arXiv:2301.10440v1 [cs.DB] 25 Jan 2023

调度空间扩展器：数据摄取查询并发控制的扩展

Sho Nakazono NTT计算与数据科学实验室 东京，日本
syou.nakazono.nu@hco.ntt.co.jp

Hiroyuki Uchiyama Recruit Co., Ltd. 东京，日本
hiroyuki-uchiyama@recruit.co.jp

Yasuhiro Fujiwara NTT通信科学实验室 神奈川，日本
yasuhiro.hujiwara.kh@hco.ntt.co.jp

Hideyuki Kawashima 庆应义塾大学环境与信息研究学院，神奈川，日本
river@sfc.keio.ac.jp

摘要

随着传感设备和物联网应用的不断发展，数据库系统需要处理频繁更新传感器数据的数据摄取查询。为了在保证事务正确性的同时处理数据摄取查询，我们提出了一种新颖的协议扩展方法——调度空间扩展器（SSE）。SSE 的核心思想是：如果某次更新已经过时且不再必要，就可以安全地省略该更新。SSE 在传统协议中增加了另一条控制流，用于检验擦除版本顺序的事务正确性；该顺序假设某个事务的所有更新均已过时且不再必要。此外，我们还提出了 SSE 的一种优化方案，称为基于纪元的SSE（ESSE），它比 SSE 更高效地生成、检验和维护擦除版本顺序。我们的方法提高了数据摄取查询的处理效率。实验结果表明，在一台 144 核机器上，我们为 Silo 和 MVTO 扩展的 ESSE 在 TATP 基准测试上的性能分别提升了 2.7× 和 2.5× 倍，而在 TPC-C 基准测试上的性能则与原始协议相当。

1 引言

现代物联网（IoT）或移动电信服务利用数十亿个传感器、联网汽车或移动设备，通过网络持续摄取来自这些来源的数据。这些服务读取实时摄取的数据，以操作现实世界中的执行器，例如装配厂机器、机器人高速公路施工标志或基于位置的数字标牌 [7]。数据摄取查询与实时操作的组合正成为网络服务提供商中的常见流程。这些查询长期以来一直在流数据库等非事务系统中运行 [13, 19]。然而，近年来，事务正确性 的重要性已得到研究 [1, 28, 39], , 因为读取

允许为个人或课堂使用而免费制作本作品的全部或部分数字或纸质副本，前提是副本不得用于或分发以谋取利润或商业利益，且副本须附上本声明和首页上的完整引文。对于本作品中不属于 ACM 所有的组件，必须尊重其版权。注明出处后可以摘录。若要以其他方式复制、重新发布、发布到服务器或分发至列表，须事先获得明确许可和/或支付费用。请通过 permissions@acm.org 申请许可。*Conference'17*，*2017*年7月，美国华盛顿特区 © 2023 计算机协会。
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

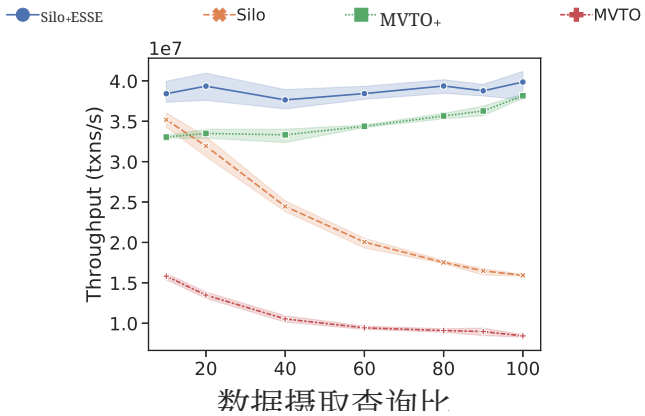


图1：TATP基准测试的吞吐量与数据摄取查询比的关系。通过改变 TATP 中 UPDATE_LOCATION 所占的百分比来表示数据摄取查询比，并使用 144 个工作线程测量吞吐量。原始百分比为 14%。每条线的带状区域表示实验进行五次所得结果的方差。

不一致或过时的数据会导致现实世界中的执行器发生故障、事故或机会损失 [6, 7, 28] 。

电信应用事务处理基准测试（TATP） [27, 30] 是事务型工作负载的一个示例，其中包含数据摄取和实时操作查询。在 TATP 中，大量移动设备会更新用户当前的位置信息或用户资料。与此同时，电信基站会将已摄取的数据作为在线事务读取，以执行归属位置寄存器数据库的迁移。TATP 中数据摄取查询的一个关键特征是**盲更新**。这不是读-修改-写；每台移动设备都会更新自己的位置信息，但不需要读取该设备自己之前写入的信息。此外，这也不是插入；我们甚至无法将该查询视为“带唯一时间戳的插入”。尽管这种方法适用于时间序列分析系统，但对于实时操作，我们需要确保每个数据项只有一个最新版本，并保证多个数据项的结果一致 [1, 7, 28]。

协议	事务正确性	省略写操作	版本存储
采用 Thomas 写规则（TWR）的时间戳排序（T/O）	并非严格可串行化	**Yes**	1VCC
Silo OCC	**严格可串行化**	No	1VCC
Cicada MVTO	非严格可串行化	No	MVCC
Silo + ESSE	**严格可串行化**	**Yes**	1VCC
MVTO +ESSE	**严格可串行化**	**Yes**	MVCC

表1：传统协议与我们提出的方法之间的差异

盲更新是我们所针对的应用与传统应用之间最显著的区别。传统工作负载（例如用于建模批发仓库管理的 TPC-C [38],）也会生成大量写入，例如新订单，但这些工作负载请求的是插入，而不是盲更新。插入可以通过分区实现扩展，因为它们写入不同的数据项；但更新原则上必须写入同一个数据项，因此难以扩展。在我们的实验中，随着数据摄取查询速率的提高，现有协议在 TATP基准测试中的性能会下降，如图1所示。这是因为为了保证事务正确性，它们必须使用锁机制，将针对同一数据项的更新请求按顺序串行执行。这种更新的串行执行会导致性能下降。如果吞吐量低于数据速度，我们就无法运行物联网/电信应用的系统和服务。

为了实时处理海量数据，可以通过负载削减或反压[13, 14], 在不使用锁机制的情况下**省略**更新。然而，这些方法无法充分保证事务正确性。例如，当传感器更新两个相邻移动对象的信息，但系统部分省略其中一个对象的信息时，执行器只能获取一个移动对象的数据，因此某些执行器可能会引发碰撞等事故。为了保证事务正确性，数据库需要使用并发控制（CC）协议。CC协议通过确保事务正确性的两个基本属性，处理并发操作的交错执行：可串行化（保证获得一致的数据快照） [12] 和线性化（保证获得非陈旧数据快照） [21]。理论上，我们可以通过找到与所有更新相关的版本顺序 [2] ，来判断省略某项更新是否符合正确性要求。如果找到版本顺序，且其正确性经协议验证，那么在保证事务正确性的同时，我们可以跳过加锁、缓冲区更新和持久化日志记录。然而，据我们所知，现有方法都没有利用版本顺序这一概念。这是因为朴素方法需要基于所有事务和所有可能的版本顺序，对所有可能的依赖图进行代价高昂的无环性检查，而这已被证明是NP完全问题 [12]。

本文提出了一种通用的协议扩展方法——**调度空间扩展器（SSE）**。SSE 的贡献有三方面。

C1: SSE 通过仅测试由 SSE 的数据结构生成的单个**擦除版本顺序**，将验证成本降低至多项式时间。借助擦除版本顺序，验证只需针对并发事务的单个子图进行。

SSE 还保留传统协议中的版本顺序及其测试算法；如果未找到擦除版本顺序，或其未通过正确性测试，SSE 就会将控制流交还给传统协议。因此，SSE 可以省略更新，但不会直接中止任何事务。这表明，SSE 只是扩展了传统协议的调度空间。

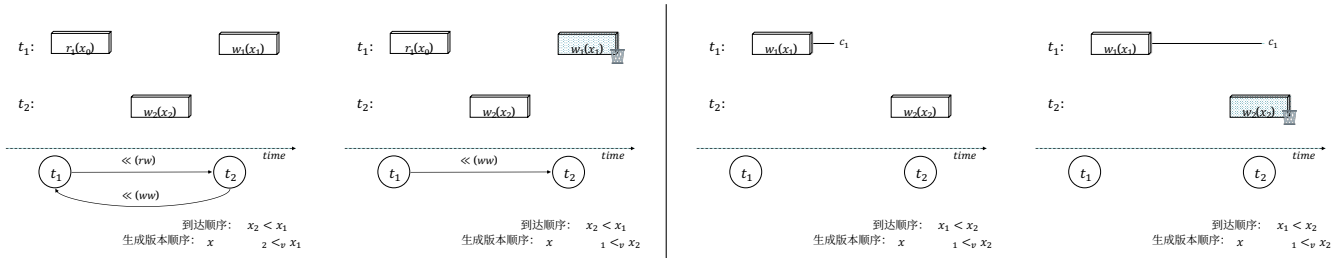
C2: 我们开发了**基于纪元的SSE（ESSE）**，为 SSE 引入优化。借助纪元框架，ESSE 减少了 SSE 验证中目标事务的数量，并将这些目标事务的足迹编码到一个 64 位数据结构中。因此，经 ESSE 扩展的协议能够生成擦除版本顺序，并以无闩锁方式执行正确性测试。如果事务通过测试，就可以通过少量原子操作（例如比较并交换）省略其写操作。

C3: 我们证明了 SSE 和 ESSE 适用于各种协议。我们将 ESSE 应用于两种最先进的 1VCC 和多版本并发控制协议（ Silo [11] 和基于Cicada的MVTO [26]），随后在 144 核环境中评估其在 TATP、YCSB [5], 和 TPC-C 基准测试上的性能。图1表明，ESSE 成功缓解了数据摄取查询的性能问题，并提升了 TATP 基准测试上的性能。这是因为 ESSE 恰当地省略了不必要的版本，如实验所示（第6节 6.1 中的图 8c）。表 1 展示了传统协议与我们提出的方法之间的差异。ESSE 扩展了 Silo 和 MVTO 等多种最先进的协议，使其能够省略写操作，同时保持事务正确性。

