ByteHTAP:字节跳动的HTAP系统,数据新鲜度高,数据一致性强

Jianjun Chen, Yonghua Ding, Ye Liu, Fangshi Li, Li Zhang, Mingyi Zhang, Kui Wei, Lixun Cao*, Dan 邹*, 杨柳*, 张磊*, 芮石*, 丁卫, 吴凯, 罗尚宇, 孙杰, 梁玉铭*

字节跳动美国基础设施系统实验室,*字节跳动公司

抽象的

近年来,在字节跳动,我们看到越来越多的业务场景需要对新导入的数据进行复杂的分析,同时需要事务支持和数据强一致性。在本文中,我们描述了构建 ByteHTAP 的过程,这是一个具有高数据新鲜度和强数据一致性的 HTAP 系统。它采用分离引擎和共享存储架构。其模块化的系统设计充分利用了字节跳动现有的OLTP系统和开源的OLAP系统。这种选择为我们节省了大量资源和开发时间,并允许将来轻松扩展,例如用其他替代方案替换查询处理引擎。

ByteHTAP 可以提供不到一秒延迟的高数据新鲜度,这为我们的客户带来了许多新的商机。客户还可以根据业务需求配置不同的数据新鲜度阈值。ByteHTAP 还通过其 OLTP和 OLAP 系统的全局时间戳提供了强大的数据一致性,这极大地减轻了应用程序开发人员自己处理复杂数据一致性问题的负担。此外,我们对 ByteHTAP 引入了一些重要的性能优化,例如将计算推送到存储层以及使用删除位图来有效地处理删除。最后,我们将分享我们在生产环境中开发和运行 ByteHTAP 的经验教训和最佳实践。

PVLDB参考格式： Jianjun Chen, Yonghua Ding, Ye Liu, Fangshi Li, Li Zhang, Mingyi Zhang, Kui Wei, Lixun Cao, Dan Zou, Yang Liu, Lei Zhang, Rui Shi, Wei Ding, Kai Wu, Shangyu Luo, Jason孙,梁玉明． ByteHTAP:字节跳动的HTAP系统,具有高数据新鲜度和强数据一致性。
PVLDB, 15(12): 3411-3424,
2022.doi :10.14778/3554821.3554832

1 简介

近年来,在字节跳动,我们开始看到越来越多的业务场景需要对新导入的数据进行复杂的分析,以及交易支持和数据强一致性。例如,字节跳动的用户增长部门需要对业务属性关系、广告费用等不断变化的数据进行复杂的 SQL 查询,最新的数据变化预计在亚秒级延迟内可见。

但是,我们无法找到满足客户需求的现成解决方案。传统的 OLAP 系统通常会定期批量加载大量数据,但会遇到数据过时的问题。相比之下,传统的OLTP系统支持DML ,可以高效执行点查查询,但缺乏海量并行处理能力。因此,它们无法有效地处理对大量数据的复杂查询。为了满足我们的业务需求,混合事务/分析处理（HTAP)系统是最合适的选择。

具体来说,我们构建了一个支持新鲜数据更改和强数据一致性的大规模实时分析 HTAP 系统,其设计目标如下:

·大规模。字节跳动的几个热门应用程序,如 TikTok、抖音和今日头条,每天都有数亿活跃用户。因此,我们希望构建一个可以扩展到 PB 级数据的分布式实时分析系统。 ·实时。我们希望 ByteHTAP 中的 OLTP 和 OLAP 查询在独立的 OLTP/OLAP 系统中运行时具有可比的性能。这对于将现有客户的工作负载迁移到 ByteHTAP 非常重要,这些工作负载目前在独立系统中运行。

·高度新鲜的数据变化。为了探索新的业务机会,一些客户希望在一秒延迟内提供最新的数据变化以供查询。这对我们的系统设计提出了很强的要求。目前,我们的客户在数据新鲜度方面有几分钟甚至几小时的延迟。

·强数据一致性。目前很多客户都是将OLTP数据库中的数据导入到数据仓库中进行数据分析。因此,他们很难跨 OLTP 和 OLAP 引擎获得一致的全局快照,应用程序开发人员不得不花费额外的精力来处理数据一致性问题。因此,客户希望 ByteHTAP 提供对强数据一致性的原生支持。

近年来,HTAP 系统在学术界和工业界得到了广泛讨论[37、51、62] 。已经开发了几个专用的 HTAP 系统,例如 SAP Hana [32] 、TiDB [34]和MemSQL [13、29]。此外,许多传统的 OLTP 和 OLAP供应商也声称他们的系统支持[38,39,44,52] 。

一般来说,HTAP 系统可以有完全不同的架构。
最近的一项调查[50]根据其架构选择将 HTAP 系统分为以下几类:

·单引擎。此类 HTAP 系统通常具有统一的 HTAP 引擎,例如 SAP Hana [32] 和 MemSQL [29]。
这些系统可以进一步分为两类

VLDB 捐赠论文集,卷。 15,第 12 期 ISSN 2150-8097。
doi:10.14778/3554821.3554832

陈建军博士为通讯作者,jianjun.chen@bytedance.com。

关于它们支持的数据格式:单一数据格式或混合数据格式。 ·独立引擎。此类别中的 HTAP 系统使用单独的查询引擎来处理 OLTP 和 OLAP 工作负载,例如 WildFire [23]和 TiDB [34]。这些系统根据其存储结构可以进一步分为两类:分离存储或共享存储。尽管前者在生产中被广泛采用,但它有一个缺点,即对于 OLAP 查询,数据新鲜度通常较低。

在仔细研究[50] 中列出的不同设计选择后,我们决定采用分离引擎和共享存储设计,原因如下: ·分离引擎。开发一个可以同时处理 OLTP 和 OLAP 工作负载的查询引擎并非易事。很少有前任

isting 开源系统可以很好地处理这种混合的工作负载。相比之下,有多个独立的 OLTP 和 OLAP系统可供我们使用,无论是专有的还是开源的。因此,我们选择独立引擎的路线,让每个引擎都发挥最大的作用,同时避免OLTP 和 OLAP 工作负载之间的干扰。我们使用专有的 OLTP 系统 ByteNDB 和开源 OLAP引擎 Apache Flink [5] 构建 ByteHTAP。 ·共享存储。字节跳动的基础设施系统通常采用计算与存储分离的云原生架构。 ByteNDB 的架构类似于 Amazon Au rora [61]。我们扩展其复制存储以支持行存储之外的列存储。以这种方式,数据更改以最小的延迟在存储层中传播。我们的列式存储还包含内存中的增量存储,以允许OLAP 引擎查询最新的数据更改。

请注意,由于我们为 OLTP 和 OLAP 查询提供了统一的 SQL API,因此架构选择对用户来说大多是透明的。
ByteHTAP 精心设计的代理将查询自动路由到相应的引擎。除了上述优点外,这样的模块化设计也便于日后的扩展。例如,我们目前正在构建一个新的分布式MPP SQL 引擎,未来可以在 ByteHTAP 中轻松替代 Flink 。我们只需要为新引擎实现一个新的连接器来与我们的存储层对话。

在本文中,我们想展示我们用一个小型工程团队构建一个高数据新鲜度和强数据一致性的 HTAP 系统的过程。我们在 2020 年初以不到 10 名开发人员开始设计和开发,并于 2021 年中期发布了1.0版本。目前,我们已经有多个内部客户在生产中使用 ByteHTAP,预计客户数量将在 2020 年显着增长2022. 我们从这段旅程中学到了很多东西,希望我们的故事能对有类似需求的读者有所帮助。

总之,我们的主要贡献如下:

·我们演示了如何使用独立引擎和共享存储架构构建具有竞争力的HTAP 系统。我们的模块化系统设计充分利用了字节跳动现有的 OLTP 系统和开源的 OLAP 系统。这种设计节省了大量资源和开发时间,并允许将来轻松扩展,例如用其他替代方案替换查询处理引擎。

· ByteHTAP 可以提供不到一秒延迟的高数据新鲜度,这为我们的客户带来了许多新的商机。客户还可以根据业务需求配置不同的数据新鲜度阈值。

· ByteHTAP 通过跨 OLTP 和 OLAP 系统的全局时间戳提供强大的数据一致性,这减轻了应用程序开发人员在其系统中处理复杂数据一致性问题的负担。

· ByteHTAP 的复制存储层利用统一的复制框架来无缝构建行和列存储。

我们还描述了 ByteHTAP 内部的一些重要性能优化,例如将计算推送到存储层以及使用删除位图来有效地处理删除。 ·我们讨论在生产环境中开发和运行 ByteHTAP 系统的经验教训和最佳实践。

本文的其余部分组织如下:第 2 节概述了相关工作。第 3 节描述了总体 ar

ByteHTAP 的架构及其关键组件的实现。第 4 节重点介绍 ByteHTAP 如何实现高数据新鲜度。第 5 节描述了 ByteHTAP 中的强数据一致性。第 6 节展示了我们在 ByteHTAP 内部所做的一些性能优化。在第 7 节中,我们提供了 ByteHTAP 的一些实证测量。第 8 节列出了我们在生产中学到的一些主要经验教训。最后,第 9 节总结了我们的工作。

2 相关工作

在过去的十年中,HTAP 在学术界和工业界得到了广泛的讨论[28, 35, 37, 47, 48, 51, 53, 57]。市场上的许多数据库产品都声称自己是 HTAP 数据库或支持 HTAP功能[6、7、9、10、20] 。在本节中,我们首先介绍几个众所周知的专用 HTAP 系统。接下来,我们将概述现有 OLTP 和 OLAP数据库产品中的 HTAP 扩展。最后,我们讨论一些关于 HTAP 的近期工作。

SAP Hana [32, 41, 46, 60]是一个同时支持 OLTP 和 OLAP 的内存中多模型数据库系统。它为OLTP/OLAP组件提供统一的接口,包括语言接口、计划树、算子框架和用于存储的表接口。它包含支持行和列格式的分层内存存储。最近的一项工作[59]扩展了 Hana 的内存列存储以支持内存和磁盘存储。
Hana 采用单引擎架构,不同于ByteHTAP 的独立引擎统一存储架构。

MemSQL [29]也采用单引擎架构。它是一个无共享和内存优化的分布式 HTAP 系统。
它提供了一个内存中的行存储和一个磁盘上的列存储。
它使用 LLVM [40]和无锁内存数据结构来快速执行查询。

