计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2015.20140302
52(2):333-342,2015

大数据分析与高速数据更新

陈世敏
(计算机体系结构国家重点实验室(中国科学院计算技术研究所) 北京 100190)
(chensm@ict.ac.cn)

Big Data Analysis and Data Velocity

Chen Shimin
(*State Key Laboratory of Computer Architecture* (*Institute of Computing Technology*, *Chinese Academy of Sciences*), *Beijing* 100190)

Abstract Big data poses three main challenges to the underlying data management systems: volume (a huge amount of data), velocity (high speed of data generation, data acquisition, and data updates), and variety (a large number of data types and data formats). In this paper, we focus on understanding the significance of velocity and discussing how to face the challenge of velocity in the context of big data analysis systems. We compare the requirements of velocity in transaction processing, data stream, and data analysis systems. Then we describe two of our recent research studies with an emphasis on the role of data velocity in big data analysis systems: 1) MaSM, supporting online data updates in data warehouse systems; 2) LogKV, supporting high-throughput data ingestion and efficient time-window based joins in an event log processing system. Comparing the two studies, we find that storing incoming data updates is only the minimum requirement. We should consider velocity as an integral part of the data acquisition and analysis life cycle. It is important to analyze the characteristics of the desired big data analysis operations, and then to optimize data organization and data distribution schemes for incoming data updates so as to maintain or even improve the efficiency of big data analysis.

Key words data updates; big data analysis; data warehouse; event log processing system; data organization and distribution

摘 要 大数据对于数据管理系统平台的主要挑战可以归纳为volume(数据量大)、velocity(数据的产生、获取和更新速度快)和variety(数据种类繁多)3个方面.针对大数据分析系统,尝试解读velocity的重要性和探讨如何应对velocity的挑战.首先比较事物处理、数据流、与数据分析系统对velocity的不同要求.然后从数据更新与大数据分析系统相互关系的角度出发,讨论两项近期的研究工作:1)MaSM,在数据仓库系统中支持在线数据更新;2)LogKV,在日志处理系统中支持高速流入的日志数据和高效的基于时间窗口的连接操作.通过分析比较发现,存储数据更新只是最基本的要求,更重要的是应该把大数据的从更新到分析作为数据的整个生命周期,进行综合考虑和优化,根据大数据分析的特点,优化高速数据更新的数据组织和数据分布方式,从而保证甚至提高数据分析运算的效率.

收稿日期:2014-11-30;**修回日期**:2014-12-10
基金项目:中国科学院"百人计划";国家自然科学基金创新研究群体项目(61221062)

关键词 数据更新;大数据分析;数据仓库;日志处理系统;数据组织与分布算法

中图法分类号 TP311.13

大数据(big data)对于数据管理系统平台的主要挑战可以归纳为 volume(数据量大)、velocity(数据的产生、获取和更新速度快)和 variety(数据种类繁多)3 个方面. 大数据这个名词中"大"(big)在日常生活中通常用于形容空间的尺度. 因此,谈到大数据,人们的第一印象是数据量一定很大,从而要求大数据问题必须满足 volume 很大的特征. 这实际上是对大数据认识的一种误区. 从科研的角度出发,数据的 volume、velocity 和 variety 3 个方面,可以看作是描述数据特征的 3 个维度. 任何一个维度的需求超过了现有系统的处理能力,这样的问题都是值得研究的,都是大数据问题[1]. 也就是说,大数据的 3 个维度是"或"的关系. 研究可以围绕大数据的某一个维度展开,在单一维度的研究中取得了一定成果的基础上,再进一步研究两个维度的综合挑战,以至研究最复杂的 3 个维度的"与"问题. 本文主要集中探讨大数据 velocity 的问题.

大数据分析可以发掘隐藏在大数据中的巨大的政治、经济、社会价值,已经引起了广泛的关注[2]. 随着 Web 服务、社交网络、移动计算、物联网、大规模科学探测与计算分析等的蓬勃发展,"数字宇宙",即全人类在生产和生活中产生、存储、使用的全部数字化信息正在呈指数级快速增长. 市场分析公司 IDC 预测从 2013～2020 年,数字宇宙的总数据量将从 4.4 ZB 增长到 44 ZB(10^{21})[3]. 在 2013 年,数字宇宙中已经被分析利用的数据只占全部数据总量的 2%,而到 2020 年为止,全部数字宇宙中将有 35%的数据通过分析可以创造价值[3]. 综合上述两个方面的预测可以发现,大数据分析的市场将有超过 100 倍的巨大增长空间.

大数据分析系统包括数据仓库系统、MapReduce 系统、图处理系统、日志处理系统等. 本文针对大数据分析系统尝试解读 velocity 的重要性和探讨如何应对 velocity 的挑战. 下面,我们首先比较不同种类的大数据处理系统,分析 velocity 挑战的异同. 然后,从数据更新与大数据分析系统的相互关系的角度出发,讨论两项近期的研究工作:1)MaSM,在数据仓库系统中支持在线数据更新[4];2)LogKV,在日志处理系统中支持高速流入的日志数据和高效的基于时间窗口的连接操作[5]. 最后,我们比较上述两项工作和总结全文.

1 大数据系统中的 Velocity

1.1 事物处理系统

事物处理系统(transaction processing system)被广泛地应用于人们的日常生活中. 在银行进行的存取款操作、在商店购物结账、预订火车票或飞机票、旅店预订、网上购物、股票交易、信用卡交易等都是典型的事物处理应用. 从关系型数据库系统诞生至今,事物处理应用一直是传统的商用数据库系统最主要的应用之一. 进入大数据时代,许多 Web 2.0 应用的前端都符合事物处理系统的特征.