本文其余部分的组织如下。第2节介绍预备知识。第3节提出可安全省略的事务这一概念及其正确性测试算法。第4节介绍调度空间扩展器（SSE）方案，该方案能够生成可安全省略的事务。第5节介绍 ESSE，这是基于纪元框架针对 SSE 的优化技术。第6节报告对所提方案的评估结果。最后，第7节介绍相关工作，第8节总结全文。

2 预备知识

我们主要采用源自 Weikum 等人 [12] 的记法。表 2 展示了常用的符号和记法。设 $t_i, x_i, w_i(x_i)$ 和 $r_i(x_j)$ 分别为第 i 个事务、数据项 x 的一个版本、写操作和读操作。t_i具有一个有序的操作集合。$w_i(x_i)$ 表示 t_i写入 x_i。 $r_i(x_j)$ 表示 t_i读取 x_j。 ws_i



(a) $x_2 <_v x_1$**：违反可串行化** **(b)** $x_1 <_v x_2$**：可安全省略** **(c)** $x_2 <_v x_1$**：违反线性化** **(d)** $x_2 <_v x_1$**：可安全省略**

图 2：具有版本顺序的调度示例。图中用虚线表示的操作带有废纸篓标记，表示省略了其版本。只有情况 (b) 和 (d) 存在可安全省略的事务（即能够确保未发布事务的事务正确性）。

记法	定义
t_i	第 i 个事务；一个有序操作集
x_i	数据项 x 的一个版本
$w_i(x_i)$	写操作； t_i 写入 x_i
$r_i(x_j)$	读操作； t_i 读取 x_j
c_i	t_i的提交操作
a_i	t_i的中止操作
rs_i	由 t_i读取的版本集合
ws_i	由 t_i写入的版本集合

表2：常用符号和记法

并且 rs_i 分别表示 t_i 的读操作和写操作集合。 c_i和 a_i分别表示 t_i 的终止操作：提交和中止。

2.1 事务正确性

我们将预期服务的**事务正确性**定义为可恢复性 [20] 和严格可串行化 [21]。我们假设任何事务都不会读取未提交事务写入的值。此约束可确保可恢复性。严格可串行化由可串行化和线性化 [21]组成。可串行化对于为预期的物联网/电信应用提供实时操作所需的一致数据快照至关重要，即读取结果中不存在不一致或不完整的更新。在多种可串行化概念中，我们采用多版本视图可串行化（MVSR），因为这一性质提供了最大的调度空间 [12]。线性化指非并发事务之间的墙上时钟排序约束。如果数据库保证线性化，就能防止过时读写，即读取和写入过时版本。

Bernstein 等人提出了多版本串行化图（MVSG） [2]，并证明一个调度是多版本视图可串行化，当且仅当存在一个无环多版本串行化图。多版本串行化图的节点对应调度中所有已提交的事务。边由给定的调度和该调度的版本顺序 添加。版本顺序有两种类型：数据项的版本顺序和调度的版本顺序。调度的版本顺序是所有数据项版本顺序的并集。当 x_i 在调度中先于 x_j 时，我们记作 $x_i <_v x_j$，并称其为数据项的版本顺序。给定调度的版本顺序后，对于每组三个互不相同的操作 $w_j(x_j)$、 $r_i(x_j)$ 和 $w_k(x_k)$，其中 $t_i \neq t_k \neq t_j$。边有三种类型：(1) $t_j \xrightarrow{wr} t_i$表示 t_j写入一个版本 x_j，而 t_i读取该版本。(2) 如果 $x_j <_v x_k$，则 $t_i \overset{\ll}{(rw)\longrightarrow} t_k$表示 t_i读取一个版本 x_j，且其版本顺序先于 t_k的版本x_k。(3) 否则， $t_k \overset{\ll}{(ww)\longrightarrow} t_j$表示 t_k写入一个版本 x_k，且其版本顺序先于 t_j的版本 x_j。请注意，多版本串行化图的原始记法[1] 不包含事务顺序的依赖类型。当然，Bernstein 的定义并没有问题。不过，我们引入上述记法，以便阐明正确性定理的证明。

[1]在原始的多版本串行化图定义中， [2],所有边都表示为 $\rightarrow$，并且调度以及每个数据项的版本顺序采用相同的记法 $\ll$；因此，当 x_i 先于 x_j时，记作“$x_i \ll x_j$ in $\ll$”。

2.2 数据摄取查询与 WRITE 省略技术

数据摄取查询用于汇总物联网/电信应用中来自传感器和移动设备的更新。这类查询长期以来一直在流数据库等非事务系统中受到讨论 [13, 14]。如果数据摄取查询用于历史分析，则不必采用事务处理。然而，如果数据被用于操控现实世界中的执行器的实时操作，我们就需要使用并发控制协议，因为此类操作要求具备事务正确性 [1, 28]。遗憾的是，对于写竞争工作负载（例如数据摄取查询），研究表明，现有的先进并发控制协议无法实现较高的吞吐量 [40, 41]，因为它们需要通过加锁来满足可串行化。

为解决数据摄取查询的性能下降问题，非事务型流数据库会采用负载削减等写入省略技术 [13]。在事务处理中，这类写入省略技术被称为托马斯写规则（TWR） [36],，它是时间戳排序（T/O）并发控制协议的一项优化规则。借助 TWR，当事务的时间戳小于已安装写入的数据项 x 的时间戳 x时，该事务可以避免安装对该数据项的写入。然而，目前尚不清楚这种省略是否满足事务正确性，也不清楚该规则能否应用于其他现代协议。

3 可安全省略的事务 NS

本节中，我们将介绍任何协议都可使用写入省略技术的**安全可省略事务**定义，以及基于多版本串行化图的验证算法。

我们首先给出以下定义：

定义 1（未发布）。未发布事务是指未执行将其写集安装到存储中并记录日志的事务。

定义 2（可安全省略）。未发布事务 t_j 若 c_j 不影响正确性，则称为可安全省略。

定义 2 的关键在于，事务无需发布其写集即可提交。并发事务和未来事务都必须不能读取它的版本。因此， t_j 对其他事务而言并非必要；对于可安全省略的事务，我们可以跳过缓冲区更新和持久化日志记录。

要判断事务 t_j 是否可安全省略，我们需要验证其正确性。这可以通过多版本串行化图的概念来实现。具体而言，我们必须为该调度生成一个版本顺序，然后检验 1）多版本串行化图是否无环，以及 2）非并发事务之间的墙上时钟顺序。图 2 用两个示例调度和四种版本顺序说明了这些检验。可安全省略的事务以灰色显示，未发布的更新则以垃圾箱标记。(a)-(b) 和 (c)-(d) 两组分别具有相同的调度，但版本顺序和事务生命周期不同。这些差异导致 (b) 和 (d) 包含可安全省略的事务，而 (a) 违反可串行化，(c) 违反线性化。请注意，操作按照图中从左到右所示的墙上时钟时间顺序到达，但我们绘制多版本串行化图时采用的版本顺序并不取决于到达顺序。

可串行化。在（a）和（b）中， t_1 对 x 执行读-修改-写操作（读取 x_0 并将 x_1 写入为下一个版本），而 t_2 则针对同一数据项执行盲写（将 x_2 写入为任意版本）。如果我们生成版本顺序 $x_2 <_v x_1$，并像（a）中那样省略 t_2，则多版本串行化图的边会构成一个环 $t_1 \rightarrow t_2 \rightarrow t_1$。这是因为$t_1$ 会将读-修改-写操作写入 x_0 的紧接下一个版本，因此，任何事务都可以将一个版本插入 x_0 和 x_1之间。然而，如果我们像（b）中那样生成版本顺序 $x_1 <_v x_2$，则可以安全地省略 t_1，因为多版本串行化图是无环的。

线性化。在（c）和（d）中，只存在盲更新。因此，两个多版本串行化图都没有边且无环。然而，如果我们生成不恰当的版本顺序并省略错误的版本，数据库就会违反线性化。在（c）中，我们生成版本顺序 $x_2 <_v x_1$

。此版本顺序与事务的墙上时钟顺序不匹配； t_1 和 t_2 是非并发事务，因此这些事务的版本顺序必须为 $x_1 <_v x_2$。此后，数据库的事务正确性因写入省略而丧失。

从以上示例中，我们可以看出创建可安全省略的事务所需遵循的规则和限制。示例(a)和(b)表明，必须至少存在一个盲更新 [12]。如果没有盲更新却省略了更新，那么无论创建何种版本顺序，正确性测试都无法通过。此外，示例(c)和(d)表明，盲更新必须由并发事务写入。因此，为了创建可安全省略的版本，我们有三个据我们所知尚未得到全面研究的问题：

Q1：如何找出并发盲更新以生成版本顺序?

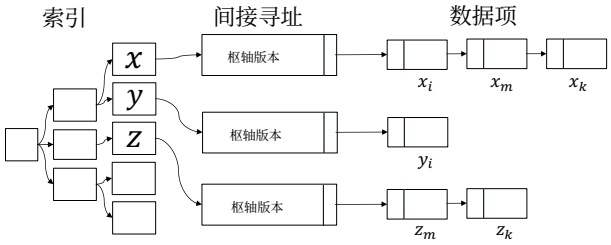


图3：SSE 实现的总体结构。

Q2：如何测试正确性?

