Alibaba Hologres:用于混合服务/分析处理的云原生服务

Xiaowei Jiang, Yuejun Hu, Yu Xiang, Guangran Jiang, Xiaojun Jin, Chen Xia, Weihua Jiang, Jun Yu, Haitao Wang, Yuan Jiang, Jihong Ma, Li Su, Kai Zeng
Alibaba Group {xiaowei.jxw, yuejun.huyj, yu.xiangy, guangran.jianggr, xiaojun.jinxj, chen.xiac, guobei.jwh, bob.yj, haitao.w, yuan.jiang, jihong.ma, lisu.sl, zengkai.zk}@alibaba-inc.com

抽象的

在现有的大数据堆栈中,分析处理和知识服务的过程通常分离在不同的系统中。在阿里巴巴,我们观察到这两个过程融合的新趋势:知识服务产生新数据,这些数据被输入到分析处理过程中,进一步微调服务过程中使用的知识库。将这种融合处理范例拆分为单独的系统会产生开销,例如额外的数据重复、不一致的应用程序开发和昂贵的系统维护。

在这项工作中,我们提出了 Hologres,这是一种用于混合服务和分析处理 (HSAP) 的云原生服务。 Hologres 解耦了计算层和存储层,允许在每一层中灵活扩展。表被划分为自我管理的分片。

每个分片相互独立地同时处理其读取和写入请求。 Hologres 利用混合行/列存储来优化 HSAP 中使用的点查找、列扫描和数据摄取等操作。我们提出执行上下文作为系统线程和用户任务之间的资源抽象。执行上下文可以以很少的上下文切换开销进行协作调度。查询被并行化并映射到执行上下文以进行并发执行。调度框架强制执行不同查询之间的资源隔离,并支持可定制的调度策略。我们进行了实验,将 Hologres 与专门为分析处理和服务工作负载设计的现有系统进行比较。结果表明,Hologres 在系统吞吐量和端到端查询延迟方面始终优于其他系统。

PVLDB 参考格式:
Xiaowei Jiang, Yuejun Hu, Yu Xiang, Guangran Jiang, Xiaojun Jin, Chen Xia, Weihua Jiang, Jun Yu, Haitao Wang, Yuan Jiang, Jihong Ma, Li Su, Kai Zeng.
Alibaba Hologres:用于混合服务/分析处理的云原生服务。 PVLDB,13(12):3272 - 3284,2020 年。
DOI:https://doi.org/10.14778/3415478.3415550

一、简介

现代商业普遍受到从海量数据中获取商业前景的驱动。从跑步的经验来看阿里巴巴内部的大数据服务栈,以及公有云的服务,我们观察到了现代商业使用大数据的新模式。例如,为支持实时学习和决策制定,现代电子商务服务背后的大数据堆栈通常会汇总购买交易和用户点击日志等实时信号,以持续获取最新的产品和用户统计数据。这些统计数据以在线和在线方式大量使用,例如:(1) 它们作为重要特征立即在线提供。传入的用户事件与这些功能相结合,为搜索和推荐系统中的实时模型训练生成样本。 (2) 它们还被数据科学家用于复杂的交互分析,以得出模型调整和营销操作的见解。这些使用模式清楚地展示了在线分析处理 (OLAP) 的传统概念无法再准确涵盖的一系列新趋势:分析处理与服务的融合。传统的 OLAP 系统通常在整个业务栈中扮演一个相当静态的角色。他们在线分析大量数据并获取知识(例如,预先计算的视图、学习的模型等),但将获取的知识交给另一个系统来为在线应用程序提供服务。不同的是,现代商业决策是一个不断调整的在线过程。衍生知识不仅被服务,而且还参与复杂的分析。对大数据的分析处理和服务需求融合在一起。

在线和在线分析的融合。现代企业需要将新获得的数据快速转化为洞察力。写入的数据必须在几秒钟内可供读取。冗长的在线 ETL 过程不再是可容忍的。此外,在所有收集的数据中,传统的从OLTP系统同步数据的方式只占很小的一部分。更多数量级的数据来自交易较少的场景,例如用户点击日志。在处理查询时,系统必须以非常低的延迟处理大量数据摄取。

现有的大数据解决方案通常使用不同系统的组合来托管混合服务和分析处理工作负载。例如,使用 Flink [4] 等系统实时预聚合摄取的数据,这些系统填充在处理多维分析的 Druid [36] 等系统中,并在 Cassandra [26] 等系统中提供服务。这不可避免地会导致跨系统的过多数据重复和复杂的数据同步,抑制应用程序立即对数据采取行动的能力,并招致重要的开发和管理开销。

在本文中,我们认为混合服务/分析处理 (HSAP) 应该统一并在单个系统中处理。在阿里巴巴,我们构建了一个名为 Hologres 的云原生 HSAP 服务。作为一种新的服务范式,HSAP 面临着与现有大数据堆栈截然不同的挑战(详细讨论请参见第 2.2 节):(1) 系统需要处理更高的查询工作负载

VLDB 捐赠论文集,卷。 13,第 12 期 ISSN 2150-8097。

DOI:https://doi.org/10.14778/3415478.3415550

比传统的 OLAP 系统。这些工作负载是混合的,具有非常不同的延迟和吞吐量权衡。 (2) 在处理高并发查询工作负载的同时,系统还需要跟上高吞吐量的数据摄取。摄取的数据需要在几秒钟内可供读取,以满足服务和分析工作对新鲜度的严格要求。 (3) 混合工作负载[illegible]性,能够迅速对这些突发事件做出反应。

为了应对这些挑战,构建 Hologres 时对系统设计进行了全面反思:存储设计。 Hologres 采用了一种将存储与计算解耦的架构。数据远程保存在云存储中。 Hologres 以表组的形式管理表,将一个表组划分为多个分片。每个分片都是独立的,manages 独立读写。与物理工作节点解耦,数据分片可以在工作节点之间灵活迁移。 Hologres以数据分片作为基本数据管理单元,通过分片迁移可以高效实现故障恢复、负载均衡、集群扩容等过程。

为了同时支持低延迟查询和高吞吐量写入,分片被设计为版本化。每个表组分片上读写的关键路径是分开的。

Hologres 使用 tablet 结构来统一存储表。 Tablets 可以是行格式或列格式,并且都以类似 LSM 的方式进行管理,以最大化写入吞吐量,并最小化数据摄取的新鲜度延迟。

并发查询执行。我们构建了一个面向服务的资源管理和调度框架,命名为 HOS。 HOS 使用执行上下文作为系统线程之上的资源抽象。

执行上下文是协同调度的,上下文切换开销很小。 HOS 通过将查询划分为细粒度的工作单元并将工作单元映射到执行上下文来并行化查询执行。该架构可以充分发挥硬件高并行性的潜力,使我们能够同时多路复用大量查询。执行上下文还有助于实施资源隔离,这样低延迟服务工作负载可以与同一系统中的分析工作负载共存而不会被停滞。 HOS 使系统可以根据实际工作负载轻松扩展。

回想起来,我们列出了以下贡献: 1. 我们为混合服务引入了一种新的大数据服务范例
ing/analytical processing (HSAP),并确定这种新范式下的新挑战。

2. 我们设计并实现了一个名为 Hologres 的云原生 HSAP 服务。 Hologres 具有新颖的存储设计,以及名为 HOS 的高效资源管理和调度层。这些新颖的设计组合帮助 Hologres 实现实时摄取、低延迟服务、交互式分析处理,还支持与其他系统(如 PostgreSQL [12])的联合查询执行。

3. 我们在阿里巴巴内部大数据栈和公有云平台部署了Hologres,并在真实工作负载下进行了全面的性能研究。我们的结果表明,即使与专门的服务系统和 OLAP 引擎相比,Hologres 也能实现卓越的性能。本文组织如下:第 2 节介绍了 Hologres 的关键设计考虑和系统概述。在第 3 节中,我们解释了数据模型和存储框架。接下来,我们在第 4 节介绍调度机制和查询处理的细节。实验结果在第 5 节介绍和讨论。最后,我们在第 6 节讨论相关研究并总结这项工作。

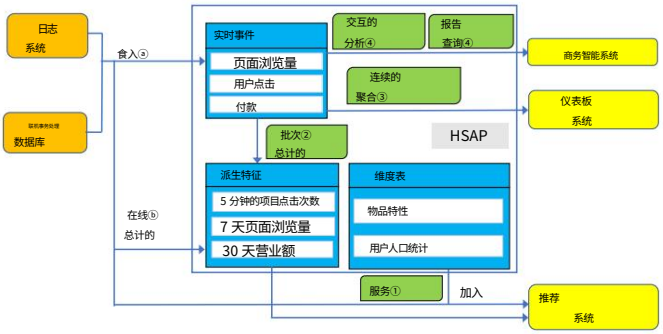


图 1:HSAP 场景示例:推荐服务背后的大数据堆栈

2. 关键设计考虑

现代企业中的大数据系统面临着越来越多的混合服务和分析处理的需求。本节我们使用阿里巴巴的推荐服务来演示一个典型的HSAP场景,总结HSAP给系统设计带来的新挑战。 en 我们提供了 Hologres 如何应对这些挑战的系统概述。

2.1 HSAP 在行动

现代推荐服务非常重视反映实时用户趋势并提供个性化推荐。为了实现这些目标,后端大数据栈已经演化到一个极其复杂和多样化的数据处理模式的状态。图 1 展示了支持阿里巴巴电子商务平台推荐服务的大数据堆栈的示意图。

为了捕捉个性化的实时行为,推荐服务严重依赖实时特征和不断更新的模型。通常有两种实时特征: 1. e 平台积极收集大量实时事件,包括日志事件(例如,页面浏览量、用户点击),以及交易(例如,从平台同步的付款)OLTP 数据库)。

