TAO：Facebook的社交图谱的分布式数据存储

Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov
丁晖, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, Harry Li, Mark Marchukov
Dmitri Petrov, Lovro Puzar, Yee Jiun Song, Venkat Venkataramani
Facebook, Inc.

摘要

我们介绍了一个简单的数据模型和专门为社交图服务的API，以及这个模型的实现--TAO。TAO是一个地理分布式的数据存储，它为Facebook的高要求工作负载提供高效和及时的DeepL，使用一套固定的查询。它被部署在Facebook，为许多符合其模型的数据类型替换了memcache。该系统在数以千计的机器上运行，广泛分布，并提供对许多PB数据的访问。TAO每秒钟可以处理10亿次的读取和数百万次的写入。

1 简介

Facebook拥有超过10亿的活跃用户，他们记录自己的关系，分享自己的兴趣，上传文字、图像和视频，并策划有关他们数据的语义信息[2]。社交应用的个性化体验来自于对*社交图谱*这一数据流的及时、有效和可扩展的获取。在本文中，我们介绍了TAO，这是一个读取优化的图数据存储，我们已经建立了一个处理苛刻的Facebook工作负载。

在TAO之前，Facebook的网络服务器直接使用MySQL来读取或写入社交图，并积极使用memcache[21]作为旁观缓存。TAO直接实现了一个图的抽象，使其能够避免looka-side缓存架构的一些基本缺陷。TAO继续使用MySQL进行持久化存储，但对数据库的访问进行调解，并使用自己的图感知缓存。

TAO作为一个单一的地理分布实例部署在Facebook。它有一个最小的API，并明确倾向于可用性和每台机器的效率，而不是强一致性；它的创新之处在于其规模。TAO可以在一个多PB的变化数据集上维持每秒10亿次的读取。

总之，本文有三个贡献。我们启动了（第2节）并描述了（第7节）一项具有挑战性的工作：对一个不断变化的图的高效和可用的最多读访问。我们描述了对象和关联，一个数据模型和API，我们用它来访问图（第3节）。最后，我们详细介绍了TAO，一个实现这个API的地理分布式系统（第4-6节），并评估了它在我们的工作负载上的性能（第8节）。

a)

b)

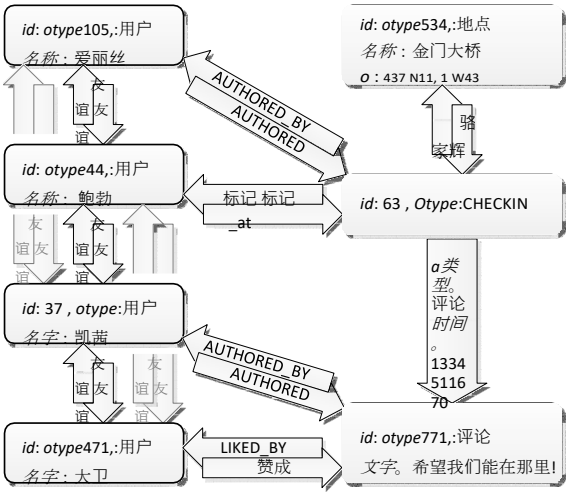


图1：一个用户的签到可能被映射到对象和关联的运行例子。

2 背景介绍

一个单一的Facebook页面可以从社交图谱中聚合并筛选出数百个项目。我们为每个用户提供为他们量身定做的内容，并对每个项目进行隐私检查，以考虑到当前浏览者的情况。这种极端的定制化使得在创建内容时执行大部分的聚合和过滤是不可能的；相反，我们解决数据的依赖性，并在每次查看内容时检查隐私。我们尽可能地拉动社交图谱，而不是推送它。这种实施策略对图数据存储提出了极高的读取要求；它必须是高效的、高度可用的，并能扩展到高查询率。

2.1 从Memcache提供图表服务

Facebook最初是通过将社交图谱存储在MySQL中，从PHP中查询，并将结果缓存在memcache中[21]。这种lookaside缓存架构很适合Facebook的快速迭代周期，因为所有的

数据映射和缓存验证的计算是在频繁部署的客户端代码中进行的。随着时间的推移，一个PHP抽象被开发出来，允许开发人员读写图中的对象（节点）和关联（边），对于符合模型的数据类型，对MySQL的直接访问被取消了。

TAO是我们构建的一个服务，直接实现了对象和关联模型。我们的动力来自于PHP API中的封装失败，来自于从非PHP服务中轻松访问图的机会，以及来自于lookaside缓存架构的几个基本问题。

不有效的边缘列表。键值缓存对于边缘列表来说不是一个好的语义工具；查询必须总是获取整个边缘列表，而对单个边缘的改变需要重新加载整个列表。基本的列表支持只是解决了第一个问题；需要更复杂的东西来协调缓存列表的并发增量更新。

分布式控制逻辑。在一个概览式的缓存结构中，控制逻辑是在客户机上运行的，而客户机之间并不互相通信。这就增加了故障模式的数量，并使其难以避免雷鸣般的群集。Nishtala等人对这些问题进行了深入的讨论，并提出了*租赁*，一个一般的解决方案[21]。对于对象和关联，固定的API允许我们将控制逻辑转移到缓存本身，在那里可以更有效地解决这个问题。

昂贵的先读后写一致性。Facebook对MySQL使用异步的主/从复制，这给使用复制的数据中心的缓存带来了问题。写作会被转发到主站，但在它们被复制到本地之前会经过一些时间。Nishtala等人的*远程标记*[21]跟踪已知过期的键，将这些键的读取转发给主区域。通过将数据模型限制为对象和关联，我们可以在写入时更新副本的缓存，然后使用图语义来解释来自并发更新的缓存维护信息。这就为所有共享缓存的客户提供了（在没有多重故障的情况下）读写一致性，而不需要区域间的通信。

2.2 TAO的目标

TAO提供对多个地区的数据中心中不断变化的图的节点和边的基本访问。它对读取进行了大量的优化，并且明确地倾向于高效性和可用性，而不是一致性。

像TAO这样的系统可能对任何需要从高度互联的数据中有效地生成细化的定制内容的应用域都是有用的。该应用不应该期望数据是

3 TAO数据模型和API

Facebook关注的是人、行动和关系。我们将这些实体和连接建模为图中的节点和边。这种表示方法非常灵活；它直接模拟现实生活中的对象，也可以用来存储应用程序的内部实现特定数据。TAO的目标不是支持一套完整的图查询，而是提供足够的表达能力来满足大多数应用的需求，同时允许可扩展和高效的实现。

考虑一下图1a中的社交网络例子，在这个例子中，Alice用她的手机记录了她和Bob对一个著名地标的访问。她在金门大桥上'签到'，并'标记'了鲍勃，表示他和她在一起。凯西添加了一条评论，大卫已经'喜欢'。社交图谱包括用户（Alice、Bob、Cathy和David），他们的关系，他们的行为（签到、评论和喜欢），以及一个物理位置（金门大桥）。

