arXiv:2303.04103v1 [cs.DB] 7 Mar 2023

迈向深度在线聚合

Nikhil Sheoran[+]*、Supawit Chockchowwat*、Arav Chheda、Suwen Wang、Riya Verma、Yongjoo Park
nikhil.sheoran@databricks.com,{supawit2,aravmc2,suwenw2,rverm2,yongjoo}@illinois.edu
Databricks[+] 和伊利诺伊大学厄巴纳-香槟分校

摘要

在探索性数据分析中，人们通常希望在实际获得答案之前，就能知道可能会得到什么答案。为此，可以设计系统，在处理过程的早期基于部分数据处理，提供数据操作结果（例如 op(data)）的估计值。随着处理的数据越来越多，这些估计值会不断完善，最终收敛到精确答案。遗憾的是，现有技术——称为在线聚合（OLA）——仅适用于单个操作；也就是说，我们无法获得 op(op(data)) 或 op(...(op(data))) 的估计值。如果深度在线聚合成为可能，数据分析师就能通过复杂的级联操作，以更具交互性的方式探索数据。

在本研究中，我们通过深度在线聚合 迈出了一步，提出了演化数据框 （edf），这是一种新颖的数据模型，通过表示一种在集合操作下封闭 的演化结构化数据（具有收敛估计值），为嵌套操作——op(...(op(data)))——提供在线聚合。也就是说，op(edf) 会生成另一个 edf；因此，我们可以自由地对 edf 连续应用操作，并为每个操作获得在线聚合输出。我们通过 Wake（一个基于 edf 的在线分析处理系统）评估其可行性，并将其与最先进的在线聚合系统和非在线聚合系统进行比较。在 TPC-H 数据集上的实验中，与传统系统相比，Wake 更快地产生首批估计值 4.93×（中位数），而获得精确答案时的 1.3× 中位数减速比为。除了通用性之外，Wake 在生成相对误差低于 1% 的估计值方面，也比现有在线聚合系统 1.92× 更快（中位数）。

ACM 引用格式：
Nikhil Sheoran、Supawit Chockchowwat、Arav Chheda、Suwen Wang、Riya Verma、Yongjoo Park。2023 年。迈向深度在线聚合。载于*ACM* 数据管理特别兴趣小组会议论文集（*SIGMOD '23*）。ACM，美国纽约州纽约市，16 页。https://doi.org/10.1145/ 1122445.1122456

1 引言

对于即席数据探索，获取洞见最快的方法是通过计算与（未来的）最终答案高度相似的中间估计值，从部分数据处理中尽可能提取有用信息，同时持续优化这些估计值，直至处理完全部数据。自 Hellerstein 等人开创性地提出这一数据处理范式以来 [39],——称为在线聚合 （OLA）——人们从多个方向对其展开研究，以

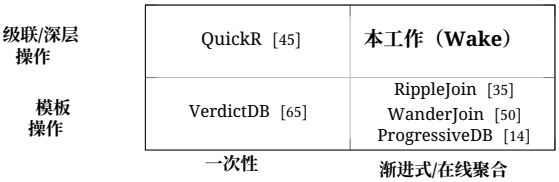


图 1：我们的系统 Wake 支持深层操作的在线聚合

借助新型架构 [14,61, 84], 新颖的连接算法 [22, 30, 35, 50, 53, 58, 81],对子查询的支持 [90], 专用索引 [28, 46, 64, 85], 等，提升其通用性和性能。

遗憾的是，现有在线聚合存在一个普遍局限，使其不 再是当今数据探索的首选。也就是说，现有在线聚合不允许对先前的在线聚合输出执行后续操作。 为说明这一点，假设有一个用类似 pandas 的方法 [55] 表示的数据分析会话[1]，如下所示：

```
lineitem = read_csv('...')
# item count for each order
order_qty = lineitem.sum(qty, by=orderkey)
# select only the large orders
lg_orders = order_qty.filter(sum_qty > 300)
# find the customers with biggest order sizes
lg_order_cust = lg_orders.join(orders).join(customer)
qty_per_cust = lg_order_cust.sum(sum_qty, by=name)
top_cust = qty_per_cust.sort(sum_qty, desc=True).limit(100)
```

第一个输出（L3）会再次进行聚合/筛选，以找出顶级客户（L5-L9）。现有在线聚合可以增量地计算第一个输出（即 `order_qty`），但在得到最终答案之前，无法再以在线聚合的方式对其进行筛选/连接/求和处理。具体来说，现有在线聚合有两个局限性。首先，它将每个查询视为独立查询，不会推理其输出可能如何被后续操作使用。其次，它无法处理任意深度的查询；也就是说，即使我们组合一个（很长的）查询来直接计算 `avg_order_size`，也无法生成聚合之上的聚合及其正确调整（注意： 这个问题不同于增量视图维护，后者始终会产生精确结果）。

在本工作中，我们通过演化数据框（或 edf）来应对这一限制。这是一种全新的数据/处理模型，旨在实现深度在线聚合——能够对先前的在线聚合输出应用后续操作，从而生成另一个在线聚合输出。对于每个操作，edf 都能为最终答案提供逐渐收敛的估计值——预期误差不断减小（§4.5）——其依据是一项共同假设：未见数据与已观测数据相似；一旦处理完全部数据，每个 edf 都会与传统数据系统所能获得的结果完全一致。为了评估我们方法的可行性，我们

特此许可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，但前提是不得为营利或商业利益而制作或分发副本，且副本必须附有本声明以及首页上的完整引用。对于本作品中由 ACM 以外的其他方拥有版权的组件，必须遵守其版权规定。在注明出处的情况下允许撰写摘要。若要以其他方式复制、重新发表、发布到服务器或重新分发到列表，须事先获得具体许可和/或支付费用。请通过 permissions@acm.org 向我们申请许可。*SIGMOD '23，2023*年*6*月*18*日至*23*日，美国华盛顿州西雅图© 2023 计算机协会。ACM ISBN 978-1-4503-XXXX-X/18/06...$15.00 https://doi.org/10.1145/1122445.1122456

[+]在伊利诺伊大学厄巴纳-香槟分校期间完成的工作。*这些作者对本研究的贡献相同。[1]此示例数据分析是对 TPC-H 第18条查询的改写版本。

表 1：已有工作的汇总。▲/✗ 表示支持有限或不支持。

系统/方法	OLA?	深度查询?	创新点	不足/差异
OLA（Hellerstein）[39]	✔	✗	首个在线聚合提案	仅适用于不含连接/子查询的简单 SQL
RippleJoin [35, 53]	✔	▲	在线聚合的连接算法	多个连接的指数复杂度
WanderJoin [50]	✔	▲	支持多个连接	需要索引 / 可能无法生成精确答案
G-OLA [90]	✔	▲	支持带子查询的筛选	有些数据需要重复处理
QuickR [45]	✗	✔	下推采样算子	非在线聚合（每个查询的答案都来自单个样本）
VerdictDB [65]	✗	▲	与平台无关	非在线聚合（每个查询的答案均来自单个样本）
我们的方案（Wake）	✔	✔	支持深度嵌套操作	可能需要更多内存（§4）；需要调整分区大小（§8.7）；没有查询优化器

实现了基于 edf 的在线分析处理系统 Wake[2]，并将其性能与现有在线聚合系统（ProgressiveDB [14], Wan- derJoin [50]）以及现代数据系统/库（Presto [75], PostgreSQL、Polars [70]）进行了比较。据我们所知，此前没有研究正式探讨过深度在线聚合的数据模型，也没有通过全面的实验评估其在实际用例中的效率。

挑战。 给定一个查询（或操作），现有在线聚合可以理解为一个将输入数据转换为一系列中间结果或最终结果的过程。值得注意的是，输入与输出属于不同类型，这正是在线聚合无法应用于在线聚合结果的根本原因。具体而言，我们观察到以下挑战。首先，现有的结构化数据模型（我们称之为数据框）不足以表示逐步变化的数据框；这类数据框可能包含近似属性值，其行数也可能发生变化。其次，现有的面向集合（或关系型）操作是为最终数据框设计的，而非近似数据框；直接将常规操作应用于演化数据框可能会产生有偏的值，因为部分数据必须视为样本。第三，高性能至关重要。对现有数据模型进行任何由在线聚合驱动的扩展都可能带来开销，而开销必须足够小，才能相比传统的一次性 方法显著提升交互式计算能力。

我们的方法。 我们的新数据模型 edf 在一类集合操作下是封闭的；也就是说，edf 表示不断演化的 OLA 输出，当通过集合操作进行转换后，会再次产生另一个 edf。具体而言，edf 具有以下关键特征。首先，每个 edf 在处理完全部数据后始终会收敛到精确的/最终答案。其次，为确保对一个 edf 执行操作后产生另一个 edf，我们的集合操作——通过 map、filter、join 和 agg 表示——在每个 edf 内维护两项独特属性，即可变属性和基数增长，这对于生成准确估计至关重要（§2.3）。第三，我们的内部处理旨在尽可能减少冗余计算。

正交工作。 OLA（包括深度在线聚合）可以理解为一种将涉及聚合的查询转换为增量计算逻辑的机制，因此它与增量计算框架 [56, 59] 以及增量视图维护 [10, 91]正交。OLA 属于近似查询处理这一更广泛的查询处理范式；例如，基于采样的近似查询处理 [8, 45, 65] 会生成单个近似答案（而不是像 OLA 那样生成一系列持续细化的答案）。

贡献。 本文分享以下发现：

(1) 我们的新数据模型 演化数据框（edf）支持连续执行在线聚合（OLA）操作。（§3）(2) 我们表示常见集合操作的处理模型，能够依靠内部推断技术将一个 edf 转换为另一个 edf（并确保其属性正确）。（§4 和 §5）(3) 扩展后的数据模型可在流水线中传播置信区间。（§6）(4) Wake 针对 OLA 的多线程实现可通过流水线并行实现高速处理。（§7）(5) 与非 OLA 相比，Wake 仅产生 1.3× 的开销，却能比相应的最终答案更快地产生中间结果 4.93×。TPC-H 数据集的首个中间答案的相对准确度中位数为 2.70%，并迅速趋近于零。（§8）

上述技术贡献是在我们介绍设计演化数据框背后的动机（§2）之后提出的。

2 动机

我们首先说明为何需要 Deep OLA 专用数据模型（§2.1）。为实现深度在线聚合，我们列举了所提出的框架必须处理的若干情形（§2.2），并讨论其独特的新属性（§2.3）。

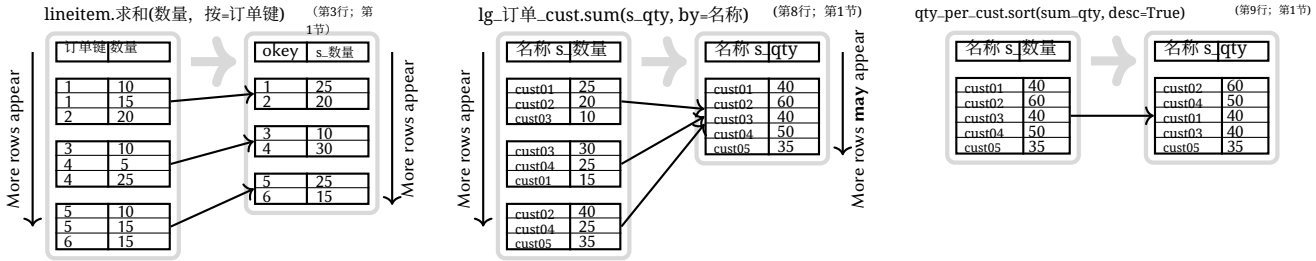
2.1 在线聚合输入/输出作为类型

我们认为，将在线聚合的输出形式化为一种类型至关重要。例如，整数（如 -1、0、1、2）在加法下封闭；因此，我们可以对先前加法的输出连续执行加法，例如 (1 + 3) + 3 = 4 + 3，而无需关心值 4 （= 1+3）最初是如何得到的。同样，数据库关系（表示二维结构化数据）在关系运算（如投影、聚合、连接）下封闭。

相比之下，现有在线聚合旨在以关系作为输入，并输出一系列关系，每个关系都代表最终答案的一个收敛估计值（误差预期会逐渐减小）。因此，在线聚合的输入与输出类型不同（也就是说，关系在在线聚合下不封闭），这使得将连续的在线聚合应用于前一个在线聚合的输出并非易事。从理论上说，可以对每个估计关系再应用一次在线聚合；然而，首先，这种做法的含义不易解释；其次，如果天真地对每个估计值都应用在线聚合，那么输出估计值的数量会随着操作次数呈指数级增长。这促使我们引入一种在在线聚合下封闭的新类型，并将其称为演化数据框（edf）。

然而，与整数或关系不同，edf 表示的是一个演化对象（而非内存中的状态）；因此，如何理解或定义将一个演化对象映射为另一个演化对象的操作，就不那么显而易见了。我们将 edf 视为由以下内容构成的对象：

[2]Wake 代表 **W**e **A**lready **K**now**E**nough（我们已经知道得足够多）。



(a) 情形 1：保序局部操作 **(b) 情况 2：带推理的洗牌操作** **(c) 情况 3：不进行推理的洗牌操作**

图 2：在线聚合的数据框转换示例。对于每种情况，输入数据框（左侧）通过顶部标注的某种操作转换为输出数据框（右侧）。随着处理的数据增多，输出基数可能会增长。

一系列 K 状态，其中每个状态都包含一个逐渐收敛于最终答案的估计值。此外，我们将 edf 上的操作视为一种映射：它根据操作的类型和参数，适当地修改每个状态中的信息（即估计值和元数据），从而将一组状态映射为另一组状态。

为了形式化这些状态及其上的操作，我们先介绍几个示例数据框，它们代表了各数据框在在线聚合过程中可能经历的不同类型转换（§2.2）；基于这些示例，我们将在后续各节（§3、§4、§5）中形式化定义 edf。