事物处理系统需要支持大量的并发用户,而每个用户操作所读写的数据只占系统的整个数据集的极少一部分,随机地分布在整个数据集中. 以银行存取款为例,有成千上万的客户在不同城市和地区的银行支行、ATM 机上进行存取款操作. 每个客户对自己的银行账户进行操作,对账户余额进行增减. 相比于银行所有账户所组成的整个数据库而言,客户的个人帐户只是整个数据集的极小一部分,是随机分布在整个数据集中的. 在大数据时代,随着网络的普及,事物处理应用的规模也在加大. 例如,Amazon 网上商店截至 2013 年底共有 2.37 亿活跃的注册用户. 在国内,淘宝、京东商城、1 号店、携程等网上商店都有大量的用户访问. 2013 年"双十一"阿里公司交易笔数超过 1 亿笔. 铁道部官方售票网站 12306.cn 在售票高峰时,每秒就有 20 万人同时在线,网站日登录数量突破 1700 万次,日点击高达 15 亿次. 在这些事务处理类应用中,用户的访问表现为大量的并发访问和随机的读写请求.

在事物处理系统中,velocity 是系统设计的核心目标,即在对每个用户的事务保证正确性和实时性的前提下,支持尽可能高的并发事务数量. 对于关系型数据库系统,TPC-C 和 TPC-E 是专门衡量事物处理效率的基准测试,它们都以并发事物数量为主要的性能指标[6].

近年来,大数据事物处理系统如何支持事物处理的原子性、一致性、隔离性、持久性(atomicity, consistency, isolation, durability, ACID)特性成为一个重要的课题. 许多分布式键值系统(例如,BigTable[7], Dynamo[8], Cassandra[9])为了追求 velocity 而简化了

设计.它们不支持 ACID,只对同一个键的读写操作保证一致性,而把不同键的读写一致性交给上层应用来实现.虽然这种简化对于这些系统最初所支持的专门应用而言降低了实现成本,提高了性能,但是为了实现更多的应用或者提供通用的服务,应用程序员需要付出更多的努力才能正确地实现应用.于是,ACID 成为分布式大数据管理系统中一个日益迫切需要解决的问题,成为许多研究工作的重点[10-12].

1.2 数据流系统

数据流系统(data stream system)与事物处理系统不同,其主要任务是分析流过系统的数据.数据流过系统,而不是存储在系统中.系统分析流过的数据,在每条流过的数据上,计算事先定义好的查询运算,例如统计、异常检测、复杂事件处理等.系统的运算是连续不断地进行的,由于不需要存储,所以流过的数据量理论上可以是无限的.

数据流系统出现于 2000 年代[13-14],被应用于电信流量监控、交通情况分析等.在大数据时代,分布式数据流系统(例如 Storm[15])广泛地用于支持社交网络、电子商务等应用.

与事务处理系统相似,velocity 也是数据流系统设计的核心目标.数据流系统希望尽可能提高系统的吞吐率,即单位时间处理的数据条数.数据流系统通常在内存中实现数据处理,一方面单位时间流过的数据量通常可以放入内存,另一方面内存处理可以实现更高的吞吐率.

1.3 大数据分析系统

大数据分析是实现大数据价值的重要途径.通过分析可以总结大数据中出现的规律,从而更好地理解现实,预测未来,实现基于数据的决策.关系型数据的数据分析系统——数据仓库——出现于 20 世纪 90 年代,新兴的大数据分析系统包括 MapReduce[16]、图处理系统[17]、日志处理系统[5]等.

大数据分析系统与事物处理系统和数据流系统很不相同.相比于事物处理系统,大数据分析系统通常只有少量的并发用户,例如企业的数据分析师、管理人员或者其他需要对数据进行分析的决策人员,其并发用户数量远远小于事物处理系统.而且每个用户进行的数据处理操作不是少量的随机读写操作,而是对大量数据的读操作.相比于数据流系统,大数据分析系统主要处理存储的数据,而不是处理流过系统的数据.由于数据不一定全部放入内存,大部分系统需要从外存读取数据.

如前所述,velocity 是事物处理系统和数据流系统的核心设计目标,因此针对这两类系统已经存在许多关于 velocity 的研究.与此呈对比的是,大数据分析系统在设计时可能只针对读操作进行了特殊的优化,而没有把 velocity 放在同等的地位进行考虑.

我们认为在大数据时代 velocity 的重要性越来越明显.数据是不断地产生、收集和加载到大数据分析系统中的,如图 1 所示.在静态数据上设计和优化的数据分析操作,一方面难以反映最新的数据,不适合许多在线应用的需求,另一方面可能受到数据更新操作的干扰,无法实现最佳的性能.因此,我们需要在大数据分析系统的设计中,不仅仅专注于大数据分析操作本身,而是把大数据从更新到分析作为数据的生命周期来对待,把 velocity 作为重要的考虑因素,体现在系统的设计中.

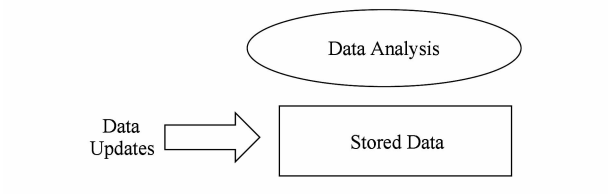

Fig. 1 Big data analysis system and high speed data updates.

图 1 大数据分析系统与高速数据更新

为了支持数据更新,最基本的要求是能够存储新到来的数据,可是这还远远不够.下面我们以近期两项工作[4-5]为基础,探讨如何有效地进行数据更新以保证甚至改善大数据分析的效率.

2 在数据仓库系统中支持在线数据更新

图 2 展示了一个典型的数据仓库系统的运行环境.数据仓库系统运行在后端,前端是各种数据源,

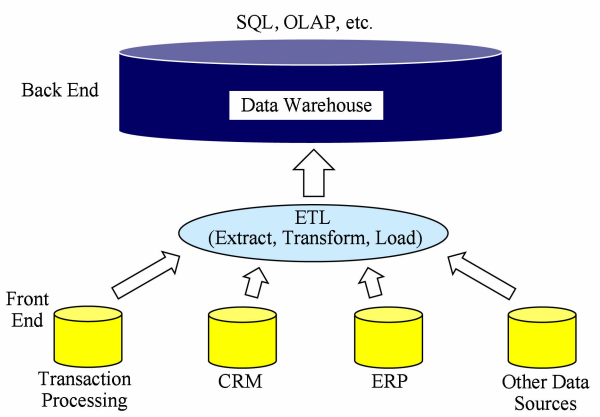

Fig. 2 Data warehouse system.