Facebook的应用服务器将查询这个事件的底层节点和边，每次它都会被重新读取。细致的隐私控制意味着每个用户可能会看到不同的签到视图：编码活动的单个节点和边可以重复用于所有这些视图，但聚合的内容和隐私检查的结果不能。

在普通情况下是陈旧的，但应该能够容忍它。许多社交网络都属于这个类别。

3.1 对象和关联

TAO*对象*是类型化的节点，TAO*关联*是对象之间类型化的有向边。对象由一个64位的整数（id）来识别，该整数在所有对象中都是唯一的，与对象类型（type）无关。关联由源对象（id1）、关联类型（type）和目标对象（id2）来识别。任何两个对象之间最多可以存在一个特定类型的关联。对象和关联都可以包含作为键值对的数据。每个类型的模式列出了可能的键、值类型和默认值。每个关联都有一个32位的时间字段，它在查询[1]中起着核心作用。

对象。(id) → (otype, → (keyvalue)∗)
Assoc.: (id1, atype, id2) → (time, (→ keyvalue)∗)

图1b显示了TAO对象和关联是如何编码这个例子的，为了清晰起见，省略了一些数据和时间。这个例子中的用户由对象表示，签到、地标和Cathy的评论也是如此。关联捕捉用户的友谊，签到和评论的作者，以及签到和其位置及评论之间的联系。

[1]时间字段实际上是一个通用的应用程序分配的整数。

行动可以被编码为对象或关联。Cathy的评论和David的 "喜欢 "都代表了用户采取的行动，但只有评论会产生一个新对象。关联自然地模拟那些最多只能发生一次或记录状态转换的动作，如接受事件邀请，而可重复的动作最好表示为对象。

尽管关联是有方向的，但关联与逆向边紧密相连是很常见的。在这个例子中，所有的关联都有一个逆边，除了COMMENT类型的链接。这里不需要逆向边，因为应用程序不需要从评论到CHECKIN对象的跟踪。一旦知道checkin的id，呈现图1a只需要重新遍历出站关联。然而，发现checkin对象需要入站边，或者一个id被存储在另一个Facebook系统中。

对象和关联类型的模式只描述实例中包含的数据。它们没有对可以连接到特定节点类型的边缘类型或可以终止边缘类型的节点类型施加任何限制。1,例如，图中的checkin对象和comment对象的au-thorship是用同一个atype来表示的。允许自边缘。

3.2 对象API

TAO的对象API提供了分配新对象和ID的操作，以及检索、更新或删除与ID相关的对象的操作。一个值得注意的遗漏是比较和设置功能，TAO的最终一致性语义大大降低了其实用性。更新操作可以应用于字段的一个子集。

3.3 协会API

社交图中的许多边是双向的，例如对称的，如例子中的朋友关系，或不对称的，如AUTHORED和AU-THORED-BY。双向的边被建模为两个独立的关联。TAO通过允许将关联类型与反向*类型进行配置，*提供了对保持关联与反向同步的支持。对于这样的关联，创建、更新和删除都会自动与反向关联的操作相联系。对称的双向类型是它们自己的逆类型。关联的写入操作是。

- **assoc add(id1, atype, id2, time, (k v)*)** - 添加或覆盖关联(id1, atype,id2)，以及它的逆向(id1, inv(type), id2)（如果已定义）。
- **assoc－delete(id1, atype, id2)** - 删除association(id1, atype, id2)和inverse(如果它存在)。
- **Assoc改变类型(id1, atype, id2, newtype)** - 如果(id1, atype, id2)存在，将关联(id1, atype, id2)改为(id1, newtype, id2)。

3.4 协会查询API

任何TAO关联查询的起点都是一个起源对象和一个关联类型。这是搜索关于一个特定对象的特定类型信息的自然结果。考虑图1中的例子。为了显示CHECKIN对象，应用程序需要列举所有标记的用户和最近添加的评论。

社交图的一个特点是，大多数数据都是旧的，但许多查询是针对最新的子集。每当一个应用程序专注于最近的项目时，就会出现这种*创建时间的局部性*。如果图1中的Alice是一个著名的名人，那么她的签到可能有成千上万的评论，但默认情况下只有最近的评论会被呈现。

TAO的关联查询是围绕*关联列表*组织的。我们将关联列表定义为具有特定id1和type的所有关联的列表，按时间字段降序排列。

协会列表。(id1, atype) → [$a_{\text{new}}a\ldots_{\text{old}}$] 。

例如，列表(*i*, COMMENT)的边是关于*i*的例子的评论，最新的在前面。

TAO对协会名单的查询。

- **assoc get(id1, atype, id2set, high?, low?)** - 返回所有的关联（id1, atype, id2）及其时间和数据，其中 id2id2set和high timelow（如果指定了）。可选的时间界限是为了提高大型关联的缓存能力。

  协商名单（见第5节）。
- **assoc count(id1, atype)** - 返回(id1, atype)的关联列表的大小，即源自id1的type型边的数量。
- **assoc range(id1, atype, pos, limit)** - 返回(id1, atype)关联列表中 *i*∈ [pos, pos + limit) 的部分。
- **assoc time range(id1, atype, high, low, limit)** - 返回(id1, atype)关联列表中的元素，从时间≤高的第一个关联开始，只返回时间≥低的边。

TAO对用于关联查询的实际限制强制执行每个类型的上限（通常是6000）。为了列举一个较长的关联列表的元素，客户必须发出多个查询，使用pos或high来指定一个起始点。

对于图1所示的例子，我们可以将一些可能的查询映射到TAO API上，如下所示。

- "关于爱丽丝签到的50条最新评论" ⇒
  Assoc range(632, COMMENT, 500,)
- "在GG桥有多少人登记？" ⇒
  赋值计数(534, CHECKIN)

–

∈

≥

≥

–

–

– –

4 TAO建筑

在这一节中，我们描述了组成TAO的单元，以及允许其跨数据中心和地理区域扩展的多层聚合。TAO被分离成两个缓存层和一个存储层。

4.1 存储层

甚至在TAO建立之前，对象和关联就已经存储在Facebook的MySQL中了；它是API的原始PHP实现的支持存储。这使得它成为TAO持久化存储的自然选择。

TAO的API被映射到一小部分简单的SQL查询中，但它也可以通过明确地维护所需的in-dexes来有效地映射到非SQL数据存储系统中的范围扫描，如LevelDB[3]。然而，在评估TAO的备份存储的适用性时，必须考虑不使用API的数据访问。这些包括备份、数据的批量导入和删除、从一种数据格式到另一种数据格式的批量迁移、副本创建、异步复制、一致性监测工具和操作调试。一个替代的存储也必须提供原子写交易，高效的颗粒写，以及很少的延迟异常值。