2.2 在线聚合操作：案例分析

结合第1节中的示例，我们讨论不同操作（如聚合、筛选、连接、限制）如何将输入数据框转换为另一种形式，这构成了§3和§4中数据模型的基础。

保序局部操作。 假设输入数据框（例如明细项表）从 CSV 文件中读取，其中包含按某个键（例如订单键）排序或聚类的原始数据。在逐行处理筛选和映射操作时，输入数据框中新出现的行不会影响已处理行的结果。第1节中的示例里，L1（读取_CSV）、L3（按键分组）、L5（筛选）和 L7（连接）都属于这一类。示意见图 2a。具体来说，设 df 是由两个分区组成的输入数据框，即 df = [df_1、df_2], , 其中"[]"表示并集/追加。对于此类局部操作 op，op(df) = [op(df_1) 、 op(df_2)]；因此，与下文将介绍的其他情况不同，计算 op(df_2) 不依赖于 df_1，从而能够增量地产生输出。同样，内连接/左连接（例如将明细项表与订单连接）也是保序局部操作，因为 join(dfa,dfb) = [join(dfa_1, dfb)、join(dfa_2, dfb)]；其物理计划可以选择不同的连接算法，例如渐进式合并 [30] 或哈希连接。

带推断的洗牌操作。 如果输入数据框按非键属性进行聚合，例如第1节中的 lg_order_cust.sum(sum_qty, by=name)，则出于三个原因，我们需要特别考虑。首先，随着我们处理来自输入的更多数据，已经生成的输出（原始）聚合值可能会发生变化。其次，原始聚合值可能需要适当缩放，才能得到准确的——理想情况下无偏的——估计值。第三，随着我们处理来自输入的更多数据，输出中可能会出现更多行（包含新的分组键值）；后续操作将使用这个输出数据框，因此需要对这些行进行定量建模。第1节中的示例里，L8（按非键属性分组）属于这一类别。图2b展示了这种情况下的数据流：输入中新出现的行可能会影响已生成的输出。具体而言，设 df = [df_1、df_2]。对于洗牌操作 op，op(df) = op(df_1) ⊕ op(df_2)，其中 ⊕ 表示基于键的合并，可表示为 A ⊕ B = agg(union(A, B), by=key).[3]

如果在在线聚合期间<style id='1'>输入数据框中可能出现更多行，则必须对该合并结果进行缩放，以生成准确估计。例如，如果输入数据框表示一个正在获取更多数据的基表，则当前观测到的部分必须被视为输入数据的样本；因此，需要根据当前行数与整个数据规模之间的比率（即采样比率）对原始求和值进行向上缩放。相反，如果输入数据框例如是某次聚合的结果（具有低基数的分组属性），我们不太可能在输入数据框中看到（许多）新行；因此，当前观测到的集合就是整个集合，无需额外缩放。虽然各个聚合值可能只是近似值，但由于它们已经是最终答案的收敛估计，原始求和值（不进行额外缩放）同样是输出的收敛估计（第4.5节）。

无需推理的重排操作。 排序和限制等操作必须使用完整的输入数据，因此无法通过特殊处理来改善输出质量。在这些情况下，输入发生变化后，只需重新计算整个输出；与情况 1 和情况 2不同，这不可避免地会导致冗余计算。第1节中的示例表明，L9（排序和限制）属于此类。然而，对于大规模聚合，这些情况 3操作通常出现在后续阶段，用于限制/排序供用户使用的结果（例如条形图）；因此，在整体计算的背景下，它们的开销并不显著。不过，如果用户的意图是对完整数据进行排序并将结果持久化到磁盘，在线聚合框架（包括我们的框架）不会提供额外收益。

2.3 必需属性

§2.2 中的案例分析揭示了两类变化：属性值的变化（例如，随着聚合更多输入行而发生的变化）以及基数增长（例如，随着输入行出现而对其进行筛选）。

可变属性。 设可变属性 是指其值可能发生变化的属性，而常量属性 是指一种属性

[3]合并操作适用于类似求和的（或可合并的）操作，这类操作可以定义加法。因此，平均值函数需要通过分别计算求和函数和计数函数来得出。一个尤为棘手的情况是去重计数，我们会维护精确集合（而不是基于 HLL 的草图 [32]）。

其值永不 变化。区分可变属性与常量属性很有用，因为输入属性类型会影响我们应如何（重新）计算输出。例如，对常量属性进行筛选（如名称 like '%east%'）可以增量地处理（情况 1），而对可变属性进行筛选（如 sum_qty > 200）则需要重新计算（情况 3）。

基数增长。正如情况 2 中所观察到的，要正确估计输出聚合结果，必须掌握输入数据框中可能出现的行数。为此，我们定义了基数增长：查询进度与分组基数（即属于某个聚合组的行数）之间的关系。研究了多种基数增长模式后，我们便可以选择最合适的增长模式来预测最终聚合结果。例如，假设我们知道分组基数随查询进度线性增长；那么，如果查询进度达到 25%，我们预计最终分组基数为 4× 行。

3 数据模型

本节从用户的角度介绍演化数据框（edf）。具体而言，我们将介绍其数据模型（§3.1）、相关操作（§3.2）以及当前的局限性（§3.3）。

3.1 演化数据框

演化数据框（edf）表示一种逐步变化的结构化数据（即数据框），其新行会不断出现，且／或属性值会发生变化。其形式化定义如下：

```
edf := t -> df  (0 <= t <= 1)
df := list((attr1, attr2, ..., attrM))
attr := constant_attr | mutable_attr
```

其中，（attr1, attr2, ..., attrM）定义了一个模式。一个或多个（恒定）属性用作主键（或简称键），以唯一标识元组。edf 的行数（即列表的长度）可能随时间增加，而可变的_属性值可能发生变化。

闭包性质。有效的 edf 必须满足两个性质，即 2C。（***1***）一致性：与 edf 相关联的所有 `df` 都具有相同的模式；也就是说，其属性列表在*t*期间保持不变。（***2***）收敛性：与 t_2 相关联的 `df`，相较于与 t_1（$t_1 < t_2$）相关联的 `df`，是对精确答案更准确的估计值；而在 $t = 1$ 处的 `df` 是精确答案。换句话说，对*edf*执行的操作必须生成一个*edf*，以确保满足这两个性质，从而保证*edf*在这些操作下具有闭包性质。

数据组织。 edf 的列表使用一个或多个分区进行组织，每个分区（简单来说）都是列表的一个子集，这些数据会一起存储或访问（例如，存储在同一存储设备上）。edf 可以具有一个聚类键，它是一个用于确定行在各分区之间分配位置的属性列表；例如，如果某个 edf 的聚类键是 orderkey，则一个分区可能包含 orderkey 介于 1 和 10 之间的行；此时，其他分区不得包含 orderkey=5 对应的行。由对其他 edf 执行操作而产生的 edf 也可以具有聚类键，详见 §4。行簇 指同处于一个分区中的那些行。

访问值。如 §2 所述，为了表示演化数据框，edf 会维护状态，其中每个状态都表示对最终答案的一个逐渐收敛的估计值，最新状态的期望准确度最高。例如，如果某个 edf 用于

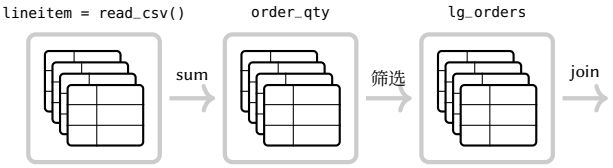


图 3：不断演变的数据框（edf）及其操作的用户视图。每个 edf 表示一个或多个状态。

lineitem.count(by=linestatus)，那么 edf 每一行中的计数值，都是同一分组最终计数值的无偏估计（也就是说，它们的期望值相等）。最新状态可通过 edf.get() 获取。如果 edf.is_final 为 true，则最新状态包含最终答案；系统一旦发现没有更多数据需要处理（收到文件结束时），便会设置 is_final 标志。最终答案可通过 edf.get_final() 获取；如果数据尚未全部处理完，该方法可能会阻塞，直至处理完成。

创建 ***EDF***。 创建 edf 有两种方式：可以直接从数据源创建 edf，也可以通过对另一个 edf 执行操作来创建 edf，具体如下：

```
read := data_source -> edf
edf_op := (edf, op) -> edf
op  := agg(attrs, by) | filter(predicate)
     | map(function) | join(df, options)
agg := sum | count | avg | count_distinct | min | max
     | var | stddev
```

其中第 3.2 节介绍了各项操作的具体细节。从数据源创建 edf 时，会从元数据中获取聚类键（§4.4）。这些操作和聚合类型足以表达全部 22 个 TPC-H 基准测试查询 [6]。

3.2 演化数据框上的操作

我们的系统（Wake）以一种独特的方式实现投影（映射）、连接、选择（筛选）和聚合等关系运算，以尽可能创造在线聚合（OLA）的机会，具体如下。

映射。 edf.map() 类似于投影操作，例如选择属性子集、创建派生属性等。我们的 map() 独特之处在于，其参数中的函数作用于一个或多个分区，而非每一行。具体而言，设 edf = [p1, p2, ..., pK]，其中 pi 是行的一个分区；那么，edf.map(func) 会创建另一个 edf2，使得 edf2 = [func([p1, p2]), func([p3, p4]), ..., func([pK-1, pK])]。也就是说，func 将一个数据框映射为另一个数据框，其中每个输入数据框都是一组分区。此处，每次调用 func 时传入的分区数量（本例中为两个）由分区大小决定。

这种设计背后有两个原因。首先，这种方法支持按分区（本地）执行操作，使一些不那么容易表达的操作变得简单，例如，在每个订单中找出排序最靠前的两种商品，其中一个订单由多个（商品，数量）元组组成。使用关系运算（在 SQL 中）表达这一操作时，可能需要用到较少见的函数（例如 group_concat [2]，find_in_set [1]）。其次，这种方法无需为按行函数添加并行化/向量化逻辑，就能轻松实现高效处理。

连接。 edf.join(edf2,options) 根据 options 中指定的设置（例如连接方法（inner/left）和连接键）将 edf 与 edf2 连接起来。Wake 会根据连接键和聚类键使用不同的连接方法

（即哈希连接或渐进式合并 [30]）。具体而言，如果 edf 和 edf2 分别按各自的连接键进行了聚类，Wake 就会执行归并连接；否则，Wake 会执行哈希连接，将 edf 作为探测表，将 edf2 作为构建表（用于创建哈希表）。如果链式执行多个连接（例如 edf.join(edf2).join(edf3)），且必须使用哈希连接，Wake 实际上会执行右深连接，同时为 edf2 和 edf3 并行构建哈希表，这对于星型模式模型很有效 [40]。

聚合。 edf.agg(cols, by_attr) 对一组行进行聚合（针对每个 by_attr），其中 agg 是允许使用的聚合函数之一。对于深度在线聚合，我们对聚合进行特殊处理，因为为了生成准确的/无偏的估计值，可能需要根据观测到的数据框大小与完整数据框大小之间的比率，对部分聚合的结果进行调整；而且，如果我们聚合的数据是另一次聚合的结果，那么完整数据框的大小也可能不确定，因此需要进一步推断。第 4 节和第 5 节将进一步介绍我们的推断逻辑。

筛选。 edf.filter(predicate) 类似于关系代数中的选择操作（或 SQL 中的 where 子句）；也就是说，该操作会生成另一个 edf，其中仅包含满足给定谓词的行。与 edf.map(...) 类似，谓词会同时应用于一个或多个分区 。一般来说，filter() 可以理解为 map() 的别名，但它可能会生成空集作为输出。具体而言，对于 edf $_2$ = [func([p1, p2]), func([p3, p4]), ..., func([pK-1, pK])], , 任何一个 func 都可能生成空集。

我们从用户的角度介绍了四种操作（即映射、连接、聚合、筛选）；然而，内部处理可能因模式/操作而异，我们将在第 4 节中介绍。

3.3 局限性

在某些情况下，为了在最大限度减少冗余计算的同时得出正确结果，某些操作必须阻塞（例如，对可变属性进行筛选/连接）。虽然我们的内部处理逻辑（第 4 节）能够区分这类情况，但对于最终用户而言，尤其是刚开始使用我们框架的用户来说，这一点可能不太直观。为了最大限度地利用深度在线聚合的机会，更高级的用户可以仔细组织数据和操作，这或许可视为一种技能（类似于在关系数据库管理系统中提供连接提示）；然而，也有人可能会认为，这意味着系统还不够智能，无法自动优化用户操作。本工作的目标是构建深度在线聚合的基础模块，而不是像采用基于成本的优化器的关系数据库管理系统那样优化端到端的声明式查询；后者留待未来工作。

4 内部处理

根据操作类型的不同，我们的系统 Wake 会采用不同的方法来更新 edf。

4.1 演化数据框的属性

除了 §3.1 中描述的模式外，每个 edf 还维护两个额外属性，即进度 和增长，用于定量表征其演化。

进度。 进度 ($0 \le t \le 1$)是迄今为止已读取/处理的（原始）输入元组数与必须处理以获得最终答案的（输入）元组总 数之比

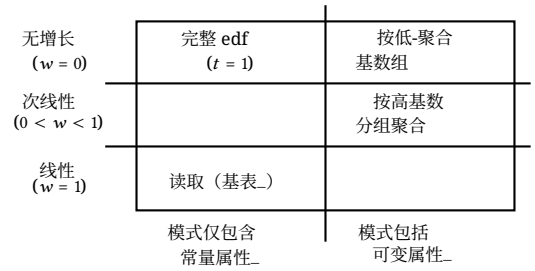


图 4：按增长程度对 edf 类型分类，Y 轴表示增长程度，X 轴表示属性类型，方框中列出示例。

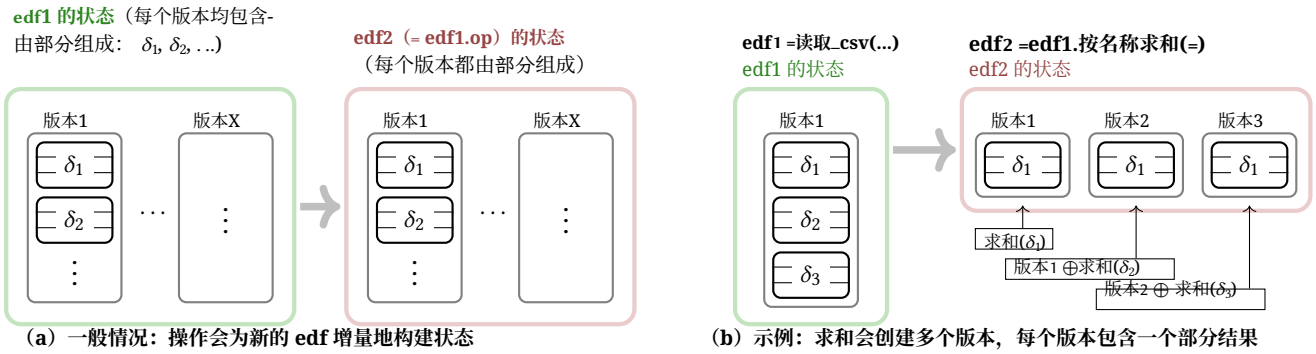
；元组总计数来自元数据（§4.4）。例如，如果一个基表由十个大小相同的分区组成，而我们只读取/处理了其中一个，t 就是 1/10（= 0.1）。另一方面，如果读取/处理了全部数据（例如，十个分区全部读取），t 就是 1。如果 t 为 1，则 edf.is_final=为 True。