TiDB [34]建立在 TiKV [34] 之上,TiKV [34] 是一种用于事务查询的分布式基于行的存储。 TiDB 使用 TiFlash [34] 增强了 TiKV， TiFlash [34] 是一种用于分析查询的基于列的存储。通过在不同的服务器上部署 TiKV和 TiFlash,TiDB 表明它可以处理隔离资源上的事务和分析查询。 TiDB将日志从 TiKV 异步复制到 TiFlash,将行格式数据转换为列格式数据,并提供日志之间的数据一致性。 TiDB 与 Byte HTAP 在架构上有一些相似之处,但在设计上存在一些差异。比如 TiDB 5.0 的

OLAP 查询引擎采用大规模并行处理 (MPP)架构,计算发生在 TiFlash 的存储节点上。
相比之下,ByteHTAP 将计算与存储分离,从而在计算层和存储层都提供了极大的弹性。

WildFire [22、23、43]利用Spark [63]作为计算引擎并使用共享存储模型。它通过简单的 DML 语句扩展 OLTP 支持以实现快速数据摄取。当完成写入 SSD 上的分片日志时,将提交 DML 语句。后台整理进程定期将日志与HDFS 中的数据合并。 SparkSQL [19]中表达的 OLAP 查询可以查询日志和 HDFS 中的数据。后来开发了Wiser [21]来为 WildFire 提供高可用性。在与 WildFire 共享一些架构相似性的同时,ByteHTAP 可以支持与 MySQL [8] 兼容的一般 OLTP工作负载。

传统的关系型数据库通常将数据按行存储,通常比 OLAP 更适合 OLTP。因此,许多关系数据库提出了特定的解决方案（例如列优先数据格式)来加速 OLAP 工作负载。 Oracle引入了 Database In-memory Option [38, 49] (DBIM) 作为双格式架构来支持 HTAP 应用程序。在 DBIM 中,行格式数据持久化在永久存储中,列格式数据纯粹存储在内存中。确保这两种格式之间的事务一致性。 Microsoft SQL Server 2016 [39]增强了列存储索引以加强 HTAP 工作负载的处理。用于 Linux、UNIX 和 Windows 的 IBM Db2 使用 BLU Acceleration [52]作为列存储来加速商业智能查询。相比之下,我们的解决方案在统一存储层上使用单独的本机 OLTP 和 OLAP 引擎,因此它对OLTP 和 OLAP 组件具有更多的隔离性和灵活性。

传统的数据仓库供应商[27, 44]也在增强他们的 OLTP 功能,以更好地支持 HTAP 工作负载。例如,Greenplum 数据库[44]使用资源组模型来分离 OLAP 和 OLTP 工作负载,并使用不同数量的资源来处理它们。实验表明,它可以在保持 OLAP 性能的同时提高 OLTP 的性能。

NoSQL 数据库也探索了 HTAP。例如,Couch base Server [24]引入了 Couchbase Analytics Service [35]来补充其查询服务以支持复杂的分析查询。
与专注于文档数据的 Couchbase Server 不同， ByteHTAP 是为关系数据开发的。

谷歌提出 F1 Lightning [62]作为其现有事务数据库系统的HTAP 增强。F1 Lightning 由三个组件组成:OLTP 数据库、Lightning 组件和联合查询引擎（F1 Query [56]）。 Lightning 组件从 OLTP 数据库中读取数据,并将它们从行优先、写入优化的格式转换为列优先、读取优化的格式。 F1 Lightning 采用 Change Data Capture [62]机制来提高 OLAP 引擎的数据新鲜度。

与 F1 Lightning 不同,ByteHTAP 对 OLTP 和 OLAP 引擎使用统一存储。ByteHTAP 的 OLAP 引擎可以直接从统一存储中读取新提交的数据。因此,ByteHTAP为 OLTP 和 OLAP 引擎提供了强数据一致性和高数据新鲜度。

最近,IBM 还为 IBM Db2 for z/OS (Db2z) 增强了 HTAP 能力,并提出了一个名为 IBM Db2 Analytics Accelerator (IDAA) [26]的新混合系统。IDAA 将 Db2 Warehouse 作为列存储添加到 Db2z 并在那里处理 OLAP 工作负载。 DB2 仓库维护 Db2z 表数据的副本,并根据请求将它们与 DB2z 同步。此外,IDAA 提出了一种新的数据复制方法,称为 Integrated Synchronization [26] ,以支持数据的增量更新。此方法提高了Db2 Warehouse 的数据新鲜度。

HTAP 系统在学术界也被广泛讨论。 BatchDB [45]是为 HTAP 工作负载设计的数据库系统。它对 OLTP 和 OLAP 工作负载采用主从副本,其中OLTP 工作负载在主副本上运行,而 OLAP工作负载在辅助副本上执行。主副本上的更新将定期传播到辅助副本,以确保副本之间的数据一致。与 BatchDB 相比,ByteHTAP 的 OLTP 引擎新提交的更改可立即用于其 OLAP 引擎。因此,ByteHTAP 具有更好的数据新鲜度。

拉扎等。 al [54]提出了一种可以动态调整其 HTAP 架构以满足数据新鲜度和运行时性能的不同要求的系统。该系统包含三个组件： OLTP 引擎、OLAP 引擎以及资源和数据交换引擎。通过调整 OLTP和 OLAP 引擎之间的资源分配， [54]可以探索一系列 HTAP 架构设计,从完全共置的 OLTP-OLAP 引擎到完全隔离的 OLTP-OLAP 引擎。与[54]使用的基于单服务器的内存存储不同， ByteHTAP 具有可扩展、分布式和持久存储。此外,ByteHTAP 的 OLTP 和OLAP 引擎具有良好的资源隔离性,因为它们独立运行在自己的资源（如 CPU 和内存)上。

VEGITO [58]是一个分布式内存 HTAP 系统。它利用数据备份来增强数据新鲜度并提高其运行时性能。具体而言,VEGITO 为其备份添加了三种新技术:八卦式日志应用方案、基于块的多版本列式数据布局和基于树的索引。与 VEGITO不同,ByteHTAP 在存储层利用日志序列号 (LSN)来提供强数据一致性。此外,ByteHTAP使用单独的 OLTP 和 OLAP 引擎为 HTAP 工作负载提供良好的隔离。

3系统架构及重要事项

成分

ByteHTAP 使用 ByteNDB 作为其 OLTP 系统。它还扩展了teNDB 的复制框架,以桥接其 OLTP 行存储和OLAP 列存储。在本节中,我们首先简要概述 ByteNDB 及其复制框架。然后,我们讨论了ByteHTAP 的架构和一些重要的组件。

3.1 ByteNDB 概述图1 显示了

ByteNDB 的整体架构。它支持Amazon Aurora [61]所做的“日志即数据库”原则,并且除了读/写主实例之外还可以有多个只读副本。

ByteNDB采用了MySQL [8]中的缓冲池、事务和锁管理组件,并进行了一些修改以实现主从同步。计算层的主要组件主要由代理和SQL引擎组成。代理知道系统配置和主副本与只读副本之间的查询路由。

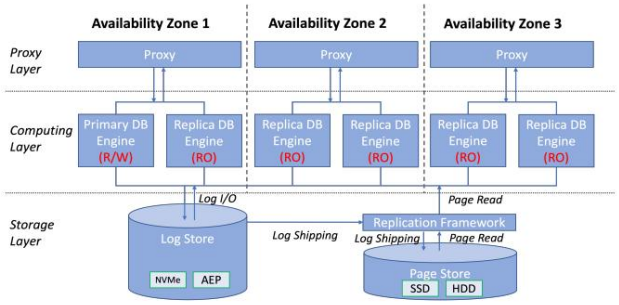


图 1:ByteNDB 架构图。

ByteNDB的复制存储层由一个持久化redo日志的Log Store和一个存储数据页版本并不断应用redo日志构建最新版本数据页的Page Store组成。这两个存储都建立在分布式存储之上,以提供高可用性和持久性。 Log Store利用append-only分布式BLOB存储提供大容量快速重做日志持久化。 Page Store提供log apply构造数据页的能力,支持页粒度的随机读取。

在 ByteNDB 的复制存储层的核心,一个 Replication Framework 适用于redo logs 的复制和分发。每个重做日志都将根据其持久化顺序分配一个唯一的 LSN。同一个事务中的重做日志被整批复制,保证原子性。来自 Log Store 的每个日志都被复制到 Page Store 中的三个存储服务器,并且使用仲裁协议[33]来保证这些副本之间的数据一致性。为了进一步保证数据的一致性,存储服务器接收到的redo log会按照LSN排序,并按顺序持久化。每个日志还包含一个与前一个日志的 LSN 的反向链接,可以立即检测到日志序列中的任何潜在漏洞（即丢失的日志）。八卦协议[31]在存储服务器之间实现,因此可以从其他存储服务器检索一个服务器上丢失的日志。结果,日志在准备好在存储服务器中应用/重放之前被排序为没有任何漏洞的序列。

为了提高读取效率,Replication Framework 会跟踪每个副本的 LSN,以便可以将查询发送到LSN 大于查询的 LSN 的单个副本,而不是从仲裁协议所需的两个副本中读取。

3.2 系统概述如图 2 所示,

ByteHTAP 采用共享存储之上的独立引擎架构。它支持一个统一的 API,查询可以由代理自动定向到 OLTP 或 OLAP 引擎。 ByteHTAP 使用 ByteNDB 作为 OLTP 引擎,使用Flink 作为 OLAP 引擎。 ByteHTAP 使用智能代理层自动将不同的查询定向到 OLTP 和 OLAP引擎。简而言之,DML、DDL、简单查询和适用于OLTP 的查询（例如在 OLTP 表上使用谓词的索引)被发送到 OLTP 引擎,而复杂的查询,例如具有多个连接和聚合的查询,被发送到 OLAP 引擎.这种方法避免了 OLTP 和 OLAP 工作负载之间的干扰,并允许查询由适当的引擎处理。