在本文中，我们将这三个问题的解决方案整合为一个名为 SSE 的 CC协议扩展。

4 SSE：调度空间扩展器

在本节中，我们提出**调度空间扩展器（SSE）**，它为传统协议增加了另一种控制流，用于生成可安全省略的事务。SSE 解决上一节所示问题的方法如下：

> **A1:** SSE 选择并管理并发盲更新，将其作为枢轴版本，以生成擦除版本顺序，该顺序假定存在可安全省略的事务（第4.1节）。**A2:** SSE 使用多版本串行化图，高效地测试擦除版本顺序的正确性（第4.2节）。

我们首先介绍擦除版本顺序，这是 SSE 生成的一种版本顺序，旨在降低正确性测试的计算成本（第4.1节）。接下来，我们概述如何测试擦除版本顺序（第4.2节）。然后，我们展示 SSE 如何通过控制流扩展协议，并提升其在数据摄取查询上的性能（第 4.3 节）。

4.1 SSE 的版本顺序生成

可串行化理论表明，如果存在一种版本顺序能够生成无环多版本串行化图，那么某个写入就是可安全省略的。然而，尝试测试所有可能的版本顺序并不现实，因为已证明，对所有版本顺序进行测试是 NP完全问题 [2, 12, 31]。为高效执行测试，SSE 在生成候选版本顺序时采用启发式限制。当活动事务 t_j到达时，SSE 会生成一种**擦除版本顺序**。这种版本顺序假设 t_j 中的所有写入都可安全省略。形式上，擦除版本顺序满足以下三个条件：（1）SSE 仅针对 t_j 正在更新的数据项更改版本顺序。根据这一条件，SSE 的正确性测试可以聚焦于多版本串行化图中包含t_j节点的子图；如果正确性遭到破坏，则原因将是与 ws_j相关的数据项的版本顺序发生了变化。（2）每个版本 x_j都是盲更新的紧前版本。我们添加这一条件，以确保可安全省略版本的未读条件成立；非最新版本会变为过时版本，后续事务不会再请求这些版本。此外，我们还要求后续版本必须是盲更新。如第3节的图 2-(a)所示，如果x_j 位于读-修改-写操作的中间，那么多版本串行化图就总会出现环。

具体而言，为构建擦除版本顺序，SSE 会为每个数据项选择一个盲更新作为枢轴版本 。枢轴版本是生成擦除版本顺序的基准点；它会告知其他事务“将你的版本放在此枢轴版本之前”。例如，在 SSE 中，事务 t_j 生成一种擦除版本顺序，使其所有版本都紧邻枢轴版本之前。如果存在枢轴版本 x_{pv}，y_{pv} ...，那么 t_j 会创建擦除版本顺序 $x_j <_v x_{pv}$，$y_j <_v y_{pv}$。图3展示了我们原型实现中枢轴版本的整体结构。我们通过为每个数据项添加一个间接引用来实现枢轴版本。我们假设数据库采用树形索引，并且每个数据项都通过其叶节点访问。在 SSE 中，每个索引叶节点都指向一个枢轴版本，该版本是指向数据项的间接寻址对象。每个数据项都表示为一个从枢轴版本开始的单向链表。SSE 仅利用枢轴版本完成对擦除版本顺序的正确性测试；如第4.2节所述，枢轴版本包含可达事务的足迹。

算法1： 正确性测试s_j

输入： t_j

输出： c_j 是否保持可串行化

```
1  T：={t_k， t_i|t_k ∈ s_j ∧ t_i在多版本串行化图中可从 t_k 到达 }      // (A)
2  对所有 t_m 属于 T，执行
3      如果 t_m 在 t_j 开始之前提交 那么                                  // (B)
4          返回 不满足严格可串行化
5      对所有 y_m 属于 ws_m，执行
6          对所有 y_n 属于 rs_j，执行
7              如果 y_m = y_n 那么                                      // (C)  ⇝ 到 t_j
8                  返回 多版本串行化图存在环
9      对于所有 y_g 属于 rs_m 执行
10         对于所有 y_j 属于 ws_j 执行
11             如果 y_g <_v y_j 那么                                    // (D)  ≪(rw)→ 到 t_j
12                 返回多版本串行化图存在环
13 返回多版本串行化图无环
```

4.2 正确性测试

采用擦除版本顺序时，SSE 能高效地测试事务的正确性。要测试可串行化，只需测试包含 t_j 中节点的多版本串行化图路径。要确保线性化，t_j 中所有出现在可串行化测试里的可达节点都必须与 t_j 并发。我们根据从 t_j 出发的边，将节点集定义为两类：后继事务和覆盖者 （分别简写为 s_j 和 o_j），如下所示：

定义3（可达事务的类型）。对于从事务 t_j 直接可达的事务，我们定义以下两个集合：

$$s_j := \{t_k | t_j \overset{\ll(ww)}{\rightarrow} t_k\},\ o_j := \{t_k | t_j \overset{\ll(rw)}{\rightarrow} t_k\}$$

由此定义可得以下定理：

定理 1（直接可达事务）。如果某个调度满足可恢复性，且存在一条边 $t_j \rightarrow t_k$，那么直接可达事务 t_k 属于 o_j 或 s_j。

证明。多版本串行化图只有三种类型的边：$t_j \overset{wr}{\rightarrow} t_k$，$t_j \overset{\ll}{(rw)}\rightarrow t_k$，以及 $t_j \overset{\ll}{(ww)}\rightarrow t_k$。由于满足可恢复性，且 t_j 处于活跃状态，因此边 $t_j \overset{wr}{\rightarrow} t_k$ 不存在。□

根据定理 1，SSE 可以通过测试从这两组事务出发的所有路径来测试正确性。因此，SSE 将正确性测试过程分为以下两个子测试。

后继事务的测试。没有研究关注后继事务的测试，因为传统协议总是将传入的 t_j 的版本写为最新版本，因此后继事务集合为空。算法1为 s_j 提供了一种测试过程。设 rs_i 是 t_i 读取的版本集合，ws_i 是 t_i 写入的版本集合。在步骤 (A) 中，收集包含于 s_j 中或可从 s_j 到达的事务。在步骤 (B) 中，为了保证线性化，测试 t_j 与收集到的每个事务 t_m 之间的并发性。在步骤 (C) 和 (D) 中，算法测试可串行化。由于每个 t_m 都是包含于 s_j 中或可从 s_j 到达的事务，因此路径 $t_j \overset{\ll}{(ww)}\rightarrow \ ...,\rightarrow t_m$ 已存在。因此，如果对每个 t_m 都不存在路径$t_m \rightarrow t_j$，则多版本串行化图无环；因此，步骤 (C) 和 (D) 重点关注最后一条此类边 $t_m \rightarrow t_j$。注意，事务 t_m 不存在路径 $t_m \overset{\ll}{(ww)}\rightarrow \ t_j$。只有当某些已提交事务读取了 ws_j 中的某个版本时，这类边才会被添加到多版本串行化图中；而为确保可恢复性，不允许执行此类读操作。由于 t_j 是一个活动事务，没有已提交事务能够读取版本 x_j。因此，我们只需检查最后几条边的类型：$t_m \overset{wr}{\rightarrow} t_j$ 和 $t_m \overset{\ll}{(rw)}\rightarrow \ t_j$。据此，在步骤 (C) 中，算法检查最后几条边 $t_m \overset{wr}{\rightarrow} t_j$。它检查 rs_j 中的 y_n 是否与 ws_m 中的 y_m 相同。这是因为，如果满足此条件，就存在一条循环路径 $t_j \overset{ww}{(\rightarrow} \ ... \rightarrow t_m \overset{wr}{\rightarrow} t_j)$。类似地，在步骤 (D) 中，算法通过确认 ws_j 中存在一个比 rs_m 中的 y_g 更新的 y_j，来检查最后几条边 $t_m \overset{\ll}{(rw)}\rightarrow \ t_j$。这是因为，如果满足此条件，就存在一条循环路径 $t_j \overset{ww}{(\rightarrow} \ ... \rightarrow t_m \overset{\ll}{)}(rw)\rightarrow \ t_j$。因此，如果步骤 (C) 和 (D) 的测试通过，则 s_j 中没有事务能够到达 t_j。

覆盖者测试。这些子集的详细算法超出了本文的讨论范围，因为我们可以使用传统协议中的现有算法，例如反依赖验证。例如，Silo [11]采用了最简单的方法之一：通过检查rs_j 中的每个版本是否已被覆盖且是否存在更新的版本，来测试 $o_j = \phi$。如果对后继者和覆盖者的测试均通过，我们就可以提交 t_j，而不会违反任何正确性，因为 t_j的所有写操作都可安全省略。

一个重要的事实是，写入枢轴版本的事务集合 t_{pv} 等价于后继事务集合 s_j。这是因为枢轴版本恰好是 t_j 的更新所对应的下一个版本。因此，我们可以通过使用事务 t_m 的读写集来实现算法1，其中包括 t_{pv} 以及从某些 t_{pv} 可达的事务。为此，每个枢轴版本都保存已读取或写入大于或等于该枢轴版本的版本的事务足迹。

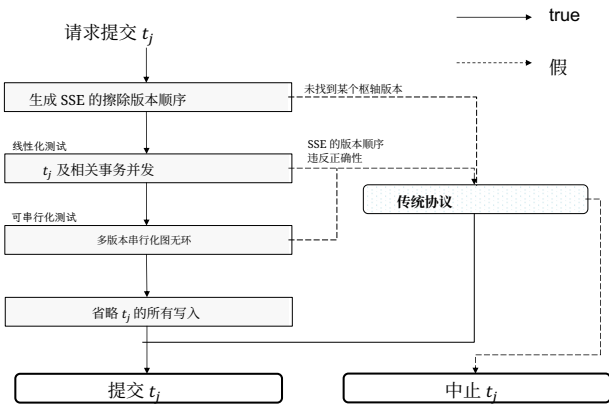


图4：采用传统协议的 SSE 控制流。

4.3 控制流

我们发现，擦除版本顺序有助于确认某个写入是否可安全省略，并且还提出了用于检验这一点的算法。接下来的挑战是，如何将这种版本顺序和检验算法应用于传统并发控制协议。我们希望不要大幅改变传统并发控制协议的性能特征和规范。然而，如果我们只是修改传统协议以生成擦除版本顺序，那么修改后的协议将毫无用处；因为擦除版本顺序总是试图写入非最新版本，我们将无法成功更新最新版本。即使同一数据项上发生竞争性更新，实际应用也需要定期更新最新版本。我们并不采用这种修改，而是提出 **SSE**，这是一种纯粹扩展传统协议调度空间的扩展方法。SSE 并非一种协议，而是对协议的扩展；它不改变传统协议的版本顺序和正确性检验算法，而是添加擦除版本顺序，并独立检验其正确性。