增长。 增长 描述当前元组计数的增长情况，以预测最终元组计数。Wake 使用历史观测值将增长简洁地建模为单项式 ct^w。图 4 给出了一些具有不同 w 值的示例。增长反映局部元组计数，而进度 t 反映查询输入比率（当前值与未来值之间的比率）。例如，如果我们正在计算平均值（不含分组属性），输出元组计数始终为 1（除非结果为空）；因此，$w = 0$ 和 $c = 1$。另一方面，$t < 1$ 如果我们仍在读取/处理输入数据。

示例。 例如，当 edf 表示一个基表时，这些变量——（c，w）和 t——彼此的联系更加紧密；此时，w 等于 1，c 等于输入大小，因为在这种情况下，该 edf 的输出（或该 edf 表示的数据）恰好与从数据源（例如目录中的 CSV 文件）检索到的输入数据量相符。然而，在其他情况下，w 可能小于 1，这表明增长是次线性的。例如，如果某个 edf 表示按对数基数分组属性进行聚合的结果——lineitem.count（按=linestatus）——输出行数不太可能增加（尽管其聚合值可能会变化）；因此，我们有 $t = 1$。图 4 根据增长程度（w）和属性类型（常量/可变）对 edf 属性类型进行了分类。表格单元格列出了一些会产生相应属性的 edf 示例。例如，如果 edf = 读取（基表_），其模式仅包含常量属性，并且其输出大小随输入数据线性增长（$w = 1$）。另一个例子是表示按高基数分组属性进行聚合结果的 edf（students.count（按=first_name））。在这种情况下，属性值可能会变化，也可能出现新的分组键（每次遇到新的名字时）。因此，其模式包含可变属性，并且 w 介于 0（无增长）和 1（线性增长）之间。

4.2 状态表示

从内部来看，edf 表示一个具有离散状态的演化数据框。状态分为两种：内在状态 和外在状态。外在状态表示收敛的无偏估计值；因此，它们会被下游 edf 或其他应用使用，而内在状态则用于在调整和/或估计之前增量地维护计算值。



（a）一般情况：操作会为新的 edf 增量地构建状态　　（b）示例：求和会创建多个版本，每个版本包含一个部分结果

图5：edf 的基于状态的表示及对其执行的操作（即，从 edf1 的外在状态到 edf2 的内在状态）。每个 edf（从概念上讲）在每次向最新版本添加部分结果或创建新版本时，都会定义一个新状态。对 edf 执行操作会创建一个新的 edf（及其状态），并将计算冗余降至最低。

示例。 假设我们要按学生的家庭所在州统计学生人数。令 edf1 表示我们正在读取的数据集；我们已读取十个大小相等的分区中的一个，第一个分区包含 2 名来自伊利诺伊州的学生和 1 名来自密歇根州的学生。edf1 的内在状态 α_1 变为 [[(id1, 伊利诺伊州), (id2, 伊利诺伊州), (id3, 密歇根州)]]。对于 edf1，其外在状态 β_1 与 α_1 相同，因为 edf1 表示基表中的元组，无需调整。令 edf2 表示 edf1.count(by=状态)；其内在状态 α_2 变为 [[(伊利诺伊州, 2), (密歇根州, 1)]]。为表示无偏估计值，在假定未观测到的（九个）分区与已观测到的（第一个）分区具有相同分布的情况下，相应地缩放 edf2 的外在状态 β_2；因此， β_2 变为 [(伊利诺伊州, 20), (密歇根州, 10)]。

我们又读取了一个分区（因此已读取两个分区）；第二个分区包含 1 名来自伊利诺伊州的学生和 1 名来自密歇根州的学生。 $\alpha_1 = \beta_1$ 变为 [[（id1，伊利诺伊州）、（id2，伊利诺伊州）、（id3，密歇根州）], [（id4，伊利诺伊州）、（id5，密歇根州）]]（请注意，新添加的元组位于单独的列表中）。为了（增量地）更新 α_2，我们首先对 β_1 的第二个列表进行聚合，暂时得到 [（伊利诺伊州，1）、（密歇根州，1）]，然后使用基于键的求和（⊕）将其合并到 α_2 中，如 §2.2 所述，最终得到 α_2 = [（伊利诺伊州，3）、（密歇根州，2）]。为了从 α_2 获得无偏估计，我们根据当前已处理元组与元组总数之间的比例（即 2:10）对各个聚合值进行缩放，从而得到 edf2 的外在状态 β_2 = [（伊利诺伊州，15）、（密歇根州，10）]。

请注意，根据 edf 的不同，我们采用了两种不同的方法来更新内在状态。对于 edf1，我们插入了新的元组，为 α_1创建了更长的列表；相反，对于 edf2，我们用另一组聚合值替换了旧的聚合值集合。我们将这些情况系统地区分为增量更新和完全更新，具体如下。

内在状态。为了支持增量更新和完全更新，我们使用版本和部分来组织 edf 的状态（一个部分是每个版本内行的子集），如图5所示。创建新版本意味着进行完整刷新，而向（某个 edf 的）每个版本追加一个或多个部分，则意味着进行增量更新。

例如，假设一个 edf——表示某个班级的（first_名称，计数）统计学——有一个版本 $\alpha_{(1)}$，且该版本当前包含一个部分结果，其中有一个元组（例如， [(mike, 4)]）。我们可以通过追加另一个部分结果来增量地更新该版本

部分结果（例如， [(sarah, 2)]）；这样，版本 $\alpha_{(1)}$ 就表示两个元组 [(mike, 4)、(sarah, 2)]，，即两个部分结果的并集。

具体而言，内在状态 $\boldsymbol{\alpha}$ 是一种二维结构（图5），由一个或多个版本（α（1） α（2）……， α（v））组成，其中每个版本 α（i）包含一个或多个部分（δ_1，……， δ_p）。这些部分在键方面彼此互斥；也就是说，这些部分划分了各个版本，而每个 edf 在执行操作时都会确保这一点（§4.3）。要获取最新的内在状态，我们可以对最新版本（$\boldsymbol{\alpha}$（v））中的所有部分取并集。

外在状态。 外在状态用于区分（外部的）估计值与（内部的）原始值。在许多操作中，例如映射/筛选/连接，外在状态只是内在状态的别名，因为这些操作不需要任何特殊调整即可获得无偏估计。外在状态主要用于聚合操作。

调整有两种类型。第一种情况是聚合不可合并 （例如去重计数），因此需要不同的聚合前表示。令 edf2 = edf1.count_distinct(name)，其中 edf1 的内在状态 α_1 由两个部分结果 δ_1 和 δ_1组成。要增量地计算去重计数（即先使用 δ_1，然后使用 δ_1更新），仅知道 δ_1 中出现的唯一值数量是不够的，因为 count_distinct(δ_1) + count_distinct(δ_2) 不 等于 count_distinct($\delta_1 \cup \delta_2$)；我们需要记录 δ_1 中的所有单个唯一值，以便正确检查δ_2 中的元组是否与 δ_1中的任何值重叠。在这种情况下，内在状态必须包含一组唯一值，然后可用它们增量地计算去重计数值（最终出现在外在状态中）。

第二种情况是：如果输入中观察到更多元组，预期聚合值会增加或变化；此时，应将当前观察到的元组视为一个样本。求和就是一个例子，正如我们已经描述过的。也就是说，将当前的原始求和结果视为样本所得的结果，并考虑当前输入基数与预计最终输入基数之比，就可以得到无偏估计。这种缩放机制（称为基于增长的缩放）将在 §5 中介绍。

表 2：各 edf 操作的状态转换。GBI：基于增长的推断。

edf 操作	内在表示	合并（⊕）	int. → ext.
map	映射后的元组	并集	恒等
join	连接后的元组	并集	恒等
筛选	筛选后的元组	并集	恒等
计数	按键计数	按键求和	GBI
sum	按键求和	按键求和	GBI
avg	按键求和/按键计数	按键求和/按键求和	GBI
不同值计数_	按键计数	按键求和	GBI
min	按键最小值	按键最小值	GBI
max	按键取最大值	按键取最大值	GBI
var	按键计算方差/总和/计数	按键计算平均值/总和/总和	GBI
标准差	按键计算方差/总和/计数	按键计算平均值/总和/总和	GBI

4.3 操作：状态转换

edf2 = edf1.op(...) 中的操作是一个状态转换过程，它将 edf1 的外在状态转换为 edf2 的内在状态（随后可以按上述方式进行可选缩放，以生成 edf2 的外在状态）。本节将介绍如何针对每种操作，将一个版本的外在状态 $\beta_1 = [\delta_1, \ldots, \delta_p]$ 增量地 转换为一个版本的内在状态 α_2。

合并。 为了针对 op 增量地构建 α_2（每次提供 δ_1、.....、δ_p），我们利用这样一个事实：存在一种内部状态表示与合并操作 (⊕) 的组合，可以满足 $\text{op}([\delta_1, \ldots, \delta_p]) = \text{op}(\delta_1) \oplus \ldots\ldots \oplus \text{op}(\delta_p)$。也就是说，给定 δ_1，我们可以先计算 $\text{op}(\delta_1)$；然后，给定δ_2，通过将 $\text{op}(\delta_2)$ 合并到之前的结果中来更新结果；对于每个部分结果，都会继续执行这一更新操作。

例如，假设我们要计算 $\text{avg}([\delta_1, \delta_2, \delta_3])$，更具体地说，是计算美国各州的平均工资。为了增量地计算平均值，我们首先根据 δ_1 计算每个州的 (计数,求和_工资)，并将其存储为内部状态。给定下一个部分结果 (δ_2)，我们再次根据 δ_2计算每个州的 (计数, 求和_工资)，然后将这些聚合结果加到每个州先前的结果中；这等同于直接根据 δ_1 和 δ_2的并集计算 (计数, 求和_工资)。请注意，对于每个 op，内部状态表示和合并操作各不相同，我们在表 2 中进行了总结。

主键。 如 §3.1 所述，一个或多个常量属性充当主键，用于唯一标识 edf 中的元组。因此，我们的转换始终会为新创建的 edf 定义主键。映射/筛选/连接会保留输入 edf 的相同键。执行聚合后，分组属性会成为新 edf 的键。

聚类键。 聚类键决定 edf 元组的物理排序。如果聚合的分组属性本身不是聚类键，那么聚类键会在聚合后发生变化。

其他属性。 除了属性类型外，新的 edf 还必须维护两个内部属性：进度和增长。由于进度是根据原始输入元组定义的比率，每个操作只需将进度值原样传递给下一个 edf，无需进行任何修改。相比之下，增长会在操作过程中重新计算（每次使用新的部分结果或版本时），以准确估计未来将新增的元组数量。我们将在 §5 中讨论这一逻辑。

4.4 基表统计信息

表示基表的 edf（通过从 CSV、Apache Parquet 或其他来源读取数据）必须提供：(1) 文件名列表，(2) 每个文件中的元组数量，以及 (3) 与正在读取的表对应、带有主键/聚类键的属性。这就是 Wake 要求从底层数据中提供的全部元数据，无需任何其他统计学信息。这些元数据信息用于计算进度 (t)。

4.5 edf 属性的闭包

Wake 的内部处理旨在确保 2C（§3.1），以保证所有 edf 在处理过程中的有效性。具体而言，我们通过以下方式满足 (1) 一致性和 (2) 收敛性。

一致性。 §3.2 中的每个操作都是一个函数，它将输入 edf 映射到具有固定模式的输出 edf。由于 edf 的来源（`read` 操作）始终会生成具有固定模式的 edf，因此所有中间 edf 和最终 edf 都具有相同的模式。

收敛性。 收敛性有两种类型：（1）可变属性变得更加准确；（2）键集（例如分组属性）收敛。属性收敛：首先，Wake 生成的所有属性值都是收敛的。也就是说，设 $\tilde{x}_n$ 是某个键所关联的 edf 在处理完第 n 个元组后的属性值，而精确值——即处理完全部数据后得到的值——为 x。那么，以下两个性质成立：第一，对于 $n \le n'$，$\text{E}[|\tilde{x}_n - x|] \le \text{E}[|\tilde{x}_{n'} - x|]$；第二，$\tilde{x}_N = x$，其中 N 是元组总数。尽管后一性质（$\tilde{x}_N = x$）十分理想，但一些依赖统计模拟的现有在线聚合系统通常无法保证该性质 [50]。此外，对于均值类聚合（例如计数、求和、平均值、标准差、方差），我们能够生成无偏估计；也就是说，$\text{E}[\tilde{x}_n - x] = 0$。对于其他聚合（例如去重计数、最小值/最大值等极值顺序统计量），我们采用文献中众所周知的估计技术，生成相当准确的估计值 [36, 82]。键集收敛：在近似计算中，产生不存在键的一个主要原因是从输入数据中获取的样本不足 [19]。然而，在我们的框架下，键集会收敛到真实键集，因为我们的操作经过设计，能够在观测到全部输入数据时生成精确答案。