正如我们从生产中观察到的那样,这些事件的数量非常大,其中大部分是较少的事务日志数据,例如 107 个事件/秒。这些事件被立即摄取到数据堆栈(a)以供将来使用,但更重要的是[illegible]时馈入推荐系统.实时连接需要以极低的延迟和高吞吐量对维度数据进行点查找,以便跟上摄取。 2. e 平台还通过聚合滑动窗口中的实时事件,以及各种维度和时间粒度(例如,5 分钟的项目点击、7 天的页面浏览量和 30 天的营业额)来导出许多特征。这些聚合根据滑动窗口粒度以批处理 (2) 或流式方式执行,并摄取到数据堆栈 (b) 中。这些实时数据还用于生成训练数据,通过在线和在线训练不断更新推荐模型。

尽管它很重要,但上述过程只是整个管道的一小部分。有一整套支持推荐系统的监控、验证、分析和改进过程。这些包括但不限于对收集到的事件进行连续的仪表板查询 (3) 以监控关键性能指标并进行 A/B 测试,以及定期批量查询 (4)

生成 BI 报告。此外,数据科学家不断对收集到的数据进行复杂的交互分析,以得出业务决策的实时见解,进行因果分析和模型改进。例如,在双十一购物节,传入的 OLAP 查询请求可以达到每秒数百个查询。上面的 e 演示了一个高度复杂的 HSAP 场景,从实时摄取（a)到批量加载（b）,从服务工作负载（1）,连续聚合（3）,到交互式分析（4）,一直到批量分析 (2)。在没有统一系统的情况下,上述场景必须由多个孤立的系统共同服务,例如通过Hive 等系统进行批量分析;通过像 Cassandra 这样的系统来处理工作负载;通过 Druid 等系统进行持续聚合； Impala 或 Greenplum 等系统的交互式分析。

2.2 HSAP 服务的挑战作为一种新的大数据服务范式,
HSAP 服务提出了 chal
就在几年前还没有那么突出。

高并发混合查询工作负载。 HSAP 系统通常面临传统 OLAP 系统前所未有的高查询并发性。在实践中,与 OLAP 查询工作负载相比,服务查询工作负载的并发度通常要高得多。例如,我们在现实生活中的应用程序中观察到服务查询可以达到每秒 107 个查询 (QPS) 的速率,这比 OLAP 查询的 QPS 高出五个数量级。此外,与 OLAP 查询相比,服务查询对延迟的要求要严格得多。如何在多路复用它们以充分利用计算资源的同时完成这些不同的查询 SLO 确实具有挑战性。

现有的OLAP系统一般采用基于进程/线程的并发模型,即使用单独的进程[5]或线程[6]来处理查询,并依赖操作系统来调度并发查询。这种设计导致的昂贵的上下文切换对系统并发性施加了硬性限制,因此不再适用于 HSAP 系统。并且它阻止系统具有足够的调度控制来满足不同的查询 SLO。

高吞吐量实时数据摄取。在处理高并发查询工作负载的同时,HSAP 系统还需要处理高吞吐量的数据摄取。在所有摄取的数据中,传统的从OLTP系统同步数据的方式只占很小的一部分,而大部分数据来自实时日志数据等不具备强事务语义的各种数据源。摄取量可能比在混合事务分析处理 (HTAP) 系统中观察到的要高得多。例如,在上述场景中,摄入率高达每秒数千万个元组。此外,与传统的 OLAP 系统不同,HSAP 系统需要实时数据摄取 写入的数据必须在亚秒级内可见 以保证分析的数据新鲜度。

高弹性和可扩展性。摄取和查询工作负载可能会突然爆发,因此需要系统具有弹性和可扩展性,并能迅速做出反应。我们在实际应用中观察到,峰值摄取吞吐量达到平均值的 2.5 倍,峰值查询吞吐量达到平均值的 3 倍。

此外,摄取和查询工作负载的爆发不一定同时发生,这需要系统独立扩展存储和计算。

2.3 数据存储在本小节中,我
们将讨论 Hologres 中数据存储的高级设计。

存储/计算解耦。 Hologres 采用计算层和存储层解耦的云原生设计。Hologres的所有数据文件和日志默认持久化在盘古,盘古是阿里云的高性能分布式文件系统。我们还支持开源分布式文件系统,例如 HDFS [3]。通过这种设计,计算层和存储层都可以根据工作负载和资源可用性独立扩展。

基于平板电脑的数据布局。在 Hologres 中,表和索引都被划分为细粒度的片。写请求被分解成许多小任务,每个任务处理对单个平板电脑的更新。相关表和索引的片进一步分组为分片,以提供有效的一致性保证。为了减少争用,我们使用无闩锁设计,每个平板电脑由一个写入器管理,但可以有任意数量的读取器。

我们可以为查询工作负载配置非常高的读取并行度,这隐藏了从远程存储读取所产生的延迟。
读/写分离。 Hologres分离读写路径,同时支持高并发读和高吞吐量写。平板电脑的作者使用类似 LSM 的应用程序

维护平板电脑图像的方法,其中记录已正确版本化。亚秒级读取可以看到新写入
潜伏。并发读取可以请求特定版本的平板电脑图像,因此不会被写入阻塞。

2.4 并发查询执行在本小节中,我们将讨论 Hologres
使用的调度机制的高级设计。

执行上下文。 Hologres 构建了一个调度框架,简称 HOS,它提供了一个称为执行上下文的用户空间线程来抽象系统线程。执行上下文是超轻量级的,可以以可忽略的成本创建和销毁。

HOS 在系统线程池之上协作调度执行上下文,上下文切换开销很小。执行上下文提供异步任务接口。 HOS将用户的读写查询划分为细粒度的工作单元,并将工作单元映射到执行上下文进行调度。这种设计还使 Hologres 能够对突然的工作负载爆发做出迅速反应。系统可以在运行时弹性放大和缩小。

可定制的调度策略。 HOS 将调度策略与基于执行上下文的调度机制解耦。 HOS 将来自不同查询的执行上下文分组到调度组中,每个调度组都有自己的资源共享。 HOS负责监控各个调度组的消耗份额,并在调度组之间强制执行资源隔离和公平。

2.5 系统概述

图 2 显示了 Hologres 的系统概览。前端节点 (FE) 接收客户端提交的查询并返回查询结果。对于每个查询,FE 节点中的查询优化器生成一个查询计划,该计划被并行化为片段实例的 DAG。协调器将查询计划中的片段实例分派给工作节点,每个工作节点将片段实例映射到工作单元（第 4.1 节）。工作节点是物理资源的单位,即CPU核心和内存。每个工作节点可以保存一个数据库的多个表组分片（第 3.2 节)的内存表。在工作节点中,工作单元作为 EC 池中的执行上下文执行（第 4.2 节）。根据预先配置的调度策略（第 4.5 节）,HOS 调度程序在系统线程（第 4.3 节)之上调度 EC 池。资源管理器管理表组分片在worker节点间的分布:一个worker节点中的资源在逻辑上被拆分成slot,每个slot只能分配给一个ta

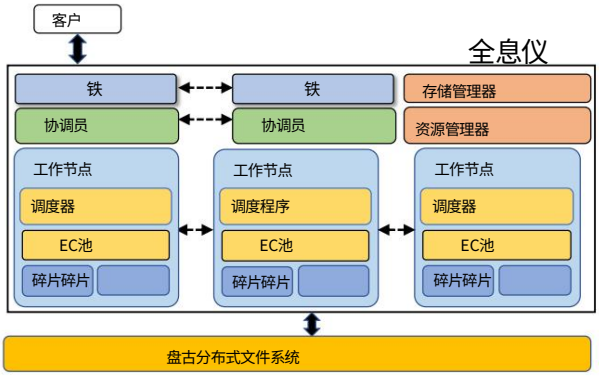


图 2:Hologres 的架构

分组分片。资源管理器还负责在 Hologres 集群中添加/删除工作节点。工作节点定期向资源管理器发送心跳。当集群中的工作节点发生故障或工作负载激增时,资源管理器会动态地将新的工作节点添加到集群中。

存储管理器维护一个表组分片目录（见第 3.1 节）,以及它们的元数据,如物理位置和键范围。每个协调器都会缓存这些元数据的本地副本,以方便查询请求的调度。

Hologres 允许执行单个查询来跨越 Hologres 和其他查询引擎（第 4.2.3 节）。例如,当片段实例需要访问未存储在 Hologres 中的数据时,协调器会将它们分发到存储所需数据的其他系统。我们设计并实现了一组用于查询处理的统一 API,以便在 Hologres 中执行的工作单元可以与其他执行引擎（如 PostgreSQL [12])进行通信。非 Hologres 执行引擎有自己独立于 Hologres 的查询处理和调度机制。

3. 储存

Hologres 支持为 HSAP 场景量身定制的混合行列存储布局。行存储针对低延迟点查找进行了优化,列存储旨在执行高吞吐量列扫描。在本节中,我们将介绍 Hologres 中混合存储的详细设计。我们首先介绍数据模型并定义一些初步概念。接下来我们介绍表组分片的内部结构,详细解释如何进行写和读。最后,我们介绍了行存储和列存储的布局,然后简要介绍了 Hologres 中的缓存机制。

3.1 数据模型

在 Hologres 中,每个表都有一个用户指定的集群键（如果未指定则为空）和一个唯一的行定位器。如果聚簇键是唯一的,则直接用作行定位符;否则,将唯一标识符附加到聚类键以构成行定位器,即 聚类键,唯一标识符 。

数据库的所有表都分组为表组。一个表组被分片成多个表组分片 (TGS),其中每个 TGS 包含每个表的基础数据分区和所有相关索引的分区。我们将基础数据分区和索引分区统一视为一个 tablet。 tablet 有两种存储格式:行 tablet 和列 tablet,分别针对点查找和顺序扫描进行了优化。基本数据和索引可以存储在行表、列表或两者中。平板电脑需要有一个唯一的密钥。因此,基本数据表的键是行定位符。而对于二级索引的 tablet,如果索引是唯一的,则索引列作为 tablet 的键;否则,通过将行定位器添加到索引列来定义键。例如,考虑一个具有单个表和两个二级索引的 TGS 一个唯一二级索引 (k1 → v1)和一个非唯一二级索引 (k2 → v2) 并且基础数据存储在行和列表中。如上所述,基本数据（行和列)片的键是 行定位器 ,唯一索引片的键是 k1 ,非唯一索引片的键是 k2 ,行定位器 。