图 2 数据仓库系统

如事务处理系统、CRM(客户关系管理系统)、ERP(企业资源计划系统)等. 前端系统的数据通过 ETL(提取、转换、载入)过程,更新后端的数据仓库系统. 数据仓库是关系型的数据分析系统,对前端收集的数据进行归档整理,支持各种数据分析操作,例如 OLAP 等.

数据仓库传统的更新方式是在夜间进行离线更新. 在白天工作时间,数据分析人员使用数据仓库执行各种数据分析操作. 晚上数据分析人员下班后,系统管理员禁止数据分析操作,进行批量离线的数据更新. 这样,数据分析和数据更新分时地访问数据仓库系统,隔离了相互的影响.

但是,这种离线的数据更新方式存在两大问题. 一方面,它牺牲了数据的新鲜性. 数据分析看到的数据是前一天的,所以无法根据新的数据立即决策,不能满足新兴的在线服务的需求. 另一方面,越来越多的公司在全球拓展业务,没有明确的"夜间"的概念. 中国的夜间是美国的白天,而美国的夜间是中国的白天. 所以,很难找到一段足够长的时间,关闭数据分析操作,完全隔离地进行离线批量数据更新.

因此,我们的研究目标是在数据仓库系统中支持在线数据更新,在完成数据更新的同时,尽量降低数据更新对数据分析性能的影响.

2.1 传统的在线更新对数据分析操作的影响

在线更新可能对数据分析操作产生怎样的影响呢? 分析大量数据的查询操作以范围读取(range scan)为主[18],在硬盘上表现为顺序读操作. 典型硬盘的顺序读性能可以达到单块硬盘 100 MBps. 而传统的数据更新直接对数据页进行数据记录插入、修改和删除操作. 数据访问符合前端系统(例如事务处理系统)的访问特征,基本上是随机的. 由于机械部件的限制,硬盘只能够支持每秒钟大约 100 次的随机访问. 如果一次访问的数据量是 4 KB,那么随机访问的性能就是 400 KBps,比顺序访问低了两个数量级. 更糟糕的是,同时进行的随机访问干扰了良好的顺序访问的行为. 硬盘的磁头需要不断地移动,执行随机访问的操作,然后再返回继续顺序访问. 这种磁头的反复移动进一步降低了数据分析的性能.

我们通过一组实验验证上述分析,如图 3 所示. 我们运行 TPC-H[6]查询操作,图 3(a)和图 3(b)分别是在一个商用的行式数据库和一个商用的列式数据库上测量得到的结果. 纵轴是归一化的运行时间. "1"代表在没有数据更新的情况下查询操作的执行时间. 这个时间也同时由每组结果中左侧的柱状条显示. 每组结果中,中间的柱状条显示了在后台进行传统的在线更新的情况下查询操作的执行时间. 实验结果表明传统的在线更新使查询操作的性能在行式和列式数据库中分别平均降低了 2.2 和 2.6 倍. 我们进一步在图 3(a)中显示了右侧白色的柱状条,它是单独进行查询操作的时间与单独完成数据更新的时间的加和. 可以看出,右侧的柱状条显著低于中间的柱状条,这表明在线的数据更新操作和数据分

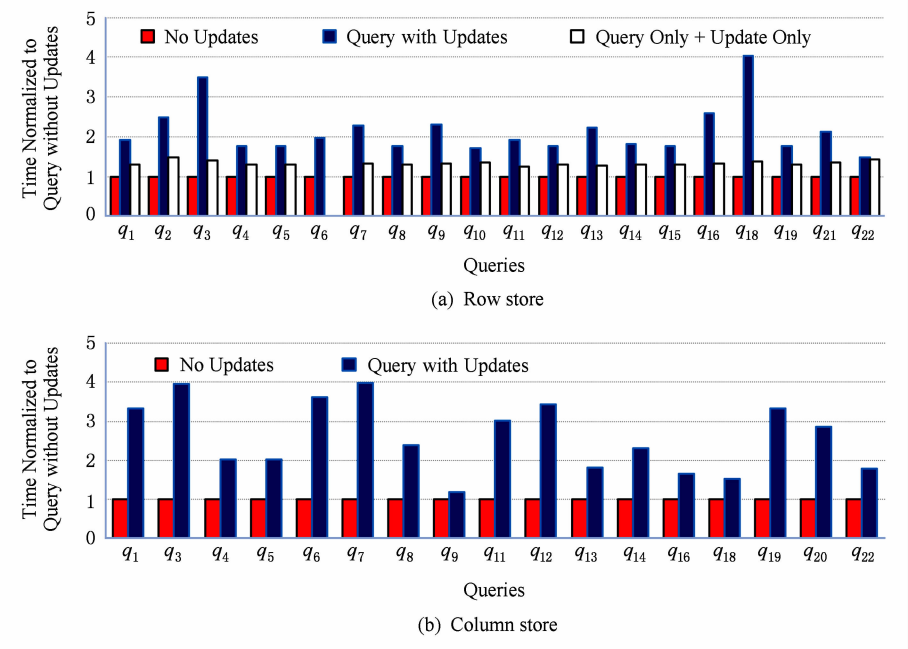

Fig. 3 Impact of traditional online data updates on the performance of TPCH data analysis queries.

图 3 传统的在线数据更新对 TPCH 数据分析操作的影响

析查询操作之间存在着相互的干扰,这种互相干扰引起了性能的进一步下降.那么如何才能够保证数据分析的效率呢?

2.2 在线更新的设计思路和设计目标

图 4 展示了我们的数据仓库在线更新设计思路.图 4 中有内存、硬盘和固态硬盘 3 种设备.数据仓库的主数据仍然存放在硬盘中.我们在系统中添加少量的固态硬盘,用以暂存新到来数据更新记录,利用固态硬盘良好的读性能降低在线更新对查询操作的影响.固态硬盘的容量只需为硬盘容量的 1%,所以其成本相对较低.