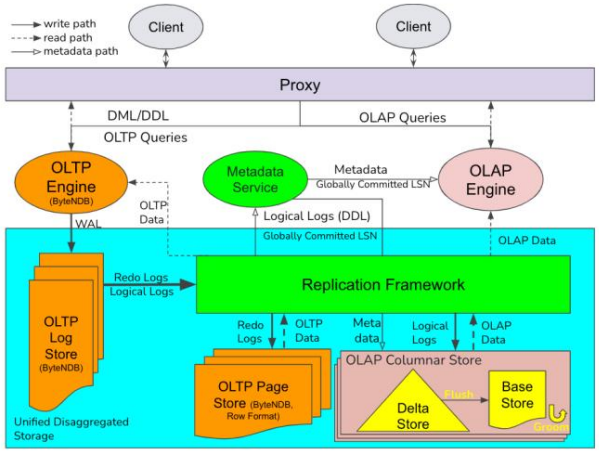


图 2:ByteHTAP 架构图。

ByteHTAP 遵循大多数数据库用户熟悉的ANSI SQL [4]标准。一个特定的要求是每个 ByteHTAP 表必须包含一个主键,基于该主键对列存储数据文件进行排序以提供高效的数据访问。用户可以发布 DML 来更新 ByteHTAP 中的主键值,数据约束由 OLTP 引擎自动执行。此外,用户还可以为 OLAP 列存储指定分区键,这使得ByteHTAP 的 OLAP 查询引擎可以并行处理查询。目前ByteHTAP的列存只支持hash分区,未来我们会支持更多的分区方案。

ByteHTAP 扩展了 ByteNDB 的 Replication Framework,它为每个分区提供了一个可靠的日志分发到多个存储节点,以构建一个列式数据存储,它可以驻留在与其对应的行存储不同的存储节点上。逻辑日志（即

用于已提交 DML 事务的 MySQL 二进制日志)会根据用户表中定义的分区方案连续分派到列式存储节点。 ByteHTAP 的 Columnar Store 由内存中的 Delta Store 和持久的 Base Store 组成。 OLAP查询使用指定的 LSN作为其快照版本扫描 Base Store 和 Delta Store。元数据服务为 OLAP 查询优化器和存储节点提供集中的元数据访问。它在启动时加载元数据并将它们缓存在内存中。

ByteHTAP 通过为其查询提供一致的数据快照来保证强数据一致性。每个 DML 和 DDL 语句在系统中生成一个唯一的 LSN。同一事务中的语句被包装在一起并以原子方式通过系统。元数据服务将全局提交的 LSN 中继到OLAP 计算引擎。此 LSN 之前的任何 LSN 都保证被 OLAP 列式存储接收（并保留）。OLAP 查询引擎中的元数据服务器客户端可以定期从存储层获取最新的全局提交的 LSN 并缓存它。

将根据全局提交的 LSN 为查询分配读取的 LSN。在大多数情况下,ByteHTAP 允许以不到一秒的延迟查询数据更改。

请注意,ByteHTAP 目前不允许使用混合 DML 和 OLAP 只读查询的事务,因为它不支持跨 OLTP 和 OLAP 引擎的分布式事务。如果客户需要此功能,我们将来可能会放宽此限制。

3.3 元数据服务

为了提供一个统一的服务,在 OLTP 和 OLAP 引擎之间对齐目录信息和分区方案,并减少其他 ByteHTAP 模块中的状态信息,实现了一个集中式元数据服务(MDS),如图 2 所示。它还提供了全局提交的来自复制框架的 LSN 作为 OLAP 查询的读取 LSN。

在 ByteHTAP 中,我们需要考虑三种不同的元数据:目录信息、 OLTP 和 OLAP 引擎的分区信息以及 OLTP 和OLAP 数据的统计信息。 ByteHTAP 的 OLTP 系统在其自己的数据存储中管理其元数据。 MDS 存储其他元数据信息,例如OLAP 存储分区架构和分区列式存储的统计信息。

MDS 服务器是建立在 Zookeeper [36]集群之上的专用进程,以实现高可用性。 MDS 客户端集成到 OLAP 计算引擎和存储服务器中,以便与 MDS 服务器通信以获取多版本元数据信息。材料数据表

与处理大多数 MySQL DDL 的 DDL 解析器集成。
承载元数据变化的 DDL 逻辑日志在OLTP 端生成,由 Replication Framework 中继,并在到达时由 MDS 解析。由此产生的目录/分区方案更改被反序列化、持久化并可用于服务。有关 DDL 处理的更多细节在第 5.3 节中讨论。

3.4 OLAP 引擎为了支持

复杂的分析查询,我们利用 Apache Flink [5]作为我们的 OLAP 计算引擎。我们评估了几个广泛使用的开源计算系统,包括 Flink [5]、Presto [11]和Apache Spark [63]。虽然我们在这些引擎中看到了类似的 TPC-H [17]和 TPC-DS [16]性能,但我们决定使用 Flink,因为它在我们公司内被广泛使用,并且它可以在未来为流式查询提供良好的支持。通过将 Flink 与我们自己的 Columnar Store 相结合,我们在较短的开发时间内为 ByteHTAP 构建了一个 OLAP 系统,我们的 OLAP 系统可以直接继承 Flink 的功能和优势。

为了使 Flink 能够高效地从我们的存储中读取数据,我们构建了一个自定义连接器,支持从列式存储中并行读取。此外,ByteHTAP 支持广泛的计算下推,包括但不限于选择谓词和聚合。为了减少我们商店的读取负载,我们在查询编译阶段主动应用分区修剪逻辑。查询优化器确定分区列上是否有任何谓词与我们的分区方案匹配,以便它可以删除不相关的分区。我们还对 Flink 的核心引擎做了一些其他重要的改进,以更好地服务于我们的查询模式(细节在第 6 节中讨论)。

我们的主要贡献之一是扩展 ByteNDB 的复制存储以支持具有高数据新鲜度和强数据一致性的行和列格式数据。在下一节中,我们将详细描述 ByteHTAP 的 Columnar Store。

4 海量数据共享存储

新鲜

在本节中,我们描述了 ByteHTAP 的列式存储,它由 Delta Store 和 Base Store 组成。 Delta Store 是分布式的

内存和行格式存储。它可以低延迟地应用日志,并提供高新鲜度的数据给OLAP 引擎进行查询。 Base Store 是一个分布式持久化的列式存储。 OLTP 端的 Delta Store 和 Base Store 加上 Log Store 和 Page Store在统一的 Replication Framework 下进行管理。它们构成了ByteHTAP 的共享存储层。我们采用多种技术来确保存储中数据的高新鲜度。

4.1 达美商城

在 ByteHTAP 中,可以对 OLAP 表进行分区,每个分区具有三个副本,以确保高可用性。因此,我们为表的每个分区副本维护一个增量存储。 Delta Store由两个列表组成:插入列表和删除列表,分别按照行的LSN顺序记录插入和删除操作。在 ByteHTAP 中,更新操作转换为删除操作,然后是逻辑日志中具有相同 LSN 的插入操作。

在 ByteHTAP 中,对 Delta Store 和 Base Store 的扫描需要检查扫描行是否已被删除,即该行是否在Delta Store 的删除列表中。因此,我们额外维护一个delete hash map来记录Delta Store中的所有删除,以加速删除列表的查找。

Delta Store 中有四个主要操作:LogApply、Flush、 Garbage Collection 和 Scan。 Delta Store中的数据结构和算法经过精心设计,使得这些操作可以并行运行,数据新鲜度高,数据一致性强。我们将在第 5 节讨论如何实现强数据一致性。

日志应用。如图2所示,Replication Framework根据表的partition key将逻辑日志分发到Delta Store。这些逻辑日志按其 LSN 排序。然后, LogApply 操作将每个插入日志和删除日志条目分别附加到插入列表和删除列表。删除日志也被插入到删除哈希映射中,其中映射的键是存储在日志中的主键。

冲洗。 Flush 是 Delta Store 中的一个后台任务,周期性地将累积的行格式数据传输到列格式,并将它们存储在位于同一存储节点上的相应持久化 Base Store 中。 Flush的过程可以描述如下。首先,我们根据数据块大小或行数方面的预定义阈值选择一个 LSN 作为此 Flush 的终点。上一次Flush的结束点和这个新的结束点组成一个fush range。其次,对于插入列表的fush范围内的所有行,我们根据它们的主键对它们进行排序,并将它们从行格式转换为列格式。第三,我们将列数据作为一个数据块写入Base Store。笔记

需要在上述过程中通过检查Delta Store中的删除hash map来排除fush范围内被删除的行。最后,我们通过更新Base Store 中数据块的删除位图来处理删除列表中的删除。删除位图的细节将在 4.2 节中描述。

垃圾收集 (GC)。 Garbage Collection 是一项后台任务,它会定期检查存储节点中的 Delta Stores。如果Delta Store 中的 fushed 数据达到阈值并且没有活动扫描

需要它们,GC 任务可以从Delta Store 中截断这些 fushed 数据以释放内存。

扫描。 OLAP 查询将同时扫描 Delta Store 和 Base Store,并将结果联合在一起。在 ByteHTAP 中,我们为每个查询提供带有读取 LSN 的快照读取,以实现数据的强一致性。

由于 Delta Store 目前不支持 spill 数据到磁盘,我们需要确保它能有效地利用内存,并且在数据变化率很高的情况下不会出现内存不足的错误。

ByteHTAP 有一个工作负载管理模块来监控资源使用情况,并管理资源使用情况以防止系统出现严重的性能问题。例如,当内存利用率高时,触发紧急 Flushes,使内存利用率恢复到正常状态。由于篇幅限制,我们在本文中跳过了工作负载管理的细节。

4.2 基础商店

Base Store 是为每个分区副本创建的持久列存储。它与关联的内存中增量存储位于存储节点上。我们明确决定不在Base Store 中存储每条数据记录的 LSN,以减少

存储开销并提高更新和扫描效率。这个决定的缺点是我们不支持读取比 Delta Store 中最旧版本更早的快照版本,这在我们当前的用例中不是问题。

Base Store 数据存储在存储节点的本地文件系统中的分区属性跨 (PAX)-类似于[18]格式。 PAX 类格式用于许多开源 OLAP 系统,例如 Kudu [42]。每个 Base Store 包含许多数据块,每个块是行的集合,默认大小为 32MB。内的数据

每个块都按表的主键排序。在每个数据块中,我们保留块级元数据和每一列的编码数据。块的元数据包括行数、键范围、主键的布隆过滤器以及每列统计信息（如最小值/最大值）,它们将在读取操作期间用于提前修剪数据。目前,我们只支持主键的基于值的索引,但我们将在未来支持二级索引。

来自 Delta Store 的 Flush 操作将生成新的 Base Store块。为了支持数据更改,一个重要的设计决策是我们使用删除位图来跟踪数据块上的删除。为了在不可变数据块上有效地应用 Delta Store 删除,我们利用 RocksDB [12]来存储从 Base Store 中删除的行的删除位图。每个数据块的删除位图作为单个键/值对存储在 RocksDB 中,以块 ID 作为键,以字节为单位的位图作为值,因为 RocksDB除了 append-only Base之外还提供了可靠且快速的查找/更新存储店铺。

