基于物化视图的 OLAP 系统性能优化研究

万瑞萍 M202173811[1)]

[1)](华中科技大学计算机科学与技术学院，武汉 430074)

摘　要　联机分析处理 OLAP(Online Analytical Processing)是一种软件技术，它使分析人员能够迅速、一致、交互地从各个方面观察信息，以达到深入理解数据的目的。OLAP 查询常常涉及到不同的维表和事实表，要得到查询结果通常需要进行多张表的连接操作。连接操作是一种非常耗时的操作，因此，如何提高 OLAP 查询效率成为数据仓库应用中的关键问题。物化视图是包括一个查询结果的数据库对象，它是远程数据的的本地副本，或者用来生成基于数据表求和的汇总表。Microsoft 研发了一种利用谓词下推实现 OLAP 系统中内容缓存的系统，Crystal，它大大改善查询延迟，同时节省了远程存储的带宽。Google 开发了一种分析数据管理系统 Napa，它具有健壮的查询处理和灵活的客户端数据库配置的特点。Netflix 开发了一种新的数据库 CDC 框架，DBLog，解决了数据同步的问题。

关键词　数据中心；缓存；OLAP

中图法分类号　TP302

Research on performance optimization of OLAP System Based on materialized view

WAN Ruiping M202173811[1)]

[1)](School of computer science and technology, Huazhong University of Science and Technology, Wuhan 430074)

Abstract　OLAP (online analytical processing) is a software technology, which enables analysts to observe information from all aspects quickly, consistently and interactively, so as to achieve the purpose of in-depth understanding of data. OLAP queries often involve different dimension tables and fact tables. To get query results, you usually need to connect multiple tables. Join operation is a very time-consuming operation. Therefore, how to improve the efficiency of OLAP query has become a key problem in the application of data warehouse. Materialized view is a database object that includes a query result. It is a local copy of remote data or used to generate summary tables based on data table summation. Microsoft has developed a system that uses predicate push down to realize content caching in OLAP system, crystal, which greatly improves query latency and saves remote storage bandwidth. Google has developed an analytical data management system Napa, which has the characteristics of robust query processing and flexible client database configuration. Netflix has developed a new database CDC framework, dblog, to solve the problem of data synchronization.

Key words　Data center;　Materialized view；OLAP

1 引言

基于关系数据库的 OLAP(ROLAP) 以关系数据库为核心，用关系二维表来存放数据。ROLAP 将多维数据库中的多维结构表划分为两类，一类是事实表，用来存储事实的度量值及各维的关键字以作为外键; 另一类是维表，维表即用户分析问题的一个角度，简单说即一个方面。以事实表为中心，通过外键与若干维表联系在一起形成星型模型。星型模型中，如果维度表是分层次的便组成雪花模型; 如果事实表共享某些维度表便组成星座模型; 而如果维度表分层次，同时事实表共享部分维表，则组成雪暴模型。无论是哪种模型，都是星型模型的扩展。星型模型是 OLAP 中被广泛采用的一种模型，该模型利于查询功能的实现，但在查询性能方面有很大的不足。用户每做一次查询，如果涉及到不同的维表和事实表，那么就要对不同的维表和事实表进行连接处理，而连接处理会花费很多额外的时间开销。例如，基于一个实际的应用项目中的 OLAP 模型为星座模型，用户的查询请求大多需要连接 3 张或 3 张以上的维表和事实表。目前，实现 OLAP 查询的方式是基于视图的查询。这种查询方式只是简化了用户查询方式，在查询的效率上并没有作特殊处理。

物化视图又叫实视图或实体化视图、，通过预先将那些费时的表连接操作或聚集操作进行预算并保存起来，是数据仓库中提高查询响应速度的一种高效技术。在 OLAP 查询优化中，可先按业务需要、相关主题或者统计分析等角度将数据抽取到物化视图中保存起来，然后让用户基于物化视图进行查询。这样做避免了从整个数据中寻找用户所需数据，这是一种典型的用空间换取时间的优化技术，它相当于部分远程数据的本地副本。物化视图对应用透明，增加和删除物化视图不会影响应用程序中 SQL 语句的正确性和有效性。

物化视图技术在传统 RDBMS 时代就已经发展得十分成熟，各大商业数据库（如 SQL server 喝 Oracle）均提供了不错的支持，只是在开源数据库中应用较少。最近重回人们的视野中，主要是由于它在分布式和实时性上有新的发展，不再是原来的单机模型了。