5 聚合推断

给定 edf 的内在状态，聚合推断会得出其外在状态。这面临两项挑战。首先，随着处理的输入数据增多，组大小（例如，来自某个州的学生人数）可能会以非线性方式增长。其次，组的数量也可能随时间增加（即州的数量）。第三，不同类型的聚合通常需要不同的估计机制。为应对这些挑战，我们将整体推断逻辑（§5.1）分解为两部分：基数估计器（§5.2）和聚合估计器（§5.3）。

5.1 问题分解

Wake 将聚合推断表述为一个无偏估计问题。利用截至当前进度 $0 \le t \le 1$ 的内在状态，聚合推断旨在求出最终进度 $T = 1$ 时逐单元的无偏估计量。假设数据框有 $m(t)$ 个分组，$X_i(t)$

表示进度 t 时第 i 个分组的分组基数（即已聚合到第 i 个分组中的元组数量）。虽然可能存在许多聚合属性，但聚合推断每次只关注一个属性，并将第 i 个分组的聚合值记为 $Y_i(t)$。由于 Wake 不知道未观测到的分区，$m(t)$、X_i 和 Y_i 是关于“时间” t 的随机过程。我们用小写字母表示截至进度 t 观测到的组基数和聚合值：$x_{i,:t}$ 和 $y_{i,:t}$。期望的无偏估计量 $\hat{y}_{i,:t}$ 满足：

$$\hat{y}_{i,:t} = \mathrm{E}\left[Y_i(T) \mid x_{i,:t}, y_{i,:t}\right] \tag{1}$$

许多已知的聚合估计器依赖当前计数 $x_{i,t}$ 以及最终计数 $x_{i,T}$；然而，当前尚不知道后者。Wake 转而根据截至目前的分组基数 $x_{i,:t}$ 计算最终分组基数的无偏估计量 $\hat{x}_{i,:t}$，详见 §5.2。

$$\hat{x}_{i,:t} = \mathrm{E}\left[X_i(T) \mid x_{i,:t}\right] \tag{2}$$

Wake 使用该估计值以及当前基数和聚合值，再通过 §5.3 中的聚合感知估计量 f 来估计 T 处的最终聚合值。

$$\hat{y}_{i,:t} = \mathrm{E}\left[Y_i(T) \mid x_{i,:t}, y_{i,:t}\right] = f(y_{i,:t}, x_{i,:t}, \hat{x}_{i,:t}) \tag{3}$$

因此，Wake 首先为所有 $i = 1, \ldots, m(t)$ 估计 $\hat{x}_{i,:t}$，然后估计 $\hat{y}_{i,:t}$。当聚合属性较多时，Wake 会复用 $\hat{x}_{i,:t}$，并应用相应的聚合估计器分别估计每个聚合。最后，它将所有聚合估计结果汇总到填充了外在状态的输出数据框中。

5.2 基数估计器

Wake 根据具有共同幂次的单项式来建模组基数，即 $\mathrm{E}[X_i(t)] \propto t^w$。其背后的推理如下。Wake 假设样本数量和组数量分别遵循两个隐藏的单项式 $\mathrm{E}[n(t)] \propto t^u$ 和 $\mathrm{E}[m(t)] \propto t^v$。因此，平均组基数为 $\frac{n(t)}{m(t)} \propto t^{u-v}$，$w = u - v$。其期望值与 $\sum_{i=1}^{m(t)} X_i(t)$ 成正比，因此 $\frac{1}{m(t)}\sum_{i=1}^{m(t)} X_i(t) \propto t^w$。这种建模涵盖了深度在线聚合中的许多场景。例如，如果输入数据框是一个表读取器，那么 Wake 会预期样本呈线性增长（$u = 1$）。如果输入位于两个表的交叉连接之后，那么 Wake 会预期其呈二次增长（$u = 2$）。筛选则会影响与其选择率对应的系数。另一方面，如果分组键与聚类键相同，那么随着读取更多分区，Wake 会发现组数量呈线性增长（$v = 1$）。低基数组键会产生一个常数（$v = 0$），而较高基数的键则会产生介于两者之间的结果（$0 < v \leq 1$）。

此外，该模型简化了估计逻辑。事实上，Wake 无需估计 $\mathrm{E}[n(t)]$，也无需估计 $\mathrm{E}[m(t)]$，只需估计 $\mathrm{E}[X_i(t)]$。Wake 分两步估计最终的分组基数。首先，它对由所有已观测到的平均分组基数组成的数据集拟合 $\frac{x_t}{t^w} = \frac{1}{m_t}\sum_{i=1}^{m_t} x_{i,t}$。具体而言，它在对数变换后的线性回归中拟合幂次以及系数 b：$\mathrm{E}[\log x_t] = \log b + w \log t$。Wake 实现了流式线性回归，每次观测的时间/空间复杂度为 $O(1)$。最后，Wake 为 $\mathrm{E}[X_i(t) \mid x_i]$ 中每个分组拟合系数：$\mathrm{E}[x_{i,t}] = x_{i,t}$ $= c_i t^w$，并利用 $T = 1$ 预测最终的分组基数：

$$\hat{x}_{i,:t} = \mathrm{E}[X_i(T) \mid x_{i,:t}] = (x_{i,t}/t^w)\, T^w = x_{i,t} \,/\, t^w \tag{4}$$

5.3 聚合估计器

Wake 根据聚合类型，从以下估计器集合中选择聚合估计器 f。此集合可与现有估计器一同扩展。

计数。使用估计的基数：f 计数$(y_{i,:t}, x_{i,:t}, \hat{x}_{i,:t}) = \hat{x}_{i,:t}$。

Sum. Scale the summation: $f_{\mathrm{sum}}(y_{i,:t}, x_{i,:t}, \hat{x}_{i,:t}) = \frac{y_{i,t}}{x_{i,t}} \hat{x}_{i,:t}$.

加权平均值。加权平均值（例如平均值、方差、标准差）是求和的特殊情况。由于我们选择了这些估计器，平均值估计器可简化为恒等函数。令 $y_{\prime i,:t}$ 为加权求和，$y_{\prime\prime i,:t}$ 为权重之和，$y_{i,t} = y_{\prime i,t}/y_{\prime\prime i,t}$ 为加权平均值：

$$f_{\mathrm{avg}}((y'_{i,:t}, y''_{i,:t}), x_{i,:t}, \hat{x}_{i,:t}) = \left(\frac{y'_{i,t}}{x_{i,t}}\hat{x}_{i,:t}\right) / \left(\frac{y''_{i,t}}{x_{i,t}}\hat{x}_{i,:t}\right) = y_{i,t} \tag{5}$$

不同值计数。Wake 采用有限总体矩估计量 [36]（在 §4.1 中记作 $\hat{D}_{MM1}$）。为简洁起见，本小节聚焦于第 i 个分组，并简化当前分组基数 $x = x_{i,t}$、最终估计的分组基数 $X = \hat{x}_{i,:t}$ 以及当前分组不同值计数 $y = y_{i,t}$ 的记法。Wake 计算 $f_{cd}(y_{i,:t}, x_{i,:t}, \hat{x}_{i,:t}) = Y$，其中 Y 满足公式（6）。

$$y_{i,t} = Y(1 - h(\hat{x}_{i,:t}/Y)) \tag{6}$$

$h(z)$ 的定义如下。为求解该方程，Wake 会执行牛顿-拉夫逊迭代，直到在给定容差范围内收敛，且最多进行有限步。每次迭代都需要计算伽马函数和双伽马函数的数值近似值。

$$h(z) = \frac{\Gamma(\hat{x}_{i,:t} - z + 1)\,\Gamma(\hat{x}_{i,:t} - x_{i,t} + 1)}{\Gamma(\hat{x}_{i,:t} - x_{i,t} - z + 1)\,\Gamma(\hat{x}_{i,:t} + 1)} \tag{7}$$

次序统计量。次序统计量包括最小值、最大值、中位数、分位数，以及第 k 小／大的值。目前，Wake 只输出最新值：$f\mathrm{order}(y_{i,:t}, x_{i,:t}, \hat{x}_{i,:t}) = y_{i,t}$ 无需任何计算成本，即可为较大的 $\hat{x}_{i,:t}$ 提供相当准确的估计值。

5.4 正确性

给定截至当前进度 t 的观测值（$y_{i,:t}$，$x_{i,:t}$），引理 1 和引理 2 共同表明，在某些条件下，Wake 的聚合推断是无偏的。

引理 1（无偏计数）。$\hat{x}_{i,:t} = \mathrm{E}[X_i(T) \mid x_{i,:t}] = \frac{x_i}{t^w}$ 是无偏的，条件是：*A*）w 是无偏的；*B*）所有操作都产生单项式，或将单项式增长输入转换为关于进度 t 呈单项式增长的输出。

引理 *2*（无偏聚合）。给定无偏分组基数估计 $\hat{x}_{i,:t} = E[X_i(T) \mid x_{i,:t}]$，*Wake* 的聚合估计器会产生无偏估计，但可能需要满足取决于聚合类型的附加条件：$E\left[Y_i(T) \mid x_{i,:t}, y_{i,:t}\right] = f(y_{i,:t}, x_{i,:t}, \hat{x}_{i,:t})$。

证明请见我们的扩展稿 [11]。

```
lineitem = read_csv('...')          # LI
# item count for each order
order_qty = lineitem.sum(qty, by=orderkey)     # OQ
# select only the large orders
lg_orders = order_qty.filter(sum_qty > 300)    # LO
# find the customers with biggest order sizes
lg_order_cust = lg_orders.join(orders) \       # OO
                  .join(customer)              # OC
# select top-100 customers
qty_per_cust = lg_order_cust.sum(sum_qty, by=name) # C
top_cust = qty_per_cust.sort(sum_qty, desc=True) \
                .limit(100)              # TC
```

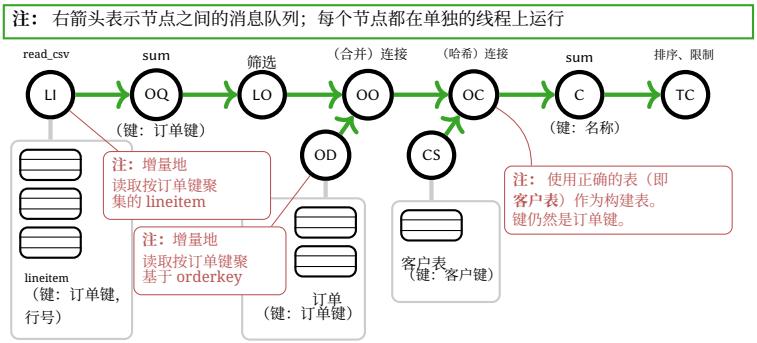


图 6：使用 edf 的数据操作示例（左），以及 Wake 用于并行处理这些操作的内部表示（右）

5.5 替代方案

本节列出了一些我们曾考虑过、但与 Wake 的整体设计不太契合的替代设计选择。

概率基数估计器。 可以对分布 $X_i(T)|x_{i,:t}$（而不是 Wake 中的期望值 $\mathrm{E}[X_i(T)|x_{i,:t}]$）建模，以表达置信度。然而，评估需要计算边际期望，对于许多聚合估计器而言，这可能代价高昂。此外，应该使用哪种分布仍是一个有待进一步研究的开放问题。

其他基数函数族。 不同的多项式族是很有吸引力的替代方案；不过，要高效地拟合其系数，就需要事先知道订单集合。混合使用指数函数和对数函数，在某些情况下可以提高准确性，但估计起来也会更加困难。相比之下，仿射函数简单，且有许多广为人知的估计算法，但它们仅适用于特定的增长模式。归根结底，单项式最简单，拟合成本也最低（在对数尺度上），同时还能提供广泛的增长曲线。

有限总体下的次序统计量。给定概率密度函数/累积分布函数，可以得到第 k 个顺序统计量 [29] 的密度函数。基于此，我们可以在 $\hat{x}_{i,:t}$ 处计算期望值，从而获得无偏估计值。对于离散变量，也可以进行类似分析。然而，通常这种方法需要付出高昂的计算代价来重建概率密度函数/累积分布函数，更不用说计算期望值了。例如，如果使用核密度估计（KDE） [69, 73] 和经验分布函数（eCDF）进行重建，则计算密度并保存所有样本需要 $O(n(t))$ 的时间和空间。这种代价难以扩展，且精度提升有限，却可能拖慢在线聚合的进度。

6 DEEPOLA 的置信区间

WAKE 的数学概念和实现可以扩展，以便在提供均值估计的同时，还提供置信区间。

扩展定义。 Wake 通过三个步骤，在整个处理过程中维护所有可变属性的“不确定性”：(1) 计算初始可变属性的不确定性；(2) 通过 edf 操作传播不确定性；(3) 根据最终的不确定性推导置信区间。具体而言，我们扩展 edf 的定义（§3.1）：

```
df_ci := (list((attr1, attr2, ..., attrM)), Σ)  (8)
```

其中，协方差矩阵 Σ 用于表示不确定性，$\Sigma_{i,\ j}$ 表示可变属性 attr_i 与 attr_j之间的协方差。

初始方差。 当可变属性首次出现时，Wake 使用现有的、针对特定聚合方式的方差估计器推断方差。具体而言，它通过计算普通最小二乘参数 [37] 的方差来测量基数幂 $\mathrm{Var}(w)$ 的方差；对求和和平均值应用中心极限定理 [71]；对不同值计数使用基于经验密度函数的泊松化 [16]；对次序统计量使用自助法 [31]；并通过拟合广义极值分布 [47] 来处理极值顺序统计量（最小值/最大值）。

方差传播。 Wake 使用一种标准统计学技术——“不确定性传播” [48]，在 edf 操作中传播 Σ。对于可微映射 $v = f(u)$ 和已知协方差矩阵Σ^U，Wake 使用一阶泰勒展开线性化 f，并计算 $\Sigma^V = J\Sigma^U J^{\mathrm{T}}$，其中 J 是雅可比矩阵（$J_{i,\ k} = \partial f_i/\partial U_k$）。公式（9）展开了这一表达式。

$$\Sigma^V_{i,j} = \sum_k \sum_l \Sigma^U_{k,l}\,(\partial f_i/\partial U_k)\,(\partial f_j/\partial U_l) \tag{9}$$