我们观察到,数据库中的大多数写入访问一些密切相关的表,也写入单个表同时更新基础数据和相关索引。通过将表分组为表组,我们可以将对 TGS 中不同 tablets 的相关写入视为原子写入操作,并且只在文件系统中保留一个日志条目。该机制通过减少日志刷新次数来帮助提高写入效率。此外,对频繁连接的表进行分组有助于消除不必要的数据洗牌。

3.2 Table Group Shard TGS是

Hologres中数据管理的基本单位。一个TGS主要由一个WAL manager和属于这个TGS中的table shards的多个tablet组成,如图3所示。

Tablets被统一管理为一棵LSM树:每个tablet由worker节点内存中的一个内存表,以及一组持久化在分布式文件系统中的不可变分片文件组成。内存表被周期性地用作分片文件。 e分片

文件被组织成多个级别,Level0， Level1， ...,LevelN 。在 Level0 中,每个分片文件对应一个 ushed 内存表。从Level1开始,本层的所有记录按key排序划分到不同的shard文件中,这样同一层的不同shard文件的key范围是不重叠的。 Leveli+1可以容纳比 Leveli 多 K 倍的分片文件,每个分片文件的最大大小为 M。行和列片的更多细节分别在第 3.3 节和第 3.4 节中解释。

平板电脑还维护一个元数据文件,用于存储其分片文件的状态。元数据文件按照与 RocksDB [13] 类似的方法进行维护,并保存在文件系统中。

由于记录是版本化的,因此 TGS 中的读写完全分离。最重要的是,我们采用无锁方法,只允许 WAL 的单个写入者,但当前在 TGS 上允许任意数量的读者。由于HSAP场景对一致性的要求比HTAP弱,Hologres选择只支持原子写和read-your-writes读来实现读写的高吞吐和低延迟。接下来,我们详细解释读写是如何进行的。

3.2.1 写入 TGS

Hologres 支持两种类型的写入:单片写入和分布式批量写入。两种类型的写入都是原子的,即写入提交或回滚。单分片写入一次更新一个TGS,并且可以以极高的速率执行。另一方面,分布式批量写入用于将大量数据作为单个事务转储到多个 TGS 中,并且通常以低得多的频率执行。

单分片写入。如图 3 所示,在接收到单个分片摄取时,WAL 管理器 (1) 为写请求分配一个 LSN,它由时间戳和递增的序列号组成,并且 (2) 创建一个新的日志条目并保存日志进入文件系统。日志条目包含重播记录的写入的必要信息。写入在其日志条目完全持久化后提交。那么,(3)写请求中的操作

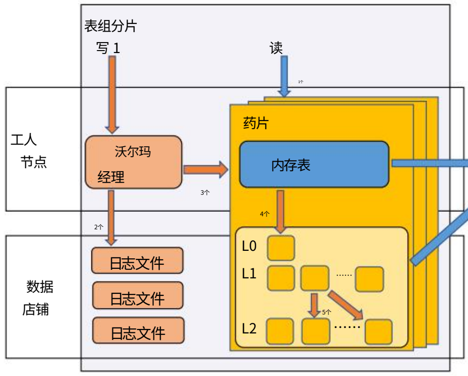


图 3:TGS 的内部结构

应用于相应平板电脑的内存表中,并对新的读取请求可见。值得注意的是,不同平板电脑上的更新可以并行进行（请参阅第 4.1 节）。一旦内存表已满,(4) 将其作为文件系统中的分片文件使用,并初始化一个新的。最后,(5) 分片文件在后台异步压缩。在压缩或内存表刷新结束时,平板电脑的元数据文件会相应更新。

分布式批量写入。我们采用两阶段提交机制来保证分布式批量写入的写入原子性。

接收到 bath 写请求的 FE 节点将锁定所涉及的 TGS 中所有访问过的 tablets。 en 每个 TGS：(1)为该批写入分配一个 LSN，(2)刷新所涉及的 tablet 的内存表,以及（3)在 gestion 的单分片过程中加载数据并将它们作为分片文件写入。请注意,步骤（3)可以通过构建多个内存表并将它们并行写入文件系统来进一步优化。一旦完成,每个 TGS 都会投票给 FE 节点。当 FE 节点收集到来自

参与的 TGS,它承认他们最终的提交或中止决定。在收到提交决定后,每个 TGS 都会保留一个日志,指示该批写入已提交;否则,将删除该批写入过程中所有新生成的文件。当两阶段提交完成后,相关平板电脑的锁就会被释放。

3.2.2 读取 TGS

Hologres 支持行和列平板电脑中的多版本读取。读请求的一致性级别是read-your-writes,即客户端总是会自己看到最新提交的写。每个读取请求都包含一个读取
时间戳,用于构造 is LSNread用于过滤掉不可见的记录到一个
LSNread 。此读取,即 LSN 大于LSNread的记录。

为了方便多版本读取,TGS为每个表维护一个LSNre f ,它存储了为该表中的tablet
维护的最旧版本的LSN和 LSNre f根据用户指定的保留期定期更新。(1)合并LSN等于
table flush.le compaction 过程中,对存储在 key. in memory
或小于LSNre f的记录； (2) LSNs 大于LSNre f的记录保持原样。

3.2.3 分布式TGS 管理在我们目前的实现中,一个TGS 的写

入者和所有读取者共同位于同一个工作节点上,以共享这个TGS 的内存表。如果工作节点正在经历工作负载突发,Hologres 支持将一些 TGS 迁移到过载的工作节点（参见第 4.4 节）。

我们正在研究一种解决方案,该解决方案维护与相应编写器远程的 TGS 只读副本,以进一步平衡

并发读取。我们计划支持两种类型的只读副本：(1)完全同步的副本维护 TGS 的内存表和元数据文件的最新副本,并且可以服务于所有读取请求； (2) 部分同步的副本仅维护元数据文件的最新副本,并且只能为写入文件系统的数据提供读取服务。可以根据读取版本将对 TGS 的读取分派到不同的副本。请注意,两个只读副本都不需要复制分片文件,如果需要,分片文件会从分布式文件系统加载。

如果 TGS 失败,存储管理器向资源管理器请求一个可用槽,同时向所有协调器广播 TGS 失败消息。恢复 TGS 时,我们从最新的 ushed LSN 重播 WAL 日志以重建其内存表。一旦所有的内存表都被完全重建,恢复就完成了。之后,存储管理器得到确认,然后向所有协调器广播包含新位置的 TGS 恢复消息。协调器暂时保留对失败的 TGS 的请求,直到它恢复。

3.3 排片

Row tablets 经过优化以支持给定键的有效点查找。图 4(a) 说明了行表的结构：我们将内存表维护为 Masstree [30],在其中我们按记录的键对记录进行排序。不同的是,分片文件是块式结构。分片文件由两种类型的块组成:数据块和索引块。分片文件中的记录按键排序。连续的记录被分组为一个数据块。为了帮助通过键查找记录,我们进一步跟踪每个数据块的起始键及其在分片文件中的集合作为索引块中的一对 key, block off set 。

为了支持多版本数据,存储在行tablet中的值被扩展为 value col s, del bit, LSN ： (1) value col s是非键列值； (2)del位表示这是一条删除记录； (3)LSN 为对应的写LSN。给定一个键,内存表和分片文件都可以有多个具有不同 LSN 的记录。

阅读行片。每个读入行的 tablet 都包含一个键和一个LSNread 。通过并行搜索tablet 的内存表和分片文件得到结果,仅搜索键范围与给定键重叠的分片文件。在搜索期间,
如果记录包含给定的键并且具有等于或小于 LSNread的
LSN,则该记录被标记为候选记录。 e个候选记录按照LSN的顺序合并为结果记录。如果结果记录中的 del 位等于 1,或者没有找到候选记录,则在LSNread的版本中不存在给定键的记录。否则,返回结果记录。

写在行片中。在行 tablets 中,插入或更新由键、列值和一个 LSNw rite组成。一次删除包含一个键、一个特殊的删除标记和一个LSNw ri te 。每个写入都被转换成一对键值对的行表。对于插入和更新,del 位设置为 0。对于删除,列字段为空,del 位设置为 1。首先将键值对附加到内存表中。一旦内存表已满,它就会作为 Level0 中的分片文件被写入文件系统。如果 Leveli已满,则可能会进一步触发从 Leveli到 Leveli+1 的级联压缩。

3.4 柱片

列片旨在促进列扫描。如图 4(b) 所示,与行 tablets 不同,列 tablet 由两个组件组成,一个列 LSM 树和一个删除映射。

存储在列 LSM 树中的值以格式扩展 of: value col s, LSN ,其中value col s为非键列,LSN为对应的写LSN。在一列 LSM 树中,