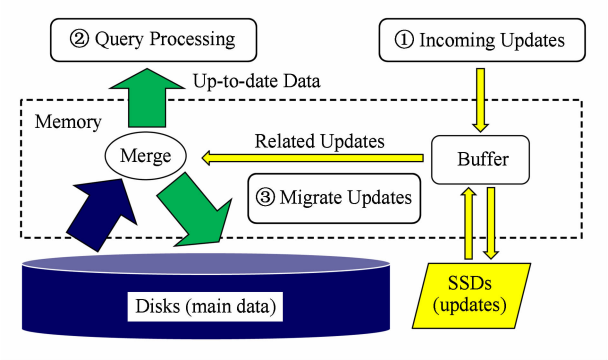

Fig. 4 Our design to support online data updates.

图 4 在线更新的设计思路

进入系统的每条在线数据更新记录包含需要更新的数据记录的主键、操作(插入/修改/删除)以及具体的更新数值(插入的值/修改后的新值).我们首先在内存中缓冲到来的数据更新记录.当缓冲区满时,我们把缓冲的更新记录写入固态硬盘.查询操作一方面从硬盘中读入(有些过时的)主数据,另一方面从固态硬盘中读入与需要的主数据相关的更新记录.在内存中实时地归并这两部分数据,在读入的主数据上完成更新记录所指明的插入、修改和删除操作,产生最新的数据.这样,上层的查询运算就能够处理最新的数据.当固态硬盘将满或者积累的数据更新超过事先设制的容量限制时,我们需要迁移数据更新,即更新硬盘上的主数据,并清除固态硬盘中的数据更新记录.这个操作可以利用现有的归并机制,把归并的结果写回硬盘.

为了实现上述设计方案,我们提出以下 5 个设计目标.本文将重点叙述如何实现前 3 个目标,后 2 个目标的具体实现请详见文献[4].

1) 对查询的影响小.这是在线更新的主要设计目标.在算法设计中,我们将利用固态硬盘的特性,减少在线更新对查询操作的影响.

2) 内存占用少.可用内存的大小影响着数据分析运算的性能.一方面,内存可以用于缓冲数据,从而减少 I/O 操作.另一方面,以排序或者哈希为基础的算法,对于不同的内存大小,算法性能可能有很大的变化.当数据可以完全放入内存中时,这些算法通常只需要读一次数据.可是当数据超出内存容量时,这些算法通常需要一个额外的数据划分步骤带来额外的开销.所以,我们希望设计的算法尽可能地减少对内存的占用.

3) 适合固态硬盘的特性.与普通硬盘相比,固态硬盘没有机械移动部件,所以其随机读性能可以达到普通硬盘的 100 倍或更高.固态硬盘的顺序读和顺序写的性能也要高于普通硬盘.但是,固态硬盘中的闪存只支持有限次数(例如,5 000~10 000 次)的擦除操作,整个固态硬盘写数据量超过(可擦除次数×容量)就会报废.另一方面,固态硬盘的随机写性能很差,而且经过一段时间的随机写操作,固态硬盘各方面的性能都会降低.所以,我们要尽可能减少对固态硬盘的写操作,同时避免随机写操作.

4) 高效的迁移操作.在时间方面,由于已经积累了大量的数据更新记录,可能每个主数据页中都存在一定的更新,所以迁移操作可以进行顺序写操作,而不是随机写.在空间方面,数据迁移可以一部分一部分地进行,只使用少量的额外的硬盘空间,而不需要占用主数据两倍大小的空间.

5) 支持 ACID.这是保证数据读写一致性所需要的.思路是在数据更新记录、主数据页和查询操作上增加时间戳,通过比较时间戳,判断数据更新记录是否已经被迁移而改变了相关的主数据页,以及查询操作是否应该看到数据更新.

2.3 MaSM 算法

我们提出了一组新的数据仓库在线更新算法 MaSM(materialized sort merge).该算法基于 LSM (log structured merge tree)[19]的基本思路,把在线更新存储在内存和固态硬盘两层的数据结构中.在归并操作时,我们需要把数据更新记录按照主键的顺序进行排序.但是,每个查询操作都进行一次外存排序显然会引起较大的代价.实际上,多次查询所需要的数据更新的排序结果基本相同,不同的部分只是新增的数据更新记录.于是,我们可以简化外存排序,固化(materialize)和重复利用排序产生的中间结果(sorted run).算法的名称由此产生.

图 5 展示了 MaSM-2M 算法.其中 M 是一个参数.当固态硬盘容量是 SSD 个数据页时,$M=\sqrt{SSD}$.算法需要使用 $2M$ 个内存页.如图 5 所示,

其中 M 个数据页用于在内存中缓冲新到来的数据更新记录,另 M 个数据页用于提供内存缓冲以支持查询运算操作.

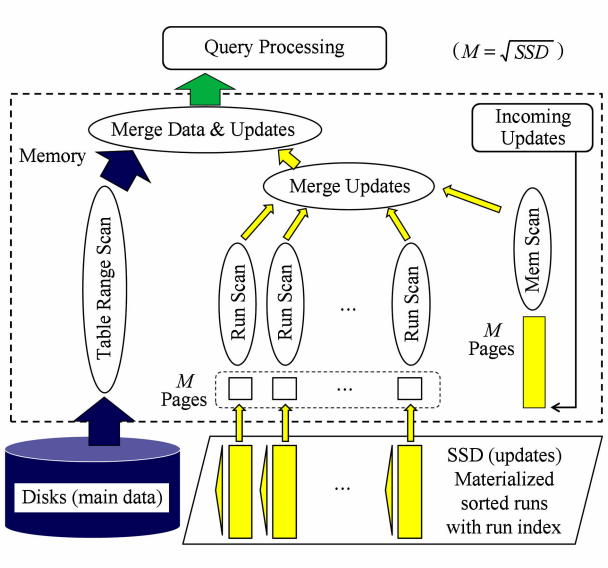

Fig. 5 MaSM-2M algorithm.