物化视图技术要点在于对内容的增量维护，也就是以最小的成本来更新内容，本文调研的在这方面都有自己的研究进展。

2 原理和优势

本文中调研了三篇文章，分别来自于 Microsoft、Google 和 Netflix。第一篇文章是《Crystal: A Unified Cache Storage System for Analytical Databases》[1]，是 Microsoft Research 发表在 VLDB21 上的一篇文章，它介绍了一种用于分析数据库的统一缓存存储系统 Crystal ，能够大大改善查询延迟，同时节省了远程存储的带宽。第二篇文章是《Napa: Powering Scalable Data Warehousing with Robust Query Performance at Google》[2]，是 Google 发表在 VLDB21 上的一篇文章，Google 开发并部署了一个分析数据管理系统 Napa，它可以在可伸缩性、亚秒级查询响应时间、可用性和强一致性等极其苛刻的要求下存储和服务这些全球规模的数据集。第三篇文章是《DBLog: A Watermark Based Change-Data-Capture Framework》[3]，是 Netflix 发表在 arXiv 上的一篇文章，Netflix 开发了一种新的 CDC 数据库框架，DBLog。它可以将事件传递到任何输出，无论它是数据库、流还是 API。这些特性为同步多个数据系统开辟了新的途径。由于 Netflix 运营着数百个具有独立数据需求的微服务，DBLog 已经成为 Netflix 数据同步和丰富平台的基础。

Crystal 是一种新的智能缓存存储系统架构，该架构与计算节点位于同一位置。Crystal 的缓存区域可以被认为是物化视图或语义缓存，因此继承了这一丰富的工作。Crystal 的客户端是具有下推谓词的特定于 dbms 的“数据源”。在本质上类似于 DBMS, Crystal 集成了查询处理和优化组件，重点关注称为区域的单表超矩形的高效缓存和服务。结果表明，Crystal 使用一个小的特定于 dbms 的数据源连接器，可以显著改善未修改 Spark 和 Greenplum 上的查询延迟，同时还节省了远程存储的带宽。

Napa 实现健壮查询性能的方法包括积极使用物化视图，当跨多个数据中心接收新数据时，物化视图得到一致维护。这与其他系统通过有效扫描基本数据来实现性能的当前趋势形成了鲜明对比。如果没有索引物化视图，为我们的大多数工作负载交付健壮的次秒响应时间是极其困难的。查询工作负载的物化视图的覆盖范围决定了查询性能，而视图刷新的速度影响到新鲜度。通过组合，改变视图覆盖的工作负载的大小以及刷新它们的频率，客户端

可以选择不同的成本/性能权衡。

DBLog 利用了一种基于水印的方法，它允许我们将事务日志事件与我们直接从表中选择的行交织在一起，以捕获完整的状态。作者的解决方案允许日志事件在处理选择时继续进行，而不会停止。可以在任何时候对所有表、特定表或表的特定主键触发 select。DBLog 以块的形式执行选择并跟踪进度，允许暂停和恢复选择。水印方法不使用锁，对源代码的影响最小。目前，Netflix 的数十家微服务公司都在使用 DBLog。

接下来，本文将详细介绍上述三篇文章的研究进展。

3 研究进展

3.1 Crystal:用于分析数据库的统一缓存存储系统

3.1.1 研究背景

在其灵活性和现收现付能力的驱动下，我们正在见证分析数据库系统向云计算的范式转变。此类数据库采用分层或分解的存储模型，其中弹性计算层访问持久存在于独立可伸缩的远程云存储上的数据。考虑到远程存储相对较高的延迟和较低的带宽，在计算节点上缓存数据变得非常重要。因此，我们见证了高速缓存技术在分析方面的新发展，在这种技术中，热门数据被保存在计算层的有限大小的快速本地存储(例如 SSD)中。

3.1.2 现有方法的不足之处

为了简单起见，这些缓存解决方案通常在文件或块级别上作为黑箱操作，使用标准缓存替换策略(如 LRU)来管理缓存。尽管它们很简单，但这些解决方案并没有解决云数据库的几个架构和性能挑战:

- 如今，每个 DBMS 都根据其特定需求实现了自己的缓存层，这导致了系统间大量的工作重复，重新定义了缓存什么、缓存在哪里、何时缓存以及如何缓存等选项。
- 数据库越来越多地支持对原始数据格式和面向行二进制格式的分析。与 Apache Parquet 等二进制柱状格式相比，这些格式上的数据处理更慢，并导致成本增加，即使数据已经缓存在计算节点上。与此同时，将所有数据在存储上转换为二进制列格式的成本很高，特别是因为查询只积极使用和访问一小部分数据并不断变化。
- 在现有的解决方案中，缓存利用率很低，因为即使页面中需要一个记录或值，也需要检索和缓存整个页面，从而浪费缓存中宝贵的空间。
- 最近，云存储系统提供了谓词下推作为一种本地功能。下推允许我们将谓词发送到远程存储并避免检索所有块，但加剧了如何利用它进行有效本地缓存的问题。

3.1.3 Crystal 的设计

作者提出了一种新的“智能”存储中间件 Crystal，它与数据库分离，位于数据库和原始存储之间。Crystal 可以被看作一个小型 dbms，或缓存管理系统(CMS)，用于存储。它作为两个子组件运行:

- Crystal CMS 运行在计算节点上，本地“客户端”可以访问它，并能够与远程存储进行交互。
- Crystal 的客户端(称为连接器)是特定于 db 的适配器，它们本身使用下推谓词实现数据源 API，类似于今天的 CSV 和 Parquet 数据源。