马夫。 Base Store 实现了一种包含压缩和垃圾收集 (GC) 的修饰机制,以减少磁盘使用率并提高查询性能。由于 Base Store 的删除操作仅在删除位图中标记与删除行对应的位,但不会从其数据块中删除它们,因此Base Store 的磁盘使用量将在没有修饰过程的情况下继续增长。此外,分区的后续Flushes从其Delta Store中生成的数据块可能在主键范围内重叠。当查询针对给定的键范围扫描 Base Store 时,可能需要扫描多个键范围重叠的数据块,这将大大降低查询效率。

压实。 ByteHTAP运行时,后台线程会周期性地测量数据块中删除的百分比和不同数据块之间主键的重叠程度,以选择数据块进行Compaction。具体来说,删除率高或者重叠率高的数据块会优先进行Compaction。后台线程聚合两个或多个选中数据块的数据行,并写入

聚合结果到新的数据块中。新数据块不包含已删除的行,并最大限度地减少键范围内的重叠。
新块生成后,后台线程相应地自动更新新块的元数据。在Compaction结束时,后台线程也将旧数据块插入到一个GC列表中。存放在GC列表中的数据块会在异步GC时被回收。

垃圾收集。后台线程定期检查存储在 GC 列表中的数据块,如果没有活动查询仍在访问它们,则通过永久删除它们来回收存储空间。详细信息将在第 5 节中描述。

4.3 高数据新鲜度我们将数据新鲜度定

义为 OLTP (TP) 系统最近更改的数据（插入/更新/删除)对OLAP (AP) 查询可见的时间。在 HTAP 系统中查询最近更改的数据至关重要,尤其是对于实时分析工作负载。在ByteHTAP 中,在大多数情况下,AP 查询可以在不到一秒的时间内读取 TP 系统中的已提交日志。我们通过采用以下创新来实现这种高数据新鲜度：

高效的日志复制。如图 2 所示,Replication Framework 会将事务日志复制到统一存储层中的 TP 和 AP 数据存储。如第 3 节所述,日志分发采用基于 Quorum 的投票协议以实现良好的性能和高可用性。在日志分发过程中,如果由于副本不一致而出现空洞,我们会应用gossip协议,使用pull-and-push同步方式来填补空洞。因此,日志复制的延迟通常很低。

具有高效删除处理的快速 LogApply。在 ByteHTAP 中,更新操作被删除和插入操作取代。
ByteHTAP 使用软删除方法来实现快速 LogApply 和 Flush,通过将删除处理推送到查询扫描时,这有助于提高数据新鲜度。在 LogApply 期间,删除日志被插入到删除列表和删除哈希映射中,两者在 Delta Store 中的成本都是恒定的。使用删除

位图,Delta Store Flushes 不需要改变现有的数据块。另外,我们的TP系统保证了逻辑日志的合法性,节省了LogApply时对insert/delete日志的验证成本。例如,LogApply 不需要验证插入是否可能包含与Base Store 中现有数据重复的主键,因为在OLTP 系统中执行 DML 语句时会发现此类主键违规。

使用 Arenas 向量的高效内存管理。
使用竞技场向量的高效内存管理提高了 LogApply 和 Delta Store GC 的性能。在 LogApply 期间有大量内存分配,在Delta Store GC 期间有大量内存释放。这些操作的成本对

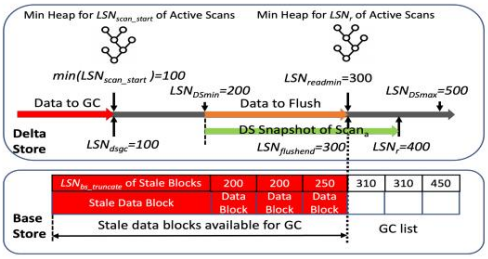


图 3:保存在增量存储和数据基本存储中的 LSN
一致性

数据新鲜度高。为了有效地管理内存中的Delta Store,我们引入了用于Delta Store内存管理的arenas向量。每个 Delta Store 最初都有一个可配置起始大小的竞技场。如果 Delta Store用完了当前 arena 中的内存,一个新的 arena 将被分配双倍大小（上限为 1MB)并添加到 vector 中。为了有效地分配和释放竞技场,我们将竞技场大小限制为 1 兆字节。每个区域都与一个 LSN 相关联,该 LSN 是保存的最后一行的 LSN

在这个舞台上。在 Delta Store GC 期间,我们释放那些LSN 小于 GC LSN 的 arena。

5 基于 LSN 的强数据一致性

如第 3 节所述,ByteHTAP通过基于 LSN 的版本控制机制保证了强数据一致性。在高层次上,我们的 OLAP 存储可以提供一致的快照读取。 Delta Store 保留最新版本的有限历史记录以支持并发查询。另一方面,Base Store 只保留一个版本,即 Delta Store最近一次 Flush 生成的最大数据版本。每个查询都携带一个从元数据服务获得的读取 LSN ,它可以使用它来获取其读取快照。此外， OLAP 端的数据操作通过各种 LSN 仔细协调,因为我们的共享存储中可以有许多并发操作。此外,不在 Base Store 中保留记录级版本也为在 ByteHTAP 中支持快照带来了额外的挑战。在本节中,我们将详细描述支持一致数据操作的算法。

5.1 ByteHTAP 中的重要 LSN作为版本控制机制
的基本构建块,如图 3 所示,在 Delta Store 中维护了几个 LSN,包括: :Delta 中活动数据条目的最低 LSN

- 还没有被 fush 到 Base Store 的 Store。
- : 增量中活动数据条目的最高 LSN
  店铺。
- :扫描的读取LSN。它也是扫描快照的上限。 ·
  :跟踪所有活动扫描的读取LSN 的最小 堆。 · 所有主动扫描。我们用一分钟 :
  扫描快照的下限,即扫描到达Delta Store 的时间。我们也

为 系统维护一个最小 的所有活动扫描 对于 – 中
堆。

- :下一次Flush中Delta Store的数据条目的最大LSN 。一旦一次Flush成功,按道理来说,Base Store的版本就会升级到那次Flush的版本值
- : Delta Store 的 LSN 小于此 LSN 的数据条目
  将被下一个 Delta Store GC 截断。 ·
  _ :在Base Store Compaction结束时分配给陈旧数据块的基于LSN的版本,描述当时Delta Store的状态并用于Base Store GC。

这些 LSN 与传统的并发控制技术一起,确保了数据分区上并发扫描的快照读取一致性,即使在同一分区内同时进行多个前台和后台数据操作时也是如此。

5.2 查询和 DML 处理在本节中,我们通过检
查扫描和数据分区上的其他数据操作之间的交互来解释如何保证快照读取一致性。

扫描。从整个系统的角度来看,表上的 OLAP 扫描被分配为每个分区的一个扫描任务,在所有任务中使用相同的读取 LSN（即 ）。对于单个分区,分区的Delta Store中所有未fushed数据的LSN形成一个LSN窗口[ ]。存在三种可能的情况:

- > :并非所有可见的扫描日志都已应用到 Delta Store 中,Replication Framework 将暂停此扫描,直到数据可用。
- < :快照在这种情况下不再有效,将返回错误。 OLAP 计算引擎将从元数据服务获取更新的 LSN,并使用最新的 · <= <= 重试查询

: 这是最常见的
case 和数据的可见性包括Delta Store 中[ ]中的所有插入和删除,以及 Base Store 中的所有数据。由于 Base Store 不维护数据的

记录,当 ByteHTAP 生成执行计划时,它通过获取所有相关持久数据块的名称并在这些块上制作当前删除位图的副本来拍摄快照。

此外,我们需要确保分区上的其他并发操作不会影响活动扫描的现有快照。
日志应用。 LogApply 是一个前台操作,它将一批插入和删除附加到增量存储列表的末尾。它会
只增加 但不会影响扫描快照。
冲洗。 Flush 是后台操作,将窗口 [
] 中的 作为

在新的 Flush 开始时,以防止后来到达的扫描遭受无效快照（即 ）。在Flush 结束时,它会自动将新的数据块添加到Base Store,更新删除现有数据块的位图,持久化到 <

并相应地更新新的 最旧的活动数据条目（如果增量存储中没有数据,则设置为 +
1）。 Scan 和 Flush 中的上述过程通过并发控制的适当锁定来同步。 ByteHTAP 用户可以将刷新频率配置为

控制要在增量存储中保留多少数据。 Delta Store 中保存的数据越多,ByteHTAP 可以支持的有效快照窗口范围就越大,但代价是消耗更多的内存。持久化的 也标记了最新的数据已经被持久化,作为故障恢复时 Replication Framework同步逻辑日志的起点。 存在的问题是长时间运行的查询可能会在很长一段时间内阻塞增量存储刷新。目前,长时间运行的查询在超过ByteHTAP 中预先定义的时间阈值后将被自动终止。

三角洲商店 GC。 Delta Store GC 是一个后台操作,用于删除没有活动扫描的已填充数据（即 LSN < ）。 设置为min(_),) GC 启动时。

基本存储压缩。基本存储压缩操作在完成后在新数据块和旧数据块的元数据之间进行原子切换。在原子元数据切换期间， Delta Store 中所有当前活动扫描的最大读取 LSN将被记录为陈旧数据块的 _ 以确保没有活动扫描在获得 GC-编辑。陈旧块将连同它们的 _ 附加到 Base Store GC 列表,如图3的Base Store 部分所示。

基地商店 GC。为防止删除正在被主动扫描访问的磁盘数据， Base Store GC 线程将当前的

只有 _ < 的数据块才能保证不提供任何主动扫描,因此可以安全地进行 GC-ed。

我们可以用一个例子来说明上面讨论的 LSN 的用法。图 3 说明了分区中的特定时间点。

此时Delta Store的 分别是200和500。假设和 =400 刚刚到达,而 Flush、Delta Store GC 和 Base Store GC 计划同时发生。假设有一个正在进行的扫描与=390 到达,

IE a a a是带有A 时,它在100时拍摄了其快照。因此,当前的

最小值（ _)为100， concurrent Flush 会将 [200, 300] 中的数据条目 fush 到 Base Store。在 Flush 结束时,新的 将被设置为下一个数据条目的 LSN,例如 301。同时， 将使用 和 原子地拍摄 Delta Store 和 Base Store 数据块（未显示）的快照插入两分钟

分别堆。所以如果 在Flush结束前拍摄快照,它将有一个 [200, 400] 的快照窗口,其中包含旧元数据、Base Store 数据块和删除位图,否则它将有一个 [301, 400] 的快照窗口,其中包含新刷新修改的元数据、数据块和删除位图。