图 5 MaSM-2M 算法

到来的数据更新记录被缓存在内存的 M 个数据页的缓冲区中.当缓冲区满时,算法对缓冲区中的数据更新记录按照主键进行排序(我们在这里集中讨论一个数据表上的更新操作,多个数据表的更新处理与此类似).然后,把排序后的数据更新记录写入固态硬盘,形成一个新的文件,即一个新的 materialized sorted run(MSRun).MSRun 创建之后不再修改,直到迁移操作后被删除.产生 MSRun 的同时,算法生成一个静态的索引,包含 MSRun 中每个数据页的第 1 个数据更新记录的主键.

因为固态硬盘的容量为 $SSD=M^2$,每个 MSRun 的大小为 M,所以在固态硬盘中至多有 M 个 MSRun 文件.在查询操作时,需要从每个 MSRun 文件中读取数据更新记录,所以需要为每个 MSRun 分配一个内存数据页作为缓冲,至多需要 M 个数据页.

对于指定主键范围的查询操作,算法创建一个 table range scan 操作运算部件.MSRun 文件是按照时间顺序生成的,所以与这个范围查询相关的数据更新记录可能存在于每个 MSRun 文件中.算法在每个 MSRun 文件上创建一个 Run scan 运算部件.使用 MSRun 的静态索引可以查找主键范围,把数据更新记录的读取范围精确到固态硬盘数据页.此外,对于内存缓冲区中的相关更新记录,也要通过一个 memory scan 运算部件读取.于是,在读取主数据的同时算法读取数据更新记录,经过归并操作就可以获得最新的数据.

MaSM-2M 算法基本实现了我们在 2.2 节中提出的设计目标.对于目标 3,MaSM-2M 对固态硬盘的写操作只用于生成 MSRun 及其索引.这是顺序写,而且每个数据更新只在固态硬盘上写了一次.

对于目标 1,我们分两种情况进行讨论.当主键区间范围很大时(例如占整个表的 50%),查询操作将对数据表进行大量的顺序读操作.与此相应,查询操作也将读取相应比例(例如,50%)的数据更新记录.因为固态硬盘容量是主硬盘的 1%,那么从固态硬盘中读取的数据也大约为主硬盘的 1%,而固态硬盘的顺序读性能高于普通硬盘,所以数据更新的读操作完全可以被主数据的读操作所覆盖,查询性能几乎没有改变.另一种情况是区间范围很小时,例如只读取一个数据页,这种情况可能在使用二级索引时遇到.那么,每个 MSRun 文件根据静态索引都可能找到一个数据更新页.所以,查询操作需要从主硬盘读一个数据页,在固态硬盘上读至多 M 个数据页.这里,因为固态硬盘的随机读性能是硬盘的 100 倍,我们可以支持同时读取 100 个 MSRun 文件而不改变查询操作的性能.

对于目标 2,内存占用是固态硬盘大小平方根的 2 倍,这已经比较小了.我们可以进一步减少内存占用,上述 MaSM-2M 算法实际上是 MaSM-αM 算法的特例 $\left(\frac{2}{\sqrt[3]{M}}\leqslant\alpha\leqslant 2\right)$.图 6 显示了内存占用与固态硬盘写次数的关系.例如,当 $M=1000$,可以减少内存占用为 $0.2M$,而固态硬盘的写次数只增加一倍.

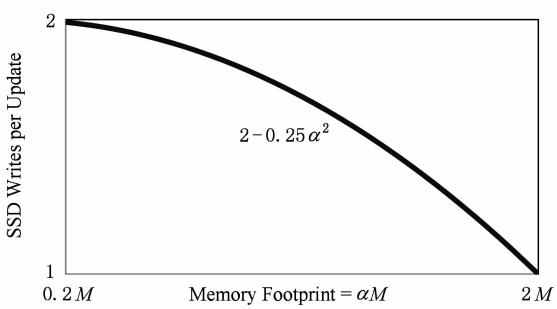

Fig. 6 Analysis of MaSM-αM algorithms.

图 6 MaSM-αM 算法分析

2.4 实验结果

我们实现了一个数据仓库原型系统,支持传统在线更新和 MaSM 在线更新.我们在原型系统上测试 MaSM 在线更新的效果.首先,我们在商用的行式数据库系统上运行 TPCH 查询操作,并记录硬盘数据块的访问踪迹数据(trace).通过分析 TPCH 数

据表的存储位置,把数据块访问对应回数据表,从而把整个查询操作分解成对多个数据表的范围读取操作.然后,我们在原型系统上建立相同的 TPCH 数据表,并按照记录的硬盘数据块访问踪迹,重复数据表上的范围读取操作.

图 7 比较了 3 种情况的查询时间.左面的柱状条是没有在线更新时查询操作的时间,中间的柱状条是在传统数据更新操作时查询操作的时间,右面的柱状条是 MaSM 在线更新的同时查询操作的执行时间.可以看出,传统的在线更新使查询操作产生了很大的性能降低,而 MaSM 对查询操作只有不到 1%的影响.

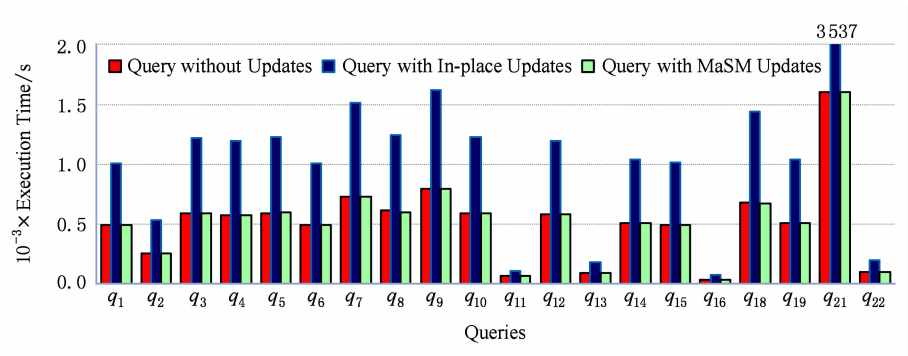

Fig. 7 TPCH performance with MaSM online updates.