Crystal 的缓存区域可以被认为是物化视图，因此继承了这一丰富的工作。但是 Crystal 的缓存有额外的限制，它们严格地是单表谓词的结果。具体来说，Crystal 的区域是每个表上谓词连接的析取。在实际应用中难免会碰到跨表谓词，而作者的技术挑战解决方案中利用了这一限制，使我们能够更好地匹配、一般化和更有效地搜索最佳缓存区域集。

系统概述

图 1 显示了 Crystal 在当今云分析生态系统中的位置。每个计算节点运行一个 DBMS 实例;Crystal 位于计算节点上，并通过数据源连接器为这些 DBMS 实例提供服务。其目的是作为大数据系统和云存储之间的缓存层，利用计算节点中的快速本地存储来减少对远程存储的数据访问。

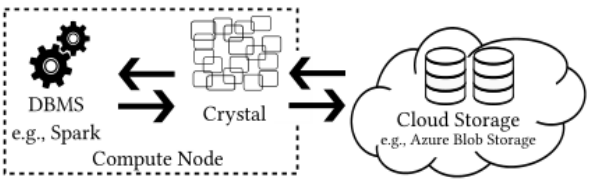


图 1 Crystal 在大数据生态系统中的位置

Crystal 被构建为两个独立的组件:一个轻量级 dbms 特定数据源连接器和 Crystal CMS。而 Crystal CMS 又同时维护了两个本地缓存——一个小的请

求区域(RR)缓存和一个大的 oracle 区域(OR)缓存——分别对应于短期和长期知识。这两个缓存都以高效的柱状开放格式(如 Parquet)存储数据。RR 缓存充当一个“缓冲”区域，在 OR 缓存收集足够的统计数据以做出长期决策之前，临时存储最近查询的缓存视图。这两个区域为缓存提供了一个高效的反应式解决方案。

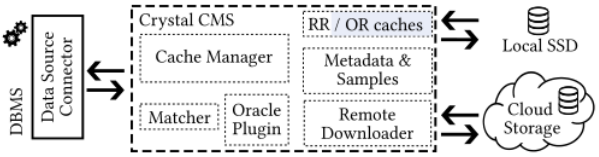


图 2 Crystal 组成

请求匹配

在区域请求期间，Crystal 搜索缓存以检索本地超集。图 3 显示了匹配请求的过程。首先，对 oracle 区域缓存进行匹配扫描。如果请求没有完全缓存，Crystal 会尝试将其与所请求的区域缓存匹配。如果查询不匹配，下载管理器将获取远程文件(可选地从文件缓存中获取)。

在匹配过程中，对一个完整超集进行优先级排序。只有在没有找到完全超集的情况下，Crystal 才会尝试满足单个的连接。Crystal 通过根据远程文件名和投影模式对缓存进行分区来优化区域匹配。文件名表示为远程文件目录的(位)集。这个集合由表进行分片。类似地，模式可以表示为(位)集。分区是在多个阶段完成的。在快速的文件名超集检查之后，将测试所有结果候选者的模式超集。只有在超集区域的这个分区中，我们才能扫描潜在的匹配。虽然在区域匹配过程中不会出现性能问题，但多维索引(例如 r -树)可以用于进一步加速查找。

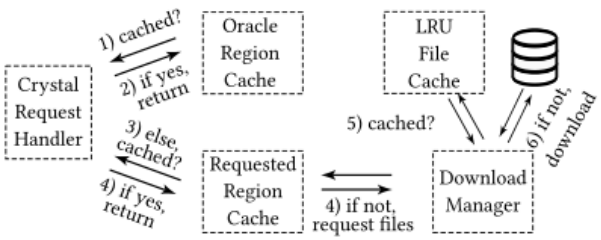
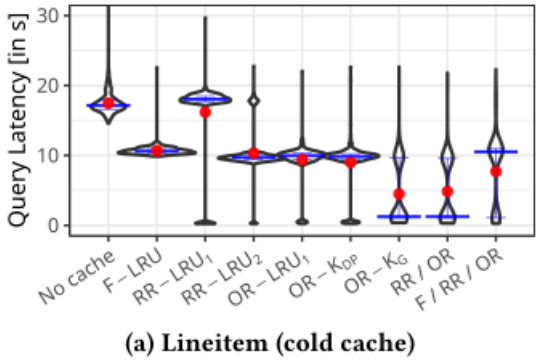