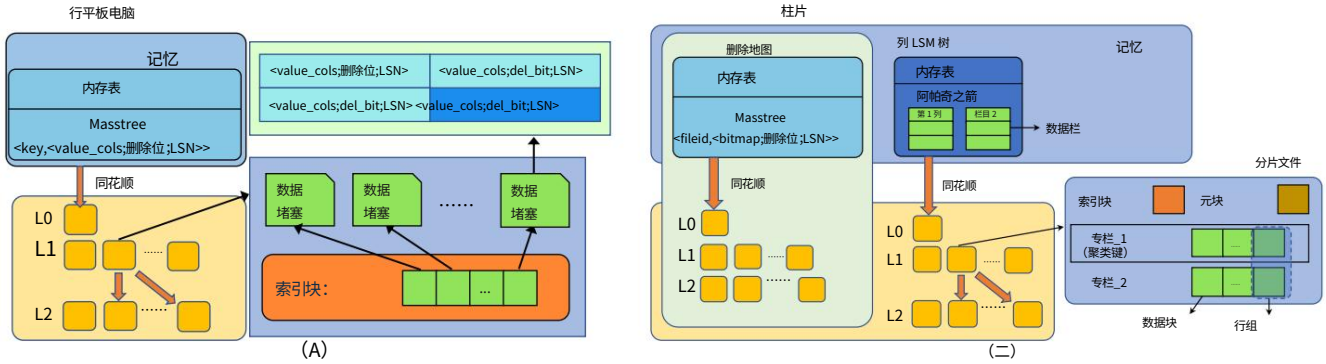


图 4:(一) e 行 tablet 的结构,以及 (b) 列 tablet 的结构

内存表以 Apache Arrow [2] 的格式存储记录。
记录按到达顺序连续添加到内存表中。在分片文件中,记录按键排序并按逻辑分成行组。行组中的每一列都存储为单独的数据块。同一列的数据块连续存储在分片文件中,以方便顺序扫描。我们在元块中维护每一列的元数据和整个分片文件,以加速大规模数据检索。元块存储:(1)对于每一列,数据块的集合,每个数据块的取值范围和编码方案,以及 (2)对于分片文件,压缩方案,总行数,LSN和关键范围。为了根据给定的键快速定位行,我们将行组的排序后的第一个键存储在索引块中。

e delete map是一个row tablet,key是column LSM tree中的一个 shard le(内存表作为一个特殊的shard le)的ID,value是一个位图,表示哪些记录是新删除的分片文件中相应的 LSN。在 delete map 的帮助下,column tablets 可以大规模并行化顺序扫描,如下所述。

读取列片。对列 tablet 的读取操作包括目标列和LSNread 。读取结果是通过扫描内存表和所有的分片文件得到的。

在扫描分片文件之前,我们将其 LSN 范围与LSNread进行比较: (1) 如果其最小 LSN 大于LSNread ,则跳过该文件; (2) 如果其最大LSN小于等于LSNread,则整个分片文件在读取版本中可见; (3) 否则,只有该文件中记录的子集在读取版本中可见。在里面

第三种情况,我们扫描该文件的 LSN 列并生成一个 LSN 位图,指示哪些行在读取版本中可见。为了过滤掉分片文件中已删除的行,我们在删除映射中执行读取(如第 3.3 节所述),使用分片文件的 ID 作为版本LSNread的键,其中合并操作合并所有候选位图.获得的位图与LSN位图相交,并与目标数据块连接,以过滤掉读取版本中已删除和不可见的行。请注意,分片文件都不可以独立读取,而无需与其他级别的分片文件合并,因为 delete map 可以有效地告诉分片文件中直到 LSNread 的所有已删除行。

写入 Column Tablets。在列表中,插入操作由一个键、一组列值和一个 LSNw rite组成。 Adelete操作指定了要删除的行的key,通过它我们可以快速的找出包含该行的文件ID以及它在该文件中的行号。我们在 delete map 中执行 insert at version LSNw ri te ,其中 key 是文件 ID,value 是行号

删除的行。更新操作的实现方式是先删除再插入。对列 LSM 树的插入和删除 map 可以触发内存表 ush 和 shard le 压缩。

3.5 分层缓存Hologres采用分层缓存机

制来降低I/O和计算成本。一共有三层缓存,分别是本地磁盘缓存、块缓存和行缓存。

每个 tablet 对应一组存储在分布式文件系统中的分片文件。本地磁盘缓存用于在本地磁盘 (SSD) 中缓存分片文件,以减少文件系统中昂贵的 I/O 操作的频率。在 SSD 缓存之上,内存中的块缓存用于存储最近从分片文件中读取的块。由于服务和分析工作负载具有非常不同的数据访问模式,我们在物理上隔离了行和列片的块缓存。在块缓存之上,我们进一步维护一个内存中的行缓存,以将最近点查找的合并结果存储在行片中。

4. 查询处理和调度在本节中,我们介绍了 Hologres 的并行查询执行范例和 HOS 调度框架。

4.1 高度并行的查询执行

图 5 说明了 Hologres 中的查询处理工作流。
在接收到查询时,FE 节点中的查询优化器生成一个表示为 DAG 的查询计划,并将 DAG 在 shufe 边界划分为片段。共有三种类型的片段:读/写/查询片段。读/写片段包含访问表的读/写运算符,而查询片段仅包含计算运算符。化为多个分片实例,例如,每个读/写分片实例处理一个TGS。 e FE 节点将查询计划转发给协调器。然后,协调器将片段实例分派给工作节点。

读/写片段实例总是被分派到托管访问的 TGS 的工作节点。查询片段实例可以在任何工作节点中执行,并在调度时考虑工作节点的现有工作负载以实现负载平衡。位置和工作负载信息分别与存储管理器和资源管理器同步。

在工作节点中,片段实例被映射到工作单元 (WU),这是 Hologres 中查询执行的基本单元。
WU 可以在运行时动态生成 WU。 e映射描述如下:

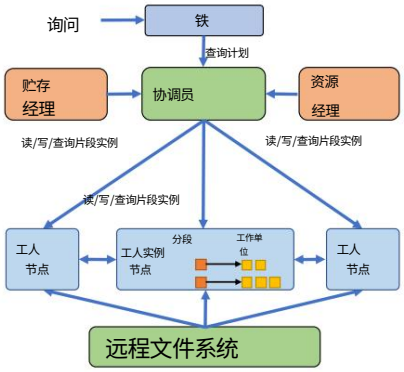


图 5:查询并行化的工作流程

· 读取片段实例最初映射到读取同步WU,它从元数据文件中获取片的当前版本,包括内存表的只读快照和分片文件列表。接下来,读取同步 WU 生成多个读取应用 WU 以并行读取内存表和分片文件,并对读取数据执行下游运算符。该机制利用高运算符内部并行性来更好地利用网络和 I/O 带宽。 · 写片段实例将所有非写操作符映射到一个查询WU,然后是一个写同步WU,将写入数据的日志条目持久化在WAL 中。然后写同步 WU 产生多个写应用 WU,每个 WU 并行更新一个 tablet。

· 查询片段实例映射到查询WU。

4.2 执行上下文作为一个 HSAP 服务,

Hologres 被设计用来同时执行不同用户提交的多个查询。并发查询的 WU 之间的上下文切换开销可能成为并发的瓶颈。为了解决这个问题,Hologres 提出了一个用户空间线程,称为执行上下文(EC),作为 WU 的资源抽象。与抢占式调度的线程不同,EC 是协同调度的,无需使用任何系统调用或同步原语。我们在 EC 之间切换的成本几乎可以忽略不计。 HOS以EC为基本调度单位。计算资源以EC的粒度进行分配,进一步调度其内部任务。 EC 将在分配给它的线程上执行。

4.2.1 EC 矿池

在工作节点中,我们将 EC 分组到不同的池中以允许隔离和优先级排序。 EC池可以分为三种类型:数据绑定EC池、查询EC池和后台EC池。 · Data-bound EC pool有两种类型的EC:WAL EC和tablet EC。在一个 TGS 中,有一个 WAL EC 和多个 tablets EC,每个 tablet 一个。 e WAL EC 执行写同步 WU,而 tablet EC 在相应的 tablet 上执行写应用 WU 和读同步 WU。 e WAL/tablet EC 以单线程的方式处理 WU,这消除了并发 WU 之间同步的必要性。 · Iequery EC pool中,每个query WU或read-apply WU映射到一个query EC。 · 在后台EC 池中,EC 用于从数据绑定的EC 中加载昂贵的任务,例如 [illegible] 和 [illegible]。包 compaction等。通过这种设计,数据绑定的EC主要用于WAL上的操作并写入内存表,因此sys tem 可以在没有锁定开销的情况下实现非常高的写入吞吐量。

为了限制后台 EC 的资源消耗,我们将后台 EC 与不同线程池中的数据绑定和查询 EC 物理隔离,并在优先级较低的线程池中执行后台 EC。

4.2.2 执行上下文的内部结构

接下来介绍EC的内部结构。

任务队列。 EC中有两个任务队列:(1)无锁内部队列,存储EC自己提交的任务,(2)线程安全提交队列,存储其他EC提交的任务。

一旦调度,提交队列中的任务将重新定位到内部队列,以方便无锁调度。内部队列中的任务按 FIFO 顺序安排。

状态。在 EC 的生命周期中,它会在三种状态之间切换:可运行、阻塞和暂停。被挂起意味着 EC 无法被调度,因为它的任务队列是空的。向 EC 提交任务会将其状态切换为可运行,这表明 EC 可以被调度。如果一个 EC 中的所有任务都被阻塞,例如,由于 I/O 停顿,则 EC 切换出去并且它的状态被设置为阻塞。一旦接收到新任务或阻塞的任务返回,阻塞的 EC 将再次变为可运行状态。 EC 可以在外部取消或加入。

取消 EC 将使未完成的任务失败并暂停它。 EC加入后,无法接收新任务,当前任务完成后自行挂起。 EC 在系统线程池之上协同调度,因此上下文切换的开销几乎可以忽略不计。

4.2.3 联合查询执行

Hologres 支持联合查询执行以与来自开源世界(例如,Hive [7] 和 HBase [6])的丰富服务进行交互。我们允许单个查询跨越 Hologres 和其他在不同进程中物理隔离的查询系统。在查询编译期间,要在不同系统中执行的运算符被编译为单独的片段,然后由 Hologres 中的协调器将其分派到它们的目标系统。

其他与Hologres交互的系统被抽象为特殊的stub WU,每一个都映射到一个EC中,在Hologres中统一管理。存根 WU 处理 WU 在 Hologres 中提交的拉取请求。除了功能方面的考虑,例如访问其他系统中的数据,出于系统安全原因,此抽象还用作隔离沙箱。例如,用户可以提交带有可能不安全的用户定义函数的查询。 Hologres 将这些功能的执行传播到 PostgreSQL 进程,这些进程在与 Hologres 中的其他用户物理隔离的上下文中执行它们。