图 7 MaSM 算法 TPCH 性能

3 LogKV:高性能日志处理系统

日志记录广泛地存在于软硬件系统中.许多硬件设备(包括网络路由器、各种传感器)和软件系统(包括 Web 服务器、数据库系统、操作系统、多种应用软件)都会产生日志记录,按时间顺序记录运行状态、测量数据等.例如,Weblog、微博、系统日志、传感器数据等都是日志数据.

在收集存储日志数据的基础上,日志处理系统可以支持多种重要的应用,例如安全管理、故障排查和用户行为分析等.安全管理可以监控局域网的机器设备的异常情况,从而发现网络攻击等安全异常.故障排查可以通过分析日志数据,帮助寻找故障的根源.用户行为分析可以分析用户的网页浏览数据,推测用户的偏好,提供个性化的服务.

我们希望设计一个高性能的日志处理系统,可以达到 100 TB/d(即平均 1.2 GBps)的日志获取能力,并且可以高效地支持相对复杂的查询运算.有的查询运算(例如过滤、统计等),可以很容易地在多台机器的数据集上并行执行.而我们的目标是支持基于时间窗口的连接操作.连接操作很可能引起机器节点间大量的数据通信,影响执行效率.例如 Web 用户行为分析,大型网站通常采用大量 Web 服务器组成机群,并由前端的负载均衡器把 Web 请求随机分发给机群中的 Web 服务器执行.同一用户的多次访问可能被多台服务器分别记录.那么,用户行为分析需要对所有用户找到其所有的访问记录,这个操作会导致大量的网络通信,可能成为性能的瓶颈.

于是,我们的设计目标是在获取日志记录的同时,把同一段时间的日志记录分布在同一个机器节点上,从而减少或避免上述性能问题.

3.1 LogKV 系统结构

因为键值系统(key value stores)可以灵活地表达多种类型的日志记录信息,而且可以提供可靠的高性能的分布式存储[7-9],我们采用键值存储系统作为日志数据的底层存储.

我们设计了一个高性能的日志处理系统 LogKV,如图 8 所示.图 8 的下面部分展示了多种日志数据源,上面部分展示了多种查询操作,其中基于时间窗口的连接操作是我们支持的重点.中间部分

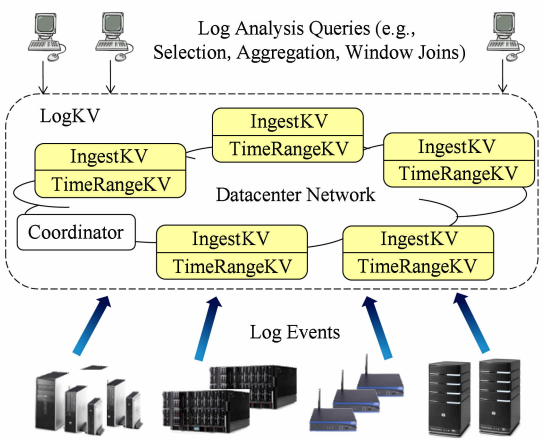

Fig. 8 LogKV system architecture.

图 8 LogKV 系统结构

是 LogKV 系统. 系统由一个协调管理节点和多个工作节点通过数据中心内网连接组成. 每个工作节点运行两个子系统: IngestKV 是基于内存的键值存储子系统,用于日志数据的收集和暂时缓冲; TimeRangeKV 存储全部日志数据,并实现系统的设计目标,即把同一段时间的日志记录分布在同一个机器节点上.

图 9 展示了 LogKV 的主要操作. 首先,LogKV 需要把日志数据源映射(mapping)到 IngestKV 节点,由 IngestKV 收集日志数据源的数据. 其次,要把日志数据从 IngestKV 重新分布(shuffling)到 TimeRangeKV 中,实现同一段时间的日志记录分布在同一个机器节点的目标. 此外,需要实现查询操作,需要考虑在 TimeRangeKV 中如何存储数据以提高空间利用率和查询效率,需要考虑系统的可靠性,保证在机器节点失效的情况下数据不丢失,并且可以继续提供查询服务. 在本文中,我们集中讨论前两方面的内容,其他操作的实现详见文献[5].

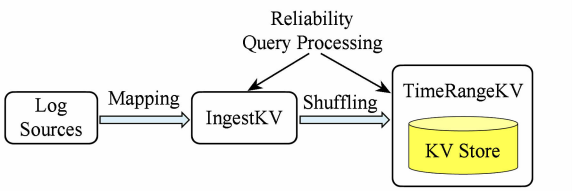

Fig. 9 LogKV main operations.

图 9 LogKV 的主要操作

3.2 从日志数据源到 IngestKV 的映射

我们希望尽可能地均衡 IngestKV 节点之间的日志数据流量. 从日志获取方法的角度,日志数据源有以下 3 种类型: 1) 日志数据源可以运行 LogKV 的代理进程,直接收集日志数据,通过网络发送到指定的一个或多个 IngestKV; 2) 日志数据源可以配置远程的服务器和端口,自动地发送日志数据到指定的服务器和端口; 3) 日志数据源把数据写入本地文件中,IngestKV 需要通过文件传输协议(如 FTP, SCP, SFTP)从日志数据源获得日志数据文件.

我们称第 1 种日志数据源为可分割的日志数据源,后两种为不可分割的日志数据源. 可分割的日志数据源可以根据系统各节点的负载分配到多个 IngestKV 上,比较灵活. 而每个不可分割的日志数据源只能映射到一个且仅一个 IngestKV 节点.

我们首先映射不可分割的日志数据源. 这是一个 NP 问题,我们设计了下面的贪心算法.

算法 1. 不可分割的日志数据源映射算法.

第 1 步: 对不可分割的日志数据源,按照日志数据流量从大到小的顺序进行排序;

第 2 步: 进入循环,依流量从大到小的顺序,每次循环确定下一个日志数据源的映射关系;

第 3 步: 在循环体中,找到已经分配的日志数据流量负载最轻的 IngestKV 节点,把本次循环需要确定映射的日志数据源分配给 IngestKV;

第 4 步: 重复第 3 步,直至所有不可分割的日志数据源的映射都已经确定为止.

