较该版本的标识符和接收到的其他操作的标识符,只有当该版本的所有因果依赖(Causal Dependencies)都已经执行完成时,才返回数据给客户端.

消息传递协议的一个潜在问题是消息延迟或丢失的可能性.如果一个消息被延迟或丢失,如果依赖该消息的其他消息已经被处理,就会导致违反因果一致性.为了减轻这种风险,消息传递协议通常包括检测和处理消息延迟或丢失的机制,如超时机制、重试和重复检测.消息传递协议的另一个问题是可能出现时钟同步问题.为了让Lamport时间戳和矢量时钟正常工作,系统中的每个节点必须有一致的时间概念.如果时钟同步不当,就会导致不正确的因果排序,违反因果一致性.为了解决这个问题,可以使用时钟同步算法来确保所有节点的时钟同步到一定的精度内.此外,消息传递协议的性能会受到网络延迟、消息量和协议实现的复杂性等因素的影响.在某些情况下,与维护和处理时间戳有关的开销会影响系统性能和可扩展性.总之,虽然使用消息传递来保证因果一致性,如果实施和使用得当是安全的,但在分布式系统中设计和部署消息传递协议时,必须考虑潜在的问题,如消息延迟或损失、时钟同步问题和性能开销.

## 协议

系统类文章的协议的正确性证明

## 检测

# 事务因果一致性

## 规约

## 理论

### Client-Assistant Lazy Dependency Resolution(CALDeR,in OCC[9])

OCC为了最大限度地减少元数据的开销和读取响应时间,采取了一种经济有效的方式暴露不稳定的项目版本,并且能够保证系统的因果一致性.这正是通过Client-Assistant Lazy Dependency Resolution(CALDeR)协议来实现的.CALDeR协议的主要思想是通过将追踪因果依赖关系的责任转移到客户端,从而将数据存储从追踪因果依赖关系的责任中解脱出来.CALDeR协议消除了分区之间的同步需求,这隐含地消除了由速度较慢的分区滞后引起的问题.下面将详细介绍该协议的做法.

客户端以客户端依赖元数据(Client-side Dependency Metadata)的形式存储执行读取时建立的因果关系的信息.如果客户端*c*读取了*Y*,并且存在*X*,使得*XY*,*c*会将*X*和*Y*之间的依赖关系记录到它的客户端依赖元数据中.当*c*执行以后的操作时,就可以通过客户端依赖元数据来读取所需的因果依赖关系.

当客户端*c*发起一个读操作(GET或RO-TX)请求时,服务器*p*需要客户端的依赖元数据来让确定它自己的状态是否与*c*的历史一致.考虑上一段中的例子,如果*c*想在读完*Y*之后再读*X*,通过*c*提供的依赖元数据和*p*本地存储的状态信息, *p*检查它是否已经收到*X*.如果*p*已经收到了*X*,那么*p*存储的最新的本地版本的*X*与*c*的历史兼容,它被返回给*c*.如果*p*还没有收到*X*,*p*必须在为*c*的请求服务之前收到它.在这种情况下,*p*会拖延*c*的请求,直到它收到*X*.

当客户端*c*发起一个写操作(PUT操请求时,需要客户端的依赖元数据来知道新创建的项目依赖于什么.我们仍然考虑前面的例子,当在服务器*p*上写入*Y*时,*p*必须已经交付了*Y*因果依赖的所有版本的*y*.也就是说,在OCC中,*p*必须等到它收到发行客户端所依赖的最新的Y的版本才能执行该写请求.

### Client’s Universal Stable Time(UST,in PaRis[11])

UST是PaRiS[10]实现的协议,以可扩展的方式识别稳定的快照.每个分区都维护着一个版本向量,表示最新应用的事务的时间戳,包括分区本身执行的事务和从远程replicas收到的事务.通过gossiping协议,同一DC内和跨DC的分区定期交换其版本向量中的最小时间戳.交换值的总体最小值确定了一个时间戳,即每个DC的所有分区都应用了时间戳小于等于该最小值的所有事务.UST使用单一时间戳来保证因果一致性,这减少了分区与分区之间以及客户与分区之间的通信开销,实现了高可扩展性和提高了效率.

仅利用UST并不能保证因果关系.事实上,通过保证分配给客户端*c*发出的事务提交时间戳(Commit Timestamp)大于分配给*T*的稳定快照(Stable Snapshot),使得提交时间戳能够反映因果关系.注意,*T*的提交时间戳可能大于 *c*请求的下一个事务所分配的快照.在这种情况下,这种快照将不包括*c*在*T*中进行的修改,这可能导致违反因果一致性所要求的read-your-writes性质.

### Binary Dependency Time(BDT,in Wren[10])

Wren实现了Binary Dependency Time(BDT)协议用来追踪数据项之间的因果依赖关系.BDT的关键特征是每个数据项只通过两个标量时间戳(Scalar Timestamp)来追踪因果依赖关系,而不管系统的规模如何.一个标量时间戳追踪本地项目(Local Items)的依赖关系,另一个标量时间戳总结远程项目(Remote Items)的依赖关系.与其他设计相比,只使用两个时间戳可以实现更高的效率和可扩展性.