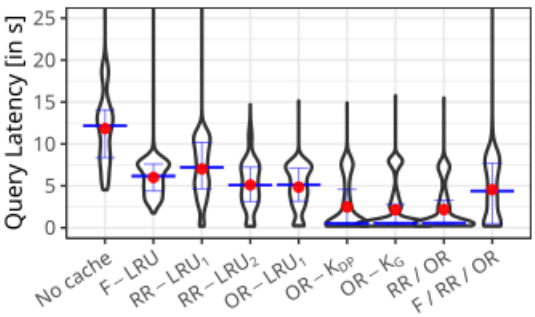
图 3 请求处理

3.1.4 实验评价

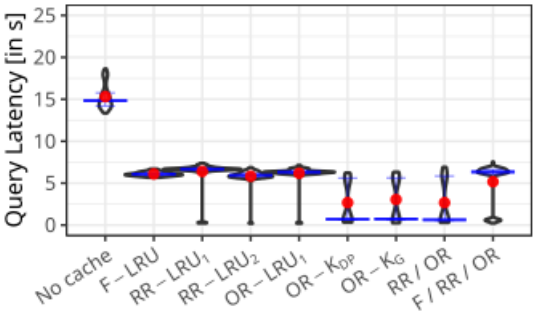
作者选用了 Lineitem、NYC Taxi、Historical Stock Prices 这三个数据集对 Crystal 进行了一系列测试。首先测试了 Crystal 在三个数据集上的长期知识工作负载，如图 4 所示。我们看到了 oracle 区域缓存的显著改进。贪婪背包及其 RR/OR 变量在三种工作负载下均优于竞争对手。



(a) Lineitem (cold cache)



(b) Taxi (cold cache)

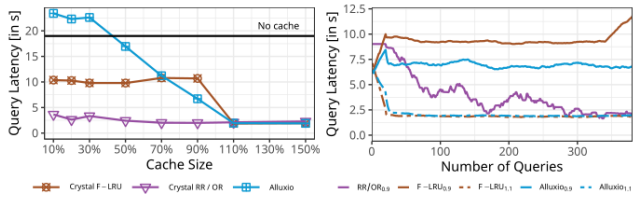


(c) Stocks (after cache warm-up)

图 1 实验结果 1

为了突出 Crystal 与传统缓存相比的性能优势，我们使用不同的缓存大小运行区域工作负载。图 5a 显示了不同缓存值的平均 lineitem 区域性能，该值由原始数据的百分比表示，可以看出只有在这种极端情况下，Alluxio 才能达到 RR/OR 的性能。图 5b 显示了使用冷缓存的 90%和 110%版本的 Alluxio 和 F-LRU 的性能随时间的变化情况。Crystal 的 RR/OR 仅绘制为 90%(与 110%没有可见差异)。传统缓存直接达到最大性能，而 RR/OR 随着时间的推移学习元组的最佳集。在查看足够多的查询之后，RR/OR

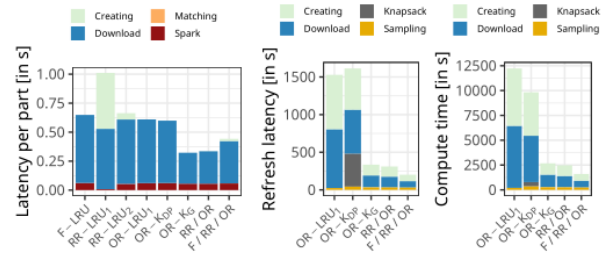
即使在 110%的场景中也能达到类似的性能。



(a) Mean lineitem regions performance on varying cache sizes after a warm-up phase. (b) Lineitem regions mean latency over time with cache sizes 90% (solid) and 110% (dashed).

图 2 实验结果 2

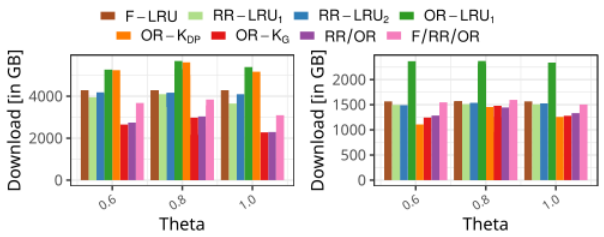
接下来，作者跑了几个 benchmark，均取得了不错的成绩。



(a) Query latency per Parquet part (b) Oracle: cache refresh latency (c) Oracle: sum of compute time

Figure 14: Operation break-down on the online end-to-end and the offline background tasks for lineitem regions.

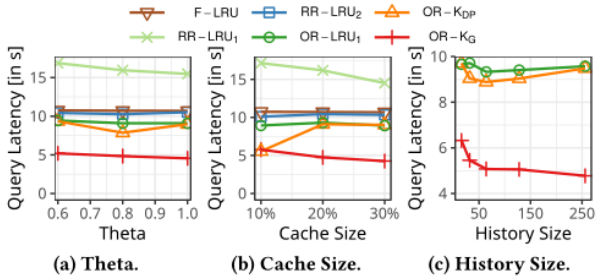
图 6 测试结果 3



(a) Regions - Lineitem (b) Regions - Stocks

Figure 15: The total network traffic used by each strategy plotted on a varying theta.

图 7 测试结果 4



(a) Theta. (b) Cache Size. (c) History Size.