例如，协方差矩阵分别通过 `count` 和 `sum` 传播如下：

$$\mathrm{Var}(f_{\mathrm{count}}) = \mathrm{Var}(\hat{x}_{i,:t}) = (\hat{x}_{i,:t}\ln(1/t))^2\,\mathrm{Var}(w) \tag{10}$$

$$\mathrm{Var}(f_{\mathrm{sum}}) = 1/x^2_{i,t}\left(\mathrm{Var}(y_{i,t})\hat{x}^2_{i,:t} + \mathrm{Var}(\hat{x}_{i,:t})y^2_{i,t}\right) \tag{11}$$

请参阅我们的扩展稿 [11]，了解其他操作（例如，加权平均、不同值计数、顺序统计量、映射/投影）。

基于方差的置信区间。 最后，Wake 根据切比雪夫不等式 [78]，利用估计值 y 的方差 $\sigma^2 = \mathrm{Var}(y)$ 推导出一个置信区间。它输出 $[y-k\sigma, y+k\sigma]$，其中 $k = \sqrt{1/(1-\delta)}$ 对应置信水平 $(1-\delta)$。例如，$k \approx 4.5$ 对应 95% 置信区间。

局限性。 上述方法适用于可微操作，其中包括我们最感兴趣的大多数操作。与均值估计一样，有限方差不确定性也假设观测数据的分布能够代表未观测数据的分布，这是机器学习和统计推断的一个基本前提。计算置信区间需要计算公式（9），会带来时间和内存开销；不过，对于 TPC-H 查询而言，这些开销相对较小，因为相关的协方差数量很少。

替代方案。可以用高阶矩替代方差传播以提高准确性；然而，其时间复杂度（以及运行时开销）会增加。其他替代方案，例如

自助法或参数分布传播，要么会产生显著开销，要么仅适用于有限的操作。

7 实现

Wake 的 Rust 实现主要分为两部分：(1) 查询服务，供用户构建查询；(2) 执行引擎，以在线聚合的方式执行构建好的查询。

7.1 查询服务

用户将查询表示为一个执行图，其中节点代表不同的操作，边代表这些节点之间的数据流路径。节点的入边数量（表示操作的输入）与操作中参数的数量相同。例如，连接操作需要两条入边，分别代表待连接的 edf；而基本的筛选操作只需要一条入边。为支持 §3.2 中描述的 edf 操作，Wake 实现了不同类型的节点，例如读取器、归并连接、哈希连接、聚合器等，使用户能够表达多种多样的查询。边通过通道实现，用于在线程间传递消息流。用户可以逐步向查询图中添加节点和边，以表示嵌套的在线聚合操作。

图 6 中的圆圈表示节点，绿色箭头表示边。图中的 LI—read_csv—充当根节点，将获取的分区传递给其订阅者（即 OQ），后者按照图中的定义继续执行。当前的叶节点（即 TC）表示查询输出，可供下游应用程序（例如渐进式可视化）使用。

7.2 执行引擎

执行引擎接收查询服务，并在指定的数据集上评估查询，生成一系列 edf 输出。为每个 read_csv 节点指定输入后，执行引擎便开始执行查询。每个节点都在独立的线程中运行，并从其输入通道读取消息。收到的消息包含：（1）指向数据框的共享指针；（2）包含查询执行进度信息的元数据（§4）。节点处理这些消息，利用元数据更新其内在状态，并将其外在状态连同元数据作为消息写入输出通道。如果节点的输入通道上没有消息，该节点就会阻塞，等待从通道读取消息。特殊消息类型 EOF 用于表示给定通道上的输入结束。执行节点一旦在其输入通道上收到 EOF 消息，就会向其输出通道发送 EOF 消息，并终止执行。

7.3 讨论

查询优化。 作为第一步，Wake 引入了深度在线分析原语，但没有使用声明式语言。我们计划采用谓词下推、连接顺序优化等现有优化技术，同时也会探索独特的优化机会。例如，连接算法（如排序归并或哈希）不仅会影响性能，也会影响中间结果的交付方式。如果后续的分组操作使用与连接相同的键，即使哈希连接能更快地产生最终结果，我们也可能选择归并连接，以便更快地提供交互式结果。

OLA 特定优化。 Wake 的设计包含在线聚合特定优化，例如：（1）对查询操作采用流水线实现；（2）当两个表按共同的聚类键分区时（如 lineitem 和订单），采用排序合并连接；（3）重复进行哈希连接时，重用右侧（构建端）表的哈希表；（4）使用数据的共享指针以减少克隆开销，等等。这些优化帮助 Wake 像其他专为精确查询处理而设计的系统一样，尽快生成精确答案。

查询内并行。 Wake 受益于数据流水线和多线程，这些技术被数据系统广泛采用，以缩短任务完成时间 [57, 80]。我们的扩展报告 [11] 研究了数据流水线的优势。未来，我们将研究深度 OLA 特定分布式处理。

8 评估

本节将 Wake 与（传统的）精确数据系统以及 OLA 系统进行评估。我们的实验表明：

- Wake 的首次估算比精确系统快 4.93×，但生成精确结果的速度慢 1.3×。（§8.2）
- Wake 首次估计结果的中位误差为 2.70 %。与最佳精确数据系统相比，Wake 平均能以快 3.17× 的速度提供误差低于 1% 的结果（最高可达 48.80×）。（§8.3）
- 与最先进的 OLA 系统相比，Wake 生成误差低于 1% 的结果快 1.92× 倍。（§8.4）
- Wake 的置信区间符合选定的置信水平，但在处理接近完成时可能较为保守。（§8.5）
- Wake 以预期的时间复杂度执行深层查询。（§8.6）
- 通过优化分区大小，可以进一步提升 Wake 的性能。（§8.7）

8.1 实验设置

所有实验均在一台 Standard D16ads v5（Azure）机器上进行，该机器配备 16 个虚拟 CPU 和 64 GB 内存。我们使用 TPC-H 基准测试来比较和评估不同系统。

基线系统。 我们采用 2 个先进的 OLA 系统和 4 个精确系统。(1) ProgressiveDB [14]：一种适用于非嵌套无连接查询的基于中间件的 OLA 系统。我们使用作者的实现 [5]（目前仅支持单表），并使用 TPC-H 查询（Q1、Q6）进行基准测试（图9a）。(2) WanderJoin [50]：一种适用于非嵌套多重连接查询的基于随机游走的 OLA 系统。我们使用作者的实现 [7] 及其修改后的查询（Q3、Q7、Q10）（图9b）。(3) Presto [75]：Presto 是一种面向多种数据源设计的数据仓库。我们在 HDFS 上使用其 Hive 连接器。(4) Postgres：一种广泛使用的关系数据库管理系统。我们根据 TPC-H 架构，在相应属性上创建适当的外键和索引。(5) Polars [70]：Polars 是一种以 Rust 编写的数据框架库，经过单指令多数据、Arrow [3], 无锁并行哈希等优化。(6) Vertica [49]：Vertica 是一种基于列式存储的分析型数据库。我们在 Azure 上使用 Vertica Data Warehouse。(7) Actian Vector [93]：Actian Vector 是一种高性能向量化分析型数据库。

文件格式。 对于 Presto、Polars 和 Wake，我们使用 Parquet [4]。WanderJoin 和 ProgressiveDB 基于 Postgres 实现。

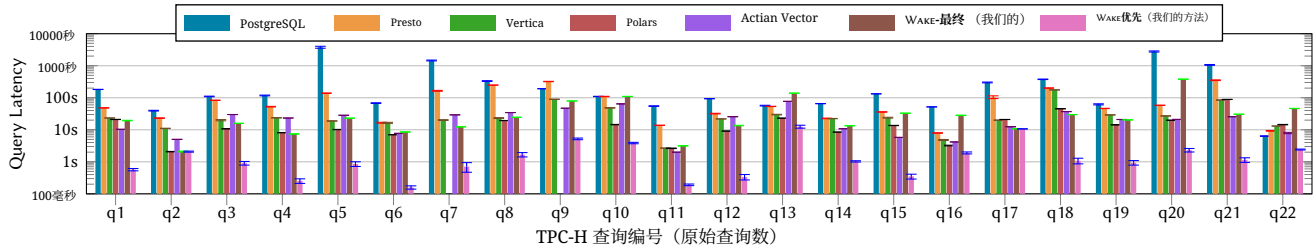


图 7：TPC-H 100 GB 数据集上不同基线系统的比较。结果为 10 次运行的平均值。

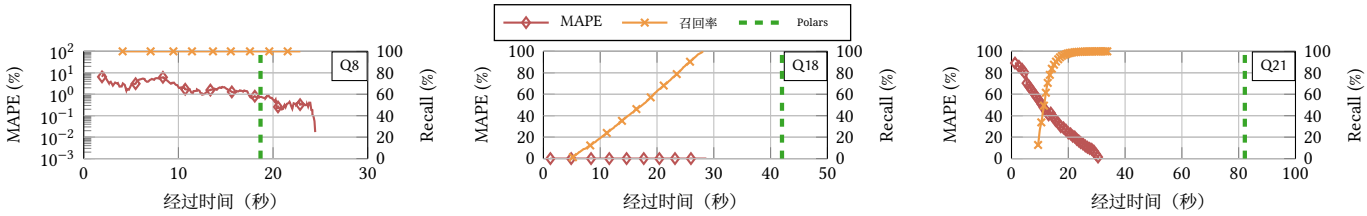


图8：Wake 的近似误差随时间变化，以平均绝对百分比误差（MAPE）和召回率衡量。竖线表示精确方法的完成时间。从左到右（Q8、Q18、Q21），Postgres 分别在（332 秒、376 秒、1061 秒）完成，而 Presto 分别在（247 秒、200 秒、352 秒）完成。

数据集。 我们使用规模100（100 GB）的 TPC-H 数据集 [6]。对于 Wake，该数据集被划分为 512 MB 的数据块，然后每个数据块都会转换为 Apache Parquet 格式。

查询。 我们使用 22 个 TPC-H 查询来评估不同系统并比较其性能。使用 TPC-H 查询有两个原因。第一，为了与已有工作保持一致，以便进行准确比较。第二，许多 TPC-H 查询可视为深层 查询，因为现有在线聚合方法无法处理其中的嵌套 `select` 语句——Q1 和 Q6 除外。与此相关的是，我们在与现有在线聚合方法进行比较时使用修改后的 Q3、Q7 和 Q10（§8.4）；不过，在 §8.2 和 §8.3 中研究我们的系统时，我们使用未经简化的原始查询。最后，我们还使用系统生成的更深层查询评估 Wake（§8.6）。

指标。计算以下指标：

- 最终结果延迟：处理完整数据集并生成正确最终结果所需的时间。• 首次估计延迟：首次估算所需的时间。

- 峰值内存使用量：对于内存中的 Polars 和 Wake，我们计算峰值内存使用量（即最大常驻集大小）。
- 平均绝对百分比误差：用于计算近似误差。

- 召回率：对于分组查询，召回率衡量正确生成的最终结果组所占的比例。

8.2 交互式查询与低开销

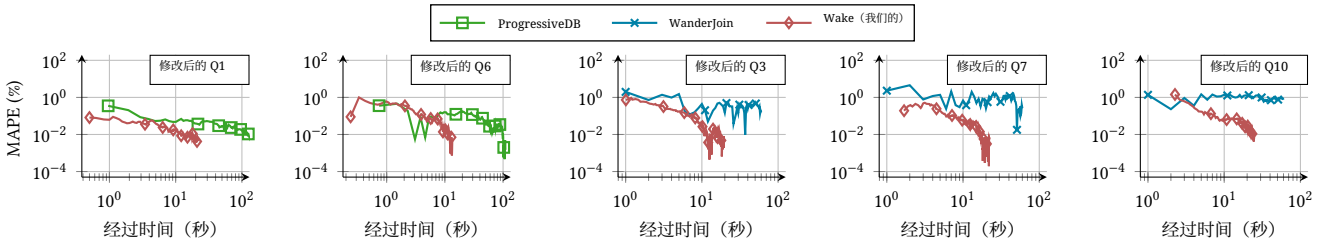
我们将 Wake 的延迟与不同精确查询处理系统进行比较，考察其生成首次估计结果和最终精确结果所需的时间。图 7 显示了 Wake 获得首次结果和最终结果所需的时间。中间结果的数量因查询所使用表的分区数量而异。与 Actian Vector 的精确答案相比，Wake 生成首次估算的速度快 4.93×；与 Polars 的精确答案相比快 11.8×；与 Vertica 的精确答案相比快 21.81×；与 Presto 的精确答案相比快 238.3×，与 Postgres 的精确答案相比快 238.3×（均为中位数）。就减速比而言，其定义为 Wake 的最终结果延迟与其他基线的最终结果延迟之比，相对于 Actian Vector 的中位数减速比为 5.3×，相对于 Polars 为 1.5×，相对于 Vertica 为 0.8×，相对于 Presto 为 0.3×，相对于 Postgres 为 0.1×；这意味着 Wake 生成精确答案的速度甚至快于 Vertica、Presto 和 Postgres。Wake 尽管是一个在线分析处理系统，但在 20 个查询中比 Postgres 更快地产生精确结果，在 17 个查询中比 Presto 更快，在 6 个查询中比 Polars 更快。

此外，Wake 的峰值内存利用率较低。对于查询 Q7 和 Q9，Polars 会在实验机器上耗尽内存，无法生成结果；而 Wake 则能成功生成精确的查询结果。平均而言，Wake 的峰值内存使用量比 Polars 少 4.3×（部分查询最多少 17.4×），因此能够处理更大的数据集。