可以证明,算法 1 保证: 单个 IngestKV 被分配的日志数据流量小于 average(日志数据流量)与 max(不可分割日志数据流量)之和.

在分配完不可分割的日志数据源后,我们对可分割的日志数据源进一步分配,尽可能地均衡负载.

3.3 从 IngestKV 到 TimeRangeKV 的分布

我们把整个时间轴切成定长的时间段. 设时间段的长度为 TRU(time range unit). TRU 是根据日志分析中的时间窗口的大小而确定的. 设 LogKV 系统中一共有 N 个工作节点. 我们把时段的所有权依次循环地分配给各个工作节点:

$$\text{TimeRangeKV node } ID = \left\lfloor \frac{TimesStamp}{TRU} \right\rfloor \bmod N.$$

给定一个日志记录,可以用上式计算此记录应该所属的 TimeRangeKV 节点. 于是,日志数据分布的目标是发送 IngestKV 中缓存的日志记录到其所属的 TimeRangeKV 上. 如何高效地实现呢?

一个最先想到的解决方案就是采用双缓冲机制实现数据的分布操作. 在 IngestKV 中保持 2 个缓冲区 A 和 B,其中缓冲区 A 接收当前 TRU 的日志记录,而缓冲区 B 保存着上一个 TRU 的日志记录. 在接收当前 TRU 日志记录的同时,我们把缓冲区 B 中的日志记录发送到其对应的 TimeRangeKV 节点上. 每个 TRU 交换一次 A 和 B.

仔细分析这个方案就会发现,实际上一个 TRU 的所有日志记录都对应于同一个 TimeRangeKV. 所以,系统中所有的 IngestKV 节点都会同时向同一个 TimeRangeKV 节点发送大量的日志记录. 这个 TimeRangeKV 就成为性能瓶颈,如图 10(a)所示.

如何能够避免性能瓶颈呢? 我们的解决方案是积累 M 个时段的数据,同时传输 M 个时段的数据,如图 10(b)所示. 于是,分布操作就有了 M 个目标节点. 我们设计了一个随机传输算法. 每个目标 TimeRangeKV 节点建立一个需要接收日志数据的 IngestKV 节点列表,每次随机地选择一个列表中的

节点接收数据,完成后将这个节点从列表中删除,然后再随机地选择下一个 IngestKV 节点接收数据,重复这个过程,直至接收完毕.运行这个随机算法,不同的目标 TimeRangeKV 节点很可能同时从不同的 IngestKV 节点接收数据,从而避免了瓶颈.

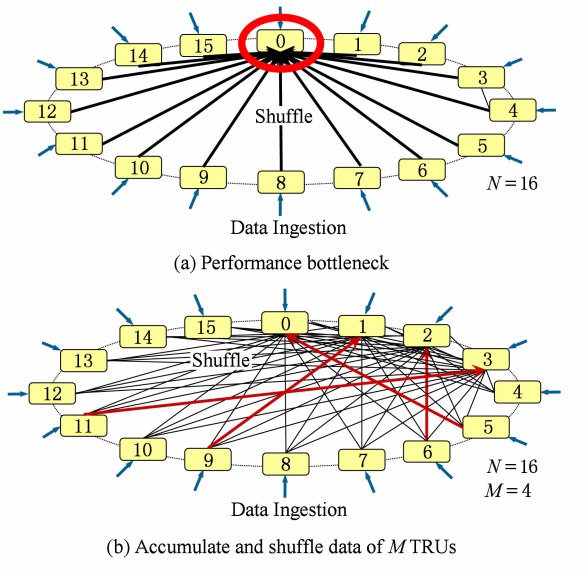

Fig. 10 Log shuffling algorithm.

图 10 日志分布算法

3.4 实验结果

我们基于 Cassandra 键值系统实现了 LogKV.图 11 展示了 LogKV 系统日志数据流量的测试结果.实验使用了 20 个刀片服务器组成的机群.图 11 中横轴是 LogKV 工作节点的数量,从 1～20.在数据重新分布的过程中,所有的工作节点都是目标节点,即 $N=M$.纵轴是日志数据流量.其中,日志记录的平均长度为 100 B.可以看出,随着工作节点的增加,系统性能呈线性增长.根据测量结果,可以估计 N 个节点的 LogKV 系统可以支持的日志数据流量是 $28N$ MBps.那么,100 TB/d(即 1.2 GBps)的设计目标,可采用大约 43 个工作节点就能实现.

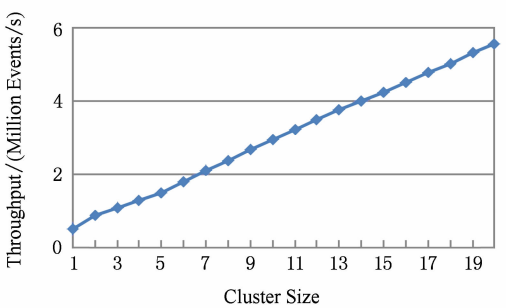

Fig. 11 LogKV data ingestion performance($N=M$).

图 11 LogKV 系统的日志输入流量($N=M$)

图 12 针对基于时间窗口的连接操作比较了 3 种实现的性能:1)Cassandra,数据存储于 Cassandra 键值系统中;2)HDFS,数据存储于 HDFS 分布式文件系统中;3)LogKV,数据存储于 LogKV 中,已经完成了重新分布.我们看到,因为避免了大量的日志数据的传输交换操作,LogKV 的性能比 Cassandra 和 HDFS 大幅度提高了 15 倍和 11 倍.

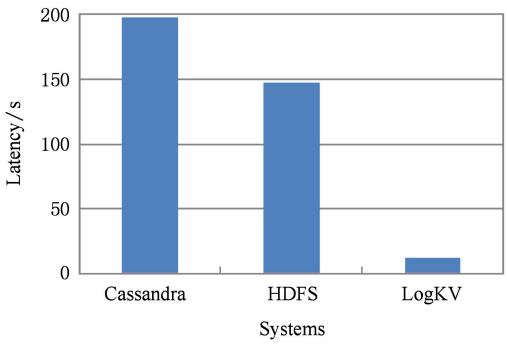

Fig. 12 Time window join performance comparison.