对于Delta Store GC， 设置为100。 _ 值为250 表示当其对应的数据块被之前的Compaction变成stale时,此时Delta Store 中active query的最大读LSN为250 ,因此当前最旧的活动扫描 必须晚于此。因此,可以安全地清除此块和其他较旧的陈旧块,但不能清除 LSN=310 或更新的数据块,因为它们可能会被 访问。

5.3 DDL 处理除了 DML 和查询操作,ByteHTAP 还需要确保其 DDL 操作能够提供强数据一致性。为实现这一点,ByteHTAP 在其元数据服务 (MDS) 中保留了多个版本的数据库元数据,这是ByteHTAP 中元数据的真实来源。

当 DDL 语句到达 ByteHTAP 时,它将首先由 OLTP 引擎以 MySQL 兼容的方式进行处理。

接下来,Replication Framework负责将DDL操作产生的DDL逻辑日志发送给MDS和AP存储服务器。 MDS会解析DDL逻辑日志生成新版本的元数据,逻辑日志的LSN作为元数据版本号。 MDS 将持续存在,然后使这个新版本的元数据与所有历史版本的元数据一起可用于 AP 查询。

AP 存储服务器集成了一个 MDS 客户端,它能够定期拉取新的可用元数据,并在本地缓存一个完整的元数据历史副本。当一个DDL逻辑日志到达一个Columnar Store 分区时,在LogApply期间,MDS客户端会根据从MDS收到的DDL LSN来获取相应的元数据。目前,我们仅支持不需要数据重组的 DDL 更改,例如ADD、DROP 和 RENAME COLUMN。更高级的支持正在开发中。

6 OLAP 查询性能优化

除了高数据新鲜度和强数据一致性之外,性能是 ByteHTAP 的另一个重要指标,尤其是当它需要处理实时分析工作负载时。在本节中,我们将讨论存储引擎和 OLAP 查询引擎的性能优化,包括删除处理、计算下推、统计收集、异步读取和并行优化。我们目前没有 OLAP 计划缓存,因此每个查询都是用自己读取的 LSN 重新编译的。我们计划在未来添加此功能。

6.1 扫描的删除优化为了支持高效的删除处理,我们存储删除位图,其中包含有关在原始 Flush 后从 Base Store 中删除的行的信息。我们已经在中描述了位图格式

第 4.2 节。

在 Base Store 扫描操作期间,扫描器首先对要扫描的块 ID 进行快照,然后根据每个块 ID 获取位图,最后扫描基列数据并应用删除。除了在 Base Store 扫描期间检查位图中的删除之外,我们还需要有效地处理仍然驻留在 Delta Store 中的删除。我们实现了两种方法来实现这一点：

- 惰性方法:由于Delta Store 中的删除存储在Delta Store 中的删除散列映射中,我们首先扫描Base Store,然后对于结果中的每一行,我们进行散列查找以查看它之前是否已被删除我们返回最终结果。
- Eager 方法:在Base Store 扫描初始化阶段,首先检索从扫描快照中的Delta Store 中删除的键。对于每个删除的键,我们利用我们的主键索引

在 Base Store 中进行索引查找并在我们扫描 Base Store 数据之前构造一个选择向量。

每种方法都有其优点和缺点。惰性方法类似于我们探查基础数据的散列连接,然后使用增量存储中的 Delete Hashmap 对每一行进行散列查找,并返回连接结果。请注意,此方法始终需要选择主键列,即使查询不需要。另一方面,急切的方法类似于基于索引的嵌套循环连接,我们首先从 Delta Store 的删除列表中读取删除,然后对于每个删除,我们在 Base Store 表上进行索引查找,然后返回合并的结果。 eager方式的好处是当Delta Store的delete不多,Base Store有大量数据的时候,可以非常高效。另一方面,当 Delta Store 包含影响 Base Store 中许多数据块的大量删除时,eager 方法效率较低。为了产生最佳的执行计划,我们建立了一个成本模型,根据实际统计数据智能地确定最佳使用方法。统计数据是在每次刷新期间从 Delta Store 和 Base Store 即时收集的。实验结果(在第 7.4 节中给出)表明我们的成本模型可以选择合适的方法并产生最佳性能。

6.2 计算下推到存储引擎如图 2 所示,ByteHTAP 将存储和计算解耦。
因此,我们设计并实现了谓词下推和聚合门下推,以减少从存储引擎到查询引擎的数据传输。

谓词下推。查询规划器通过检查谓词是否可以被存储引擎评估及其相应的成本节省来决定是否可以将扫描运算符的谓词下推到存储引擎。通过谓词下推,扫描的数据将在存储层被谓词过滤,只有满足谓词的结果才会被发送回查询引擎进行进一步处理。 Base Store 将数据块的每一列的最小/最大列值维护为元数据,这是在 Delta Store Flush 或 Base Store Compaction 期间计算的。

如果基于最小/最大值的谓词评估可以过滤出一个数据块,那么我们根本不需要为该数据块从磁盘加载数据。此外,我们还通过首先评估带有谓词的列来对 Base Store 数据执行惰性物化。如果谓词的评估可以过滤掉数据块中的所有行,我们可以跳过该数据块以避免读取不需要的列。

聚合下推。在典型的 AP 工作负载中,聚合操作广泛用于大量数据。通常,聚合操作可以分为部分聚合(局部聚合门)和最终聚合。 AP 查询优化器可以考虑将符合条件的部分聚合下推到存储引擎,例如,当聚合下只有一个表扫描时,扫描的所有过滤谓词(如果存在)也可以下推到存储引擎。在每个分区中,存储引擎将在应用过滤谓词(如果存在)后聚合扫描的数据,并将部分聚合的结果返回给计算引擎。

然后,计算引擎将对部分结果进行最终聚合。

6.3 OLAP 查询引擎优化由于我们利用 Flink 来支持复杂的分析查询,我们对 Flink 的核心引擎做了一些改进,以提高其 OLAP 能力。由于我们当前客户的 OLAP 查询具有时效性,需要很高的每秒查询数 (QPS),因此我们的优化侧重于降低查询优化和查询执行的延迟,并提高整体系统吞吐量。

以下是我们所做的一些改进。
统计数据收集。我们使用 Flink 计算表的统计信息,并将结果存储在 ByteHTAP 的元数据服务(MDS)中。
在查询编译期间,我们的 Flink 连接器将从本地缓存或 MDS 中获取这些统计信息,以帮助 Flink 基于成本的优化。对于简单的查询,我们还提供了绕过完全基于成本的优化的快速路径。为了向查询提供最新的统计信息,我们还收集来自Flush 和 Compaction 操作的增量统计信息,并将它们存储在 MDS 中。当更改看起来足够重要时,这些额外的统计信息可能会触发 Flink 重新计算完整的统计信息。

异步读取。默认情况下,Flink 连接器是一个单线程,重复读取和处理数据段,直到完成对整个数据集的处理。作为一个顺序过程,在读取和处理数据的过程中需要大量的I/O等待时间,使得整个过程的效率较低。我们将连接器拆分为两个单独的线程,一个用于从列式存储中读取,另一个用于处理数据并将它们传递给其他运算符。它们通过可调节的缓冲区进行通信。通过这种方法,我们看到 I/O 吞吐量显着提高,TPC-DS 总运行时间减少了 10%。

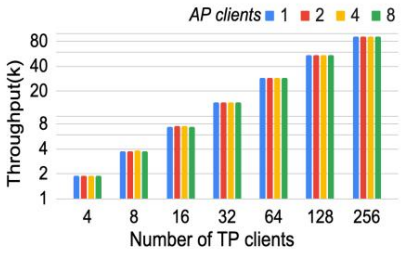
并行优化。 Flink 使用预先配置的任务并行性,不考虑数据大小和作业规模。这种设计浪费了简单查询的任务容器资源,并增加了需要更大并行度的复杂查询的端到端延迟。我们添加了优化器规则以根据数据统计信息和分区数调整源扫描并行性。另外,scan上面的算子可以根据源的并行度来调整自己的并行度。这项工作在 TPC-DS 工作负载上将我们的集群 QPS 提高了20%。

7 性能研究

在本节中,我们通过一组实验来描述我们系统的评估。首先,我们展示了为测量我们系统的 HTAP 能力而进行的实验。我们使用基准 CH-benCHmark [30] 测量OLTP 和 OLAP 引擎之间的性能干扰,并使用 Sys bench [14] 测量数据新鲜度。然后,我们使用 TPC-DS [16]评估我们系统的 OLAP 兼容性和性能。最后,我们展示了第 6 节中讨论的 OLAP 查询性能优化的结果。

所有的实验都是在一个由七台机器组成的集群上进行的。
表 1 列出了机器和集群的设置。请注意,一些 ByteHTAP 组件位于同一台机器上以节省资源。

7.1 混合 OLTP 和 OLAP 工作负载HTAP 系统在混合 OLTP/OLAP 工作负载上的性能至关重要。因此,我们在 ByteHTAP 上运行 CH-benCHmark [30]来测试其在混合工作负载上的性能。 CH-基准是



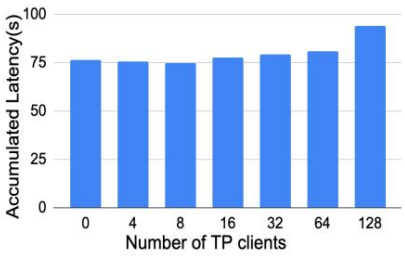


图 4:OLTP 的吞吐量图 5:CH-benCHmark中的 CH- benCHmark 查询延迟

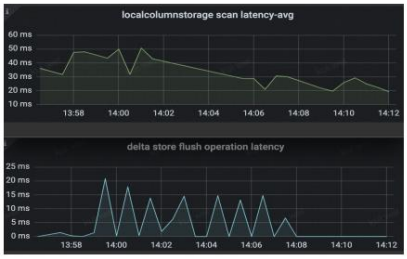


图 6:随时间变化的操作延迟

表 1:机器和集群设置。

中央处理器	英特尔(R) 至强(R) 金牌 5218（2 个 NUMA 节点,32 个核心）
记忆	378GB
缓存	22MB共享三级缓存
操作系统	Debian 4.14.81
网络	25Gbps 以太网
页面存储和日志存储	3台服务器
Columnar Store & OLAP engine 3台服务器	
OLTP 引擎和元数据服务1 台服务器	