Wren依靠BDT来实现Binary Stable Time(BiST),这是一个高效的稳定协议(Stablization Protocol),以确定更新操作何时可以包括在回复给DC内的客户提出的快照(即更新操作何时在DC内可见).BiST允许起源于一个DC的更新在该DC中立即变得可见,而不需要等待收到远程项目.相反,当一个远程更新*u*是稳定的之后才在DC内可见,这需要等到它的所有因果依赖项全部被DC接收.BiST计算出两个截止值(Cut-of Value)来分别表示那些本地和远程项目可以在一个DC内的事务中可见.BiST计算的本地部分是本地稳定时间(Local Stable Time,LST).远程分量是远程稳定时间(RST),与LST类似,表示本地DC内每个节点已经安装的远程快照的下限.通过解耦本地和远程项目,与使用单一标量时间戳进行依赖性跟踪的系统不同,BiST允许事务确定本地项目的可见性,而无需与远程DC同步.这种解耦在地理复制的情况下也能实现可用性和无阻塞的读取,因为对事务可见的快照只包括本地DC已经收到的远程项目.

通过BiST,DC内的分区定期交换他们所应用的最新本地和远程交易的提交时间戳(Commit Timestamp)后,每个分区计算LST(resp,RST)作为收到的本地(resp,远程)事务对应的时间戳的最小值.因此,LST和RST反映了本地和远程快照,这些快照已经被DC中的所有分区安装,并且事务从这些快照中读取而不被阻塞.

## 协议

系统类文章的协议的正确性证明

考虑抽象类似的几个协议；参考 chapar 的证明

## 检测

# 总结与展望

References:

[1] Burckhardt S, Gotsman A, Yang H, Zawirski M. Replicated data types: specification, verification, optimality. Proceedings of the 41st ACM Symposium on Principles of Programming Languages (POPL’2014). : 271–284.

[2] Burckhardt S. Principles of eventual consistency. Found. Trends Program. Lang., Now Publishers Inc., 2014, 1(1–2): 1–150.

[3] Ahamad M, Neiger G, Burns JE, Kohli P, Hutto PW. Causal memory: definitions, implementation, and programming. Distributed Computing, 1995, 9(1): 37–49.

[4] Lloyd W, Freedman MJ, Kaminsky M, Andersen DG. Don’t settle for eventual: scalable causal consistency for wide-area storage with cops. Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP’2011). : 401–416.

[5] Lamport L. Time, clocks, and the ordering of events in a distributed system. Commun. ACM, 1978, 21(7): 558–565.

[6] Perrin M, Mostefaoui A, Jard C. Causal consistency: beyond memory. Proceedings of the 21st ACM Symposium on Principles and Practice of Parallel Programming (PPoPP’2016). .

[7] Bouajjani A, Enea C, Guerraoui R, Hamza J. On verifying causal consistency. Proceedings of the 44th ACM Symposium on Principles of Programming Languages (POPL’2017). : 626–638.

[8] Viotti P, Vukolić M. Consistency in non-transactional distributed storage systems. ACM Comput. Surv., 2016, 49(1): 19:1-19:34.

[9] Spirovska K, Didona D, Zwaenepoel W. Optimistic causal consistency for geo-replicated key-value stores. IEEE Transactions on Parallel and Distributed Systems, 2021, 32(3): 527–542.

[10] Spirovska K, Didona D, Zwaenepoel W. Wren: nonblocking reads in a partitioned transactional causally consistent data store. 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). 2018: 1–12.

[11] Spirovska K, Didona D, Zwaenepoel W. PaRiS: causally consistent transactions with non-blocking reads and partial replication. 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). 2019: 304–316.

附中文参考文献:

[2] 陈翔,顾庆,刘望舒,刘树龙,倪超.静态软件缺陷预测方法研究.软件学报,2016,27(1):1−25. http://www.jos.org.cn/1000-9825/4923. htm [doi: 10.13328/j.cnki.jos.004923]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 江雪(1990－),女,博士生,主要研究领域为分布式计算和形式化方法. |  |  | 黄宇(1981－),男,博士,教授,CCF专业会员,主要研究领域为分布式算法和形式化方法. |
|  | 魏恒峰(1986－),男,博士,主要研究领域为分布式计算(尤其是分布式数据一致性 问题(和形式化方法. |  |  |  |

1. \* 基金项目: 国家自然科学基金(61906090, U20B2064, 61773208); 江苏省自然科学基金(BK20191287, BK20170809); 中央高校基本科研业务费专项资金(30920021131); 中国博士后科学基金(2018M632304)

   收稿时间: 2020-04-13; 修改时间: 2020-10-26; 采用时间: 2020-12-14 [↑](#footnote-ref-1)