图 12 基于时间窗口的连接操作性能比较

4 比较与总结

通过比较上述两项工作可以看出,MaSM 和 LogKV 都没有停留于简单地存储数据更新,而是把数据更新操作作为整个大数据分析过程的一个有机组成部分,放到与数据分析操作同样重要的地位进行考虑,针对系统各自支持的大数据分析查询操作的特点,本文设计了适合的数据 velocity 方案.

MaSM 和 LogKV 所支持的查询分析操作都不是事先预设的,不是完全确定的,而是即兴的(Ad-hoc),但各自具有一定的特点.具体而言,数据仓库的查询分析操作以范围读取为主,而日志处理系统中基于时间窗口的连接操作是支持的重点.所以,两个系统的 velocity 设计方案都是针对其各自查询分析操作的特点而形成的.

MaSM 和 LogKV 的具体解决方案存在很大的区别,原因如下:1)MaSM 面向单机运行环境,而 LogKV 面向分布式系统.于是,MaSM 主要关注单机中 I/O 操作的效率,而 LogKV 关注机器节点之间的通信代价.2)在数据仓库中,范围查询的顺序读数据操作对于硬盘访问已经达到最优,所以 MaSM 的设计目标是保持好的查询性能.而在日志处理系统中,现有系统对于基于时间窗口的连接操作支持得并不理想,可能引起很大的数据交换的开销,所以 LogKV 的设计目标是通过更好的数据分布来优化

基于时间窗口的连接操作的效率. 3)MaSM 所支持的数据更新操作包含插入、修改和删除,所以需要引入排序并与主数据归并. 而 LogKV 的日志数据获取是单纯的追加操作,所以不需要修改已有的数据,也不需要更加复杂的数据归并操作.

总结全文,我们再次强调 velocity 对于大数据分析系统是非常重要的,在系统设计中,应该把大数据从更新到分析作为数据的整个生命周期,进行综合地考虑和优化. 从 velocity 维度进行思考,可以对大数据分析系统产生全新的、更加深刻的认识,引出许多值得探索的课题. 在不同的应用场景下,大数据分析系统具体的查询分析操作可能多种多样,数据更新操作也可能各具特点,需要具体问题具体分析,设计最适合的 velocity 方案来更好地支持大数据分析.

参 考 文 献

[1] Abadi D, Agrawal R, Ailamaki A, et al. The beckman report on database research [R]. (2013-10-15) [2014-11-30]. http://beckman.cs.wisc.edu/

[2] Li Guojie, Cheng Xueqi. Research status and scientific thinking of big data [J]. Strategy & Policy Decision Research, 2012, 27(6): 647-657 (in Chinese)
(李国杰, 程学旗. 大数据研究:未来科技及经济社会发展的重大战略领域——大数据的研究现状与科学思考[J]. 中国科学院战略与决策研究, 2012, 27(6): 647-657)

[3] International Data Corporation. EMC digital universe study with research and analysis by IDC [R]. 2014 [2014-11-30]. http://www.emc.com/leadership/digital-universe/index.htm

[4] Athanassoulis M, Chen S, Ailamaki A, et al. MaSM: Efficient online updates in data warehouses [C] //Proc of the SIGMOD Int Conf on Management of Data. New York: ACM, 2011: 865-876

[5] Cao Z, Chen S, Li F, et al. LogKV: Exploiting Key-Value Stores for Event Log Processing [C/OL] //Proc of the 6th Biennial Conf on Innovative Data Systems Research. 2013 [2014-11-30]. http://www.cidrdb.org

[6] Transaction Processing Council. [2014-11-30]. http://www.tpc.org

[7] Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data [C] //Proc of USENIX Symp on Operating System Design and Implementation. Berkeley, CA: USENIX, 2006: 205-218

[8] DeCandia G, Hastorun D, Jampani M, et al. Dynamo: Amazon's highly available key-value store [C] //Proc of ACM Symp on Operating Systems Principles. New York: ACM, 2007: 205-220

[9] Lakshman A, Malik P. Cassandra: A decentralized structured storage system [J]. Operating Systems Review, 2010, 44(2): 35-40

[10] Baker J, Bond C, Corbett J, et al. Megastore: Providing scalable, highly available storage for interactive services [C] //Proc of the 5th Biennial Conf on Innovative Data Systems Research. 2011: 223-234. [2014-11-30]. http://www.cidrdb.org

[11] Mahmoud H, Arora V, Nawab F, et al. MaaT: Effective and scalable coordination of distributed transactions in the cloud [J]. PVLDB, 2014, 7(5): 329-340

[12] Corbett J, Dean J, Epstein M, et al. Spanner: Google's globally distributed database [J]. ACM Trans on Computer System, 2013, 31(3): No.8

[13] Chen J, DeWitt D, Tian F, et al. NiagaraCQ: A scalable continuous query system for Internet databases [C] //Proc of the SIGMOD Int Conf on Management of Data. New York: ACM, 2000: 379-390

[14] Babu S, Widom J. Continuous queries over data streams [J]. SIGMOD Record, 2001, 30(3): 109-120

[15] Apache. Apache Storm. [2014-11-30]. http://storm.apache.org

[16] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters [C] //Proc of the 6th USENIX Symp on Operating System Design and Implementation. Berkeley, CA: USENIX, 2004: 137-150

[17] Malewicz G, Austern M, Bik A, et al. Pregel: A system for large-scale graph processing [C] //Proc of the SIGMOD Int Conf on Management of Data. New York: ACM, 2010: 135-146

[18] Becla J, Lim K. Report from the First Workshop on Extremely Large Databases (XLDB 2007) [J]. Data Science Journal, 2008, 7(23): 1-13

[19] O'Neil P, Cheng E, Gawlick D, et al. The Log-Structured Merge-Tree (LSM-Tree) [J]. Acta Informatica, 1996, 33(4): 351-385

Chen Shimin, born in 1973. Received his PhD in Carnegie Mellon University in 2005. Professor and PhD supervisor at the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include data management systems, big data processing, and computer architecture.