4.3 调度机制

在本小节中,我们将详细介绍查询的 WUs
计划生成查询输出。
基于异步拉取的查询执行。查询在 Hologres 中遵循基于拉的范例异步执行。在查询计划中,叶片段使用外部输入,即分片文件,而汇片段产生查询输出。基于拉取的查询执行从协调器开始,协调器将拉取请求发送到接收器片段的 WU。在处理拉取请求时,接收者 WU 进一步向其依赖的 WU 发送拉取请求。读算子的WU,即column scan,一旦收到pull request,就会从对应的 shard le中读取一批数据,返回结果格式为 record batch,EOS ,其中 record batch为一批结果记录和 EOS 是一个

指示生产者 WU 是否已完成其工作的布尔值。在

接收到先前拉取请求的结果,协调器通过检查返回的 EOS 来确定查询是否已完成。
如果查询尚未完成,它会发出另一轮拉取请求。依赖于多个上游 WU 的 WU 需要同时从多个输入中拉取,以提高查询执行的并行性和计算/网络资源的利用率。 Hologres 通过发送多个异步拉取请求来支持并发拉取。与需要多线程协作的传统并发模型相比,这种方法更加自然和高效。

Intra-worker pull request 被实现为一个函数调用,它将一个 pull 任务插入到托管接收者 WU 的 EC 的任务队列中。
工作人员间拉取请求被封装为源工作人员节点和目标工作人员节点之间的 RPC 调用。 RPC 调用包含 ID
接收者WU的消息,目标工作节点根据该消息将拉取任务插入到相应EC的任务队列中。
背压。基于上述范式,我们实现了基于拉取的背压机制,以防止 WU 因收到过多的拉取请求而不堪重负。首先,我们限制 WU 一次可以发出的并发拉取请求的数量。其次,在为多个下游 WU 生成输出的 WU 中,处理拉取请求可能会导致为多个下游 WU 生成新的输出。这些输出被缓存起来,等待来自相应 WU 的拉取请求。为了防止 WU 中的输出缓冲区增长过快,下游比其他 WU 更频繁拉取的 WU 将暂时减慢向该 WU 发送新拉取请求的速度。

预取。 HOS 支持为未来的拉取请求预取结果以减少查询延迟。在这种情况下,一组预取任务被排队。预取任务的结果在预取缓冲区中排队。在处理拉取请求时,可以立即返回预取缓冲区中的结果,并创建一个新的预取任务。

4.4 负载均衡

Hologres 中的负载平衡机制有两个方面：(1)跨工作节点迁移 TGS，以及（2)在工作线程之间重新分配 EC。

TGS 的迁移:在我们当前的实现中,读/写片段实例总是被分派到托管 TGS 的工作节点。如果一个 TGS 成为热点,或者一个 worker 节点超载,Hologres 支持将一些 TGS 从超载的 worker 节点迁移到其他可用资源更多的节点。到

迁移一个 TGS,我们在存储管理器中将 TGS 标记为失败,然后按照标准 TGS 恢复程序（参见第 3.2.3 节)在新的工作节点中恢复它。正如第 3.2.3 节中所讨论的,我们正在为 TGS 实现只读副本,这使得读取片段实例能够平衡到位于多个工作节点中的 TGS 的只读副本。

EC 的重新分配:在一个工作节点中,HOS 在每个 EC 池内的线程之间重新分配 EC,以平衡工作负载。 HOS 执行三种类型的重新分配： (1) 新创建的 EC 始终分配给线程池中 EC 数量最少的线程； (2) HOS周期性地在线程之间重新分配EC,使得线程之间EC数量的差异最小化； (3) HOS 还支持工作负载窃取。一旦线程没有 EC 可以调度,它就会从同一线程池中具有最大 EC 数量的线程中“窃取”一个。 EC 的重新分配仅在其未运行任何任务时进行。

4.5 调度策略

HOS 的一个关键挑战是在多租户场景中保证查询级别的 SLO,例如,大规模的分析查询不应该阻止对延迟敏感的服务查询。为了解决这个问题,我们建议将调度组 (SG) 作为工作节点中数据绑定和查询 EC 的虚拟资源抽象。更具体地说,HOS 为每个 SG 分配一个份额,其价值与分配给该 SG 的资源量成正比。 SG 的资源在其 EC 之间进一步拆分,EC 只能使用分配给自己 SG 的资源。

为了将摄取工作负载与查询工作负载分开,我们将数据绑定 EC 和查询 EC 隔离到不同的 SG 中。
数据绑定 EC 处理需要所有查询共享的同步的关键操作,并且主要致力于摄取工作负载（读取同步 WU 通常非常轻量级） ,我们将所有数据绑定 EC 分组在一个数据绑定中神光。相反,我们将不同查询的查询 EC 放入单独的查询 SG 中。我们为数据绑定 SG 分配了足够大的份额来处理所有摄取工作负载。默认情况下,所有查询 SG 都分配有相同的份额以强制公平资源分配。 SG 股票是可转让的。

给定一个 SG,在一个时间间隔内分配给它的 EC 的 CPU 时间量受两个因素影响:(1) 它的份额,(2) 它在最后一个时间间隔内占用的 CPU 时间量。 SG 的份额根据其 EC 在上一个时间间隔内的状态进行调整。只有可运行的 EC 才能被调度。将 ECi 的份额表示为 EC sharei ,我们计算 EC 份额 av gi来表示一个时间间隔内 ECi 的实际份额,而 SGi 的实际份额是其 EC 份额的总和：

$$\text{gi的 EC 份额} = \text{EC sharei}\ \frac{\Delta\text{特伦}}{\Delta\text{Trun} + \Delta\text{Ts pd} + \Delta\text{Tbl k}}$$

$$\text{SG 份额gi} = \sum_{j=1}^{\text{否}} \text{g j的 EC 份额}$$

runnable、suspended 和 blocking 状态。

ΔTbl k表示 EC i 处于 Tbl k 的时间间隔， ΔTrun 的时间间隔， ΔTs pd表示

对于SGj中的ECi ,我们维护一个反映其历史资源分配状态的虚拟运行时。将最后一个时间间隔内分配给ECi 的 CPU 时间表示为 ΔCPU timei ,最后一个时间间隔内 ECi的虚拟运行时增量Δvruntimei计算如下：

$$\text{EC vsharei} = \frac{\text{EC 份额* SG 份额}}{\text{g j的 SG 份额}};$$

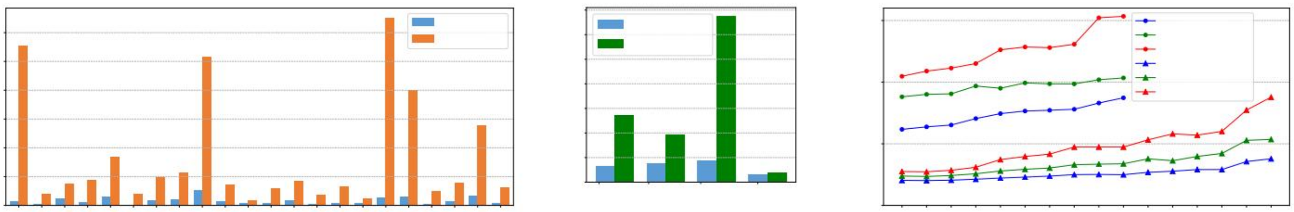
$$\Delta\text{vruntimei} = \frac{\Delta\text{CPU timei}}{\text{EC vsharei}}$$

在选择下一个要调度的 EC 时,线程调度程序总是选择具有最小 vruntime 的那个。

5. 实验

在本节中,我们进行实验来评估 Hologres 的性能。我们首先通过将 Hologres 与最先进的 OLAP 系统和服务系统（第 5.2 节)进行比较,分别研究 Hologres 在 OLAP 工作负载和服务工作负载上的性能。我们表明,即使与这些专用系统相比,Hologres 也具有卓越的性能。然后,我们展示了 Hologres 处理混合服务和分析处理工作负载的各个性能方面的实验结果： · 我们单独研究 Hologres 的设计在处理分析工作负载或单独服务工作负载时的并行化和扩展能力。我们尝试增加工作负载和计算资源（第 5.3 节）。

Machine Translated by Google



(A) (二) (C)

图 6:(a) Hologres和 Greenplum 在 TPC-H 基准测试中的分析查询延迟。 (b) 对Hologres 的性能关键特征的影响的细分研究。 (c) 在 YCSB 基准测试中提供Hologres和 HBase的查询吞吐量/延迟。

· 我们研究了HOS 性能的两个方面：(1)HOS 在处理混合服务和分析工作负载时是否可以执行资源隔离和公平调度； (2) 居屋能否对突如其来的工作量爆发做出迅速反应（第 5.4 节）。 · 我们研究了Hologres 存储设计的效率:(1) 高速数据摄取对读取性能的影响,以及(2) 多索引维护下的写入延迟和写入吞吐量（第5.5 节）。

5.1 实验设置

工作量。我们使用 TPC-H benchmark [15] (1TB) 模拟典型的分析工作负载,使用 YCSB benchmark [17] 模拟典型的服务工作负载,其中包含一个包含 1 亿条记录的表,每条记录有 11 个字段, 每个字段为 100 字节。

当对相同数据的混合服务和分析工作负载进行测试时（第 5.4.1 节）,我们使用 TPC-H 数据集并将 TPC-H 查询与 lineitem 表上的合成服务查询（点查找)混合。为了研究混合读/写请求,我们模拟了阿里巴巴的一个生产工作负载,简称 PW。 PW 有一个包含 6 亿行的购物车表,每秒更新 106 次。每条记录有 16 个字段,一条记录的大小为 500 字节。我们在实验期间重播更新。

系统配置。我们使用一个由 8 台物理机组成的集群,每台物理机有 24 个虚拟内核（通过超线程）、192GB 内存和 6T SSD。除非明确指定,否则我们在 TPC-H 和 YCSB 基准测试的实验中使用此默认设置。