图 8 测试结果 5

3.2 Napa：在Google为可扩展数据仓库提供强大的查询性能

3.2.1 研究背景

谷歌在全球拥有超过 10 亿用户，运营多项服务。在提供这些服务时，谷歌服务依赖于应用程序数据来提供更好的用户体验、提高服务质量和计费。谷歌业务用户通过复杂的分析前端与这些数据交互，以获得对其业务的洞察。这些前端对大量数据发出复杂的分析查询，并施加严格的时间限制。在某些情况下，商定的查询响应时间目标以毫秒为单位。这些数据有多个拍字节(pb)，并且通过大量行星级的更新流不断更新。用户要求查询结果保持一致和新鲜，并要求在数据中心故障或网络分区时保持持续可用性。本文描述了 Napa，一个满足这些挑战性要求的分析数据存储系统。

Napa 是用来取代 Mesa 的，Mesa 是一个早期的谷歌系统。Napa 已经运营多年了。它从 Mesa 继承了许多 pb 级的历史数据，并搭载了许多新客户。虽然 Mesa 是为具有极端延迟要求的特定关键客户而构建的，但 Napa 有更广泛的任务。我们在相关的工作部分比较了 Mesa 和 Napa:简而言之，与 Mesa 相比，Napa 的设计宗旨是在谷歌范围内使用，并服务于许多分析应用程序的多样化需求。

3.2.2 Napa 的设计

Napa 必须具有高度的可伸缩性，以处理更新流，同时以良好的性能服务数百万个查询。Napa 的一个关键设计选择是依赖于物化视图来实现可预测和高查询性能。

纳帕的高层架构由下图所示的三个主要组件组成。

图 9 Napa 的概念设计

系统架构

Napa 的高层架构由数据和控制平面组成，如图 10 所示。该架构部署在多个数据中心，管理每个数据中心的副本。数据平面包括摄入、存储和查询服务。控制平面由一个控制器组成，它协调各个子系统之间的工作。控制器还负责跨多个数据中心同步和协调元数据事务。Napa 客户端创建数据库和表及

其关联的模式。客户端可以选择为每个表创建物化视图。

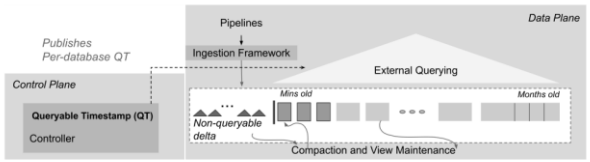


图 10 Napa 的系统结构

摄取数据

摄取框架的目标是允许摄取管道将大量的数据插入到 Napa 中，而不会产生很大的开销。Napa 的关键技术之一是将摄取从视图维护和索引中分离出来，为客户提供跨新鲜度、查询性能和成本的权衡。摄取框架通过图 11 所示的两种机制促成了这种设计。首先，摄取框架的目标是接受数据，执行最少的处理，并使其持久，而不考虑后续视图维护的速度。所有摄入的行都被分配一个元数据时间戳进行排序，然后在满足其他持久性条件(如复制)后标记为已提交。其次，摄取框架允许配置增加或减少接受数据并执行批处理、聚合和复制摄取工作的任务数量，从而限制机器的峰值成本。

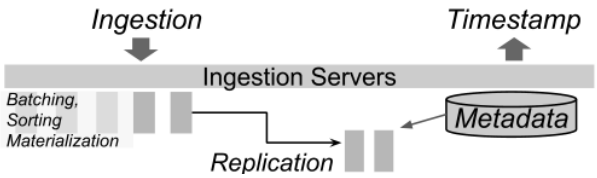


图 11 Napa 摄取数据

可查询时间戳

表的可查询时间戳(QT)是一种时间戳，它指示可查询数据的新鲜程度。如果 QT(表)= X，则客户机可以查询 X 时间之前摄入到表中的所有数据，X 时间之后的数据不属于查询结果的一部分。换句话说，表的新鲜度是[Now() - QT]。QT 充当了一个屏障，在 X 之后获取的任何数据对客户端查询都是隐藏的。一旦在(Y-X)范围内摄取的数据被优化以满足查询性能要求，QT 的值将从 X 提升到 Y。反过来，客户端可以使用 Napa 的配置选项，这个单一的客户端可见指标来调整新鲜度、查询性能和成本。例如，如果客户想要高的查询性能和低的成本，但是可以权衡新鲜度，那么系统会优先使用更少的机器资源来进行视图维护以降低成本，而 QT 的进程可能会很慢，从而表明数据的新鲜度降低。