图4展示了扩展协议的控制流。SSE 在活跃事务 t_j 发出提交请求时开始处理。首先，SSE 按照第4.1节所述生成一个擦除版本顺序。如果找不到任何版本顺序，SSE 就将 t_j 的处理委托给传统协议。如果找到擦除版本顺序，SSE 接着会检查正确性。对于线性化，SSE 会检查 t_j 与所有相关事务之间的并发性。对于可串行化，SSE 会检查多版本串行化图是否无环。如果两项检查均通过，SSE 就省略 t_j 的所有版本并提交 t_j。否则，SSE 会将 t_j 的处理委托给传统协议。请注意，SSE 不会更改传统协议生成的版本顺序；SSE 可以提交 t_j，但不会直接中止 t_j。即使 SSE 无法按其擦除版本顺序提交 t_j，也仍有可能按照传统协议生成的版本顺序提交 t_j。这表明，SSE 纯粹扩展了传统协议的调度空间。

在最佳情况下，我们只需访问枢轴版本即可省略 t_j 的更新，因为这些间接寻址数据结构包含正确性测试所需的全部数据。否则，如果更新$w_j(x_j)$未找到 x的枢轴版本，或者某项测试失败，我们就会移交控制权，并退出以执行 SSE 测试。在这种情况下，枢轴版本的开销几乎可以忽略不计，相较于传统协议；它只为每个数据项增加一个间接引用。

5 ESSE：基于纪元的SSE

本节介绍SSE的一种优化方法，称为**基于纪元的SSE（ESSE）**。我们首先说明，SSE的朴素实现会在对后继事务进行正确性测试时遇到性能问题。如第3节所述，这项测试需要获取所有可达事务 t_m的读集（rs_m）和写集（ws_m），因而会产生巨大的开销。具体而言，在算法1的步骤（A）中，来自足迹 rs_m和 ws_m 的内存消耗会随着被测目标事务t_m 数量的增加而增加。此外，步骤（C）和（D）存在同步问题：足迹 rs_m 和 ws_m 必须采用并发数据结构，才能并行提交事务。

为解决这两个性能问题，我们提出ESSE，这是一种采用基于纪元的组提交 [4, 11]的优化版SSE实现。基于纪元的组提交将墙上时钟时间划分为纪元 ，并将每个事务分配到一个纪元。同一纪元内事务的提交操作会同时延迟。由于这些事务具有相同的提交点，因此可视为并发；而不同纪元中的事务则不并发。ESSE充分利用了纪元内这种并发特性。具体而言，ESSE选择每个纪元中各数据项的首次盲更新作为枢轴版本。于是，线性化测试变得简单；只需检查枢轴版本是否与 t_j的版本在同一纪元中写入，即可轻松完成测试。纪元与枢轴版本的紧密关联有助于缓解SSE此前的性能问题——巨大的内存消耗。对于旧纪元中的事务，无需保留其足迹，因为在对这些事务进行可串行化测试之前，SSE就会在线性化测试中失败。

ESSE 还包含以下两项优化。(1) 它通过使用纪元创建一条优化规则，以减少需要进行测试的目标事务数量 t_m （第 5.1 节）。(2) 它提供了一个**可达性标志**，用于高效地测试后继事务；这是对 SSE 枢轴版本的一种无锁实现，旨在解决 SSE 的后续性能问题——同步的必要性（第 5.2 节）。介绍完 ESSE 后，我们将展示如何将 ESSE 实际应用于传统协议，并说明哪类协议更适合 ESSE（第 5.3 节）。

5.1 优化规则

为了减少需要进行测试的目标事务数量，我们利用 ESSE 的枢轴版本添加了优化规则。对于一个纪元，我们将一组枢轴版本定义为**枢轴屏障**。利用枢轴屏障，我们得出以下定理，该定理为减少目标事务数量提供了重要性质。

定理 2（枢轴屏障违例）。设 t_j 是纪元 e_1 中一个活跃的事务，正在尝试在 *ESSE* 上提交。设 et_1 为写入 e_1 的枢轴屏障（该纪元中的枢轴版本）的事务集合。如果从 et_1 中的任何事务都无法到达 t_j，则 $successors_j$ 的正确性测试通过。

证明。为反证，我们首先假设不存在满足定理 2 两个条件的事务，并且

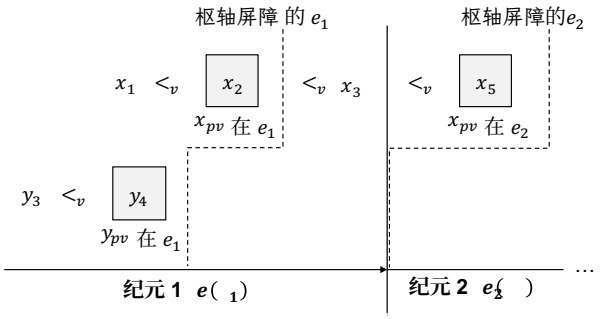


图5：两个纪元的枢轴版本和枢轴屏障。所有版本均不可省略，并安装在内存中。一个纪元的枢轴屏障由其枢轴版本组成。灰色方框表示枢轴版本，虚线表示各纪元的枢轴屏障。 t_3 未通过 SSE 的正确性测试，并在枢轴屏障的右侧和左侧都安装了版本。这样的事务会为后续事务造成一个 MVSG 环。

对$successors_j$ 的测试检测到了一个 MVSG 环。由于 MVSG 中存在一个环，因此存在一个事务 t_i， t_j 可从该事务直接到达，这满足定理 2 中的直接可达条件。根据假设， t_i 不满足枢轴屏障条件的右侧；因此，任何 t_{pv}都无法到达 t_i。所以，MVSG 环通过枢轴屏障的左侧以 $t_j \rightarrow ..., \rightarrow t_i \rightarrow t_j \rightarrow t_{pv}$的形式闭合。我们关注从 t_j出发的第一条边。由于ws_j 中的所有版本都是枢轴版本的直接前驱，因此不存在事务能够写入一个大于属于 t_j 的版本、但小于枢轴版本的版本。因此，要在枢轴屏障左侧构成环，从 t_j 出发的环边类型不能是$\ll(ww)\rightarrow$ 。此外，边的类型也不能是 $\overset{wr}{\rightarrow}$，因为 t_j 是一个活跃事务，且正试图省略其写入。因此，边的类型必须是$\ll(rw)\rightarrow$ 。于是，根据定理 1 和上述假设， t_i 既不在$successors_j$ 中，也不在其可达事务中。因此， t_i 不是对$successors_j$进行正确性测试的目标事务， t_j 也就无法检测到 MVSG 环。□

定理 2 阐明了在测试后继事务时会引入 MVSG 环的事务类型。由于 MVSG 环总是包含读取或写入枢轴屏障右侧某个版本，且从 t_j 可达的事务，因此 ESSE 只需存储此类事务的足迹。如果 t_j在已保存的足迹中检测到此类事务的存在，ESSE 就不会提交 t_j；反之，其他事务是否存在对 t_j 并不重要。因此，我们只需管理满足定理 2 中两个条件的事务的足迹。具体而言，我们会存储满足第一个条件（位于枢轴屏障右侧）的事务的足迹，因为在没有入边事务的情况下，我们无法测试某个事务是否满足第二个条件

t_j。满足第一个条件的事务

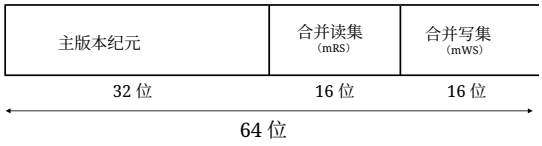


图6：可达性跟踪器的布局

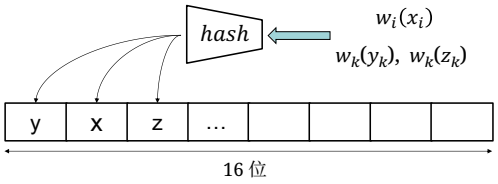


图 7：合并写集示例

假设它们可以从传入的 t_j 到达，因此会将其读写集存储到枢轴版本中。

图5展示了每个纪元中的枢轴版本及其版本顺序示例。所有版本都已安装在共享内存中，且所有事务 t_1、 t_2、 t_3、 t_4 和 t_5 均已提交。设 t_j 为属于纪元 e_1 且请求写入 x_j 和 y_j的活动事务。对于 t_j的后继事务正确性测试而言，t_3 的存在至关重要：由于 t_3 同时满足定理 2 中的右侧条件和直接可达条件，因此会在多版本串行化图中生成一个环，并在使用 ESSE 的擦除版本顺序提交 t_j时破坏可串行化。因而，在提交 t_j之前，ESSE 必须从 t_3的足迹中检测出这样的多版本串行化图环。相反，由于t_1 不满足右侧条件，根据定理 2，无需保存其足迹。此外，通过使用纪元，ESSE 减少了所需足迹的数量，并实现了定期垃圾回收。无需管理旧纪元的枢轴版本和枢轴屏障。具体而言，根据线性化，在 e_1的所有事务终止后，除了 t_5之外，无需管理任何事务的足迹； e_1 完成后，我们不再启动属于 e_1的任何事务，因此也无需管理 e_1的枢轴版本和足迹。

5.2 无锁实现

在 ESSE 中，每个数据项的枢轴版本都表示为一个**可达性跟踪器**，它是一种 64 位数据结构。这是图3中所述枢轴版本的具体实现。可达性跟踪器由一个纪元和两个字段（合并读集和合并写集）组成。图6展示了可达性跟踪器的布局。

- **纪元**。此字段以 32 位整数的形式存储枢轴版本的纪元。每个事务在事务开始时从全局纪元获取并分配自己的纪元。当事务执行盲更新并成为枢轴版本时，ESSE 会将该事务的纪元存储到此字段中。此字段用于正确性测试。如果所有收集到的可达性跟踪器的纪元都与活动事务的纪元相同，则满足线性化要求。