鉴于TAO需要处理的数据量远大于单个MySQL服务器所能存储的数据量，我们将数据分为逻辑*分片*。每个分片都包含在一个逻辑数据库中。数据库服务器负责一个或多个分片。在实践中，分片的数量远远超过了服务器的数量；我们调整了分片与服务器的映射，以平衡不同主机的负载。默认情况下，所有对象类型存储在一个表中，所有关联类型存储在另一个表中。

每个对象的ID都包含一个嵌入的分片ID，用于识别其托管分片。对象在其整个生命周期内被绑定到一个分片上。一个关联被存储在其id1的分片上，因此每个关联查询都可以从一个服务器上得到服务。两个id不太可能映射到同一个服务器上，除非它们在创建时被明确地放在一起。

4.2 缓存层

TAO的缓存为客户实现了完整的API，处理与数据库的所有通信。缓存层由多个*缓存服务器组成*，它们共同构成一个*层*。一个层级能够共同响应任何TAO请求。(我们也把一个区域内的数据库集合称为一个层。)每个请求都使用类似于4.1节中描述的分片方案映射到一个缓存服务器。没有要求各层有相同数量的主机。

客户端直接向相应的高速缓存服务器发出请求，然后由其负责完成读取。

或写。对于缓存缺失和写入请求，服务器会联系其他缓存和/或数据库。

TAO的内存缓存包含对象、关联列表和关联计数。我们按需填充缓存，并使用最近使用的最小值（LRU）策略驱逐项目。缓存服务器理解其内容的语义，并使用它们来回答查询，即使确切的查询之前没有被处理过，例如，缓存中的计数为零就足以回答一个范围查询。

对一个有逆向的关联的写操作可能涉及到两个分片，因为正向边缘被存储在id1的分片上，逆向边缘被存储在id2的分片上。从客户端接收查询的层级成员向托管id2的成员发出RPC调用，该成员将联系数据库以创建逆向关联。一旦反写完成，缓存服务器就会为id1向数据库发出一个写。TAO不提供这两个更新之间的原子性。如果发生故障，正向可能存在而没有反向；这些*挂起的*关联由异步作业安排修复。

4.3 客户端通信栈

在渲染一个Facebook页面时，查询数百个对象和关联是很常见的，这很可能需要在短时间内与许多缓存服务器通信。由此产生的全对全通信的挑战与我们的memcache池所面临的挑战相似。TAO和memcache共享Nishtala等人[21]所描述的大部分客户端栈。TAO请求的延迟可能比mem- cache的延迟高得多，因为TAO请求可能会访问数据库，所以为了避免多路连接上的线头阻塞，我们使用了一个具有失序响应的协议。

4.4 领导者和追随者

理论上来说，只要分片足够小，单个缓存层可以被扩展到处理任何可预见的总请求率。但在实践中，大的层级很可能是不现实的，因为它们更容易出现热点，而且它们的所有连接数会呈二次增长。

为了在限制最大层数的同时增加服务器，我们将缓存分成两层：一个*领导者*层和多个*跟随者*层。TAO相对于旁观者缓存架构的一些优势（如2.1节所述）依赖于每个数据库有一个单一的缓存协调器；这种分割允许我们将协调器保持在每个区域的单一层中。与单层配置一样，每层都包含一组缓存服务器，它们共同能够响应任何TAO查询；也就是说，系统中的每个分片都映射到每层的一个缓存服务器上。领头羊（领头羊层的成员）的行为如以下所述
§ 4.2,从存储层读取和写入存储层。左右

低层（追随者层的成员）将转发读失误和写失误到一个领导者。客户端与最近的追随者层进行通信，从不直接与领导者联系；如果最近的追随者不可用，他们就会失效到另一个附近的追随者层。

鉴于这种两级缓存的层次结构，必须注意保持TAO缓存的一致性。每个分片都由一个领导者主持，所有对分片的写入都要经过这个领导者，所以它自然是一致的。另一方面，追随者必须被明确告知通过其他追随者层级进行的更新。

TAO通过异步从领导者向追随者发送缓存维护信息来提供最终的一致性[33，35]。领导者中的对象更新将无效消息排到每个相应的跟随者中。发出写的追随者在领导者的回复中被同步更新；缓存维护消息中的版本号允许它在以后到达时被识别。由于我们只对关联列表的连续序号进行缓存，因此无效的关联可能会截断列表并丢弃许多边。相反，领导者会发送一个*refill*消息来通知追随者关于一个关联的写入。如果追随者已经缓存了关联，那么refill请求会触发对领导者的查询，以更新追随者已经过时的关联列表。§ 第6.1节讨论了这种设计的一致性，以及它是如何容忍失败的。

领导者对来自追随者的并发写入进行序列化处理。因为一个领导者调解所有对id1的请求，它也是保护数据库不受雷鸣般的群组影响的理想位置。领头羊确保它不会向数据库发出并发的重叠查询，同时也对一个分片的最大待决查询数量进行了限制。

4.5 地理上的扩展

领导者和追随者的配置允许TAO扩展到处理高的工作负载，因为读取吞吐量与所有层中追随者服务器的总数成比例。然而，设计中隐含的假设是，从跟随者到领导者以及领导者到数据库的网络延迟很低。如果客户被限制在一个单一的数据中心，或者甚至是一组相距不远的数据中心，这个假设是合理的。然而，在我们的生产环境中，这并不是真的。

随着我们的社交网络应用的计算和网络需求的增长，我们不得不超越单一的地理位置：今天，我们的低层可以相隔数千英里。在这种构架下，网络往返时间很快就会成为整个架构的瓶颈。由于在我们的工作负载中，追随者的失读次数是写入次数的两倍25，所以我们选择了一个主/从架构，要求将写入发送到主站，但允许将写入的数据发送到主站。

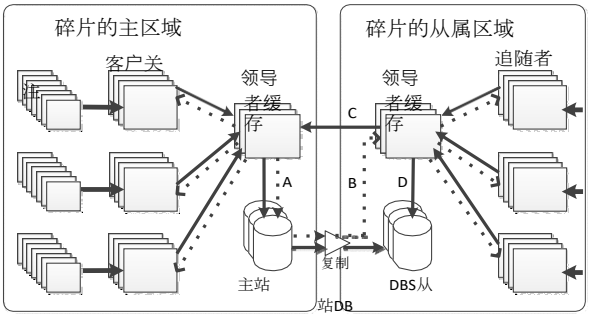


图2：多区域TAO的配置。主区域向主数据库（A）发送读取失误、写入和嵌入式一致性信息。当复制流更新从属数据库时，一致性消息被传递给从属领导（B）。从属领导向主领导（C）发送写信息，并向复制数据库（D）发送读失误。主站和从站的选择是针对每个分片单独进行的。

读取失误在本地得到服务。与领导者/追随者的设计一样，我们以异步方式传播更新通知，以最大限度地提高性能和可用性，但要牺牲数据的新鲜度。

社交图是紧密互联的；不可能对用户进行分组，从而使跨分区请求很少。这意味着每个TAO追随者都必须在本地拥有一个完整的社会图的多字节副本的数据库层。在每个数据中心提供完整的副本是非常昂贵的。