标准 HTAP 基准,它弥合了用于 OLTP 的标准 TPC-C 基准和用于 OLTP 的 TPC-H 基准之间的差距
联机处理程序。 CH-benCHmark 建立在未修改版本的
TPC-C 基准测试,其 OLAP 部分包含从 TPC-H 继承的 22 个分析查询。它允许在一个数据库中的一组共享表上运行 OLTP 和 OLAP查询。我们的实验是基于 CH-benCHmark 的 100 个仓库数据进行的

设置[30]。数据首先加载到 ByteHTAP 的 OLTP 数据存储中,然后复制到 OLAP 列式存储中。

图 4 显示了具有不同数量的 TP 和 AP 客户端的 TP 引擎的吞吐量。吞吐量以每分钟事务数 (tpmC [15]) 来衡量。如图所示， TP引擎的吞吐量随着 TP客户端数量的增加几乎呈线性增长。对于固定数量的 TP 客户端,TP吞吐量几乎与不同数量的 AP 客户端相同。

结果表明,由于其独立引擎设计,ByteHTAP 的 OLTP 性能几乎不受 OLAP 工作负载的影响。

接下来,我们查看 CH-benCHmark 上的 AP 性能,其中使用 22 次分析查询的累积查询延迟作为性能指标。当我们通过多个TPC-C事务查询不断从TP端更新数据时,二进制日志被复制到触发Flush操作的AP端的Delta Store 。如图 5 所示,当 TP 客户端从 0 增加到 64 时,性能仅降低约 5%。这是因为大多数 fush 延迟都小于 20 毫秒,并且

如图 6所示,扫描延迟在刷新操作期间保持较低。

然而,当 TP 客户端的数量增加到图 5 中的 128 个时,累积的查询延迟显示出明显的增加。
这是因为在这种情况下,TP 和 AP 之间的日志复制

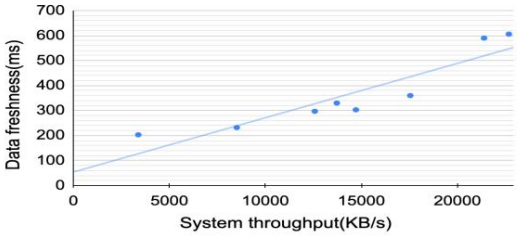


图 7:不同吞吐量的数据新鲜度

成为瓶颈,可能会在日志复制中引入 2 到 3 秒的延迟。发生这种情况时,如果 AP 查询想要获取具有最新提交的 LSN 的数据,则需要等到直到该 LSN 的最新日志被复制并对 AP 可见。我们可以通过添加更多机器资源来解决这个问题,以避免复制框架过载。到目前为止,这在我们的生产场景中很少发生。

综上所述,ByteHTAP 的 AP 性能在大多数情况下是稳定的,
即使在执行 Flush 和 Compaction 操作期间。

7.2 高数据新鲜度实验数据新鲜度是 HTAP 系统的一个重要指标。但是,我们找不到一个共同的衡量标准。以前的一些工作采用新鲜度度量[25, 55]。但是,我们决定使用新鲜度时间度量,即OLTP 端的数据修改对 OLAP 端的查询可用的速度有多快。

从我们的角度来看,这个指标更适合我们的用户,因为它非常符合他们的要求。

如前所述,事务的逻辑日志将从TP 存储同步到 AP 存储服务器,并在应用到 Delta Store 后对 AP 查询可见。因此,这个过程的总时间可以用作我们的新鲜度时间指标 。

根据我们的设计目标,对于典型的工作负载， 应该小于一秒。

我们使用表 1 中给出的相同实验设置。我们根据 SysBench [14]基准设计我们的数据新鲜度实验,并使用具有 257 个分区的散列分区表。

我们选择使用 16 个并发写入线程,这是在生产环境中观察到的典型 TP 客户端数量之一。我们通过在实验期间调整一个事务中的数据操作数来改变事务大小。我们从 1 次更新、1 次删除和 1 次插入开始,然后加倍

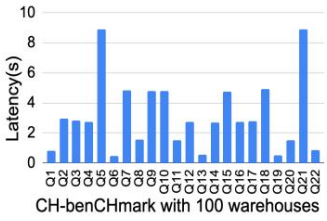


图 8:CH-benCHmark OLAP 性能

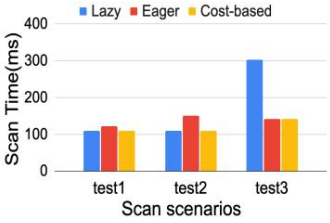


图 9:基于成本的模式优化

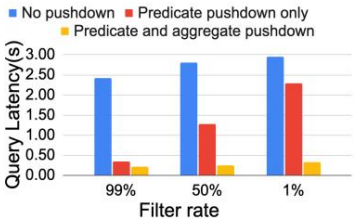


图 10:计算下推改进

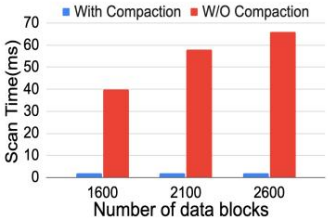


图 11:压缩对范围扫描的影响

每个操作随后直到 128。吞吐量是通过将事务大小乘以从 SysBench 获得的每秒事务数来计算的。

图 7 显示了具有不同事务吞吐量的257 个分区的平均新鲜度时间指标。正如预期的那样,新鲜度时间指标通常随着总数据吞吐量的增加而增加。请注意,在 16 个写入线程和22681 KB/s 的相对较高数据吞吐量的情况下,我们仍然观察到606 毫秒的低 。这满足了我们的设计目标。我们还对 32 个写入线程进行了实验,并观察到类似的模式。

7.3 OLAP 兼容性和性能在本节中,我们将讨论评估 Byte HTAP 的 OLAP 性能的实验。我们首先使用 TPC DS [16]基准测试 ByteHTAP,这是通用决策支持系统的标准基准。结果表明我们的系统可以支持所有的 TPC-DS 查询。支持复杂 OLAP 查询的能力对于用户采用 ByteHTAP 非常重要。我们还评估了一个知名开源软件1的 TPC-DS 性能。结果

市场上的 HTAP 系统,简称 HTAP-T,表明HTAP-T 仅支持 103 个 TPC-DS 查询中的 66 个。由于篇幅限制,我们省略了基准测试结果。

我们接下来讨论 ByteHTAP 在 CH-benCHmark 上的表现。目前,ByteHTAP 支持所有 22 个分析查询,但在 HTAP-T 中,查询 16 在其 OLTP 和OLAP 引擎上的结果不一致。图 8 显示了我们的系统在 CH benCHmark 上具有 100 个仓库的性能。我们可以看到大部分查询的延迟小于5秒， 21个查询的累积延迟与HTAP-T相当。我们还在CH-benCHmark 上测试了 Byte HTAP 的性能,其中有 1000 个仓库和七个以上的 OLAP 节点。所有查询都在 10 秒内完成。

由于篇幅限制,我们省略了细节。

7.4 性能优化评估在本节中,我们进行了多个实验来评估第 6 节中提出的性能优化。

扫描优化。我们进行了一个实验来测试第 6.1 节中描述的三种删除处理模式:惰性模式、急切模式和基于成本的模式。在实验中,我们创建了一个base case（test1）,在Base Store中有500、000行， Delta Store中删除了一半。如图 9 所示,当我们在 test1 中扫描少量行（即 500)时,惰性模式的扫描时间稍微好一些,因为急切模式会支付相对较高的前期费用

预处理增量存储删除的成本。在 test2 中,我们将Delta Store 中的删除次数增加到 99%。结果表明 eager模式支付的前期成本相应增加,而lazy模式的时间变化不大。在 test3 中,我们没有改变 delete 的数量,而是将扫描的行数增加到50, 000。lazy 模式的扫描成本比 eager 模式增长得更快,因为它需要支付固定的哈希查找成本每个扫描行。

图 9 还显示,我们的基于成本的模型可以根据收集的统计信息在懒惰模式和急切模式之间巧妙地进行选择,以获得良好的扫描性能。

计算下推。为了评估从查询引擎到存储引擎的计算下推的有效性,我们使用 TPC-DS [16]基准进行实验。我们在 TPC-DS 的表 store_sales 上使用带谓词和聚合的查询:

```
选择总和（ss_net_profit)作为利润
来自store_sales
其中ss_ticket_number < C
按ss_store_sk分组
ORDER BY利润DESC ;
```

查询中的常数 C 选择不同的过滤率:99%、50% 和 1%。对于每个过滤率,我们有三种模式:无下推、谓词下推以及谓词和聚合下推。

图 10 显示谓词下推和聚合下推可以显着提高 ByteHTAP 的性能。

压实。图 11 显示了使用和不使用压缩的扫描性能。在实验中,我们开发了单元测试,可以精确地生成在主键范围内重叠的数据块。我们在具有表 1 中描述的规范的服务器上运行测试。使用相同的模式定义了三个表,并填充了不同数量的数据。三个表的数据块总数分别为1500、2000、2500 。我们让每个表中的数据块在它们的主键范围内重叠。 key 范围在 0 到 100 之间的行被进一步删除并重新插入 100 次以添加额外的 100 重叠的数据块。对每个表执行范围扫描,选择主键范围在0到100之间的数据。

图 11 显示使用压缩时扫描时间显着减少,因为如果不压缩,则必须访问具有重叠主键的所有数据块。相反,压缩合并重叠的数据块并删除已删除的数据块。因此,扫描的数据块更少,扫描效率得到显着提高。

1 由于限制性专有许可协议,我们省略了供应商的名称。

Delta Store 与 Base Store 扫描性能对比。在此实验中,我们测量了 Base Store 中列格式数据与 Delta Store 中行格式数据的扫描性能。测试查询扫描TPC-DS (100GB) 的 web_sales 表中的 34 列之一。

表 2 显示了不同刷新率下的扫描性能： Delta Store 中的所有数据都被刷新到 Base Store (100%),一半数据被刷新 (50%),没有数据被刷新 (0%)。表 2显示 Base Store 中的扫描性能优于 Delta Store ,尽管 Delta Store 中的数据驻留在内存中,并且 Scan Speedups随着 Flushed Data (%) 的增加而增加。在 ByteHTAP 系统中,大部分数据通常被 fush 到 Base Store 并以列格式存储,只有最近的数据暂时驻留在 Delta Store 中并以行格式存储。