具体来看，Q9、Q10 和 Q13 需要先为较小的右表构建哈希表，才能生成首个结果，因此提升幅度较小。Q2 和 Q17 需要计算子查询的聚合，因此收益微乎其微（但开销几乎为零）。就总查询延迟而言，计算开销在 Q10 和 Q13（由于对高基数 *c_custkey* 进行分组）以及 Q20（由于对 *partsupp* 重复筛选）中最为显著。对于首次估计延迟，Wake 的中位误差为 2.70%。Wake 提供的估计结果平均绝对百分比误差低于 1%，3.17× 比最佳基线的最终结果耗时更快。WAKE 的查询性能得益于我们的手动优化，例如谓词下推以及对连接方法的谨慎选择（图 6）；正如预期的那样，对于较差的查询计划，例如连接后筛选，我们观察到了更高的延迟。这也促使我们开展未来工作，详见第 7.3 节。

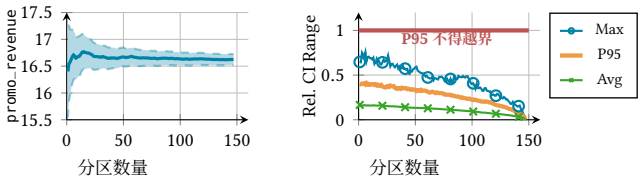
8.3 Wake 的近似误差分析

在本实验中，我们分析 Wake 的 OLA 输出在随时间处理更多数据时的近似误差，使用平均绝对百分比误差和



(a) 与 ProgressiveDB 对比（使用单表 Q1 和 Q6） **（b）与 WanderJoin 对比（使用与 [50] 中相同的修改后的 Q3、Q7 和 Q10）**

图 9：与最先进的 OLA 系统相比，近似误差随时间的变化



（a）置信区间收敛 **（b）置信区间正确性**

图 10：Wake 对 Q14 的 95% 置信区间（$k \approx 4.5$）。

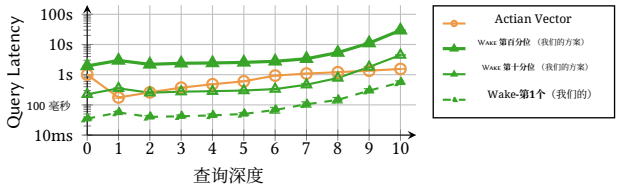


图 11：查询深度对其性能的影响

召回率误差作为衡量指标。图 8 展示了三个不同类别中若干代表性查询的时间误差曲线，如下所示。

第一类包括针对低基数非聚类分组键的查询。总体而言，随着 Wake 观察到更多数据，其平均绝对百分比误差曲线会随时间下降，而召回率则很早便达到 100%。TPC-H 中的许多查询都属于这一类：Q1、Q4–Q9、Q12、Q14、Q17、Q19 和 Q22。具体而言，Q8（图 8 左）涉及对多个连接表进行加权平均分组聚合。Wake 能在 1.9 秒时给出首次估算，误差为 6.5%。当 Polars 完成计算时（19 秒），Wake 的误差为 0.87%。

第二类查询涉及聚类分组键，因此其聚合值是精确的（平均绝对百分比误差为 0%），而随着 Wake 在不同分区中检索键，其召回率会不断提高。Q3、Q18（图 8 右）和 Q20 就是这类查询的例子：随着检索到或观察到的键越来越多，它们的召回率呈线性增长。

第三，第三类是前两类的组合；也就是说，可以用平均绝对百分比误差、召回率和/或精确率来理解它们的误差。例如，Q10、Q16 和 Q21（图 8 右侧）的召回率曲线迅速上升，而其平均绝对百分比误差曲线仅呈线性下降，因为它们的分组键各不相同，导致每组的样本数量较少，因而预测能力较弱。虽然未展示相关图表，但 Q11 的平均绝对百分比误差得分为满分，不过其召回率/精确率曲线在末尾迅速上升。Q2 和 Q15 使用最小值和最大值的自变量，因此召回率和精确率呈现开关式变化。最后，Q13 对高基数非聚类键（c_custkey）执行计数，随后按内层计数进行外层分组。由于内层计数在不同分区间会发生变化，外层各组内的增长可能并非单调，从而违反 Wake 的基数估计器假设，导致平均绝对百分比误差相对较大。我们将在未来解决这一局限。

8.4 比现有在线聚合更快、更准确

本节在 ProgressiveDB 和 WanderJoin 各自支持的所有 TPC-H 查询集上，将 Wake 与其他 OLA 系统进行比较：ProgressiveDB 支持 Q1 和 Q6，WanderJoin 支持 Q3、Q7 和 Q10。图9a展示了 ProgressiveDB 在 Q1 和 Q6 上的结果。

尽管 ProgressiveDB 和 Wake 的初始估计值相近，但 Wake 收敛到相对误差低于 1% 的速度比 ProgressiveDB 2.5× 快。图9b展示了与 WanderJoin 的比较。尽管首次估算的误差相当，但 Wake 收敛到相对误差低于 1% 的速度比 WanderJoin 1.51× 快。此外，Wake 很快就会收敛到精确答案，而 WanderJoin 的相对误差则维持在约 1%；这是意料之中的，因为它的基于随机游走的采样机制并非为此而设计。我们认为，Wake 所采用的这种收敛至精确答案的方法对最终用户更有利。

8.5 置信区间正确性

我们通过执行 Q14 来实证验证 Wake 的置信区间推导（§6），并使用打乱的输入分区来模拟输入顺序异常的情况。Q14 表示对两个带有筛选条件的表进行连接后计算加权平均值。我们的计算结果显示，置信区间逐渐收敛于均值估计，如图 10a 所示。此外，我们还研究了这些置信区间的质量（图 10b），其中包括相对置信区间范围：实际绝对误差相对于置信区间大小的比例 $\lceil y - y \rceil / (k\sigma)$。起初，我们的 P95相对置信区间范围约为 0.4，这符合预期，因为基于 Chebyshev 的置信区间分配了 $k \approx 4.5$。随后，相对置信区间范围随着分区数增加而减小；尽管这种方法过于保守，但它能够安全地界定真实答案。

8.6 合成深层查询上的性能

我们研究了 Wake 的内部处理如何随查询复杂度扩展。我们使用合成方式生成了一个包含 100M 行、11 个整数列的 100分区数据集。其中 10 列是分组列，具有 4_{10} 种唯一组合。合成查询在求和聚合和最大值聚合之间交替进行，查询深度 $d = 0, \ldots\ldots, 10$。例如，如果 $d = 2$，则查询为 `df.max(x, by=(ci,cii)).sum(max_x, by=ci).sum(sum_max_x)`。图 11 在对数纵轴上展示了 Wake 处理第 1 个、第 10 个和第 100 个（最终）结果的延迟。在所有深度下，Wake 处理每个分区所需的时间都相近，并以稳定的节奏输出结果。随着深度增加，Wake 的执行时间随主分组基数扩展（即，对于深度 d，为 $O(4^d)$ 个组），因为它

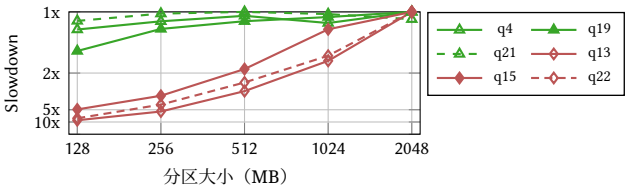


图12：分区大小的影响（红色查询——q13、q15、q22——合并开销高于绿色查询——q4、q19、q21）

需要将新的聚合合并到现有聚合中。通常，对于 n 行、每个分区 B 行且深度为 d 的情况，Wake 的时间复杂度为 $O(4^d n/B + n)$，而非在线聚合引擎（例如 Actian Vector）的复杂度预计为 $O(n)$。

8.7 数据分区大小至关重要

为了解分区大小对整体查询延迟的影响，我们使用不同的分区大小（128 MB、256 MB、.....、2048 MB）在规模100（100 GB）数据上评估 Wake。随着单个分区大小增加，生成首个结果所需的时间会增加，而最终结果延迟往往会降低，因为合并多个分区的开销较低。图12展示了多个 TPC-H 查询的延迟，以该查询在不同分区大小下观察到的最佳性能的倍数表示。

对于合并操作开销较小的查询，分区大小按预期不会影响最终结果延迟。例如，Q1、Q4、Q6、Q7、Q12、Q19（分组聚合中的组较少）、Q18 和 Q21（对 o_订单键进行流式处理）。

对于合并开销较高的查询（例如，包含大量组的分组），分区大小会显著影响性能。较大的分区大小会减少分区数量，从而降低计算开销，进而缩短最终结果延迟。例如，Q13、Q15、Q16（分组聚合中的组较多）和 Q22（对 c_客户键进行剪枝）。

对于不太适合在线聚合的查询（例如 Q17），较大的分区大小有助于缩短首个结果和最终结果的延迟。因此，合适的分区大小取决于查询本身以及目标——无论是将首个结果延迟还是最终结果延迟降至最低。然而，本研究并未探讨此类优化器。

9 相关工作

近似查询处理（AQP）使用户能够以远低于执行精确查询的成本，交互式地分析大型数据集。尽管近似查询处理经过多年研究，在支持深度嵌套查询方面仍不够成功 [20]。

在线聚合。 Hellerstein 等人 [39] 首次提出了在线聚合的概念。此后，各种研究工作 [30, 35, 44, 50, 53]在此基础上扩展了所支持查询的范围，重点关注连接查询。RippleJoin [35] 逐步连接多个表， [53]改进了针对低基数或组数较多查询的在线连接算法。 [30] 为基于排序的连接算法提供了在线聚合支持。 [43, 44] 提出了一种可扩展的基于磁盘的方法，同时为基于排序的连接算法提供了更好的统计界。WanderJoin [50] 利用索引高效处理包含多个连接的查询。现有文献对嵌套聚合和子查询的支持仍然有限。G-OLA [90] 将在线聚合推广到嵌套谓词，依据运行时估计值将已处理元组划分为确定集合和不确定集合。由于其效率取决于估计值的质量，错误的估计可能导致重新计算。一些相关工作 [84] 也研究了分布式环境中的在线聚合。 [25, 61, 77] 进一步将在线聚合应用于 MapReduce 作业，但仅限于基本的查询模板。

增量视图维护。 物化视图 [9, 15, 21]能够提升查询性能，但需要额外的存储空间和维护开销。基表变更需要通过增量查询来更新物化视图。已有研究提出了多种技术 [10, 33, 34, 51, 60, 74, 87] ，用于对物化视图执行增量更新。视图维护和索引 [23, 24, 63]与在线聚合有本质区别：与在线聚合不同，它们的目标不是估计未来结果，而在线聚合需要管理/传播多个操作中的不确定性。

近似查询处理。 近似查询处理包括基于概要数据的技术 [27]，例如使用样本 [8, 12, 13, 18, 38, 64、小波 [17], 、直方图 [42, 72], 、摘要 [26], 等。近期基于机器学习的方法将近似查询处理表述为一个数据学习问题——通过密度估计器 [41, 54],、回归模型 [54],和生成模型 [62, 79, 86]实现。最近，查询感知生成模型被用于降低低基数查询 [76, 92]的近似误差。这些方法面临的挑战包括模型维护和重新训练成本高、支持的查询类型有限，以及难以提供正确性界限。

基数估计。 基数估计是查询优化中的一个基本问题。传统方法包括使用直方图、摘要和样本等摘要统计 [27, 68]。近年来，涉及深度自回归模型 [88, 89],和基于集成的方法 [52]的基于学习的方法逐渐受到关注。 [83] 提供了全面的比较。基数估计的任何改进也都能提高 Wake 的准确性。

10 结论

在本文中，我们通过引入一种新型数据模型，向深度在线聚合（DeepOLA）迈出了一步。该数据模型在面向集合的操作（例如 map/filter/join/agg）下是闭合的，从而支持对先前在线聚合输出执行嵌套操作。我们通过 Wake 展示了该模型的可行性。Wake 是一个使用 Rust 实现的深度在线聚合系统。具体而言，我们在 TPC-H（100 GB）上评估了 Wake，并将其与最先进的在线聚合引擎和传统数据系统进行了比较。实验表明，与传统系统计算精确答案相比，Wake 能够更快地提供首次估计结果 4.93×（中位数），同时保持 2.70% 的相对误差中位数。此外，Wake 在生成精确答案时产生的额外开销很小（1.3× 中位数减速比）。事实上，对于某些查询，Wake 中不同操作的流水线实现通常能够提供比精确查询处理引擎更短的总延迟。未来，我们计划扩展 Wake，使其支持带有自动查询优化的类 SQL 声明式接口。我们还计划将 Wake 扩展到分布式部署，使深度在线聚合具备更强的可扩展性。