我们对这个问题的解决方案是选择数据中心的位置，这些位置只集中在几个*区域*，区域内的延迟很小（通常小于1毫秒）。这样，每个区域存储一份完整的社会图就足够了。图2显示了主/从TAO系统的整体架构。

跟随者在所有区域的行为都是相同的，将读取的失误和写入的内容转发给本地区域的领导者层。领导者查询本地区域的数据库，不管它是主站还是从站。然而，写是由本地领导者转发到拥有主数据库的区域中的领导者那里。这意味着读取延迟与区域间延迟无关。

主区域对每个分片都是单独控制的，并且自动切换以恢复数据库的故障。在切换过程中失败的写入会被报告给客户端，并不会被重试。请注意，由于每个缓存承载了多个分片，一台服务器可能同时是一个主站和一个从站。我们更倾向于将所有的主数据库放在一个单独的re-gion中。当一个逆向关联在不同的区域被掌握时，TAO必须穿越一个额外的区域间链接来转发逆向写入。

TAO在数据库复制流中嵌入了无效和重新填报信息。在事务被复制到从属数据库后，这些消息会立即在一个区域内传递。提前传递这些消息会造成缓存的不一致，因为从本地数据库中读取的数据是过时的。在Facebook，TAO和memcache使用相同的管道来传递无效信息和重填信息[21]。

如果一个转发的写是成功的，那么本地领导者将用新的值更新其缓存，即使本地从属数据库可能还没有被异步复制流更新。在这种情况下，追随者将从写中收到两个无效或重写，一个是写成功时发送的，一个是写的事务被复制到本地从属数据库时发送的。

TAO的主/从设计确保所有的读取都能在一个区域内得到满足，但代价是有可能向客户返回过时的数据。只要用户持续查询相同的追随者层，用户通常会对TAO状态有一个一致的看法。我们将在下一节讨论这方面的例外情况。

5 实施

前几节描述了TAO服务器是如何处理大量数据和查询率的。本节详细介绍了对性能和存储效率的重要优化。

5.1 缓存服务器

TAO的缓存层作为客户和数据库之间的中介。它积极地缓存对象和关联，以提供良好的读取性能。

TAO的内存管理是基于Facebook定制的memcached，如Nishtala等人[21]所述。TAO有一个板块分配器来管理同等大小的板块项目，一个线程安全的哈希表，同等大小的项目之间的LRU驱逐，以及一个动态的板块再平衡器来保持所有类型的板块之间的LRU驱逐年龄相似。一个板块项可以容纳一个节点或一个边缘列表。为了提供更好的隔离，TAO将可用的RAM划分为*竞技场*，通过对象或关联类型选择竞技场。这使得我们可以延长重要类型的缓存寿命，或者防止不良的缓存市民驱逐更好的类型的数据。到目前为止，我们只是手动配置了竞技场来解决特定的问题，但应该有可能实现自动的规模的竞技场，以提高TAO的整体命中率。

对于小的固定大小的项目，如关联计数，主哈希表中的桶状项目的指针的内存开销变得非常大。我们将这些项目分开存储，使用直接映射的8路关联缓冲区，不需要指针。在每个缓存中的LRU顺序

通过简单地将条目向下滑动来跟踪桶。我们通过增加一个表来实现额外的内存效率，该表将每个活动的atype映射到一个16位的值。这让我们可以在14个字节内将（id1, atype）映射为32位的计数；一个负的条目，即记录一个（id1, atype）没有任何id2的情况，只需要10个字节。这种优化使我们在给定的系统配置下，可以在缓存中多容纳大约20%的项目。

5.2 MySQL映射

回顾一下，我们将对象和关联的空间划分为分片。每个分片被分配到一个逻辑的MySQL数据库，该数据库有一个对象表和一个关联表。一个对象的所有字段都被序列化为一个单一的 "数据 "列。这种方法允许我们将不同类型的对象存储在同一个分片中，从不同的数据管理政策中受益的对象被存储在不同的自定义表中。

关联的存储方式与对象类似，但为了支持范围查询，它们的表有一个基于id1、type和time的额外索引。为了避免潜在的昂贵的SELECT COUNT查询，关联计数被存储在一个单独的表中。

5.3 缓存分片和热点

使用一致的散列法[15]将碎片映射到一个层中的缓存服务器上。这简化了层的扩展和请求路由。然而，这种半随机分配分片到缓存服务器的方式会导致负载不平衡：一些跟随者会比其他人承担更多的请求负载。TAO通过*分片克隆夹*重新平衡追随者之间的负载，其中对分片的读取由一个层中的多个追随者提供。克隆的分片的一致性管理信息被发送到所有托管该分片的跟随者。

在我们的工作负载中，一个受欢迎的对象被查询的频率比其他对象高几个数量级是很常见的。克隆可以将这种负载分配给许多跟随者，但是这些对象的高命中率使得将它们放在一个小型的客户端缓存中是值得的。当追随者对一个热点项目的查询做出反应时，它包括该对象或关联的访问率。如果访问率超过一定的阈值，TAO客户端会缓存数据和版本。通过在随后的查询中包括版本号，如果数据自上一个版本以来没有变化，追随者可以在回复中省略该数据。访问率也可以用来节制客户端对非常热的对象的请求。

5.4 高程度的物体

许多对象都有超过6,000个相同类型的关联从它们身上发出，所以TAO不缓存

完整的关联列表。同样常见的是，在执行soc get查询时，结果是空的（指定的id1和id2之间不存在边）。不幸的是，对于高等级的对象，这些查询会以各种方式进入数据库，因为被查询的id2可能在关联列表的未缓存尾部。

我们通过修改被观察到的发出有问题的查询的客户端代码来解决缓存实施中的这个缺陷。解决这个问题的一个办法是使用assoc count来选择查询方向，因为检查逆向的边是等价的。在某些情况下，如果一条边的两端都是高度节点，我们也可以利用应用域的知识来提高缓存能力。许多关联将时间字段设置为其创建时间，许多对象将其创建时间作为一个字段。由于一个节点的边缘只能在该节点被创建后才能被创建，我们可以将id2搜索限制在时间比对象的创建时间长的关联上。只要缓存中存在比对象更早的边，那么这个查询就可以由TAO的跟随者直接回答。

6 一致性和容错性

TAO的两个最重要的要求是可用性和性能。当故障发生时，我们希望能继续渲染Facebook，即使数据是陈旧的。在这一节中，我们将描述TAO在正常操作下的一致性模型，以及TAO如何在故障模式下牺牲一致性。

6.1 一致性

在正常操作下，TAO中的对象和关联最终是一致的[33, 35]；在写操作后，TAO保证最终向所有层交付无效或重新填充。如果有足够的时间让外部输入安静下来，TAO中的所有数据副本将是一致的，并反映出对所有对象和关联的所有成功写操作。复制滞后通常小于一秒。