算法 2： 使用$successors_j$进行正确性测试可达性跟踪器

输入： t_j
输出： c_j 是否保持可串行化

```
对x_j 中的 ws_j 执行
    RT：= 获取_可达性_跟踪器_的（x）                    // (A)
    如果 RT.epoch ≠ t_j 的纪元 那么
        返回 不满足严格可串行化                          // (B)
    如果 rs_j 中的任意键存在于 RT.mWS 那么               // (C)
        返回多版本串行化图可能不是无环的
    如果ws_j 中存在任何键，且其存在于 RT.mRS 则          // (D)
        返回多版本串行化图可能不是无环的
返回满足严格可串行化
```

- **合并读集 (mRS) 和合并写集 (mWS)**。这些字段表示 16 位布隆过滤器。每个过滤器都存储满足定理 2 中两个条件的所有事务足迹；它存储读取或写入枢轴版本的较新或相同版本的所有事务的读集或写集中的键的并集。图 7 展示了 mWS 的布局示例。每个数据项 x，y，z都通过哈希函数 h映射到相应的槽位。ESSE 使用此结构测试后继事务的可串行化。

可达性跟踪器将 SSE 的枢轴版本功能封装到 64 位数据结构中；它为事务的并发检测器和可达事务的足迹提供了一种方案。通过这种压缩，ESSE 使用比较并交换（CAS）等原子操作，以无闩锁方式访问可达性跟踪器。

算法 2 是算法1的详细实现，基于使用可达性跟踪器。两个算法中的所有步骤（A-D）都等价。在步骤（A）中，它收集ws_j中数据项的可达性跟踪器，以生成擦除版本顺序。在步骤（B）中，算法通过检验 t_j 与可达性跟踪器是否处于同一纪元来检查线性化。如果检测到纪元不同，则线性化的正确性测试失败。在步骤（C）和（D）中，算法通过检验 t_j 是否有入边来检查可串行化。可达性跟踪器利用布隆过滤器，因此可能产生假阳性，但不会产生假阴性。当哈希函数将不同的键分配到同一槽位时，它可能会报告一个并不存在的环。

为了使用可达性跟踪器进行正确性测试，我们需要更新它们，以维护从枢轴版本可达事务的合并读集/写集。因此，我们在每个事务提交之前添加此更新。算法 3 描述了活动事务 t_j 的 ESSE 提交协议 [2]。在步骤 (1) 中，它测试 o_j 和 s_j 集的严格可串行化。如果测试失败，算法将使用 ESSE 的版本顺序退出对 c_j 的处理，并将后续处理交由传统协议。在步骤 (2) 中，如果 t_j 不满足定理 2 的条件，ESSE 不会存储足迹。否则，ESSE 会将 t_j 的足迹存储在可达性

[2]这里，“提交协议”指的是用户应用程序不再向事务中添加任何操作时进行的处理。它并非指乐观并发控制（OCC）中的“提交阶段”。

算法 3： ESSE 的提交协议

输入： t_j
输出： 提交或中止

```
t_j.omittable：= true
# 开始原子区段
如果 没有对 t_j 的写入是盲更新则
    t_j.omittable：= false
如果正确性测试失败 则
    t_j.omittable：= false                              // (1)
如果 t_j.omittable == false 则
    # 尝试传统协议
    如果传统协议中止 t_j 则
        返回 中止
如果 t_j 满足定理 2 的第一组条件则                      // (2)
    对所有 x_i 在rs_j 中执行
        RT：= 获取_可达性_跟踪器_(x)
        RT.mRS.merge(rs_j.keys)
        RT.mWS.merge(ws_j.keys)
                                                        // (3)
    forall x_j 属于 ws_j do
        RT := 获取_可达性_跟踪器_of(x)
        如果 RT.epoch ≠ t_j.epoch 那么
            如果 w_j(x_j) 是盲更新 则                   // (4)
                RT.纪元 := t_j.纪元
                使用rs_j初始化 RT.mRS
                使用 ws_j初始化 RT.mWS
        else
            RT.mRS.设置(rs_j.键)
            RT.mWS.set(ws_j.键)
                                                        // (5)
# 结束原子区段
如果 t_j.omittable 则
    清除(ws_j)
return提交
```

跟踪器中。在步骤 (3) 中，它将从枢轴屏障到 t_j 的所有边存储到数据项的合并读集中。具体而言，对于 rs_j 和 ws_j 中的每个数据项，算法分别在合并读集和合并写集中设置位标志。它还会为 rs_j 的可达性跟踪器中的所有位设置位标志。在步骤 (4) 中，它更新合并写集。如果 x_j 是该纪元中的第一次盲更新，算法会将数据项 x 的枢轴版本重置为 x_j。否则，它会将 t_j 的足迹以及从枢轴屏障中的版本到该数据项合并写集的所有边添加进去。请注意，步骤 (3) 和 (5) 会合并由 t_j 访问的可达性跟踪器中 mRS/mWS 的位，以保持从枢轴版本的可达性；当 t_j 访问 x 和 y 并提交时，分别存在从 x 和 y 的枢轴版本到 t_j 的路径，因此，两个数据项的可达性跟踪器都应通过合并来存储这些路径。

在整个算法中，我们利用 64 位数据布局，以无闩锁方式原子地访问可达性跟踪器。在提交协议开始时，ESSE 会将所有可达性跟踪器复制到另一个位置。在对 mRS/mWS 进行测试和修改后，它会对所有位置执行 CAS 操作，以更新

协议	基于纪元的组提交	读写集	覆盖者的测试方法	乐观并发控制
Silo OCC	Yes	Yes	Yes	Yes
Cicada MVTO	No	Yes	No	Yes
2PL	No	No	No	No
ESSE 所需	Yes	Yes	Yes	较为理想

表 3：ESSE 所需组件列表及协议比较。若要将 ESSE 扩展到某个协议中，我们需要添加缺少的组件。尽管乐观并发控制并非必需条件，但从性能角度而言，它是 ESSE 的理想特性。

所有字段。如果 CAS 操作失败，ESSE 会从头开始重试提交协议。如果可达性跟踪器中的位数组在修改前后完全一致，我们就可以通过加载而非 CAS 来验证期间未发生并发更改，从而以轻量方式保证原子性。

5.3 扩展细节

若要将 ESSE 应用于传统协议，我们需要添加一些组件：基于纪元的组提交、每个事务的读写集、可达性跟踪器，以及对超额写入的正确性测试。表 3 总结了将 ESSE 扩展到某个协议所需的组件。如果协议已经具备必要组件，ESSE 就可以直接使用它们。例如，Silo 具备 ESSE 所需的所有组件，因此是较为理想的协议之一。请注意，表 3 未列出可达性跟踪器，但如图 3所述，ESSE 会为所有协议添加该组件。与 Silo 相比，传统的两阶段锁协议（2PL）[18] 不具备任何必要组件。此外，2PL 无法利用 ESSE 的版本省略技术，因为它会在事务提交前立即将版本写入数据项。而且，如果 2PL 与某些日志算法（例如 ARIES [29],）一起使用，它会立即持久化日志。ESSE 协议在未提交版本已经安装并持久化后才开始。在这种情况下，如果接受擦除版本顺序，2PL+ESSE 必须在解锁前撤销已安装的未提交版本。因此，从性能角度而言，乐观并发控制是 ESSE 的理想特性。

我们将两种乐观协议：Silo 和 MVTO，分别扩展为 Silo+ESSE 和 MVTO+ESSE。选择这两种协议，是因为它们分别是 1VCC 和多版本并发控制类型中较新的高性能协议。除 2PL 外，我们的实验没有纳入采用 TWR 的时间戳排序。时间戳排序需要集中式计数器来生成单调递增的时间戳。在众核环境中，使用此类计数器的协议会导致性能严重下降 [41]。

Silo 和 Silo+ESSE。Silo [11] 是一种乐观协议，在读密集型工作负载上取得了最先进的性能。当应用程序请求提交事务时，Silo 会在安装新版本之前，为事务写集中的所有数据项获取锁。我们将 Silo 的覆盖者正确性测试移植到了 Silo+ESSE。Silo 的测试会检查 $o_j = \phi$；如果 ^{rs}j中的某个数据项被覆盖，Silo 和 Silo+ESSE 就会中止 t_j。请注意，原始 Silo 会将记录直接存储在索引的叶节点中。由于 ESSE 的可达性跟踪器采用间接寻址，我们实现的 Silo 和 Silo+ESSE 可能会产生一次缓存未命中的开销。

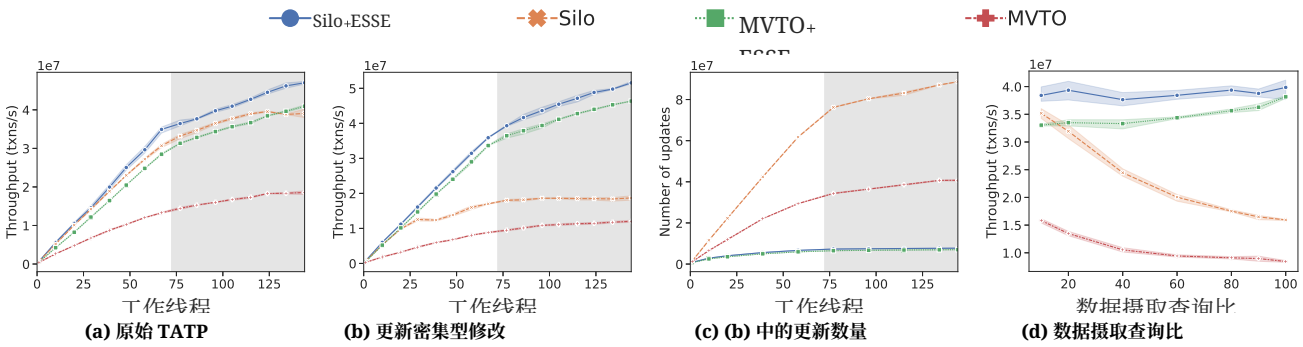
MVTO 和 MVTO+ESSE。MVTO [3, 26] 是一种基于时间戳的并发控制协议，采用多版本存储。我们基于 Cicada [26], 这一最先进的 MVTO 协议实现了 MVTO。它采用每线程分布式时间戳生成和自适应退避、用于乐观多版本控制的读写集，以及快速垃圾回收。请注意，原始 Cicada 不保证严格可串行化，而是保证因果一致性 [33]。我们将基于纪元的组提交应用于该协议，以确保严格可串行化；原始协议使用每事务 64 位时间戳，而我们将其缩短为 32 位，并在高 32 位中加入纪元编号，以便在纪元之间同步并避免过时读取。为了测试覆盖者，我们将 Silo 的实现移植到 MVTO+ESSE；当 ^{rs}j 中的某个版本 x_i 不是数据项 x的最新版本时，MVTO+ESSE 无法通过其擦除版本顺序来测试 ^{o}j，并将其控制权交给 MVTO。