致谢

本研究部分由 Microsoft Azure 资助。

参考文献

[1] 访问日期：2022-10-01。MySQL 8.0 参考手册 - FIND_IN_SET。https://dev.mysql. com/doc/refman/8.0/en/string-functions.html#function_find-in-set。[2] 访问日期：2022-10-01。MySQL 8.0 参考手册 - GROUP_CONCAT。https: //dev.mysql.com/doc/refman/8.0/en/aggregate-functions.html#function_group- concat。[3] 访问日期：2022-10-15。Apache Arrow。https://arrow.apache.org/。[4] 访问日期：2022-10-15。Apache Parquet。https://parquet.apache.org/。[5] 访问日期：2022-10-15。ProgressiveDB。https://github.com/DataManagementLab/ progressiveDB。[6] 访问日期：2022-10-15。TPC-H：决策支持基准。https://www.tpc. org/tpch/。[7] 访问日期：2022-10-15。XDB：近似数据库（XDB）。https://github.com/ InitialDLab/XDB。[8] Sameer Agarwal、Barzan Mozafari、Aurojit Panda、Henry Milner、Samuel Madden 和 Ion Stoica。2013。BlinkDB：在超大规模数据上实现有界误差和有界响应时间的查询。载于《第8届 ACM 欧洲计算机系统会议论文集》，29–42。[9] Divyakant Agrawal、Amr El Abbadi、Ambuj Singh 和 Tolga Yurek。1997。数据仓库中的高效视图维护。ACM SIGMOD Record 26, 2（1997），417–427。[10] Yanif Ahmad 和 Christoph Koch。2009。DBToaster：一种用于内存数据库中高性能增量处理的 SQL编译器。VLDB基金会会刊 2, 2（2009），1566–1569。[11] 匿名作者。2022。迈向深度在线聚合（扩展稿）。arXiv 预印本 arXiv（2022）。[12] Brian Babcock、Surajit Chaudhuri 和 Gautam Das。2003。用于近似查询处理的动态样本选择。载于《2003年 ACM SIGMOD数据管理国际会议论文集》，539–550。[13] Johes Bater、Yongjoo Park、Xi He、Xiao Wang 和 Jennie Rogers。2020。SAQE：面向数据联邦的实用隐私保护近似查询处理。VLDB基金会会刊 13, 12（2020），2691–2705。[14] Lukas Berg、Tobias Ziegler、Carsten Binnig 和 Uwe Röhm。2019。ProgressiveDB：作为中间件的渐进式数据分析。VLDB基金会会刊 12, 12（2019），1814–1817。[15] Jose A Blakeley、Per-Ake Larson 和 Frank Wm Tompa。1986。高效更新物化视图。ACM SIGMOD Record 15, 2（1986），61–71。[16] Leonid V. Bogachev、Alexander V. Gnedin 和 Yuri V. Yakubovich。2008。占用箱数量的方差。应用数学进展 40, 4（2008），401–432。https://doi.org/ . / j.aam. . .[17] 002 Kaushik Chakrabarti、Minos Garofalakis、Rajeev Rastogi 和 Kyuseok Shim。2001。使用小波进行近似查询处理。VLDB期刊 10, 2（2001），199–223。[18] Surajit Chaudhuri、Gautam Das 和 Vivek Narasayya。2007。用于近似查询处理的优化分层抽样。ACM 数据库系统汇刊（TODS）32, 2（2007），9–es。[19] Surajit Chaudhuri、Bolin Ding 和 Srikanth Kandula。2017。近似查询处理：没有灵丹妙药。载于《2017年 ACM 数据管理国际会议论文集》，511–519。[20] Surajit Chaudhuri、Bolin Ding 和 Srikanth Kandula。2017。近似查询处理：没有灵丹妙药。载于《2017年 ACM 数据管理国际会议论文集》（美国伊利诺伊州芝加哥）（SIGMOD '17）。计算机协会，美国纽约州纽约市，511–519。https://doi.org/ 10 1145 3035918 3056097 . / . Surajit Chaudhuri、Ravi [21] Krishnamurthy、Spyros Potamianos 和 Kyuseok Shim。1995。利用物化视图优化查询。载于《第11届国际数据工程会议论文集》。IEEE，190–200。[22] Shimin Chen、Phillip B Gibbons 和 Suman Nath。2010。Pr-join：一种具有统计保证、可提高早期结果率的非阻塞连接。载于《2010年 ACM SIGMOD数据管理国际会议论文集》，147–158。[23] Supawit Chockchowwat、Wenjie Liu 和 Yongjoo Park。2022。自动寻找最优索引结构。arXiv 预印本 arXiv:2208.03823（2022）。[24] Supawit Chockchowwat、Chaitanya Sood 和 Yongjoo Park。2022。Airphant：面向云的文档索引。载于《2022年 IEEE第38届数据工程国际会议》（ICDE）。IEEE，1368–1381。[25] Tyson Condie、Neil Conway、Peter Alvaro、Joseph M Hellerstein、John Gerth、Justin Talbot、Khaled Elmeleegy 和 Russell Sears。2010。MapReduce 中的在线聚合和连续查询支持。载于《2010年 ACM SIGMOD数据管理国际会议论文集》，1115–1118。[26] Graham Cormode。2011。用于近似查询处理的草图技术。数据库基础与趋势。NOW出版社（2011），15。[27] Graham Cormode、Minos Garofalakis、Peter J Haas、Chris Jermaine 等。2011。海量数据的摘要统计：样本、直方图、小波和摘要。数据库基础与趋势® 4, 1–3（2011），1–294。[28] Andrew Crotty、Alex Galakatos、Emanuel Zgraggen、Carsten Binnig 和 Tim Kraska。2016。交互式数据探索加速器（IDEAs）的必要性。载于《人在回路中的数据分析研讨会论文集》，1–6。

[29] Herbert A David 和 Haikady N Nagaraja。2004。《次序统计量》。John Wiley & Sons。[30] Jens-Peter Dittrich、Bernhard Seeger、David Scot Taylor 和 Peter Widmayer。2002。《渐进式归并连接：一种通用的非阻塞式基于排序的连接算法》。载于《第28届超大型数据库国际会议论文集》。299–310。[31] Bradley Efron。1979。《自助法：再谈刀切法》。《统计年刊》 7, 1 1979 1 26 10 1214 1176344552 ( )，–。[32] 2007 https://doi.org/ 。/aos/ Philippe Flajolet、Éric Fusy、Olivier Gandouet 和 Frédéric Meunier。。《HyperLogLog：近乎最优基数估计算法的分析》。载于《离散数学与理论计算机科学》。离散数学与理论计算机科学，137–156。[33] Jonathan Goldstein 和 Per-Åke Larson。2001。《使用物化视图优化查询：一种实用且可扩展的解决方案》。《ACM SIGMOD Record》30，2（2001），331–342。[34] Ashish Gupta、Inderpal Singh Mumick 和 Venkatramanan Siva Subrahmanian。1993。《增量维护视图》。《ACM SIGMOD Record》22，2（1993），157–166。[35] Peter J Haas 和 Joseph M Hellerstein。1999。《用于在线聚合的涟漪连接》。载于《1999年ACM SIGMOD数据管理国际会议论文集》。287–298。[36] Peter J. Haas、Jeffrey F. Naughton、S. Seshadri 和 Lynne Stokes。1995。《基于采样的属性不同值数量估计》。载于 VLDB。Morgan Kaufmann，311–322。[37] Fumio Hayashi。2000。《计量经济学》。普林斯顿大学出版社。27–32页。[38] Wen He、Yongjoo Park、Idris Hanafi、Jacob Yatvitskiy 和 Barzan Mozafari。2018。《VerdictDB 演示：平台无关的近似查询处理系统》。载于《2018年国际数据管理会议论文集》。1665–1668。[39] Joseph M Hellerstein、Peter J Haas 和 Helen J Wang。1997。《在线聚合》。载于《1997年ACM SIGMOD数据管理国际会议论文集》。171–182。[40] Ian Hellström。访问日期：2022-10-01。《面向开发者的 Oracle SQL 与 PL/SQL 优化》。https://oracle.readthedocs.io/en/latest/sql/joins/hash-join.html。[41] Benjamin Hilprecht、Andreas Schmidt、Moritz Kulessa、Alejandro Molina、Kristian Kersting 和 Carsten Binnig。2019。《DeepDB：从数据学习，而非从查询学习！》arXiv 预印本 arXiv:1909.00607（2019）。[42] Yannis E Ioannidis 和 Viswanath Poosala。1999。《集合值查询答案的基于直方图的近似》。载于 VLDB，第99卷。174–185。[43] Chris Jermaine、Subramanian Arumugam、Abhijit Pol 和 Alin Dobra。2008。《使用 dbo 引擎实现可扩展的近似查询处理》。《ACM 数据库系统汇刊》（TODS）33，4（2008），1–54。[44] Christopher Jermaine、Alin Dobra、Subramanian Arumugam、Shantanu Joshi 和 Abhijit Pol。2006。《排序-归并-收缩连接》。《ACM 数据库系统汇刊》（TODS）31，4（2006），1382–1416。[45] Srikanth Kandula、Anil Shanbhag、Aleksandar Vitorovic、Matthaios Olma、Robert Grandl、Surajit Chaudhuri 和 Bolin Ding。2016。《QuickR：在大数据集群中惰性近似复杂的临时查询》。载于《2016年国际数据管理会议论文集》。631–646。[46] Albert Kim、Eric Blais、Aditya Parameswaran、Piotr Indyk、Sam Madden 和 Ronitt Rubinfeld。2015。《带有排序保证的可视化快速采样》。载于《VLDB基金会超大型数据库国际会议论文集》，第8卷。美国国立卫生研究院公共访问，521。[47] Samuel Kotz 和 Saralees Nadarajah。2000。《极值分布》。帝国理工学院出版社出版，世界科学出版公司发行。https://doi.org/10.1142/p191 arXiv:https://www.worldscientific.com/doi/pdf/ 10 1142 191 。/p Harry H. Ku。。[48] 2010 《误差传播公式的使用说明》。载于《美国国家标准局研究杂志，C辑：工程与仪器》，第2卷。[49] Andrew Lamb、Matt Fuller、Ramakrishna Varadarajan、Nga Tran、Ben Vandier、Lyric Doshi 和 Chuck Bear。2012。《Vertica 分析型数据库：C-store 七年之后》。arXiv 预印本 arXiv:1208.4173（2012）。[50] Feifei Li、Bin Wu、Ke Yi 和 Zhuoyue Zhao。2016。《漫游连接：通过随机游走实现在线聚合》。载于《2016年国际数据管理会议论文集》。615–629。[51] Zhaoheng Li、Xinyu Pi 和 Yongjoo Park。2023。《S/C：使用有界内存加速数据物化》。载于《2023年IEEE第39届国际数据工程会议》（ICDE）。IEEE。[52] Jie Liu、Wenqian Dong、Qingqing Zhou 和 Dong Li。2021。《Fauce：用于基数估计的快速、准确且具备不确定性的深度集成》。《VLDB基金会会刊》14，11（2021），1950–1963。[53] Gang Luo、Curt J Ellmann、Peter J Haas 和 Jeffrey F Naughton。2002。《一种可扩展的哈希涟漪连接算法》。载于《2002年 ACM SIGMOD数据管理国际会议论文集》。252–262。[54] Qingzhi Ma 和 Peter Triantafillou。2019。《Dbest：借助机器学习模型重新审视近似查询处理引擎》。载于《2019年