据我们所知,目前还没有 HSAP 系统。
为了研究 Hologres 的性能,我们将其与用于分析处理和服务的专门系统进行了比较。对于分析处理,我们与 Green plum 6.0.0 [5] 进行了比较;对于服务,我们与 HBase 2.2.4 [6] 进行了比较。各系统详细配置说明如下： (1) Greenplum集群共有48个Segment,平均分配在8台物理机上。每个段分配 4 是 Greenplum 官方文档 [11] 的推荐设置,考虑到查询执行期间的查询内（查询中的多个计划片段)和查询间并发性。 Greenplum使用本地磁盘存储数据文件,数据以列的形式存储。 (2) e HBase集群有8台region 核心。 server,每台region server部署在一台物理机上。 HBase 将数据文件存储在 HDFS 中,使用本地磁盘进行配置。 HBase 以行格式存储数据。 (3) e Hologres集群有8个worker节点,每个worker节点独占一台物理机。为了与 Greenplum 和 HBase 进行公平比较,Hologres 也配置为使用本地磁盘。数据在 Hologres 中以行和列格式存储。

PW 工作负载的实验是在具有 1, 985 个内核和 7, 785 GB 内存的云环境中进行的。我们用盘古阿里云远程分布式文件系统来存储数据。购物车表的基础数据以列的形式存储。 is 表还有一个以行格式存储的索引。

实验方法。所有实验都以 20 分钟的预热期开始。对于每个报告的数据点,我们重复实验 5 次并报告平均值。

在实验中,我们使用标准的 YCSB 客户端对 YCSB 数据进行所有实验。对于 TPC-H 和 PW 数据的实验,我们实现了一个类似于 YCSB 的客户端。更具体地说,客户端连接异步提交查询请求。我们可以配置单个连接可以提交的最大并发查询数（表示为 W）。多个客户端连接同时提交查询请求。除非明确说明,否则我们在整个实验过程中设置 W = 100。

5.2 整体系统性能

在这组实验中,我们分别研究了 Hologres 在分析工作负载和服务工作负载上的性能,并与专门的 OLAP 和服务系统进行了比较。

分析工作负载。在这个实验中,我们使用 TPC-H 数据集比较 Hologres 和 Greenplum。为了准确测量查询延迟,我们使用单个客户端并将 W 设置为 1。图 6(a) 报告了 22 个查询的平均端到端延迟。

如图所示,Hologres 在所有 TPC-H 查询上都优于 Greenplum：Hologres 的查询延迟平均仅为 Greenplum 的 9.8%。对于第一季度,Hologres 比 Greenplum 快 42 倍。原因如下：(1)HOS 为查询执行提供了灵活的高运算符内部并行性。读取并行度可以达到表中分片文件的数量。灵活性允许 Hologres 对所有查询具有正确的并行性。另一方面,Greenplum的并行度是由段数决定的,不能为所有查询（例如,Q1)充分利用CPU。 (2) 列片布局支持高效编码和索引。如果查询具有可以下推到数据扫描的过滤器（例如,Q13）,这些存储布局优化可以大大提高性能。 (3) Hologres 采用高效向量化执行,支持AVX-512 指令集[8],可以进一步加速受益于向量化执行的查询[1] （例如Q15）。 (4) Hologres 动态过滤器之类的运行时优化（例如,Q20）加速连接的优化共同有助于提高 Hologres 中分析处理的性能。

为了验证上述性能关键技术（1）-（4)的效果,我们使用TPC-H bench进行击穿实验

1我们主要在以下方面使用 AVX-512:(1) 算术表达式（例如,加法、减法、乘法、除法、等于、不等于）； (2) 过滤； (3)位图操作； (4)哈希值计算； (5) 批量复制。

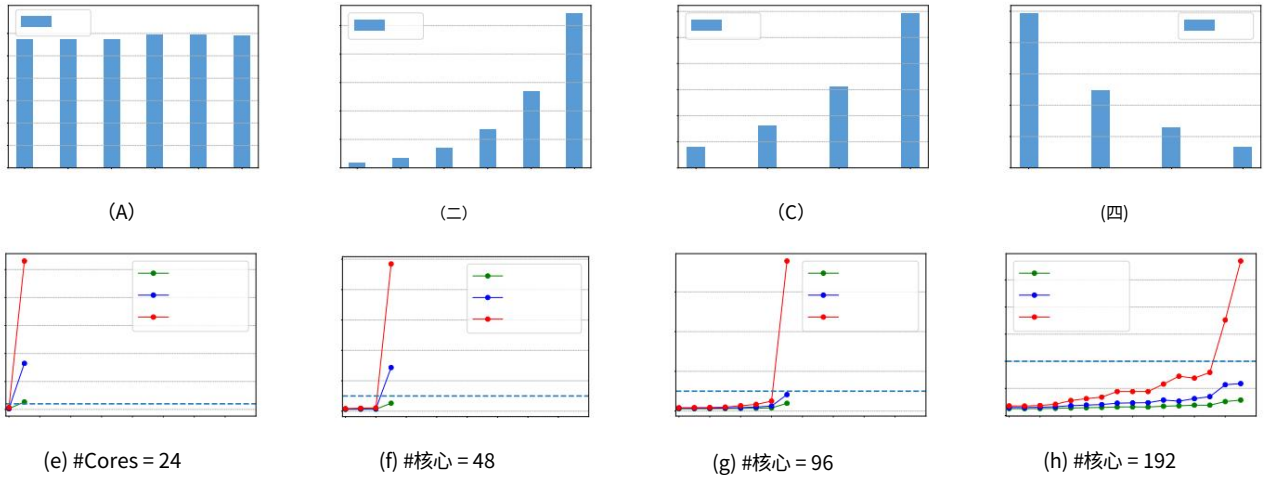


图 7:核心 e 分析工作负载在 (a)(b) 不同数量的并发查询和 (c)(d) 不同情况下的吞吐量和延迟数。 (e)(f)(g)(h) e 不同内核数量下服务工作负载的吞吐量/延迟曲线。

标记。对于每种技术,我们选择一个有代表性的查询,并将 Hologres 中的查询延迟与打开和关闭的技术进行比较。具体来说:对于 (1),我们使用 Q1,并且为了将特征 o 我们将并行度设置为 Greenplum 中的段数。对于 (2),我们使用 Q13,并关闭字典编码功能。对于 (3),我们使用 Q15,并使用没有 AVX-512 的构建来关闭该功能。对于 (4),我们使用 Q20,并关闭动态过滤器优化功能。结果如图 6(b) 所示,其中 (1) (2) (3) (4) 分别表示为 Q1-DOP、Q13-Storage、Q15-AVX512 和 Q20-Plan。正如我们所见,这些技术将性能提升了 1.2 倍到 7.6 倍。

我们还通过使用 TPC-H 基准测试 (100GB) 将 Hologres 与 Vectorwise (Actian Vector 5.1 [1]) 进行比较,对单机性能进行了微基准测试。实验在单机 32 核 128GB 内存上进行。运行所有 22 个 TPC-H 查询需要 Hologres 84 秒,而 Vectorwise 需要 27 秒。结果表明 Hologres 仍有性能提升的空间。但是,Vectorwise 中的优化技术适用于 Hologres,在未来的工作中,我们会将它们集成到 Hologres 中。

服务工作负载。在此实验中,我们使用 YCSB 基准在吞吐量和延迟方面比较 Hologres 和 HBase。我们逐渐将查询吞吐量从 100K QPS 提高到 1600K QPS。对于每个吞吐量,我们在图 6(c) 中报告了两个系统查询延迟的相应平均值、95% 和 99% 百分位数。我们将 99% 延迟 SLO 设置为 100ms,并且不报告超过 SLO 的数据点。

首先要注意的是,HBase 不会扩展到大于 1000K QPS 的吞吐量,因为查询延迟超过了延迟 SLO。而即使在 1600K QPS 下,Hologres 的 99% 延迟仍在 6ms 以下,95% 延迟甚至低于 1.18ms。对于 1000K QPS 以下的吞吐量,Hologres 的平均、95% 和 99% 延迟平均分别比 HBase 好 10 倍、22 倍和 57 倍。这是因为 HBase 中基于线程的并发模型在面对高并发服务负载时会产生很大的上下文切换开销。相反,Hologres 中的执行上下文非常轻量级,可以很少的上下文切换开销进行协作调度。这种设计也使得调度得到很好的控制,保证了查询延迟的稳定性。例如,在吞吐量 = 800K QPS 时,HBase 的 99% 延迟是比其平均延迟高 10.5 倍;相反,在 Hologres 中这个差异仅为 1.8X。

上述实验清楚地表明,通过新的存储和调度设计,Hologres 始终优于最先进的专业分析系统和服务系统。

5.3 Hologres 的并行性和可扩展性

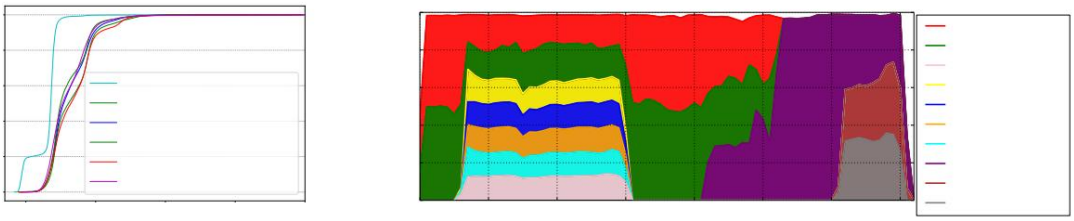
接下来,我们分别研究 Hologres 在处理分析工作负载和服务工作负载时的并行性和可扩展性。

分析工作负载。对于分析工作负载,我们研究了两个方面:(1) Hologres 并行化分析查询的能力如何,以及 (2) Hologres 在更多计算资源下的可扩展性如何。我们选择 TPC-H Q6 作为对大量数据进行顺序扫描的代表性 OLAP 查询。