6 评估

我们的实验使用了一个以 C++ 编写的轻量级、非分布式、嵌入式事务型键值存储原型。它由内存存储、并发控制协议（Silo、Silo+ESSE、MVTO 和 MVTO+ESSE）、一个从 Masstree [24], 分支出来的树索引，以及一个依照 SiloR [42]规范实现的并行日志管理器组成。实验在一台 72 核机器上运行，该机器配备四颗英特尔至强 E7-8870 CPU 和 1 TB 动态随机存取存储器。每个 CPU 插槽有 18 个物理核心；启用超线程后，共有 36 个逻辑核心。线程数超过 72 时，结果呈现次线性扩展，这是由于物理核心内部存在争用；在结果图中，我们用灰色背景表示这种情况。每个插槽都有一个 45 MB 的 L3 共享缓存。每个工作线程的事务日志均单独存放，并刷新到同一块固态硬盘中。由于所有查询都在构建时编译，因此没有使用联网客户端或 SQL 解释器。每个工作线程都配有一个线程本地工作负载生成器，使其能够输入自己的事务。

我们选取了由基准测试规范生成的三种工作负载：TATP [27], YCSB [5], 和 TPC-C [38] 基准测试。我们选取 TATP 作为适用于预期应用（例如物联网/电信应用，包括数据摄取查询）的基准测试。就性能而言，我们选取 YCSB 作为 ESSE 的理想场景，因为它包含大量盲更新。最后，我们选取 TPC-C 作为对照基准测试，其性能收益最小，因为它不包含盲更新。

TATP基准测试。 TATP代表一家电信公司的工作负载。其中包括16%的数据摄取查询，这些查询



(a) 原始 TATP (b) 更新密集型修改 (c) (b) 中的更新数量 (d) 数据摄取查询比

图 8：TATP基准测试结果

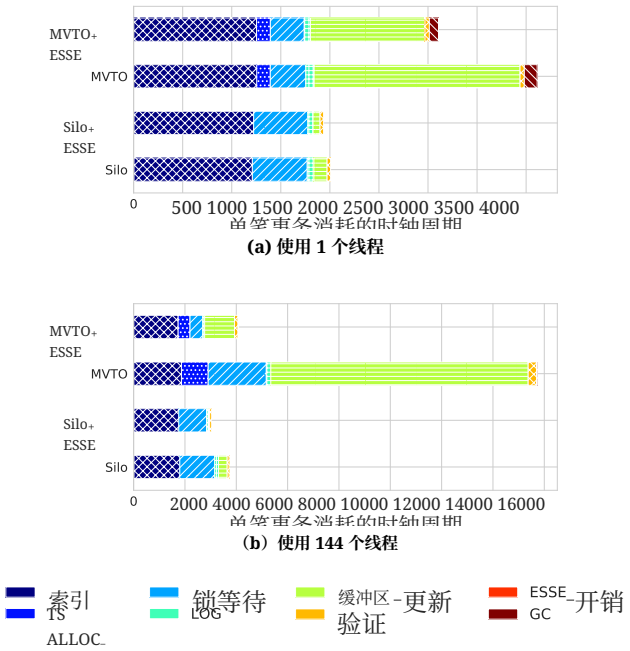
会为管理用户当前位置或用户资料数据的变更生成大量盲更新。其余查询中有70%由GET_SUBSCRIBER_DATA和GET_ACCESS_DATA组成。二者都会检索由数据摄取查询更新的最新且正确的数据快照，以操作电信基站。我们根据其规范实现了该基准测试。除遵循原始规范的工作负载外，我们还添加了盲更新占比从原始的16%变化的多种工作负载，以模拟我们预期的物联网/电信应用。我们调整占比的查询是UPDATE_LOCATION。

YCSB 基准测试。 该工作负载生成器能够代表传统的大规模在线基准测试。由于原始 YCSB 不支持包含多个操作的事务，我们在原型中实现了一个类似 YCSB 的工作负载生成器，与 DBx1000 [41] 的做法类似。具体而言，每个事务会根据参数为 θ 的齐夫分布随机访问四个数据项。每个数据项都有一个主键和一个 8 字节的附加列。我们将原型实现部署为一张包含 100K 个数据项的单表。

TPC-C 基准测试。 这是在线事务处理的行业标准。它由六个表和五个事务组成，用于模拟批发商的信息系统。需要注意的是，TPC-C 基准测试不包含任何盲更新；所有写操作都是插入或读-改-写。因此，SSE 无法使用其擦除版本顺序提交任何事务。我们实现了包含全部五个事务的 TPC-C 全混合负载。我们采用与 Silo 相同的方法来防止幻读异常：在每个事务提交时再次扫描树索引。

6.1 TATP 基准测试结果

图 8a 和图 8b 分别展示了原始 TATP 及其更新密集型修改版本的结果。在这两种情况下，ESSE 都提升了原始协议的性能，而对更新密集型修改版本的提升尤为显著。由于大多数盲更新都是可安全省略的版本，因此不会被写入物理内存，Silo+ESSE 和 MVTO+ESSE 的性能分别提升了 2.7× 倍和 2.5× 倍。相比之下，Silo 在



(a) 使用 1 个线程

(b) 使用 144 个线程

图 9：图 8a 的运行时间细分

更新密集型修改版本上的性能因锁竞争而大幅下降。虽然 Silo 是一种无需读锁的协议，但 Silo 的写操作需要加锁，从而降低了并行性。MVTO 在原始 TATP 和修改版本上的性能都是最差的。虽然 MVTO 的写操作无需加锁，但由于它会分配内存来创建新版本，其吞吐量因此下降。SSE 克服了 Silo 和 MVTO 的这些缺陷，因此大幅提升了性能。

图 8c 展示了采用更新密集型修改时 TATP 的更新次数。对于 Silo 和 Silo+ESSE，我们统计了带锁的原位更新次数。对于 MVTO 和 MVTO+ESSE，我们统计了通过创建新版本进行非原位更新的次数。

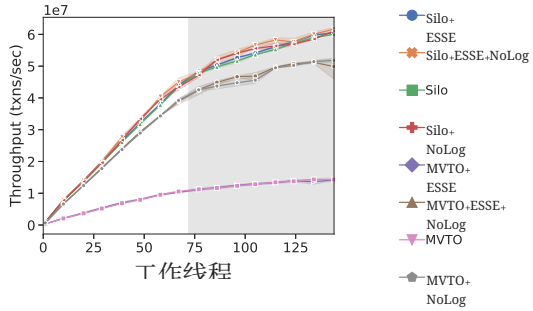


图 10：禁用日志协议的 TATP 结果

结果表明，这两种 ESSE 协议很少执行实际更新，因为大多数写入都会生成可安全省略的版本。

图8d展示了 TATP基准测试在不同数据摄取查询速率和固定为 144 个工作线程时的结果。当数据摄取查询所占比例超过原来的 16% 后，Silo 和 MVTO 的吞吐量均有所下降。原因在于，Silo 在同一数据项上遭遇了锁竞争，而 MVTO 则受到了物理内存中多个版本管理的影响。相比之下，ESSE 协议的性能优于原始协议，并且其吞吐量不会随着这一比例的增加而下降。此外，当这一比例超过 80% 时，这些扩展协议的性能有所提升。这是因为 SSE 提供了缓存效率；在这种情况下，客户端请求执行的盲写几乎都针对相同的数据项，因此 ESSE 生成了大量可安全省略的版本。因此，它很少安装新版本，几乎所有读操作都获取到相同的版本，而这些版本很少会从 CPU缓存中被逐出。

图 9 展示了图 8a 所示结果的运行时间分解。使用单个线程时，CPU 时钟周期的主要消耗项是 Silo 和 Silo+ESSE 的索引块，以及 MVTO 和 MVTO+SSE 的缓冲区_更新块。对于这两种 ESSE 协议，ESSE 的开销都可以忽略不计。使用 144 个线程时，CPU 时钟周期的主要消耗项仍与单线程时相同，ESSE 的开销也同样可以忽略不计。值得注意的是，ESSE 大幅减少了等待加锁块和缓冲区_更新块所消耗的 CPU 时钟周期。这两项开销通过使用可达性跟踪器得以降低。使用 ESSE 的擦除版本顺序提交事务时，写操作无需加锁，并以无闩锁方式仅访问可达性跟踪器。

如第 1 节所述，我们的目标应用包括数据摄取查询和实时操作。不过，有时也需要支持对所有已提交版本进行历史分析 [8] 。为了支持此类历史分析查询，我们需要将每个版本累积的数据刷新到持久化日志中。即使我们在并发控制协议中省略某些版本，也仍需为可安全省略的版本持久化日志。图 10 展示了禁用日志功能的协议在 TATP基准测试上的性能结果。我们将数据摄取查询的比例设为 80%。可以看到，“NoLog”协议的性能与启用日志功能的协议几乎相同。