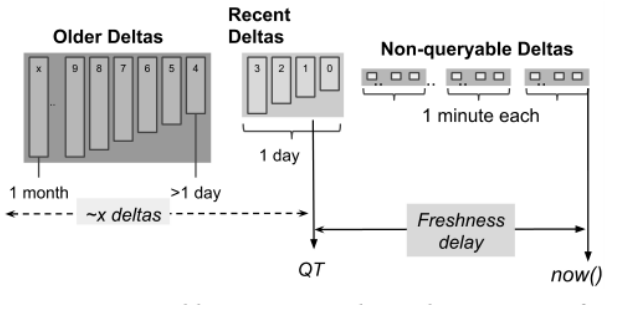


图 12 可查询的时间戳

3.2.3 实验评价

健壮的查询服务性能

图 13 显示了(a)随着视图数量的增加查询延迟的减少，(b)查询允许跨越的增量的数量和相应的延迟影响(越低越好)。

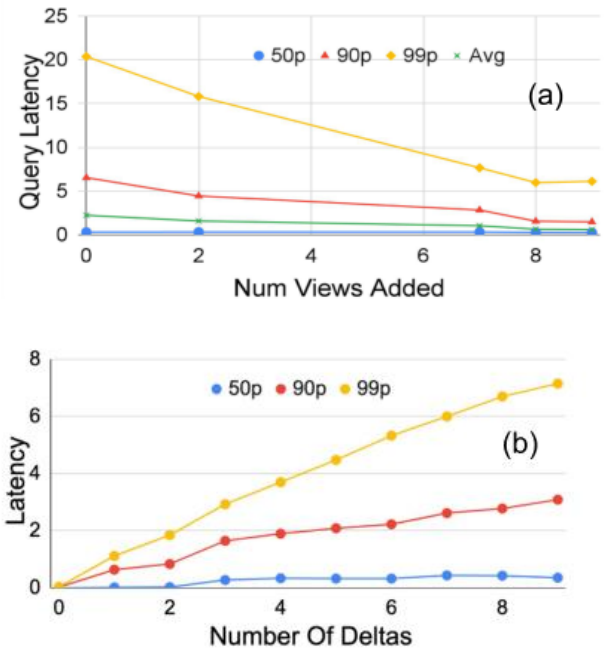


图 13 测试结果 1

图 14 显示 Napa 能够保证其客户端稳定的查询性能，即使在摄入负载发生变化或出现基础设施中断时也是如此。图 14 显示了客户机在几个小时内的工作负载。Napa 将摄取从视图维护和查询中分离出来，这允许我们优化查询延迟的低方差，在某些情况下，通过交换数据的新鲜度。图 14(a)显示客户端不断地向 Napa 发送数据，在一周的过程中输入率有一些差异。图 14(b)显示了视图维护性能在 X 和 Y 之间的持续时间下降，表明了影响任务更新视图的基础设施问题。但是，查询服务延迟在整个过

程中保持不变(图 14(d))。在这个特殊的示例中，客户机查询仍然很快，但是，对于停机期间的某些部分，数据新鲜度受到影响，如图 14(c)所示，延迟值很高。

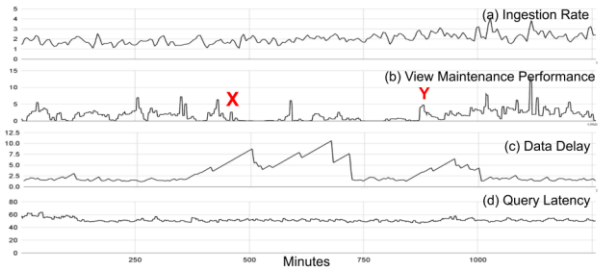


图 14 测试结果 2

图 10(a)-(d)显示了纳帕如何为客户提供优化不同性能和成本指标的灵活性。这些生产客户端有不同的需求：

客户 A:权衡新鲜度。客户端 B:权衡查询性能。客户 C:权衡成本。

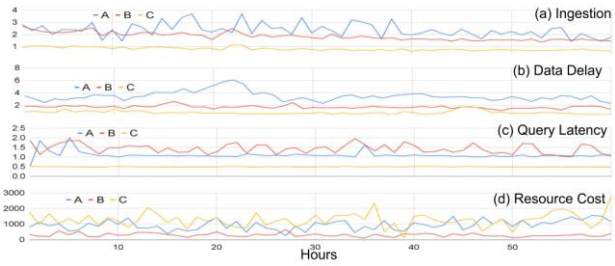


图 15 来自三个客户端工作负载的生产指标(a)摄入负载，(b)数据延迟，(c)查询延迟，(d)每摄入单位的资源成本。

3.3 DBlog：基于水印的变更数据捕获框架

3.3.1 研究背景

Netfix 使用数百个微服务，每天在数据层执行数万亿次操作。由于没有一种数据库设计能够满足所有的需求，所以每个微服务都可以利用多个异构数据库。为了能够保持多个数据库同步，我们开发了一个数据丰富和同步平台，即 Delta。关键需求之一是从源存储库到派生存储库的传播延迟较低，并且事件流是高度可用的。实现这一目标的关键需求是拥有更改数据捕获(Change-Data-Capture, CDC)，CDC 允许近实时地从数据库中捕获更改的行，并最终将这些行传播到下游消费者。CDC 在需要保持多个异构数据库同步的用例中变得越来越流行，并解决了传统技术(如双写和分布式事务)所存在的挑战。