在第一个实验中,我们使用默认集群设置(8 个工作节点,每个节点有 24 个核心)。我们使用单个客户端提交查询,但逐渐将并发查询的数量 W 从 1 增加到 32。结果如图 7(a) 和 7(b) 所示。正如我们所看到的,随着并发查询数量的增加,吞吐量保持稳定。结果清楚地表明,即使是单个查询,所有[illegible]共享,延迟呈线性增加。

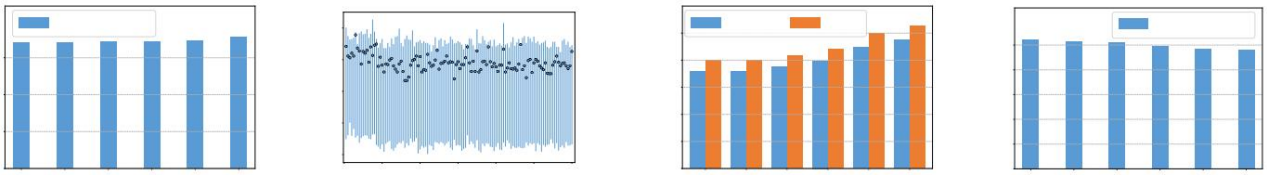
在第二个实验中,我们 x 并发查询数 W = 8,但横向扩展了资源。具体来说,我们使用 8 个工作节点,并逐渐将每个工作节点的核心数从 3 个增加到 24 个。结果如图 7(c) 和 7(d) 所示,表明吞吐量线性增加,同时查询延迟随着核心数量的增加而减少。同样,这表明 Hologres 的高算子内并行机制可以自动使硬件并行饱和。

服务工作负载。在这组实验中,我们通过改变资源量来评估 Hologres 在服务工作负载上的吞吐量和延迟。同样,我们使用 8 个工作节点,并逐渐将每个工作节点中的核心数量从 3 个增加到 24 个。对于每个集群设置,我们都会增加吞吐量,直到 99% 的延迟超过 2 毫秒的延迟 SLO。我们使用 8 个客户端连续提交查询。我们报告每个吞吐量的相应查询延迟。结果分别显示在图 7(e)-7(h) 中。我们从这些数字中得到两个观察结果。一、最大通过



(A) (二)

图 8:(a) 混合工作负载:不同后台分析工作负载下前台服务查询的延迟 CDF。 (b) 分配给并发查询的 CPU 时间HOS的动态份额。



(A) (二) (C) (四)

图 9:(a) 不同后台写入工作负载下读取查询的前台延迟。 (b) 在 PW 工作负载中随时间推移的每个 TGS 写入吞吐量的分布。 (c)(d)e 维护不同数量的二级索引时的写入延迟/吞吐量。

放 Hologres 可以实现随着 core 数量的增加线性增加。例如,我们可以看到 #core=192 时的最大吞吐量是#core=24 时最大吞吐量的 8 倍。

其次,在系统达到最大吞吐量之前,查询延迟保持在一个稳定的水平。以 #core=192为例,随着吞吐量的增长,平均、95%和99%的延迟增长非常缓慢。这是因为 Hologres 可以完全控制用户空间中执行上下文的调度。

5.4 HOS 的性能在本小节中,我们研究了

HOS 的两个性能方面:(1)混合服务和分析工作负载下的资源隔离,以及 (2)突发工作负载下的调度弹性。

5.4.1 混合工作负载下的资源隔离

HSAP 服务中的一个关键调度要求是延迟敏感的服务查询不受资源消耗分析查询的影响。为了研究这一点,我们生成了一个混合服务/分析工作负载,它有两个部分:(1)背景:我们在后台不断提交分析查询(TPC-H Q6 具有不同的谓词)。我们通过将并发查询的数量 W 从 0 增加到 16 来改变后台工作负载。 (2)前台:我们在前台提交服务查询并测量查询延迟。为了准确测试延迟,我们设置并发查询数 W = 1。对于后台工作负载的每个设置,我们收集 50K 数据点,并绘制它们的 CDF。

图 8(a) 显示了结果。我们可以看到:通过将后台查询的数量从 0 增加到 1,服务查询的延迟有一个小的增量;但进一步增加后台工作负载(从 1 到 16)不会带来任何增量。

它清楚地表明分配给不同查询的资源被 HOS 很好地隔离,因为不同查询的执行上下文被分组到单独的调度组中。因此,分析查询和服务查询可以在同一个系统中共存,而它们的延迟 SLO 仍然可以得到满足。

5.4.2 突发事件下的调度弹性

在这个实验中,我们展示了 HOS 对突发工作负载的反应有多好。通过在时间 0 同时发出 Q1 和 Q2 开始实验。在时间 5,我们发出 5 个新查询(Q3-Q7)。 Q3-Q7大致在30时结束。在40时查询Q8进入系统。 Q1 和 Q2 大致在时间 50 结束。最后,在时间 60,我们提交 Q9 和 Q10,并让 Q8-Q10 运行完成。

所有查询都分配有相同的优先级。图 8(b) 显示了时间轴上每个查询使用的 CPU 比例。

请注意,在时间 5,HOS 快速调整资源分配,以便所有 7 个查询具有相同的 CPU 份额。在时间 30,在 Q3-Q7 执行完毕后,HOS 立即调整调度并将 CPU 平均分配给仍在运行的 Q1 和 Q2。在时间 40、50 和 60 可以观察到类似的行为。这是实验的亮点,HOS 可以根据系统中的实时并发工作负载动态及时地调整其调度行为,始终保证公平共享。

5.5 Hologres 存储性能

在这组实验中,我们评估了读写分离对查询延迟的影响,并研究了 Hologres 中索引维护下的写入性能。

5.5.1 分离读/写操作

为了研究写入对查询延迟的影响,我们在由两部分组成的 PW 工作负载上生成混合读/写工作负载:(1) 背景:我们在 PW 中重放元组写入以模拟 20 分钟的后台工作负载。我们通过将写入客户端的数量从 1 增加到 32 来改变写入吞吐量。e 写入在 TGS 之间均匀分布。例如,对于写入客户端数量为 32 的情况,我们每 10 秒对写入吞吐量进行一次采样,并在图 9(b) 中报告所的平均/最小/最大写入吞吐量。 (2)前台:我们使用 16 个客户端提交 OLAP 查询作为前台工作负载。为了准确测量查询延迟,每个客户端都将其 W 设置为 1。我们重新

在图 9(a) 中移植每个吞吐量设置下的平均查询延迟。如图所示,尽管写入吞吐量有所增加,但 OLAP 查询的延迟是稳定的。结果证明高吞吐量写入对查询延迟几乎没有影响。这是因为 Hologres 中的读写分离。版本化的平板电脑保证读取不会被写入阻塞。

5.5.2 写入性能接下来,我们使用

YCSB 基准测试在索引维护下研究 Hologres 的写入性能,其中我们为 YCSB 表创建了多个二级索引。我们将二级索引的数量从 0 更改为 10。对于每个设置,我们将系统推至其最大写入吞吐量并报告写入延迟的 95% 和 99% 百分位。

如图9(c)和9(d)所示,随着索引数量的增加,写入延迟和写入吞吐量保持相当稳定,变化很小。与没有二级索引的情况相比,维护 10 个二级索引只会导致写入延迟增加 25%,写入吞吐量减少 8%。结果表明,Hologres 中的索引维护非常高效,对写入性能的影响非常有限。主要原因有三方面:(1) Hologres 通过在 TGS 中的所有索引片之间共享 WAL 来优化写入性能。因此,添加更多索引不会导致额外的日志更新。 (2) 对于每次写入 TGS,每个索引都由一个单独的写入应用 WU 并行更新。 (3) Hologres 通过加载到后台 EC 池来积极并行化内存表刷新和文件压缩等操作。有了足够的计算资源,这种设计就消除了性能瓶颈。

6. 相关工作OLTP 和 OLAP 系统。

OLTP 系统 [10、12、35] 采用行存储来支持快速事务,这些事务经常对少量行执行点查找。 OLAP 系统 [34、37、14、27、24、22、36] 利用列存储来实现高效的列扫描,这是分析查询中的典型数据访问模式。不同于上面的OLTP/OLAP系统,Hologres支持混合行列存储。表可以以行和列两种存储格式存储,以有效支持 HSAP 工作负载所需的点查找和列扫描。

像 Greenplum [5] 这样的 MPP 数据库通常将数据分成大段,并将数据段与计算节点放在一起。
在扩展系统时,MPP 数据库通常需要重新分片数据。相反,Hologres 管理 TGS 中的数据,TGS 是一个更小的单元段。 Hologres 将 TGS 动态映射到工作节点,并且可以在工作节点之间灵活地迁移而无需重新分片数据。此外,工作节点只需要将托管 TGS 的内存表保存在内存中,但根据需要从远程文件系统中获取 TGS 的分片文件。在多租户调度方面,[5]在不同的进程中处理不同的请求,并依赖操作系统来调度并发查询,很容易对查询并发设置硬限制。相反,Hologres 在一组用户空间线程上复用并发查询,从而实现更好的查询并发性。 [31, 29] 研究分析工作负载的高度并行查询处理机制。它们将查询执行分解为小任务,并跨固定在物理内核中的一组线程安排任务。Hologres 采用类似的高并行方法。但是Hologres采用分层调度框架,工作单元的抽象降低了在多租户场景下调度大量任务时的复杂度和开销。执行上下文和调度组提供了一个强大的机制来确保不同租户之间的资源隔离。 [19] 迪斯

讨论了在多租户数据库中用于性能隔离的 CPU 共享技术。它强调数据库即服务环境中所需的绝对 CPU 预留。而 Hologres 只需要相对的 CPU 预留,这足以防止分析查询形成延迟服务查询。