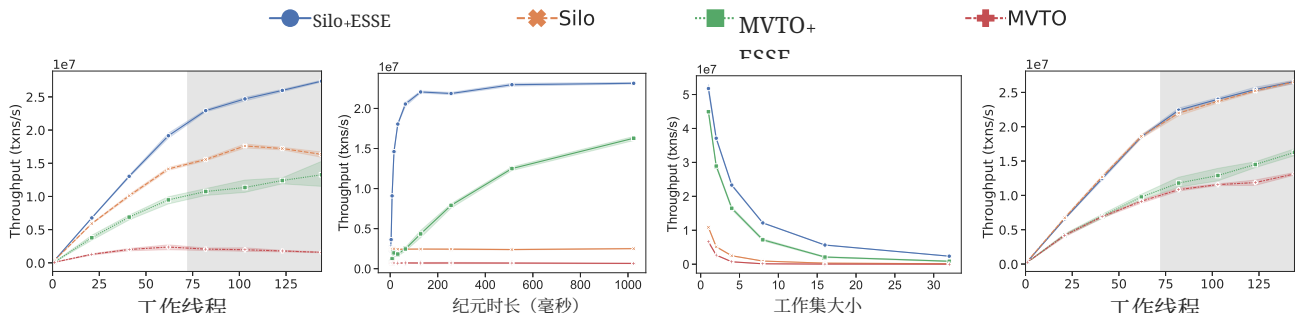
这表明，SSE 的性能提升并非来自避免日志持久化；相反，SSE 是通过减少内存消耗和避免使用锁机制来提升性能的。

6.2 YCSB 基准测试结果

图 11a 展示了中等争用率（$\theta = 0.6$）下 YCSB-A 工作负载的结果。YCSB-A 定义的操作比例为 50% 读和 50% 盲写。以往研究表明，传统协议在此类写争用工作负载上的表现不佳 [16, 23, 40, 41]。请注意，YCSB 会在基准测试前填充数据库；所有数据项都在测量前插入。因此，在 YCSB 中，所有写操作都是盲写。因此，一旦标记了每个 40 毫秒纪元的枢轴版本，ESSE 的正确性测试就很少失败。我们预期 ESSE 协议能够避免安装大量盲更新，从而相应提升性能。事实上，如图 11a 所示，ESSE 协议的吞吐量高于原始协议。在最佳情况下，使用 144 个线程时，MVTO+ESSE 的吞吐量比原始协议高出 20×。与 TATP 相比，该工作负载会产生更多写后写冲突，因此 MVTO 的链表往往更长。由于链表变长会增加读写操作中的版本遍历开销，MVTO 的性能因此下降。相比之下，MVTO+ESSE 通过利用 ESSE 省略盲更新，避免了不必要的版本分配，从而缩短了链表。

图11b和图11c展示了改变 ESSE 的两个重要参数时，YCSB-A 的结果。我们使用 72 个线程和 $\theta = 0.6$对这些工作负载进行了测试。图11b展示了不同纪元大小下的结果。如第5.2节所述，ESSE 使用基于纪元的组提交，并通过纪元测试事务之间的并发性。因此，较长的纪元时长会使更多事务并发执行，并减少 ESSE 正确性测试失败的次数。因此，为了研究纪元时长的影响，我们使用不同的时长测试了 YCSB-A。如图11b所示，ESSE 协议的吞吐量随时长增加而提高。因此，只要应用允许，我们就可以通过延长纪元时长来提升性能。接下来，图11c展示了不同读写集大小下的结果。随着读写集大小增加，ESSE 协议的性能下降至原始协议的水平。这是因为第5.2节所述的可达性跟踪器使用两个大小仅为16位的布隆过滤器（mRS/mWS）；随着事务中的操作数量增加，由于过滤器中出现假阳性，后继事务的正确性测试会更频繁地失败。

图11d展示了低争用率（$\theta = 0.2$）下读操作为主的 YCSB-B 工作负载的结果。由于该工作负载中的大多数操作都是读操作，且很少发生竞争，因此几乎没有可安全省略的版本。因此，ESSE 带来的性能提升几乎没有益处；相反，ESSE 的开销可能会对性能造成不利影响。然而，图11d显示，ESSE 协议的性能与原始协议相近。这表明 ESSE 具有低开销特性。在此类工作负载中，事务会迅速放弃 ESSE 协议，生成擦除版本顺序，



(a) YCSB-A：可扩展性 (b) YCSB-A：纪元 (c) YCSB-A：读写集大小 (d) YCSB-B：低并发下的可扩展性-争用率（$\theta = 0.2$）

图11：YCSB 基准测试结果

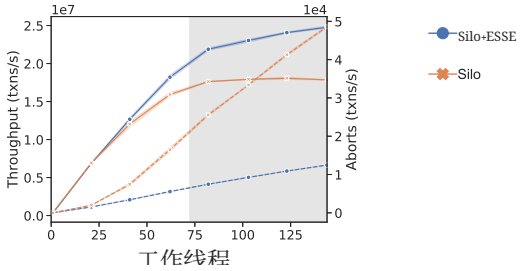


图 12 θ = 0.9：YCSB-B：高竞争下的可扩展性（）——吞吐量（实线）和中止数量（虚线）

因此不会执行正确性测试。如第 4.3 节所述，事务会先检查每个数据项间接寻址中的纪元编号。ESSE 要求所有纪元编号与事务的纪元相同；但在此工作负载中，这一条件很少得到满足。因此，事务会迅速放弃 ESSE 协议，并将控制权交给基线协议。

图12展示了高竞争率（$\theta = 0.9$）下 YCSB-B 的性能结果。虚线表示吞吐量，实线表示中止次数。由于 YCSB-B 将读操作的比例设为 95% [5]，原本预计该工作负载不适合用于测试我们的方法；SSE 和 ESSE 可以提升盲更新的性能，但 YCSB-B 很少执行这类操作。尽管如此，ESSE 协议的性能仍与原始协议相当，而且令人惊讶的是，Silo+ESSE 的性能优于原始 Silo。这一提升表明，版本省略技术对其他事务也有益。在这种情况下，Silo+ESSE 将对最新版本的覆盖操作改为省略过时版本，从而降低了中止率，因为最新版本被覆盖时，其验证会失败。在图12中，当线程数为25时，Silo 的中止次数高于其吞吐量。这是因为只有5%的写操作就会迫使 Silo 中止那95%的读操作。相比之下，Silo+ESSE 的中止率低于原始 Silo，并提升了其性能

在这种读为主的工作负载下，因为 ESSE 阻止了对最新版本的写入。

6.3 TPC-C 基准测试结果

TPC-C 基准测试中的五个查询都不包含盲写，插入操作除外。这意味着对于此工作负载，没有事务提交会采用 SSE 的擦除版本顺序。尽管我们的目标是物联网/电信应用及其包含大量盲更新的数据摄取查询，但我们也在此基准测试中测试了我们的方法，以展示 ESSE 的低开销特性。

为了分析这一低开销特性，我们使用单个仓库运行了 TPC-C 基准测试。这个高竞争场景代表了 ESSE 的最坏情况，因为即使可达性跟踪器从未被使用，每个数据项的可达性跟踪器仍必须频繁更新。图13展示了吞吐量随线程数的变化。与原论文中的实验结果类似，Silo 和 Silo+ESSE 可扩展至 32 个核心。MVTO+ESSE 的开销可以忽略不计，但其性能与原始协议相当。MVTO 在 TPC-C 上的性能瓶颈是版本遍历或缓冲区更新，因此可达性跟踪器的开销并未影响性能。相比之下，Silo+ESSE 的吞吐量比原始协议低约 0.75× [3]。这是因为 TPC-C 中的大多数查询都会对最新版本执行读-修改-写操作。由于对大于枢轴版本的版本执行读-修改-写操作的事务可能满足定理 2 的两个条件，因此必须将其足迹存储在每个数据项的可达性跟踪器中。

7 相关工作

数据摄取查询用于汇总物联网/电信应用中来自传感器和移动设备的更新。这类查询长期以来一直在流数据库等非事务系统中得到讨论 [13, 19]。然而，近年来，人们开始研究事务处理（严格可串行化）的重要性 [1, 28, 39]，因为读取不一致或过时的数据会给现实世界中的执行器带来严重问题。我们需要高效且一致地处理从传感器和移动设备摄取的数据。为了

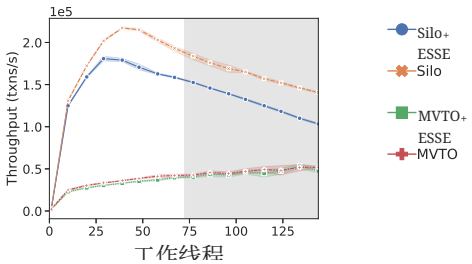


图13：TPC-C：单个仓库

高效地处理如此大量的更新，省略读操作不需要的更新至关重要。流系统使用负载削减 [13] 或反压 [14] 来限制数据提交速率。然而，这些技术无法确保事务正确性；它们会在数据提交到数据库之前将其省略。因此，我们无法使用并发控制协议来选择版本顺序，以提供正确的数据快照。在事务处理中，托马斯写规则 [36] 可以省略数据并确保可串行化。然而，据我们所知，TWR 无法保证严格可串行化。此外，TWR 仅适用于单版本时间戳排序协议，而该协议在现代内存数据库中已过时。SSE 也会执行写入省略，并在保持事务正确性的同时采用这些方法，而且适用于各种并发控制协议。

可交换理论是提高写操作并行性的另一个例子。CRDT [9] 和 Doppel [10], 等可交换系统定义了可交换操作集，例如 ADD 或 INCR。在可交换系统中，这些操作可以并行执行，同时保持一致性。相比之下，SSE 基于传统的页面模型接口（仅包含 READ 和 WRITE），专注于不可交换操作。因此，SSE 可以优化不包含可交换操作的物联网/电信应用的性能。