在正常操作中（一个请求最多遇到一次故障），TAO在单一层内提供读写一致性。TAO用本地写入的值同步更新缓存，当写入成功时，主领导者会返回一个*变化集*。这个变化集通过从属领导（如果有的话）传播到发起写入查询的从属层。如果为一个关联配置了反向类型，那么对该类型的关联的写入可能会影响到id1的和id2的分片。在这种情况下，主领导者返回的变化集包含这两个更新，从属领导者（如果有的话）和转发写的追随者必须在返回给调用者之前将变化集发送到各自层的id2的分片上。

变更集不能总是安全地应用于跟随者的缓存内容，因为如果第二个跟随者的更新中的refill或invalidate尚未交付，那么跟随者的缓存可能是过时的。在大多数情况下，我们用持久性存储和缓存中的版本号来解决这个竞争条件。版本号在每次更新时都会被递增，所以如果变化集显示其更新前的值是过时的，追随者可以安全地使其数据的本地副本失效。版本号不会暴露给TAO客户端。在从属区域，这个方案容易受到缓存驱逐和存储服务器更新传播之间罕见的竞赛条件的影响。从属存储服务器可能持有比缓存服务器所缓存的数据更早的版本，所以如果更改后的条目被从缓存中驱逐，然后从数据库中重新加载，客户端可能会观察到一个值在单个从属层的时间上的倒退。这种情况只有在从属区域的存储服务器接收更新的时间长于缓存项目从缓存中被驱逐的时间时才会发生，而这种情况在实践中是很少见的。

虽然TAO不为其客户提供强一致性，但由于它同步写入MySQL，主数据库是一个一致的真相来源。这使我们能够为需要一致性的一小部分请求提供更强的一致性。TAO的读取可能被标记为*关键*，在这种情况下，它们将被代理到主区域。我们可以在认证过程中使用关键读数，例如，这样复制滞后就不允许使用过时的证书。

6.2 故障检测和处理

TAO可以扩展到多个图形位置的数千台机器，因此瞬时和永久性故障是很常见的。因此，TAO检测潜在的故障并绕过它们是很重要的。TAO服务器采用积极的网络超时，以避免继续等待可能永远不会出现的响应。每个TAO服务器维护每个目的地的超时，如果有几个连续的超时，就将主机标记为停机，并记住停机的主机，以便主动中止后续请求。这种简单的故障检测器工作得很好，尽管它并不总是在断电的情况下保持全部容量，例如限制TCP吞吐量的突发性数据包掉落。在检测到故障的服务器后，TAO以最大努力的方式绕过故障，以保持可用性和性能的一致性为代价。我们主动探测故障机器，以发现它们何时（如果）恢复。

数据库故障。在全局配置中，如果数据库崩溃，或因维护而停工，或从主数据库复制而落后太多，则数据库会被标记为停机。当一个

如果主数据库发生故障，它的一个从属数据库会自动晋升为新的主数据库。

当一个区域的从属数据库发生故障时，缓存缺失会被重定向到托管数据库主站的区域内的TAO领导。然而，由于缓存一致性信息被嵌入到数据库的复制流中，它们不能被主机制传递。在从属数据库停机期间，在主数据库上运行一个额外的binlog tailer，并在区域间传递重新填充和失效信息。当从属数据库重新启动时，中断时的无效和重填信息将再次被传递。

领导者失败。当一个领导者的缓存服务器发生故障时，跟随者会自动在它周围路由读和写请求。跟随者将未读请求直接转发到数据库。与此相反，对一个失败的领导者的写请求会被重新路由到领导者层的一个随机成员。这个替换的领导者执行写和相关的动作，比如修改反向关联和向跟随者发送无效信息。替换的领导者也会排队向原领导者发出异步失效，以恢复其一致性。这些异步失效被记录在协同节点上，并插入到复制流中，在那里它们被储存起来，直到领导者变得可用。如果失效的领导者是部分可用的，那么在领导者的一致性被恢复之前，后面的人可能会看到一个陈旧的值。

补发和废止的失败。领导者以异步方式发送补发和失效信息。如果追随者无法到达，领导者会将消息排入磁盘，以便在稍后的时间交付。需要注意的是，如果这些消息由于每个领导者的失败而丢失，那么跟随者可能会被留下陈旧的数据。这个问题可以通过一个批量失效操作来解决，该操作会使一个分片id的所有对象和as-sociations失效。在一个失败的领导盒被替换后，所有映射到它的分片必须在跟随者中被无效化，以恢复一致性。

追随者失败。在TAO追随者失败的情况下，其他层的追随者分担为失败主机的碎片服务的责任。我们为每个TAO客户端配置了一个主从层和备份从层。在正常操作中，请求只被发送到主服务器。如果为某一特定请求托管分片的服务器由于超时而被标记为停机，那么请求将被发送到备份层的分片服务器上。因为故障转移请求仍然会被发送到托管相应分片的服务器上，它们是完全可缓存的，不需要额外的一致性工作。来自客户端的读和写请求以同样的方式进行故障转移。请注意，在不同层之间的故障转移可能会导致写后读的一致性被违反，如果读在写的重新填满或无效之前到达故障转移目标。

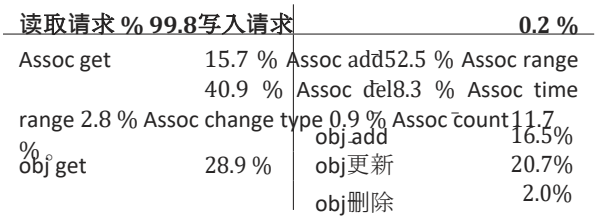


图3：所有Facebook产品对TAO的客户端请求的相对频率。读取几乎占了对API的所有调用。

7 生产工作量

Facebook有一个生产中的TAO实例。像TAO这样的多租户系统使我们能够摊薄运营成本，并在客户之间分享多余的能力。它也是快速产品创新的一个重要因素，因为新的应用程序可以与现有的数据相连接，而且当一个应用程序从一个用户增长到数亿用户时，不需要移动数据或pro- vision服务器。多租户对对象特别重要，因为它允许统一处理整个64位id空间，而不需要额外的步骤来解决otype。

TAO系统包含许多追随者层，分布在几个地理区域。每个地区都有一套完整的数据库，一个领导者缓存层，以及至少两个跟随者层。我们的TAO部署每秒钟持续处理10亿次的读取和数百万次的写入。我们不知道是否有另一个如此规模的地理分布式图形数据存储。

为了描述TAO的工作量，我们在40天内随机抽取了一百万个6.5请求。在本节中，我们将描述对该样本的分析结果。

在高层次上，我们的工作负荷显示出以下特点。

- 读取比写入要频繁得多。
- 大多数边缘查询的结果为空；以及
- 查询频率、节点连通性和数据大小具有长尾的分布。

图3分解了TAO上的负载。读取占主导地位，只有0.2%的请求涉及写入。大多数关联读取的结果是空的关联列表。对assoc get的调用只有19.6%的时间找到了关联，在我们的跟踪中，对assoc range的调用有31.0%的结果不是空的，对assoc time range的调用只有1.9%返回任何边。

图4显示了assoc计数的返回值的分布。45%的调用返回0。在非零值中，虽然小值是最常见的，但有1%的返回值≥500,000.