3.3.2 DBlog 设计

DBLog 是一个基于 java 的框架，能够从数据库的事务日志中捕获更改的行，并通过对表执行选择来捕获数据库的完整状态。选择以块的形式执行，并与日志事件交织在一起，这样日志事件处理不会停滞很长一段时间。这是通过利用基于水印的方法实现的。选择可以通过 API 在运行时执行。这允许启动时使用完整状态的 DBLog 输出，或者在以后进行修复时使用。如果输出是启用了日志压缩功能的 Kafka，那么下游消费者可以通过从 Kafka 读取包含完整数据集的事件来引导，并通过添加从源捕获的更改行来不断更新。对于只有一个消费者的用例，DBLog 还可以直接向数据存储或 API 发出事件。

事务日志捕捉

每个事件都被序列化为 DBLog 事件格式，并被附加到一个输出缓冲区中，该缓冲区位于内存中，是 DBLog 进程的一部分。然后，另一个线程使用输出缓冲区中的事件，并按顺序将它们发送到实际的输出。输出是一个简单的接口，允许插件任何目的地，比如流、数据存储，或者通常任何类型的具有 API 的服务。

我们还捕获模式更改。模式更改捕获的性质因数据库而异，因此日志中可能有模式更改的增量，或者数据库可能在每个发出的事件中包含模式信息。由于篇幅限制，本文没有讨论我们在 blog 中实现模式捕获的方法。

完整状态捕获

Netflix 开发了一个解决方案，它只使用常用的数据库特性，并且尽可能少地影响源数据库。它以块的形式从表中选择行，并将这些块放置在内存中的事件旁边，这些事件是从事务日志中捕获的。这样做确实保留了日志事件的历史。这个解决方案允许在任何时候通过 API 提取所有表、特定表或表的特定主键的完整状态。选择是按每个表执行的，并且按照配置的大小按块执行。通过按照升序的主键顺序对表进行排序，并包括那些主键大于前一个块的最后一个主键的行来选择块。为了将对源数据库的影响降到最低，这个查询必须高效地运行。由于这些原因，DBLog 需要数据库对主键提供有效的范围扫描，DBlog 只允许对具有主键的表进行选择。图 16 用一个简单的例子说明了块选择。