国际数据管理会议论文集》。1553–1570。[55] Wes McKinney 等。2011。《pandas：用于数据分析与统计学的基础 Python 库》。《高性能与科学计算中的 Python 》 14，9 （ 2011 ），1–9。[56] Frank McSherry 、 Derek Gordon Murray、Rebecca Isaacs 和 Michael Isard。2013。《差分数据流》。载于 CIDR。[57] John Meehan、Nesime Tatbul、Stan Zdonik、Cansu Aslantas、Ugur Cetintemel、Jiang Du、Tim Kraska、Samuel Madden、David Maier、Andrew Pavlo 等。2015。《S-Store：流处理与事务处理相遇》。《VLDB基金会会刊》8，13（2015）。[58] Mohamed F Mokbel、Ming Lu 和 Walid G Aref。2004。《哈希归并连接：一种可快速生成早期连接结果的非阻塞连接算法》。载于《第20届国际数据工程会议论文集》。IEEE，251–262。[59] Derek G Murray、Frank McSherry、Rebecca Isaacs、Michael Isard、Paul Barham 和 Martín Abadi。2013。《Naiad：及时数据流系统》。载于《第二十四届ACM操作系统原理研讨会论文集》。439–455。[60] Milos Nikolic、Mohammed Elseidy 和 Christoph Koch。2014。《LINVIEW：复杂分析查询的增量视图维护》。载于《2014年ACM SIGMOD数据管理国际会议论文集》。253–264。[61] Niketan Pansare、Vinayak Borkar、Chris Jermaine 和 Tyson Condie。2011。《大型 MapReduce 作业的在线聚合》。《VLDB基金会会刊》4，11（2011），1135–1145。[62] Noseong Park、Mahmoud Mohammadi、Kshitij Gorde、Sushil Jajodia、Hongkyu Park 和 Youngmin Kim。2018。《基于生成对抗网络的数据合成》。arXiv 预印本 arXiv:1806.03384（2018）。[63] Yongjoo Park、Michael Cafarella 和 Barzan Mozafari。2015。《邻居敏感哈希》。《VLDB基金会会刊》9，3（2015），144–155。[64] Yongjoo Park、Michael Cafarella 和 Barzan Mozafari。2016。《面向超大型数据库的可视化感知采样》。载于《2016年IEEE第32届国际数据工程会议》（ICDE）。IEEE，755–766。[65] Yongjoo Park、Barzan Mozafari、Joseph Sorenson 和 Junhao Wang。2018。《VerdictDB：实现近似查询处理的普适化》。载于《2018年国际数据管理会议论文集》。1461–1476。[66] Yongjoo Park、Jingyi Qing、Xiaoyang Shen 和 Barzan Mozafari。2019。《BlinkML：具备概率保证的高效最大似然估计》。载于《2019年国际数据管理会议论文集》。1135–1152。[67] Yongjoo Park、Ahmad Shahab Tajik、Michael Cafarella 和 Barzan Mozafari。2017。《数据库学习：迈向一个日益智能的数据库》。载于《2017年ACM国际数据管理会议论文集》。587–602。[68] Yongjoo Park、Shucheng Zhong 和 Barzan Mozafari。2020。《QuickSel：使用混合模型进行快速选择率学习》。载于《2020年ACM SIGMOD国际数据管理会议论文集》。1017–1033。[69] Emanuel Parzen。1962。《概率密度函数及众数的估计》。《数学统计年刊》33，3（1962），1065–1076。[70] pola rs。访问日期：2022-10-14。《Polars：适用于 Rust 和 Python 的闪电般快速数据框库》。https://www.pola.rs/。[71] Georg Pólya。1920。《关于概率论中心极限定理与矩问题》。《数学杂志》8（1920），171–181。[72] Viswanath Poosala、Venkatesh Ganti 和 Yannis E. Ioannidis。1999。《使用直方图进行近似查询应答》。《IEEE数据工程通报》22，4（1999），5–14。[73] Murray Rosenblatt。1956。《关于密度函数的一些非参数估计的评述》。《数学统计年刊》（1956），832–837。[74] Kenneth Salem、Kevin Beyer、Bruce Lindsay 和 Roberta Cochrane。2000。《如何滚动连接：异步增量视图维护》。《ACM SIGMOD Record》29，2（2000），129–140。[75] Raghav Sethi、Martin Traverso、Dain Sundstrom、David Phillips、Wenlei Xie、Yutian Sun、Nezih Yegitbasi、Haozhun Jin、Eric Hwang、Nileema Shingte 和 Christopher Berner。2019。《Presto：万物皆可运行 SQL》。载于《2019年 IEEE第35届国际数据工程会议》（ICDE）。1802–1813。https://doi.org/10.1109/ICDE.2019.00196[76] Nikhil Sheoran、Subrata Mitra、Vibhor Porwal、Siddharth Ghetia、Jatin Varshney、Tung Mai、Anup Rao 和 Vikas Maddukuri。2022。《基于条件生成模型的谓词感知查询近似》。《AAAI人工智能会议论文集》36，8（2022年6月），8259–8266。https://doi.org/10.1609/aaai.v36i8.20800[77] Yingjie Shi、Xiaofeng Meng、Fusheng Wang 和 Yantao Gan。2012。《借助 COLA 尽早停止：云端聚合查询的在线处理》。载于《第21届ACM国际信息与知识管理会议论文集》（毛伊岛，夏威夷，美国）（CIKM '12）。计算机协会，纽约，美国纽约州纽约市，1223 1232 10 1145 2396761 2398423 –。[78] https://doi.org/ 。 1867 / 。P. Tchébychef。。《平均值》（俄文译文，N. de Khanikof）。《纯粹与应用数学杂志》（1867），177–184。http://eudml.org/doc/234989[79] Saravanan Thirumuruganathan、Shohedul Hasan、Nick Koudas 和 Gautam Das。2020。《使用深度生成模型进行数据探索的近似查询处理》。载于《2020年IEEE第36届国际数据工程会议》（ICDE）。IEEE，1309–1320。[80] Ankit Toshniwal、Siddarth Taneja、Amit Shukla、Karthik Ramasamy、Jignesh M Patel、Sanjeev Kulkarni、Jason Jackson、Krishna Gade、傅茂松、Jake Donham 等。2014。Storm@ twitter。载于*2014*年*ACM SIGMOD*数据管理国际会议。147–156。[81] Tolga Urhan 和 Michael J Franklin。2000。XJoin：反应式调度的流水线连接算子。技术委员会公告（2000），27。[82] A. W. van der Vaart。1998。渐近统计。剑桥大学出版社。https://doi.org/10.1017/CBO9780511802256[83] Xiaoying Wang、Changbo Qu、Weiyuan Wu、Jiannan Wang 和 Qingqing Zhou。2020。我们是否已准备好进行学习型基数估计？*arXiv*预印本 *arXiv:2012.06743* （2020）。[84] Sai Wu、Shouxu Jiang、Beng Chin Ooi 和 Kian-Lee Tan。2009。分布式在线聚合。*VLDB*基金会会刊2，1（2009），443–454。[85] Sai Wu、Beng Chin Ooi 和 Kian-Lee Tan。2010。针对多查询在线聚合的连续采样。载于*2010*年*ACM SIGMOD*数据管理国际会议。651–662。[86] Lei Xu、Maria Skoularidou、Alfredo Cuesta-Infante 和 Kalyan Veeramachaneni。2019。使用条件生成对抗网络对表格数据建模。载于神经信息处理系统进展。[87] Jian Yang、Kamalakar Karlapalem 和 Qing Li。1997。数据仓库环境中的物化视图设计算法。载于 *VLDB*，第97卷。136–145。[88] Zongheng Yang、Amog Kamsetty、Sifei Luan、Eric Liang、Yan Duan、Xi Chen 和 Ion Stoica。2020。NeuroCard：适用于所有表的基数估计器。*arXiv*预印本 *arXiv:2006.08109* （2020）。[89] Zongheng Yang、Eric Liang、Amog Kamsetty、Chenggang Wu、Yan Duan、Xi Chen、Pieter Abbeel、Joseph M Hellerstein、Sanjay Krishnan 和 Ion Stoica。2019。深度无监督基数估计。*arXiv*预印本 *arXiv:1905.04278* （2019）。[90] Kai Zeng、Sameer Agarwal、Ankur Dave、Michael Armbrust 和 Ion Stoica。2015。G-OLA：面向大数据交互式分析的通用在线聚合。载于*2015*年*ACM SIGMOD*数据管理国际会议。913–918。[91] Kai Zeng、Sameer Agarwal 和 Ion Stoica。2016。IOLAP：通过管理不确定性实现高效的增量式联机分析处理。载于*2016*年数据管理国际会议。1347–1361。[92] Meifan Zhang 和 Hongzhi Wang。2021。基于条件生成模型的分组查询近似查询处理。*arXiv*预印本 *arXiv:2101.02914* （2021）。[93] Marcin Zukowski、Mark Van de Wiel 和 Peter Boncz。2012。Vectorwise：一种向量化分析型数据库管理系统。载于*2012*年*IEEE*第*28*届数据工程国际会议。IEEE，1349–1350。

A 推理正确性证明

以下重述了聚合推断（§5）的引理及正确性证明。给定截至当前进度 t 的观测值（$y_{i,:t}$，$x_{i,:t}$），引理 1 和引理 2 共同表明，在某些条件下，Wake 的聚合推断是无偏的。

引理 1（无偏计数）。$\hat{x}_{i,:t} = \mathrm{E}[X_i(T)|x_{i,:t}] = \frac{x_i}{\sum_{t} w^t}$是无偏的，条件是 A） w 是无偏的，且 B）所有操作都会生成单项式，或将单项式增长输入转换为关于进度 t的单项式增长输出。

证明。为说明 A），若满足以下充分条件，则 w 是无偏的：$\hat{X}_t = \frac{1}{m}\sum_{i=1}^{m_{(t)}} X_i(t) = c_i t^w e^\epsilon$ 其中 e^ϵ 是乘法误差，其期望为 $\mathrm{E}[\epsilon] = 0$，且与观测值相互独立。注意，$\log X_t = \log c_i + w \log t \mp \varepsilon$。根据普通最小二乘估计量的无偏性，估计的 w与 w的期望相同。

为证明 B)，我们说明 §3 中的所有操作都能满足该性质，但可能需要根据操作类型附加一些条件。首先，数据源通过实现生成单项式行，因为它会根据已读取的行数除以最终行数来计算进度 t。映射显然会保留基数。如果每个元组的选择率均匀，且选择率独立于进度，那么连接和筛选条件会保留单项式关系。最后，如果分组键在整个进度范围内均匀分布，或者是聚簇键，那么聚合会保留单项式关系。

如果 B) 成立，则对于所有 edf（包括聚合的输入），$\mathrm{E}[X_i(t)] = c_i t^w$。由于 Wake 已经在 t 处观察到了 $x_{i,t}$，因此 $\mathrm{E}[X_i(t)|x_{i,:t}] = \mathrm{E}[X_i(t)|x_{i,t}] = x_{i,t}$，这意味着 $c_i = x_{i,t}/t^w$，并由此得到 Wake 在 $t = T = 1$ 处求值时的分组基数估计器。□

引理 2（无偏聚合）。给定无偏分组基数估计值 $\hat{x}_{i,:t} = E[X_i(T)|x_{i,:t}]$，*Wake* 的聚合估计量会产生无偏估计值，但根据聚合类型，可能需要满足额外条件：$E\left[Y_i(T) \mid x_{i,:t},\ y_{i,:t}\right] = f(y_{i,:t},\ x_{i,:t},\ \hat{x}_{i,:t})$。

证明。对 f_{count} 而言，这成立，因为给定了 $\hat{x}_{i,:t}$。

假设求和中的项 $\{U_{i,j}\}_{j=1}^{X_i(t)}$ 是平稳的，且其 $\mathrm{E}[U_{i,j}] = \mathrm{E}[U_i]$ 为常数，则 f_{sum} 是无偏的。令 $Y_i(t) = \sum_{j=1}^{X_i(t)} U_{i,j}$，我们可以利用平稳性将 $\mathrm{E}[Y_i(T) \mid x_{i,:t},\ y_{i,:t}] = \mathrm{E}[X_i(T) \mid x_{i,:t}]\mathrm{E}[U_i \mid x_{i,:t},\ y_{i,:t}]$ 分解。通过展开条件期望，得到 $E[U_i \mid x_{i,:t},\ y_{i,:t}] = \frac{y_{i,t}}{x_{i,t}}$，这使最终结果等于 f_{sum}。

假设元素与权重相互独立，根据期望的线性性，加权平均估计值 f_{avg} 是无偏的。

假设等频假设成立（即样本均匀分布于 Y 个不同值中），f_{cd} 是如 [36]所述的无偏估计值。此外，同一研究还推导了偏斜频率下的估计值，留待未来改进和实验。

第 q 个样本分位数围绕第 q 个总体分位数渐近正态分布，其方差为 $q(1-q)/(X_i(t)p_q^2)$，其中 p_q 是 [82]第 21.2 节中所研究分位数值的密度。假设样本量足够大 $x_{i,t} = \omega(1/p_q^2)$，鉴于数据规模较大，这在在线聚合中很常见，因此样本分位数是无偏的，且方差较小。□

B 方差传播列表

计数与基数估计。 参见 f 计数和公式（4）：

$$\mathrm{Var}(f_{\mathrm{count}}) = \mathrm{Var}(\hat{x}_{i,:t}) = (\hat{x}_{i,:t}\ln(1/t))^2\,\mathrm{Var}(w) \tag{12}$$

求和。 回顾 f 求和 $\frac{y_{i,t}}{x_{i,t}}\hat{x}_{i,:t}$。如果对可变属性求和，则应用 $\mathrm{Var}(A+B) = \mathrm{Var}(A) + \mathrm{Var}(B) + 2\,\mathrm{Cov}(A, B)$。否则，如果对常量属性求和：

$$\mathrm{Var}(f_{\mathrm{sum}}) = \frac{1}{x_{i,t}^2}\left[\mathrm{Var}(y_{i,t})\hat{x}_{i,:t}^2 + \mathrm{Var}(\hat{x}_{i,:t})y_{i,t}^2\right] \tag{13}$$

加权平均。 参见公式（5））。这与在考虑协方差时，分别对加权求和与权重求和应用公式（13）一致：

$$\mathrm{Var}(f_{\mathrm{avg}}) = \left[\frac{y'_{i,t}}{y''_{i,t}}\right]^2\left[\frac{\mathrm{Var}(y'_{i,t}/x_{i,t})}{(y'_{i,t}/x_{i,t})^2} + \frac{\mathrm{Var}(y''_{i,t}/x_{i,t})}{(y''_{i,t}/x_{i,t})^2}\right] \tag{14}$$

不同值计数。 参见公式（6）。不同值计数的方差传播规则可以用 $\frac{\partial Y}{\partial y_{i,t}}$ 和 $\frac{\partial Y}{\partial x_{i,:t}}$ 表示，后者同样基于公式（9）。这两个项都可以通过对公式（6）进行隐式求导得到。令 $h(z) = \frac{dh}{dz}$ 用双伽马函数表示（为保证数值稳定性，以对数函数形式计算）。以下是两个隐式求导方程。

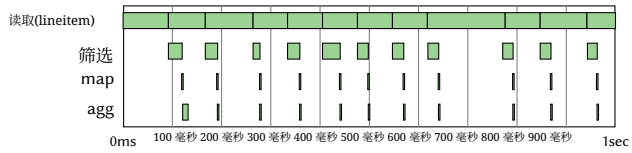


图 13：示例查询 Q6 的流水线执行

$$1 = \frac{\partial Y}{\partial y_{i,t}}(1 - h(\hat{x}_{i,:t}/Y)) + \frac{\partial Y}{\partial y_{i,t}}\frac{\hat{x}_{i,:t}}{Y}h'(\hat{x}_{i,:t}/Y) \tag{15}$$

$$0 = \frac{\partial Y}{\partial \hat{x}_{i,:t}}(1 - h(\hat{x}_{i,:t}/Y)) + \left(\frac{\partial Y}{\partial \hat{x}_{i,:t}}\frac{\hat{x}_{i,:t}}{Y} - 1\right)h'(\hat{x}_{i,:t}/Y) \tag{16}$$

Solving them gives us the desired derivatives.

$$\frac{\partial Y}{\partial y_{i,t}} = \frac{1}{(1 - h(\hat{x}_{i,:t}/Y)) + (\hat{x}_{i,:t}/Y)h'(\hat{x}_{i,:t}/Y)} \tag{17}$$

$$\frac{\partial Y}{\partial \hat{x}_{i,:t}} = \frac{h'(\hat{x}_{i,:t}/Y)}{(1 - h(\hat{x}_{i,:t}/Y)) + (\hat{x}_{i,:t}/Yh')(\hat{x}_{i,:t}/Y)} \tag{18}$$

最后，应用公式（9）。

$$\mathrm{Var}(f_{\mathrm{cd}}) = \mathrm{Var}(Y) = \frac{\mathrm{Var}(y_{i,t}) + \mathrm{Var}(\hat{x}_{i,:t})[h'(\hat{x}_{i,:t}/Y)]^2}{[(1 - h(\hat{x}_{i,:t}/Y)) + (\hat{x}_{i,:t}/Yh')(\hat{x}_{i,:t}/Y)]^2} \tag{19}$$

次序统计量。 输出估计方差 $\mathrm{Var}(y_{i,t})$。

映射与投影。 依靠自动微分来计算映射中所有可变属性的偏导数。对于当前属性值处导数未定义的映射函数（例如，$|x|$ 在 $x = 0$ 处，或阶跃函数），此方法不适用，此时 Wake 可将置信区间标记为“不稳定”。投影是映射的一种特殊情况，其投影方差等于源方差。

C 流水线查询处理

尽管合并计算会带来额外开销，Wake 的总查询延迟仍可与其他精确处理系统相当或更低，一个关键因素是对不同操作进行流水线处理。由于每项操作都可以根据其输入 edf 独立运行，流水线执行能够提高可用计算资源的利用率，从而缩短总耗时。图 13 展示了在一次查询执行过程中，不同节点如何处理数据分区。虽然查询执行的时间更长，为清晰起见，我们将观察时间线限制为 1 秒。操作 agg 是查询 Q6 的最终节点。因此，agg 操作时间线上的每个区块都代表一个已获得的输出。