图中显示5的是，在中国境内，有多少人是被派往国外的？

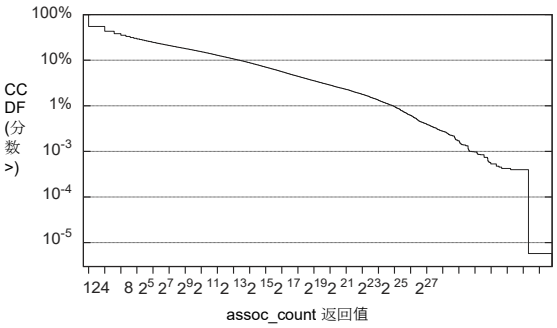


图4：我们生产环境中的assoc计数频率。1%的返回计数为≥512K。

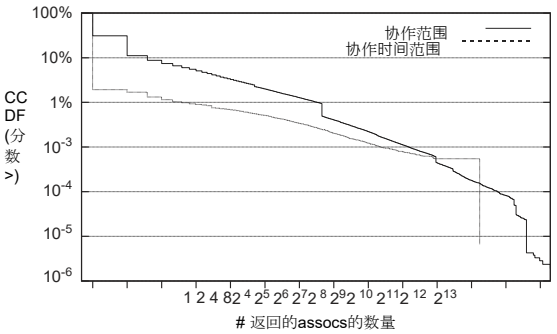
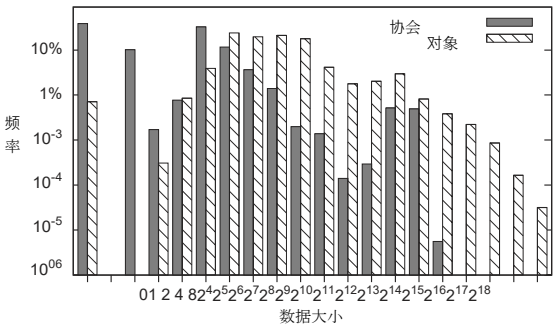


图5：assoc范围和assoc时间范围查询所返回的边的数量。64%的非空的结果有边1，其中13%的边的限制为1.

在范围和时间范围查询中返回的关联，以及在返回的关联中达到极限的子集。大多数范围和时间范围的查询有很大的客户提供的限制。12%的查询有限制=但1, 其余95%的查询有限制 少1000. ≥
在有限制的查询中，超过1%的返回值实际1达到了限制。

尽管对不存在的关联的查询很常见，但对于对象来说却不是这样的。一个有效的ID只在对象创建过程中产生，所以obj get只能在对象被删除或者对象的创建还没有被复制到当前区域的情况下返回一个空结果。在我们的追踪中，这两种情况都没有发生；每个对象的读取都是成功的。这并不意味着对象从未被删除--它只是意味着从来没有尝试过读取一个被删除的对象。

图6显示了TAO查询结果的数据大小分布。39.5%的客户查询到的关联不包含任何数据。我们的实现允许对象存储1MB的数据，关联存储64K的数据（尽管必须为存储超过字节255的数据的关联配置一个自定义表）。



大多数对象和关联的实际大小要大得多。

图6：TAO API返回的关联和对象中存储的数据大小。关联通常比对象存储的数据少得多。在返回的60.5%的关联中，平均关联数据大小为97.8字节，其中有一些数据。对象的平均数据大小为字节673。

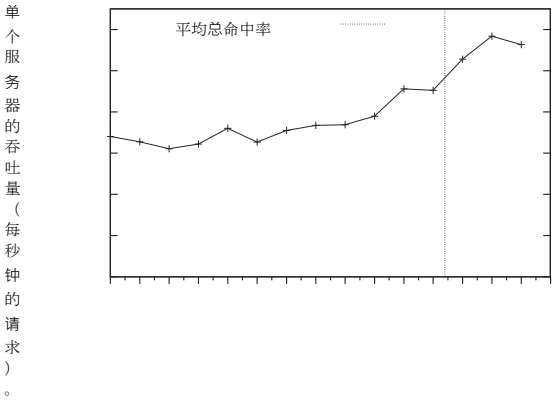


图7：在我们的生产环境中，单个追随者的吞吐量。缓存未命中和写入比缓存命中更昂贵，所以峰值查询率随着命中率上升。写入作为非命中请求包括在此图中。

较小。然而，大数值的出现足够频繁，系统必须有效地处理它们。

8 业绩

为整个Facebook运行单一的TAO部署，使我们能够从规模经济中获益，并使新产品能够很容易地与社交图的现有部分整合。在本节中，我们将报告TAO在实际工作负载下的性能。

可用性。在90天的时间里，从网络服务器上测得的TAO查询失败的比例是4。910^{-6}在解释这个数字时必须小心，因为一个TAO查询的失败可能会阻止客户发出另一个查询。

在第一种情况下，动态数据的依赖性。TAO的故障也可能与其他依赖系统的故障相关。

×

	命中时间。	*(msec)miss*
lat.(msec)	*操作50%平均*	*99%50%平均*
99%assoc计数	1.12.528.9	5.026.2
	186.8	
拨款获得	1.02.4	25.95.814.5
143.1		
角色范围	1.12.3	24.85.411.2
93.6		
拨款时间范围	1.33.2	32.85.811.9
47.2		
obj get	1.02.4	27.08.275.3
186.4		

图8：客户观察到的TAO读取请求的延迟（以毫秒为单位），包括客户API开销和网络工作穿越，按高速缓存命中和高速缓存失误分开。

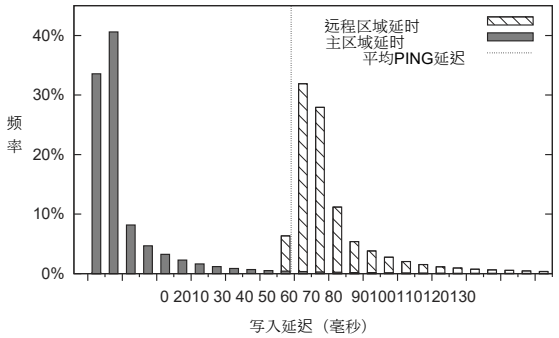


图9：来自与数据库主站在同一区域的客户的写入延迟，以及来自msec58以外区域的写入延迟。

追随者的能力。追随者的峰值吞吐量取决于其命中率。图7显示了我们在生产中观察到的最高的15分钟平均吞吐量，我们目前的硬件配置有144GB的内存，英特尔2至强核心E5-82660 CPU运行在2.2Ghz的超线程，以及10千兆以太网。