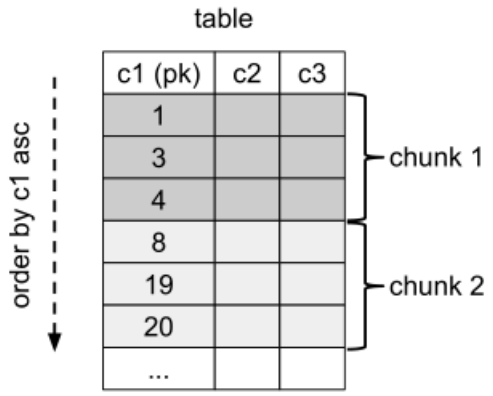


图 16 将 3 列 c1-c3 和 c1 作为主键(pk)的表分组。Pk 列的类型为整数，块大小为 3。以条件 c1> 4 选择 Chunk 2。

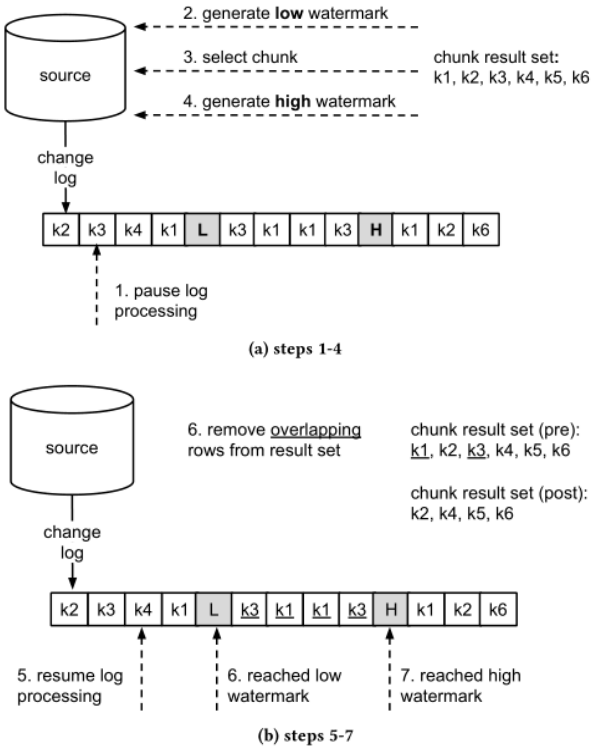


图 17 Watermark-based 块选择

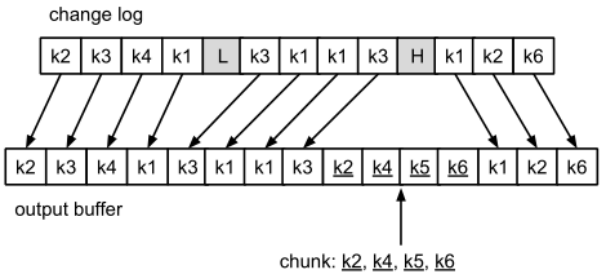


图 18 输出写的顺序。将日志捕获与完整数据捕获交织在一起。

数据库支持

到目前为止，DBlog 支持 MySQL, PostgreSQL 和 Aurora。在所有情况下，日志事件都是由数据库按照提交顺序提供的，对于单个选择事务，可以通过读取提交隔离级别进行非陈旧的读取。使用 SQL 和 JDBC 集成了完整的状态捕获，只需要实现块选择和水印更新。同样的代码用于 MySQL 和 PostgreSQL，也可以用于其他支持 JDBC 的数据库。转储处理本身不依赖于 SQL 或 JDBC，并且允许集成满足 DBLog 框架要求的数据库，即使它们不是 RDBMS 数据库。

3.3.3 DBlog 优势

DBLog 功能可以从数据库事务日志中实时捕获更改的行，还可以作为集成产品的一部分提取数据库的完整状态。此外，DBLog 为用户提供了请求完整状态并在任何时候执行它的端点，而不会停止日志事件处理。同时，由于基于水印的方法，在任何时候都保持了原始的历史顺序，而没有对源数据库使用锁。此外，还设置了控制块选择的控件，或者在需要时暂停和恢复。在捕获非常大的表上的完整状态和进程崩溃时，就不需要从头开始重复这个过程。设计 DBLog 的目的是将事件传递到任何输出，无论它是数据库、流还是 API。这些特性为同步多个数据系统开辟了新的途径。

由于 Netflix 运营着数百个具有独立数据需求的微服务，DBLog 已经成为 Netflix 数据同步和丰富平台的基础。它消除了应用程序开发人员维护多个数据存储的复杂性。DBLog 及其基于水印的方法是针对 RDBMS 类数据库设计的。下一步，我们正在开发其他 CDC 框架，以支持不属于 DBLog 框架的数据库，如 Apache Cassandra 等多主 NoSQL 数据库。其目标是支持与 DBLog 类似的功能，即:在任何时间捕获完整状态的能力，与日志事件交叉，并在源上产生最小的影响。

4 总结与展望

应该说，最近在数据库领域逐渐热起来的一个主题就是物化视图了。在年初的 TiDB Hackathon 当中就有很多队伍不约而同的选择了这个主题。

Google 也在今年的 VLDB 上发表了一篇 Napa，也是利用了相关的技术。

物化视图的基本原理就是预先将用户指定的查询（View）的内容计算好并存储起来，因此当用户查询的时候可以快速的得到计算好的结果。它在传统 RDBMS 时代就已经发展的十分成熟，各大商业数据库（如 SQL Server 和 Oracle）均提供了不错的支持，只是在开源数据库中应用较少。这次重新回到人们的视野之中，主要是在分布式和实时性上有新的发展，不再是原来的单机模型了。

物化视图技术要点存在于内容的增量维护，也就是通过抓取源数据表 DML 操作后的 Delta 数据，以最小的成本来更新内容。物化视图回到人们视野中央恰恰就是从流处理系统开始的：流处理查询本身就是物化视图的增量维护的一个应用。

虽然物化视图的理论研究已经很充分了，但实现上还是有研究的空间的。包括但不限于以下方向：

1.如何平衡计算复杂性和查询自由度。如果允许执行任意 SQL，有的查询在计算上是有本质的难度的，比如有些 Join 查询在源表修改的情况下可能会出发大规模的视图重刷，这在于需要低延迟的场景下是应该避免的。

2.如何在分布式系统（特别是 HTAP 数据库系统）中集成的问题。比如如何把对 OLTP 的性能影响降到最低、如何在分布式系统下实现低处理延迟、如何实现更好的可扩展性之类的。

3.物化视图的一致性。目前来看，很多系统只是将物化视图当作一个 ETL 过程因而没有特别关注隔离级别。但是长远来看，具备更强的一致性对于应用来说是更优的。特别是与分布式数据库结合起来，如果能保证源表上事务的 ACID 是一个很好的特性。

物化视图的逐渐流行可能也跟最近 CDC 相关技术的发展和成熟有关。Netflix 就在这方面做了一些尝试。

5 参考文献

[1] A. Agiwal et al, "Napa: Powering scalable data warehousing with robust performance at google," Proceedings of the VLDB Endowment, vol. 14, (12), pp. 2986-2998, 2021.

[2] T. Akidau et al, "Watermarks in stream processing systems: Semantics and comparative analysis of apache flink and google cloud dataflow," Proceedings of the VLDB Endowment, vol. 14, (12), pp. 3135-3147, 2021.

[3] D. Durner, B. Chandramouli and Y. Li, "Crystal: A unified cache storage system for analytical databases," Proceedings of the VLDB Endowment, vol. 14, (11), pp. 2432-2444, 2021.