表 2:不同刷新率的扫描性能。

刷新数据 (%)	扫描加速
100%	2.90
50%	1.78
0%	1.00

从生产中吸取的 8 个经验教训

自从 ByteHTAP 全面上市以来,吸引了众多有 HTAP 需求的字节跳动内部客户。在 ByteHTAP之前,许多客户的管道由多个连接的系统组成。例如,在某些设置中,他们的行数据通常来自大量 MySQL 数据库,并由提取-转换-加载 (ETL) 系统预先聚合。然后,数据被加载到专门的 OLAP 系统中进行分析。

整个过程强加了 1-2 小时以上的延迟,分析师才能得到报告,当数据新鲜度要求低于几分钟时,这就变得不合适了。另一个缺点是客户必须为上述处理维护多个系统和 ETL 管道,这会产生巨大的运营开销。切换到 ByteHTAP 后,他们看到数据新鲜度从几小时缩短到不到 1 分钟,同时管理开销也大大减少。

虽然 ByteHTAP 仍处于产品采用的早期阶段,但我们从我们的生产经验中吸取了一些重要的教训。随着越来越多的客户开始使用 ByteHTAP,我们希望在未来了解更多。

允许 ByteNDB 客户轻松升级到 Byte HTAP。许多 ByteHTAP 的潜在客户来自ByteNDB 的现有客户,这些客户通常有多个在线的 ByteNDB 集群,每天产生数百 GB 的数据。

由于 ByteHTAP 的解耦 OLTP/OLAP 设计,迁移这些 ByteNDB 集群相当容易,因为 OLAP 组件可以单独部署和启用。现有在线OLTP集群无需数据迁移,无需时间,整个过程对客户透明。一般来说,我们认为在实践中许多 HTAP 用例来自现有的 OLTP客户。由于这一观察,我们决定为 ByteNDB 客户添加一个按钮升级选项,以将他们的OLTP 数据库升级到 ByteHTAP。

跨OLTP数据库查询能力。客户通常有多个 ByteNDB 数据库用于他们的 OLTP 工作负载,例如每个部门一个。但是,当他们执行 OLAP 分析时,他们希望能够跨这些数据库进行查询。从一些早期客户那里了解到这一需求后,我们进行了简单的更改,以允许 ByteHTAP OLAP 表从 ByteNDB 内部的多个数据库同步,这使客户的生活更加轻松。一般来说,在 OLTP 和 OLAP 数据映射之间提供一些灵活性似乎是一个不错的选择。

高效的数据导入。在将大型数据库实例从传统 OLTP 系统迁移到 HTAP 系统的情况下,我们需要创建基于 TP 数据库的初始 AP 存储。一种天真的方法是简单地将历史逻辑日志流式传输到 OLAP 列式存储,这种方法非常慢且在实践中不实用。我们开发了一个工具,从Page Store获取一致性快照,然后解析快照中的数据页,直接批量写入Base Store创建实例数据,大大减少了大实例迁移的时间。

Flink 增强功能。随着我们的用户将越来越多的工作负载迁移到我们的 ByteHTAP 系统,我们看到 QPS 的高速增长和更多样化的查询类型暴露了 Flink 查询引擎的性能瓶颈。因此,我们需要修改 Flink 来解决这些问题。例如,我们将 Flink 的计划生成从单线程的方式改进为并行的方式。

我们还重新设计了 Flink 的核心调度器,去掉了任务管理器这个沉重的角色,极大地提升了任务调度的吞吐量,最高可达 200%。通过所有这些努力,我们看到与Flink 的开源版本相比,TPC-H 查询的性能提高了 25%。这项工作已在 Flink Forward Asia 2021 [1-3] 上发表。

9 结论

ByteHTAP 是一个大规模的实时分析系统,支持新的数据变化和强数据一致性,旨在满足字节跳动不断增长的业务需求。在本文中,我们展示了如何构建具有独立引擎和共享存储架构的具有竞争力的 HTAP 系统。我们的模块化系统设计充分利用了字节跳动现有的 OLTP系统和开源 OLAP 系统。 ByteHTAP 可以提供不到一秒延迟的高数据新鲜度,这为我们的客户带来了许多新的商机。自2021 年年中左右推出以来,我们看到越来越多的内部客户使用它来替换他们以前由 OLTP 数据库、OLAP 数据库和其他 ETL 管道组合组成的系统。ByteHTAP通过一致的数据和更少的运营开销为我们的客户提供实时洞察。

致谢

感谢匿名审稿人提出的宝贵意见。我们衷心感谢所有为ByteHTAP 系统的设计和开发做出贡献的人:Ron Hu, Jie Zhou, Yupeng Jin, Liwen Shao, Xikai Wang, Shicai Zeng, Xiahao Zhang, Fangwen Su, Xiangrui Meng, Yong Fang, Weihua Hu , 曹迪洲, 何润康, 张光辉.最后,我们感谢 Vipul Gupta 对这份手稿的仔细校对。

参考

[1] 2021. Flink Forward Asia 2021. Retrieved February 23, 2022 from https://fink forward.org.cn/ [2] 2021. Job Scheduler and Query Execution on Flink OLAP 的改进。 2022 年 2 月 23 日从https://www.bilibili.com/video/BV1j34y1B72o 检索？ p=7

[3] 2021。使用 Apache Flink 在 ByteDance 为 HTAP 提供支持。 2022 年2 月 23 日从https://www.bilibili.com/video/BV1j34y1B72o?p=3 [4] 2022.ANSI SQL 标准中检索。 2022 年 2 月 23 日从https://webstore 检索。 ansi.org/Standards/ISO/ISOIEC90752016

[5] 2022. Apache Flink。 2022 年 2 月 7 日从https://fink.apache.org [6] 2022.BaikalDB 检索。 2022 年 2 月 7 日检索自https://github.com/baidu/ 贝加尔数据库

[7] 2022. 微软 Azure 突触分析。 2022 年 2 月 23 日从https://azure.microsoft.com/en-us/services/synapse-analytics/ [8] 2022.MySQL 中检索。 2022 年 2 月 23 日从https://www.mysql.com/ [9] 2022.OceanBase 检索。 2022 年 2 月 7 日从https://open.oceanbase.com 检索[10] 2022. PolarDB-X. 2022 年 2 月 7 日从https://www.alibabacloud 检索。 com/product/polardb-x [11] 2022.Presto。 2022 年 2 月 23 日从https://prestodb.io [12] 2022 中检索。RocksDB。 2022 年 2 月 18 日从http://rocksdb.org/ [13] 2022.SingleStore 检索。 2022 年 2 月 7 日从https://www.singlestore.com检索[14] 2022. 系统基准。 2022 年 2 月 11 日从https://github.com/akopytov/检索

系统台

[15] 2022. TPC-C 规范。 2022 年 2 月 23 日从http://www.tpc 检索。 org/tpc_documents_current_versions/pdf/tpc-c_v5.11.0.pdf [16] 2022.TPC-DS。 2022 年 2 月 11 日从http://www.tpc.org/tpcds/ [17] 2022.TPC-H 检索。 2022 年 2 月 11 日从http://www.tpc.org/tpch/ [18] Anastassia Ailamaki、David J DeWitt 和 Mark D Hill 检索。 2002.深内存层次结构上关系数据库的数据页面布局。 VLDB 期刊 11, 3 (2002), 198–215。

[19] Michael Armbrust, Reynold S Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K Bradley, Xiangrui Meng, Tomer Kaftan, Michael J Franklin, Ali Ghodsi, et al. 2015. Spark sql:spark 中的关系数据处理。在 2015 年ACM SIGMOD 数据管理国际会议论文集中。 1383–1394。

[20] Hillel Avni, Alisher Aliev, Oren Amor, Aharon Avitzur, Ilan Bronshtein, Eli Ginot, Shay Goikhman, Eliezer Levy, Idan Levy, Fuyang Lu, 等。 2020. 使用主内存和多核的工业级 OLTP。 VLDB 捐赠论文集 13, 12 (2020), 3099–3111。

[21] Ronald Barber、Christian Garcia-Arellano、Ronen Grosman、Guy Lohman、C Mohan、Rene Muller、Hamid Pirahesh、Vijayshankar Raman、Richard Sidle、Adam Storm 等。 2019. Wiser:用于物联网应用程序的高可用性 HTAP DBMS。 2019年IEEE大数据国际会议（Big Data）。 IEEE，268–277。

[22] Ronald Barber、Christian Garcia-Arellano、Ronen Grosman、Rene Mueller、Vijayshankar Raman、Richard Sidle、Matt Spilchen、Adam J Storm、Yuanyuan Tian、Pinar Tözün 等。 2017. 为新一代大数据应用开发的数据库。 在 CIDR 中。

[23] Ronald Barber、Matt Huras、Guy Lohman、C Mohan、Rene Mueller、Fatma Özcan、Hamid Pirahesh、Vijayshankar Raman、Richard Sidle、Oleg Sidorkin 等。 2016. Wildfre:并发的炽热数据摄取和分析。在 2016 年国际数据管理会议论文集中。 2077–2080 年。

[24] Dipti Borkar、Ravi Mayuram、Gerald Sangudi 和 Michael Carey。 2016. 拥有你的数据并查询它:从键值缓存到大数据管理。在2016 年国际数据管理会议论文集中。 239-251。

[25]莫克兰·布泽古布。 2004. 数据新鲜度分析框架。 2004 年信息系统信息质量国际研讨会论文集， 59-67。

[26] Dennis Butterstein、Daniel Martin、Knut Stolze、Felix Beier、Jia Zhong 和Lingyun Wang。 2020. 以变化的速度复制:用于近实时 HTAP 处理的快速、可扩展的复制解决方案。 VLDB 捐赠会论文集13, 12 (2020), 3245–3257。

[27] Le Cai, Jianjun Chen, Jun Chen, Yu Chen, Kuorong Chiang, Marko A. Dimitrijevic, Yonghua Ding, Yu Dong, Ahmad Ghazal, Jacques Hebert, Kamini Jagtiani, Suzhen Lin, Ye Liu, Demai Ni, Chunfeng Pei , Jason Sun、Li Zhang、Mingyi Zhang 和Cheng Zhu。 2018. FusionInsight LibrA:华为企业云数据分析平台。过程。 VLDB 捐赠。 11 (2018), 1822–1834。

[28]陈建军，陈宇，陈志标，Ahmad Ghazal，李国良，李思豪，欧伟杰，孙杨，张明义，周敏琪。 2019.华为数据管理:最近的成就和未来的挑战。 2019年IEEE第35届国际数据工程会议（ICDE）。 IEEE，13-24。