命中率和延时。作为第7节中描述的数据收集过程的一部分，我们测量了客户端应用程序的延迟；这些测量包括所有的网络延迟和穿越PHP TAO客户端堆栈的时间。所有地区的请求都以相同的速率取样。TAO的总体读取命中率为96.4%。图8显示了客户端观察到的读取延迟。obj_get的失误延迟比其他读取高，因为对象通常有更多的数据（见图6）。

TAO的写操作是同步进行的，所以从其他地区的写操作包括一个区域间的往返。图9比较了两个相距58.1毫秒的数据中心的延迟（平均往返时间）。在与主站同一区域的平均写延迟为12.1毫秒；在远程区域的平均写延迟为=74.4+58.1毫秒16.3。

复制滞后。TAO的区域间异步复制写入是一种设计上的权衡，它更倾向于

读取性能和吞吐量的一致性。我们观察到，在85%的追踪过程中，TAO的从属存储服务器落后于它们的主机不到一秒1，99%的时间不到几秒3，99.8%的时间不到10秒。

故障转移。当领导者不可用时，追随者缓存直接联系数据库；在我们的样本中，有0.15%的追随者缓存失误使用了这种故障转移路径。写入请求的故障转移包括将这些请求委托给一个随机的领导者，这发生在0.045%的关联和对象写入中。由于计划内的维护或计划外的停机，从属数据库有0.25%的时间被提升为主数据库。

9 相关工作

TAO是一个地理分布式的最终一致的图存储，为读取进行了优化。以前的分布式系统工作已经探索了松弛一致性、图数据库和读优化的存储。据我们所知，TAO是第一个在一个单一系统中大规模结合所有这些技术的系统。

最终一致性。Terry等人[33]描述了最终一致性，即TAO所使用的宽松的一致性模型。Werner将读写一致性描述为最终一致性的一些变体的属性[35]。

地理上的分布式数据存储。Coda文件系统使用数据复制来提高性能和可用性，以应对缓慢或不可靠的网络[29]。与Coda不同，TAO不允许在系统断开的部分进行写入。

Megastore是一个存储系统，它在地理分布的数据中心使用Paxos来提供强大的一致性保证和高可用性[5]。Span- ner是Google在Megastore之后开发的下一代全球分布式数据库，它引入了时间API的概念，暴露了时间的不确定性，并利用它来提高提交的吞吐量，为读取提供快照隔离[8]。TAO解决了一个非常不同的用例，没有提供一致性保证，但处理的请求多出很多数量级。

分布式哈希表和键值系统。非结构化的键值系统是一种有吸引力的扩展分布式存储的方法，因为数据可以很容易地被分区，而且分区之间不需要什么通信。亚马逊的Dynamo[10]展示了它们如何被用于构建灵活和强大的商业系统。从Dynamo中获得灵感，LinkedIn的Voldemort[4]也实现了一个分布式的键值存储，但是是为社交网络服务的。TAO接受了比Dynamo更低的写入可用性，以换取避免多主机冲突解决所带来的编程复杂性。键值存储的简单性

从Facebook对memcache的使用中可以看出，价值存储也允许积极的性能选择[21]。分布式哈希表的许多贡献都集中在路由方面[28, 2432,25,]。Li等人[16]描述了DHTs的性能，而Dabek等人[9]专注于在广域网中设计DHTs。TAO利用了我们的数据中心安置所带来的集群间延迟的层次性，并假设了一个有少量成员或集群的连接环境。拓扑结构的变化。

许多其他的工作都集中在键值存储所提供的一致性标准上。Gribble等人[13]通过利用两阶段提交提供了一个缓存数据的一致性视图。Glendenning等人[12]建立了一个可线性化的键值存储，可以容忍流失。Sovran等人[31]实现了地理复制的交易。

COPS系统[17]通过跟踪客户端上下文访问的所有键的所有挂起，在一个高可用的键值存储中提供因果一致性。Eiger[18]通过跟踪列族数据库中待定操作之间的冲突来改进COPS。如果可以提高每台机器的效率，Eiger中使用的技术可能适用于TAO。

分层连接。Nygren等人[22]描述了Akamai内容缓存如何通过将边缘集群分组为区域组，共享一个更强大的 "父 "集群来优化延迟，这与TAO的跟随者和领导者层类似。

结构化存储。TAO遵循最近的趋势，从关系型数据库转向结构化存储方法。虽然是松散的定义，这些系统通常提供比传统ACID属性更弱的保证。谷歌的BigTable[6]、雅虎的PNUTS[7]、亚马逊的SimpleDB[1]和Apache的HBase[34]都是这种更具扩展性的方法的例子。这些系统都在每记录或行层面上提供了与TAO的对象和关联语义类似的一致性和交易，但没有提供TAO的读取效率或图语义。Es- criva等人[27]描述了一个*可搜索的*键值存储。Redis[26]是一个内存存储系统，提供了一系列的数据类型和一个表达式的API，用于完全在内存中的数据集。

图谱服务。由于TAO是专门为社会图谱服务而设计的，因此它与现有的图谱数据库作品有共同的特点也就不足为奇了。Shao和Wang的Trinity努力[30]在内存中存储其图结构。Neo4j[20]是一个流行的开源图数据库，它提供了ACID语义和在多台机器上分片数据的能力。Twitter也使用其FlockDB[11]来存储其社交图的一部分。据我们所知，这些系统的规模都不足以支持Facebook的工作负荷。

Redis[26]是一个具有丰富选择的键值存储。

价值类型，足以有效地实现对象和关联的API。然而，与TAO不同的是，它要求数据集完全保存在内存中。Redis的复制是只读的，所以如果没有像Nishtala等人的远程标记这样的高级系统，他们就不能提供读写一致性[21]。

图形处理。TAO目前不支持高级图形处理API。有几个系统试图支持这样的操作，但它们不是为了直接接收来自客户端应用程序的工作负载而设计的。PEGASUS[14]和雅虎的Pig Latin[23]是在Hadoop之上对图进行数据挖掘和分析的系统，PEGASUS专注于peta规模的图，而Pig Latin专注于更有表达力的查询语言。同样，谷歌的Pregel[19]也处理了很多相同的图分析问题，但使用了自己的更具表达力的作业分配模型。这些系统专注于大型任务的吞吐量，而不是大量的更新和简单查询。Facebook也有类似的大型平面图处理系统，对从TAO的数据库复制的数据进行操作，但这些分析工作并不在TAO本身执行。

10 总结

总的来说，本文有三个贡献。首先，我们描述了一个具有挑战性的Facebook工作负载：需要高吞吐量、低延迟地读取访问大型、不断变化的社交图的查询。其次，我们描述了Facebook社交图的对象和关联数据模型，以及为其服务的API。最后，我们详细介绍了TAO，我们的地理分布式系统，实现了这个API。