另一个能够生成可省略版本的协议实例是确定性数据库 [35, 37]。确定性数据库使用批处理方法进行并发控制：集中式事务管理器收集事务并将其划分为多个批次。Faleiro 等人设计了惰性事务执行 [15], , 使确定性数据库能够生成可省略版本。通过延迟操作的执行，它们只执行最终成为每个批次中最新版本的盲写。之后，数据库会删除其他版本。然而，与惰性求值不同，SSE 适用于非确定性协议，因为它不需要集中式事务管理器，也不需要预先了解事务。

多版本并发控制协议 [17, 25, 26] 可以为每个数据项保留多个版本。多版本读取可以避免单版本协议的高中止率，尤其适用于包含长事务 [23, 40, 41] 的工作负载。理论上，所有 MVCC 协议都可以利用多版本串行化图生成多个版本顺序。然而，传统协议 [22, 23, 25, 26, 34] 只能生成单一版本顺序。这是因为从所有可能的顺序中找出合适顺序的决策过程是 NP完全 [2, 32], , 如第4.2节所述。SSE 同样无法给出精确解，但它只增加一个候选版本顺序，从而降低了计算成本；该版本顺序更适合用于生成可安全省略的版本。

版本顺序，这有利于生成可安全省略的版本。

8 结论

我们提出了一种协议扩展方法——调度空间扩展器（SSE），以及其名为 ESSE 的优化实现。SSE 和 ESSE 可以扩展各种协议，使其理论上能够测试额外的版本顺序，从而生成可安全省略的版本，同时保持严格可串行化和可恢复性。为了评估我们的方法带来的性能提升，我们扩展了两种现有协议 Silo 和 MVTO，将 ESSE 纳入其中。我们预计，SSE 和 ESSE 能够帮助加速采用数据摄取查询的新兴系统。

参考文献

[1] Ronald Barber、Matt Huras、Guy Lohman、C. Mohan、Rene Mueller、Fatma Özcan、Hamid Pirahesh、Vijayshankar Raman、Richard Sidle、Oleg Sidorkin、Adam Storm、Yuanyuan Tian 和 Pinar Tözun。2016。Wildfire：并发高速数据摄取与分析。载于 SIGMOD会议。2077–2080。[2] Philip A Bernstein 和 Nathan Goodman。1983。多版本并发控制——理论与算法。TODS 8，4（1983），465–483。[3] Philip A Bernstein、Vassos Hadzilacos 和 Nathan Goodman。1987。数据库系统中的并发控制与恢复。Addison-Wesley Pub. Co. Inc.，马萨诸塞州雷丁。[4] Badrish Chandramouli、Guna Prasaad、Donald Kossmann、Justin Levandoski、James Hunter 和 Mike Barnett。2018。FASTER：支持原位更新的并发键值存储。载于 SIGMOD会议。275–290。[5] Brian F Cooper、Adam Silberstein、Erwin Tam、Raghu Ramakrishnan 和 Russell Sears。2010。使用 YCSB 对云服务系统进行基准测试。载于 SoCC。143–154。[6] Feilong Tang 等。2011。迈向情境感知的普适事务处理：模型与算法。载于 ICC。IEEE。https://doi.org/10.1109/icc.2011.5963475[7] Murat Demirbas 等。2008。TRANSACT：无线传感器/执行器网络编程事务框架。载于 IPSN。https://doi.org/10.1109/ IPSN.2008.44[8] M. Poess 等。2018。TPCx-IoT 分析：首个面向物联网网关系统的行业标准基准测试。载于 ICDE。1519–1530。[9] Marc Shapiro 等。2011。无冲突复制数据类型。载于 SSS。https: // doi.org/ 10 1007 978 3 642 24550 3_29 . / - - - - Neha Narula 等。阶段协调：[10] 2014 用于竞争性内存事务。载于 OSDI，Jason Flinn 和 Hank Levy（编）。https://www.usenix.org/conference/osdi14/technical-sessions/presentation/narula [11] Tu Stephen 等。2013。多核内存数据库中的高速事务。载于 SOSP。[12] Weikum Gerhard 等。2001。事务信息系统：并发控制与恢复的理论、算法和实践。爱思唯尔。[13] Yi-Cheng Tu 等。2006。流数据库中的负载削减：基于控制的方法。载于 PVLDB。http://dl.acm.org/citation.cfm?id =1164195 [14] 2018 Yad Tahir 等。BRPL：面向高吞吐量和移动物联网的反压RPL。IEEE移动计算汇刊（ 2018 10 1109 2017 2705680 ）。[15] 2014 https://doi.org/ . /TMC。. Jose M Faleiro、Alexander Thomson 和 Daniel J Abadi。事务在数据库系统中的惰性求值。载于 SIGMOD会议。15–26。[16] Hua Fan 和 Wojciech Golab。2019。Ocean Vista：用于高速地理分布式事务的基于流言协议的可见性控制。PVLDB 12，11（2019），1471–1484。[17] Alan Fekete、Dimitrios Liarokapis、Elizabeth O'Neil、Patrick O'Neil 和 Dennis Shasha。2005。使快照隔离达到可串行化。TODS 30，2（2005），492–528。[18] Jim Gray 和 Andreas Reuter。1992。事务处理：概念与技术。爱思唯尔。[19] Raman Grover 和 Michael J. Carey。2015。AsterixDB 中的数据摄取。载于《第18届数据库技术扩展国际会议论文集》，EDBT 2015，比利时布鲁塞尔，2015年3月23日至27日，Gustavo Alonso、Floris Geerts、Lucian Popa、Pablo Barceló、Jens Teubner、Martín Ugarte、Jan Van den Bussche 和 Jan Paredaens（编）。OpenProceedings.org，605–616。https://doi.org/10. 5441/002/edbt.2015.61[20] Vassos Hadzilacos。1988。数据库系统中的可靠性理论。JACM 35，1（1988），121–145。[21] Maurice P Herlihy 和 Jeannette M Wing。1990。线性化：并发对象的正确性条件。TOPLAS 12，3（1990），463–492。[22] Alfons Kemper 和 Thomas Neumann。2011。HyPer：基于虚拟内存快照的混合型OLTP与OLAP主存数据库系统。载于 ICDE。195–206。

[23] Kangnyeon Kim、Tianzheng Wang、Ryan Johnson 和 Ippokratis Pandis。2016。Ermia：面向异构工作负载的高速内存优化数据库系统。载于 *SIGMOD*会议 1675–1687。[24] Eddie Kohler。2020。Masstree 代码。https://github.com/kohler/masstree-beta。[25] Larson、Per-Åke，Blanas、Spyros，Diaconu、Cristian，Freedman、Craig，Patel、Jignesh M 和 Zwilling、Mike。2011。面向主存数据库的高性能并发控制机制。*PVLDB* 5，4（2011），298–309。[26] Hyeontaek 等 Lim。2017。Cicada：可靠而高速的多核内存事务。载于 *SIGMOD*会议 21–35。[27] IBM 软件集团信息管理部门。2009。TATP基准测试规范。http://tatpbenchmark.sourceforge.net/index.html。[28] John Meehan、Cansu Aslantas、Stan Zdonik、Nesime Tatbul 和 Jiang Du。2017。互联世界的数据摄取。。载于 *CIDR*。[29] C Mohan、Don Haderle、Bruce Lindsay、Hamid Pirahesh 和 Peter Schwarz。1992。ARIES：一种支持细粒度锁定和部分回滚、采用预写日志的事务恢复方法。*TODS* 17，1（1992），94–162。[30] Thomas Neumann、Tobias Mühlbauer 和 Alfons Kemper。2015。面向主存数据库系统的快速可串行化多版本并发控制。载于 *SIGMOD*会议。ACM，677–689。[31] Christos Papadimitriou。1986。数据库并发控制理论。Computer Science Press Inc.，马里兰州罗克维尔。[32] Christos H Papadimitriou 和 Paris C Kanellakis。1982。关于多版本并发控制。载于 *PODS*。76–82。[33] Karin Petersen、Mike Spreitzer、Douglas B. Terry、Marvin Theimer 和 Alan J. Demers。1997。弱一致性复制的灵活更新传播。载于 第*16*届 *ACM* 操作系统原理研讨会论文集，*SOSP 1997*，法国圣马洛，*1997*年*10*月*5*日至*8*日，Michel Banâtre、Henry M.Levy 和 William M. Waite（编）。ACM，288 301 10 1145 268998 266711 –。2012 [34] https://doi.org/。/。Dan R K Ports 和 Kevin Grittner。。Post- greSQL 中的可串行化快照隔离。*PVLDB* 5，12（2012），1850–1861。[35] Kun Ren、Alexander Thomson 和 Daniel J Abadi。2014 年。确定性数据库系统的优势与劣势评估。*PVLDB* 7，10（2014），821–832。[36] Robert H Thomas。1977 年。多副本数据库并发控制的多数共识方法。技术报告。BOLT BERANEK AND NEWMAN INC CAMBRIDGE MA。[37] Alexander Thomson、Thaddeus Diamond、Shu-Chun Weng、Kun Ren、Philip Shao 和 Daniel J Abadi。2012 年。Calvin：面向分区数据库系统的快速分布式事务。在 *SIGMOD*会议 1–12。[38] 事务处理性能委员会（TPC）。2010 年。事务处理性能委员会（TPC）：标准规范。TPC-C 基准测试版本 5.11.0（2010）。http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-c_ v5.11.0.pdf。[39] Xikui Wang 和 Michael J. Carey。2019 年。一个 IDEA：AsterixDB 的数据增强摄取框架。*Proc. VLDB Endow.* 12，11（2019），1485–1498。[40] Yingjun Wu、Joy Arulraj、Jiexi Lin、Ran Xian 和 Andrew Pavlo。2017 年。内存多版本并发控制的实证评估。*PVLDB* 10，7（2017），781–792。[41] Xiangyao Yu、George Bezerra、Andrew Pavlo、Srinivas Devadas 和 Michael Stonebraker。2014 年。凝视深渊：千核并发控制的评估。*PVLDB* 8，3（2014），209–220。[42] Wenting Zheng、Stephen Tu、Eddie Kohler 和 Barbara Liskov。2014 年。借助多核并行实现快速持久化与恢复的快速数据库。在 *OSDI*。465–477。