[29] Jack Chen、Samir Jindel、Robert Walzer、Rajkumar Sen、Nika Jimsheleishvilli 和 Michael Andrews。 2016. MemSQL 查询优化器:用于分布式数据库中实时分析的现代优化器。 VLDB 捐赠论文集9, 13 (2016), 1401–1412。

[30] Richard Cole、Florian Funke、Leo Giakoumakis、Wey Guy、Alfons Kemper、Stefan Krompass 、Harumi Kuno、Raghunath Nambiar、Thomas Neumann、Meikel Poess 等。 2011. 混合工作负载 CH-benCHmark。在第四届会议记录中测试数据库系统国际研讨会。 1–6。

[31] Alexandros G. Dimakis、Soummya Kar、José MF Moura、Michael G. Rabbat 和Anna Scaglione。 2010. 分布式信号处理的八卦算法。过程。 IEEE 98、11、1847–1864。

[32] Franz Färber、Sang Kyun Cha、Jürgen Primsch、Christof Bornhövd、Stefan Sigg和 Wolfgang Lehner。 2011. SAP HANA 数据库:现代业务应用程序的数据管理。 VLDB 捐赠论文集 40, 4 (2011), 45–51。

[33]戴维·吉福德。 1979. 复制数据的加权投票。第七届 ACM 操作系统原理研讨会论文集，150-162。

[34]黄东旭, 刘奇, 崔秋, 方竹和, 马晓宇, 徐飞, 沉力, 唐刘,周宇兴, 黄梦龙, 等． 2020. TiDB:基于 Raft 的 HTAP数据库。 VLDB 捐赠论文集 13, 12 (2020), 3072–3084。

[35] Murtadha Al Hubail、Ali Alsuliman、Michael Blow、Michael Carey、Dmitry Ly chagin、Ian Maxon 和 Till Westmann。 2019. Couchbase 分析:用于可扩展 NoSQL 数据分析的 NoETL。 VLDB 捐赠论文集 12, 12 (2019), 2275–2286。

[36] Patrick Hunt、Mahadev Konar、Flavio P Junqueira 和 Benjamin Reed。 2010. ZooKeeper:互联网规模系统的无等待协调。 2010年USENIX年度技术会议（USENIX ATC 10）。

[37] Alfons Kemper 和 Thomas Neumann。 2011. HyPer:基于虚拟内存快照的混合OLTP&OLAP 主内存数据库系统。 2011年IEEE第27届数据工程国际会议。 IEEE，195–206。

[38] Tirthankar Lahiri、Shasank Chavan、Maria Colgan、Dinesh Das、Amit Ganesh、 Mike Gleeson、Sanket Hase、Allison Holloway、Jesse Kamp、Teck-Hua Lee 等。 2015. Oracle 内存数据库:一种双格式内存数据库。 2015年IEEE第31届数据工程国际会议。 IEEE，1253–1258。

[39] Per-Åke Larson、Adrian Birka、Eric N Hanson、Weiyun Huang、Michal Nowakiewicz 和 Vassilis Papadimos。 2015.使用 SQL Server进行实时分析处理。 VLDB 捐赠论文集 8, 12 (2015), 1740–1751。

[40] Chris Lattner 和 Vikram Adve。 2004. LLVM:终身程序分析和转换的编译框架。在代码生成和优化国际研讨会上，2004 年。CGO 2004。IEEE，75–86。

[41] Juchang Lee、SeungHyun Moon、Kyu Hwan Kim、Deok Hoe Kim、Sang Kyun Cha和 Wook-Shin Han。 2017. SAP HANA 中的跨格式并行复制，用于扩展混合 OLTP/OLAP 工作负载。 VLDB 捐赠会论文集10, 12 (2017), 1598–1609。

[42] Todd Lipcon、David Alves、Dan Burkert、Jean-Daniel Cryans、Adar Dembo、Mike Percy、Silvius Rus、Dave Wang、Matteo Bertozzi、Colin Patrick McCabe 等。 2015. Kudu:用于快速数据快速分析的存储。 Cloudera 公司 28 (2015)。

[43] Chen Luo, Pinar Tözün, Yuanyuan Tian, Ronald Barber, Vijayshankar Raman, and Richard Sidle. 2019. Umzi:大规模的统一多区域索引 HTAP。在 EDBT 中。 1-12。

[44] Zhenghua Lyu, Huan Hubert Zhang, Gang Xiong, Gang Guo, Haizhou Wang, Jinbao Chen, Asim Praveen, Yu Yang, Xiaoming Gao, Alexandra Wang, et al. 2021. Greenplum:用于事务和分析工作负载的混合数据库。在2021 年 ACM SIGMOD 国际数据管理会议论文集中。 2530–2542。

[45] Darko Makreshanski、Jana Giceva、Claude Barthels 和 Gustavo Alonso。 2017. BatchDB:用于交互式应用程序的混合 OLTP+ OLAP 工作负载的高效隔离执行。在 2017 年 ACM 国际数据管理会议论文集中。 37–50。

[46] Norman May、Alexander Böhm 和 Wolfgang Lehner。 2017. SAP HANA –内存中 DBMS 从纯 OLAP 处理向混合工作负载的演变。商业、技术和网络数据库 (BTW 2017) (2017)。

[47] John Meehan、Nesime Tatbul、Stan Zdonik、Cansu Aslantas、Ugur Cetintemel、 Jiang Du、Tim Kraska、Samuel Madden、David Maier、Andrew Pavlo 等。 2015. S-Store:流媒体满足事务处理。 VLDB 捐赠论文集8, 13 (2015)。

[48] Barzan Mozafari、Jags Ramnarayan、Sudhir Menon、Yogesh Mahajan、Soubhik Chakraborty、Hemant Bhanawat 和 Kishor Bachhav。 2017. SnappyData:用于流式处理、事务和交互分析的统一集群。在 CIDR 中。

[49] Niloy Mukherjee、Shasank Chavan、Maria Colgan、Dinesh Das、Mike Gleeson、 Sanket Hase、Allison Holloway、Hui Jin、Jesse Kamp、Kartik Kulkarni 等。 2015. oracle数据库内存分布式架构。 VLDB 捐赠论文集 8, 12 (2015), 1630–1641。

[50] Fatma Özcan、Yuanyuan Tian 和 Pinar Tözün。 2017. 混合事务/分析处理:一项调查。在 2017 年 ACM 国际数据管理会议论文集中。 1771–1775。

[51] Andrew Pavlo、Gustavo Angulo、Joy Arulraj、Haibin Lin、Jiexi Lin、Lin Ma、 Prashanth Menon、Todd C Mowry、Matthew Perron、Ian Quah 等。 2017. 自动驾驶数据库管理系统。在 CIDR，卷。 4. 1.

[52] Vijayshankar Raman、Gopi Attaluri、Ronald Barber、Naresh Chainani、David Kalmuk、Vincent KulandaiSamy、Jens Leenstra、Sam Lightstone、刘少荣、 Guy M Lohman 等。 2013. 具有 BLU 加速功能的 DB2:不仅仅是一个列存储。 VLDB 捐赠论文集 6, 11 (2013), 1080–1091。

[53] Jags Ramnarayan、Barzan Mozafari、Sumedh Wale、Sudhir Menon、Neeraj Kumar、Hemant Bhanawat、Soubhik Chakraborty、Yogesh Mahajan、Rishitesh Mishra和Kishor Bachhav。 2016. Snappydata：基于 spark 构建的混合事务分析存储。在 2016 年国际数据管理会议论文集中。 2153–2156。

[54] Aunn Raza、Periklis Chrysogelos、Angelos Christos Anadiotis 和 Anastasia Ailamaki。 2020. 通过弹性资源调度的自适应 HTAP。在2020 年 ACM SIGMOD 数据管理国际会议论文集中。
2043–2054 年。

[55] Aunn Raza、Periklis Chrysogelos、Angelos Christos Anadiotis 和 Anastasia Ailamaki。 2020. 通过弹性资源调度的自适应 HTAP。 2020 年 ACM SIGMOD 数据管理国际会议论文集，2043-2054 。

[56] Bart Samwel、John Cieslewicz、Ben Handy、Jason Govig、Petros Venetis、Chanjun Yang、Keith Peters、Jef Shute、Daniel Tenedorio、Himani Apte 等。 2018. F1查询：大规模声明式查询。 VLDB 基金会论文集 11, 12 (2018), 1835–1848。

[57] Hemant Saxena、Lukasz Golab、Stratos Idreos 和 Ihab F Ilyas。 2021. HTAP 工作负载的实时LSM 树。 arXiv 预印本 arXiv:2101.06801 (2021)。

[58]沉思杰, 陈蓉, 陈海波, 臧斌余. 2021. 改造高可用性机制以驯服混合事务/分析处理。在第 15 届{USENIX}操作系统设计与实现研讨会({OSDI} 21)。 219–238。

[59] Reza Sherkat、Colin Florendo、Mihnea Andrei、Rolando Blanco、Adrian Dra gusanu、Amit Pathak、Pushkar Khadilkar、Neeraj Kulkarni、Christian Lemke、 Sebastian Seifert 等。 2019. SAP HANA 的原生商店扩展。 VLDB 捐赠会论文集 12, 12 (2019), 2047–2058 。

[60] Vishal Sikka、Franz Färber、Wolfgang Lehner、Sang Kyun Cha、Thomas Peh 和 Christof Bornhövd。 2012. SAP HANA 数据库中的高效事务处理：列存储神话的终结。在 2012 年 ACM SIGMOD数据管理国际会议记录中。 731–742。

[61] Alexandre Verbitski、Anurag Gupta、Debanjan Saha、Murali Brahmadesam、Kamal Gupta、Raman Mittal、Sailesh Krishnamurthy、Sandor Maurice、Tengiz Kharatishvili 和 Xiaofeng Bao。 2017. Amazon aurora：高吞吐量云原生关系数据库的设计注意事项。在 2017 年ACM 国际数据管理会议论文集中。 1041–1052。

[62] Jiacheng Yang, Ian Rae, Jun Xu, Jef Shute, Zhan Yuan, Kelvin Lau, Qiang Zeng, Xi Zhao, Jun Ma, Ziyang Chen, et al. 2020. F1 Lightning：HTAP 即服务。 VLDB 捐赠会论文集 13, 12 (2020), 3313–3325。

[63] Matei Zaharia、Mosharaf Chowdhury、Michael J Franklin、Scott Shenker 和 Ion Stoica。 2010. Spark：使用工作集进行集群计算。在第二届 USENIX云计算热门话题研讨会 (HotCloud 10) 中。