TAO是在Facebook内部大规模部署的。缓存和持久化存储的分离使得这两层可以独立设计、扩展和运行，并在我们的组织中最大限度地重复使用组件。这种分离也允许我们在这两层中选择不同的效率和一致性的权衡，并使用一个empotent的缓存失效策略。事实证明，TAO的限制性数据和一致性模型对我们的应用开发者来说是可用的，同时允许高效和高可用性的实现。

鸣谢

我们要感谢Rajesh Nishtala、Tony Savor和Barnaby Thieme阅读了本论文的早期版本并做出了许多改进。我们感谢许多Facebook的工程师，他们建立、使用并扩展了对象和关联API的原始实现，为我们提供了设计见解和工作负荷，从而促成了TAO。也感谢我们的审稿人和我们的牧羊人Phillipa Gill的详细评论。

参考文献

[1] 亚马逊SimpleDB。http://aws.amazon.com/simpledb/。

[2] Facebook - 公司信息。http://newsroom.fb.com。

[3] LevelDB。https://code.google.com/p/leveldb。

[4] 伏地魔计划。http://project-voldemort.com/。

[5] J.Baker, C. Bond, J. C. Corbett, J. Furman, A. Khorlin, J. Larson, J-M.Leon, Y. Li, A. Lloyd, and V. Yushprakh.Megastore。为互动服务提供可扩展的、高度可用的存储。*在创新数据系统再搜索会议论文集*，CIDR。2011.

[6] F.Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M.Burrows, T. Chandra, A. Fikes, and R. Gruber.Bigtable。一个结构化数据的分布式存储系统。在*第七届USENIX操作系统设计与实施研讨会的论文集中*，OSDI。USENIX Assoc, 2006.

[7] B.F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P.Bohannon, H.-A.Jacobsen, N. Puz, D. Weaver, and R. Yerneni.PNUTS。雅虎的托管数据服务平台。*PVLDB*，1(2), 2008.

[8] J.C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W.Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D.Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M.Szymaniak, C. Taylor, R. Wang, and D. Woodford.Spanner:谷歌的全球分布式数据库。在*第十届USENIX操作系统设计和实施会议论文集*，OSDI，美国加州伯克利。2012.

[9] F.Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris.设计一个低延迟和高吞吐量的DHT。在*第一届网络系统设计和实施研讨会上*，NSDI，美国加州伯克利，美国。2004.

[10] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels。Dynamo:亚马逊的高可用键值存储。在*第21届ACM操作系统原理研讨会论文集*，SOSP，美国纽约。2007.

[11] FlockDB 。 http://engineering.twitter.com/2010/05/introducing-flockdb.html。

[12] L.Glendenning, I. Beschastnikh, A. Krishnamurthy, and T. Anderson.Scatter中的可扩展一致性。在*第23届ACM操作系统原理研讨会上*，SOSP，美国纽约，纽约。2011.

[13] S.D. Gribble, E. A. Brewer, J. M. Hellerstein, and D. Culler.用于互联网服务构建的可扩展、分布式数据结构。In *Proceedings of the 4th Symposium on Operating System Design and Implementation*, OSDI, Berkeley, CA, USA, 2000.

[14] U.Kang, C. E. Tsourakakis, and C. Faloutsos.PEGASUS: mining peta-scale graphs.*Knowledge Information Systems*, 27(2), 2011.

[15] D.Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, 和 D.Lewin.Consistent Hashing and Random trees:缓解万维网热点的分布式缓存协议.在*第29届ACM年度计算理论研讨会论文集中*，STOC。1997.

[16] J.Li, J. Stribling, T. M. Gil, R. Morris, and M. F. Kaashoek.竞争分布式哈希表在流失情况下的性能。在*第三届点对点系统国际会议论文集*，IPTPS，柏林，海德堡。2004.

[17] W.Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen.不要满足于最终：使用COPS的广域存储的可扩展因果一致性。In T. Wobber and P. Druschel, editors, *Proceedings of the 23rd ACM Symposium on Operating System Design and Implementation*, SOSP.ACM, 2011.

[18] W.Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. 用于低延迟地理复制存储的更强语义。在*第10届USENIX网络系统设计和实施会议上*，NSDI。 2013.

[19] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn,
N.Leiser, and G. Czajkowski.Pregel: a system for large-scale graph processing.In A. K. Elmagarmid and D. Agrawal, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data*.ACM。 2010.

[20] Neo4j.http://neo4j.org/。

[21] R.Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T.Tung, and V. Venkataramani.在Facebook扩展Memcache。在*第10届USENIX网络系统设计与实施会议的论文集中*，NSDI。 2013.

[22] E.Nygren、R. K. Sitaraman和J. Sun。Akamai网络：高性能互联网应用的平台。*SIGOPS操作系统评论*，44（3），8月。 2010.

[23] C.Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins.Pig latin: a not-so-foreign language for data processing.In J. T.-
L.Wang, editor, *Proceedings of the ACM SIGMOD International Conference on Management of Data*.ACM, 2008.

[24] V.Ramasubramanian和E. G. Sirer.Beehive:O(1)的查找性能，用于点对点覆盖中的幂律查询分布。在*第一届网络系统设计与实施研讨会论文集中*，NSDI，美国加州伯克利市。 2004.

[25] S.Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker.一个可扩展的内容可寻址网络。在*2001年计算机通信的应用、技术、架构和程序协议会议上*，SIGCOMM，美国纽约，纽约。 2001.

[26] Redis。http://redis.io/。

[27] E.G. S. Robert Escriva, Bernard Wong.Hyperdex:用于云计算的分布式、可搜索键值存储。技术报告，康奈尔大学计算机科学系，伊萨卡，纽约，12月 2011.

[28] A.I. T. Rowstron和P. Druschel。Pastry:可扩展的、分散的对象定位，以及大规模点对点系统的路由。In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, Middleware, Long-don, UK, UK, 2001.

[29] M.Satyanarayanan.The evolution of coda.*ACM Transactions on Computer Systems*, 20(2), May 2002.

[30] B.Shao 和 H. Wang 。 三 位 一 体 。 http://research.microsoft.com/en- us/projects/trinity/。

[31] Y.Sovran, R. Power, M. K. Aguilera, and J. Li.地域复制系统的事务性存储。*第23届ACM操作系统原理研讨会论文集*，SOSP，美国纽约。 2011.

[32] I.Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakr-ishnan.Chord:用于互联网应用的可扩展点对点查询服务。In *Proceedings of the 2001 conference on Appli- cations, technologies, architectures, and protocols for computer communications*, SIGCOMM, New York, NY, USA, 2001.

[33] D.B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser。管理Bayou中的更新冲突，一个弱连接的复制存储系统。在*第15届ACM操作系统原理研讨会论文集中*，SOSP，美国纽约，纽约。 1995.

[34] 阿帕奇软件基金会。http://hbase.apache.org。 2010.

[35] W.沃格尔斯最终一致。*Queue*, 6(6), Oct. 2008.