HTAP 系统。近年来,随着对更多实时分析的需求快速增长,我们看到很多研究对提供针对大数据集的混合事务/分析处理 (HTAP) 解决方案感兴趣。 [33] 研究混合行和列格式如何帮助提高数据库对具有各种数据访问模式的查询的性能。 SAP HANA [21]、MemSQL [9]、HyPer [23]、Oracle 数据库 [25] 和 SQL Server [20、28] 等后续系统支持事务处理和分析处理。他们通常将行格式用于 OLTP,将列格式用于 OLAP,但需要在行格式和列格式之间转换数据。由于这些转换,新提交的数据可能不会立即反映在列存储中。相反,Hologres 可以将表格存储在行和列平板电脑中,并且每次写入表格都会同时更新两种类型的平板电脑。 Hologres 同时并行写入所有 tablets 以实现高写入吞吐量。此外,HSAP 场景的摄取率比 HTAP 场景中的交易率高得多(例如,用户通常在进行购买交易之前生成数十个页面浏览事件),但通常对一致性要求较弱。 Hologres特意只支持原子写和read-your-write读,通过避免复杂的并发控制实现了更高的读/写吞吐量。

[32] 研究 HTAP 系统中高并发工作负载的任务调度。对于 OLTP 工作负载,它会调整并发级别以使 CPU 饱和,因为 OLTP 任务包括大量使用同步。但是Hologres采用了latch-free的方式,避免了频繁阻塞。对于 OLAP 工作负载,它使用并发提示来调整分析工作负载的任务粒度,可以将其集成到 Hologres 中以调度执行上下文。

新SQL。 Hologres 采用的分片机制类似于 BigTable [16] 和 Spanner [18]。 BigTable 使用 table tablet 的抽象来促进对排序数据的范围搜索。 Spanner 是一个全局分布式的键值存储,支持强锥数据分片,Spanner 中以 Spanner 为基本存储单元。保持数据一致性与分布式数据Hologres有意避免支持较弱的要求,以追求更好的性能。

7. 结论和未来的工作

在现代大数据处理中,服务和分析处理 (HSAP) 的融合出现了许多新趋势。在阿里巴巴,我们设计并实现了云原生HSAP服务Hologres。Hologres采用新颖的基于tablet的存储设计,基于执行上下文的调度机制,以及存储/计算和读/写的清晰解耦。这使 Hologres 能够实时提供高吞吐量数据摄取,并为混合服务和分析处理提供卓越的查询性能。我们对 Hologres 和许多大数据系统进行了全面的实验研究。我们的结果表明,Hologres 在与其用于分析或服务场景的最先进系统。要在 HSAP 中获得更高的性能,存在许多开放式挑战。这些挑战包括针对读取密集型热点的更好横向扩展机制、内存子系统和网络带宽的更好资源隔离以及分布式环境中的绝对资源预留。我们计划将探索这些问题作为未来工作的一部分。

8. 参考文献

[1] Actian 向量。 https://www.actian.com。
[2] 阿帕奇箭头。 https://arrow.apache.org。
[3] 阿帕奇高清文件系统。 https://hadoop.apache.org。
[4] 大。 https://flink.apache.org。
[5] 绿梅。 https://greenplum.org。
[6] 数据库。 https://hbase.apache.org。
[7] 蜂巢。 https://hive.apache.org。
[8] 英特尔 avx-512 指令集。 https://www.intel.com/内容/www/us/en/architecture-and-technology/avx-512-overview.html。
[9] 内存数据库。 http://www.memsql.com/。
[10] 数据库。 https://www.mysql.com。
[11] 关键的青梅。 https://gpdb.docs.pivotal.io/6-0/admin_guide/workload_mgmt.html。
[12] PostgreSQL。 https://www.postgresql.org。
[13] 岩石数据库。 https://github.com/facebook/rocksdb/wiki。
[14] 泰拉数据。 http://www.teradata.com。
[15] Tpc-h 基准测试。 http://www.tpc.org/tpch。
[16] F. Chang、J. Dean、S. Ghemawat、WC Hsieh、DA Wallach、M. Burrows、T. Chandra、A. Fikes 和 RE Gruber。 Bigtable:结构化数据的分布式存储系统。 ACM 跨。电脑。系统, 26(2), 2008 年 6 月。
[17] BF Cooper、A. Silberstein、E. Tam、R. Ramakrishnan 和 R. Sears。使用 ycsb 对云服务系统进行基准测试。在第一届 ACM 云计算研讨会论文集中,SoCC 2010,美国纽约州纽约市,2010 年。计算机协会。
[18] JC Corbett、J. Dean、M. Epstein、A. Fikes、C. Frost、JJ Furman、S. Ghemawat、A. Gubarev、C. Heiser、P. Hochschild 等。Spanner:Google 的全球分布式数据库。ACM 跨。电脑。系统, 31(3), 2013 年 8 月。
[19] S. Das、VR Narasaya、F. Li 和 M. Syamala。 CPU共享多租户关系数据库即服务中的性能隔离技术。 PVLDB, 7(1):37–48, 2013。
[20] C. Diaconu、C. Freedman、E. Ismert、P.-A。 Larson、P. Mittal、R. Stonecipher、N. Verma 和 M. Zwilling。 Hekaton:Sql server的内存优化oltp引擎。在 2013 年 ACM SIGMOD 数据管理国际会议论文集中,第 1243–1254 页,2013 年。
[21] F. Farber、N. May、W. Lehner、P. Große、I. Muller、H. Rauhe 和 J. Dees。 e sap hana 数据库 架构概述。IEEE 数据工程师。公牛, 35(1):28–33, 2012。
[22] J.-F。 Im、K. Gopalakrishna、S. Subramaniam、M. Shrivastava、A. Tumbde、X. Jiang、J. Dai、S. Lee、N. Pawar、J. Li 等。Pinot:5.3 亿用户的实时 olap。在 2018 年国际数据管理会议论文集中,SIGMOD 2018,美国纽约州纽约市,2018 年。计算机协会。
[23] A. Kemper 和 T. Neumann。 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 届国际数据工程会议,第 195-206 页。 IEEE,2011 年。
[24] M. Kornacker、A. Behm、V. Bittorf、T. Bobrovytsky、C. Ching、A. Choi、J. Erickson、M. Grund、D. Hecht、M. Jacobs、I. Joshi、L. Ku , D. Kumar, A. Leblang, N. Li, I. Pandis, H. Robinson, D. Rorke, S. Rus, J. Russell, D. Tsirogiannis, S. Wanderman-Milne 和 M. Yoder。 Impala:用于 hadoop 的现代开源 SQL 引擎。在 CIDR 2015 中,第七届创新数据系统研究双年会议,美国加利福尼亚州阿西洛马,2015 年 1 月 4 日至 7 日,在线会议记录。www.cirdb.org,2015 年。
[25] T. Lahiri、S. Chavan、M. Colgan、D. Das、A. Ganesh、M. Gleeson、S. Hase、A. Holloway、J. Kamp、T. Lee、J. Loaiza、N. Macnaughton、V. Marwah、N. Mukherjee、A. Mullick、S. Muthulingam、V. Raja、M.S. Roth 、E. Soylemez 和 M. Zait。Oracle 内存数据库:一种双格式内存数据库。 2015 年 IEEE 第 31 届国际数据工程会议,第 1253–1258 页,2015 年。
[26] A. Lakshman 和 P. Malik。 Cassandra:一个去中心化的结构化存储系统。 SIGOPS 歌剧。系统。修订版,44(2),2010 年 4 月。
[27] A. Lamb、M. Fuller、R. Varadarajan、N. Tran、B. Vandiver、L. Doshi 和 C. Bear。 e vertica 分析数据库:C-store 7 年后。 PVLDB, 5(12):1790–1801, 2012。
[28] P.-r。 Larson, A. Birka, EN Hanson, W. Huang, M. Nowakiewicz 和 V. Papadimos。使用sql server进行实时分析处理。 PVLDB, 8(12):1740–1751, 2015。
[29] V. Leis、P. Boncz、A. Kemper 和 T. Neumann。 Morsel 驱动的并行性:多核时代的 numa 感知查询评估框架。在 2014 年 ACM SIGMOD 数据管理国际会议论文集中,SIGMOD 2014,美国纽约州纽约市,2014 年。计算机协会。
[30] Y. Mao、E. Kohler 和 RT Morris。用于快速多核键值存储的缓存机制。在第七届 ACM 欧洲计算机系统会议论文集中,EuroSys 2012,美国纽约州纽约市,2012 年。计算机协会。
[31] JM Patel、H. Deshmukh、J. Zhu、N. Potti、Z. Zhang、M. Spehlmann、H. Memisoglu 和 S. Saurabh。 Quickstep:基于扩展方法的数据平台。 PVLDB, 11(6):663–676, 2018。
[32] I. Psaroudakis、T. Scheuer、N. May 和 A. Ailamaki。任务调度高度并发的分析和事务性主内存工作负载。在第四届使用现代处理器和存储架构加速数据管理系统 (ADMS 2013) 的国际研讨会论文集中,编号 CONF,2013 年。
[33] R. Ramamurthy、DJ DeWitt 和 Q. Su。破碎镜子的案例。在第 28 届超大型数据库国际会议论文集中,第 430-441 页。 VLDB 捐赠基金,2002.
[34] V. Raman、G. Attaluri、R. Barber、N. Chainani、D. Kalmuk、V. KulandaiSamy、J. Leenstra、S. Lightstone、S. Liu、GM 洛曼等人。 Db2 with blu acceleration:不仅仅是一个列存储。 PVLDB, 6(11):1080–1091,2013 年。
[35] M. Stonebraker 和 A. Weisberg。 e voltdb 主内存 dbms。 IEEE 数据工程师。公牛, 36(2):21–27, 2013。
[36] F. Yang、E. Tschetter、X. Leaute、N.Ray、G. Merlino 和 D. Ganguli。 Druid:实时分析数据存储。在 2014 年 ACM SIGMOD 数据管理国际会议论文集中,SIGMOD 2014,美国纽约州纽约市,2014 年。计算机协会。
[37] M. Zukowski 和 PA Boncz。 Vectorwise:超越列存储。 IEEE 数据工程师。公牛, 35:21–27, 2012.