基于动态镜像的实时数据仓库存取预处理技术研究

毛莺池[1,2] 闵 伟[1] 接 青[1] 朱沥沥[1]

(河海大学计算机与信息学院 南京 211100)[1] (河海大学淮安研究院 淮安 223001)[2]

摘 要 实时数据仓库是数据仓库技术的重要分支,而实时数据查询和实时数据导入引发的查询竞争问题一直是实时数据仓库技术研究的重点之一。查询竞争问题严重影响了查询分析的精度和效率,还降低了数据仓库的性能。提出了一种在数据仓库外部构建动态存储区域的方法,它采用动态镜像技术,有效地缓解查询竞争问题。同时,为了提高实时 OLAP 上的查询分析操作的性能,提出了蝇量级物化方法及蝇量级物化下的表连接算法 FWMJoin(Fly-Weight Materialization Join)。基于 TPC-H 基准的实时数据仓库测试系统,针对动态镜像技术下的动态存储区域的 OLAP 性能进行分析与评估,并对实验结果进行总结。

关键词 查询竞争,动态镜像,实时数据仓库,联机在线分析

中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.12.029

Real-time Data Warehouse Pre-processing Based on Dynamic Mirror Replication

MAO Ying-chi[1,2] MIN Wei[1] JIE Qing[1] ZHU Li-li[1]

(College of Computer and Information, Hohai University, Nanjing 211100, China)[1]

(Huaian Research Institute of Hohai University, Huaian 223001, China)[2]

Abstract Real-time data warehouse is one of the important research field in the data management. Real-time data query and import can bring about the problem of query contention. Query contention will not only seriously affect the accuracy of query analysis, but also reduce the performance of the real-time data warehouse. In this paper, combining an external dynamic storage area, a dynamic mirror replication technology was proposed to effectively solve the query contention problem. Meanwhile, the fly-weight materialization method and the fly-weight materialization join algorithm were proposed to improve the query and analysis performance in the real-time OLAP. Based on the TPC-H benchmark, the proposed dynamic mirror replication technology was evaluated. The experimental results demonstrate the proposed solution can get better performance in terms of effectiveness.

Keywords Query contention, Dynamic mirror replication, Real-time data warehouse, OLAP

1 引言

近年来,电子信息数据在企业的运营中越来越重要,企业需要对电子信息数据进行高效、及时、精确的分析。传统的数据仓库通常只支持历史数据的分析与查询,不能实时捕获数据源中的变化,因为它们采用 ETL 工具周期性地从数据源中抽取数据,对其处理后加载到数据仓库,而数据抽取的频率通常为一个月一次、一周一次或者一天一次[1]。在实时数据仓库中,实时数据导入与实时数据查询的冲突将严重影响联机在线分析(On-Line Analysis Processing, OLAP)的精度和效率。

当前解决这个问题的方法有很多,包括提高数据库的性能、增加外部实时数据缓存[8]、即时(just in time)合并外部数据缓存信息、反向即时数据合并、实时分区、主动分区等等。但是上述方法在查询结果的精度和效率上不能兼顾。

为解决查询竞争并提高查询结果精度,本文提出一种动态镜像技术。动态镜像技术是在实时数据仓库的外部创建一个动态存储区域,源系统中数据导入数据仓库时,先在动态存储区域创建一个分区并在其中创建一个节点,同时将数据存入节点。当这部分数据同时有更新需求和查询需求时,则利用复制机制新建一个与存储该数据的节点物理结构、逻辑结构都相同的节点。新数据存入新的节点,当新数据导入完成后,再次进行查询,保证了查询结果的精度,且避免了查询竞争。

改进的实时数据仓库架构与传统数据仓库不同的是,改进架构的 ETL[2] 过程被分割为定期 ETL 与实时 ETL 两个部

到稿日期:2015-02-13 返修日期:2015-04-08 本文受国家自然科学基金项目(61272543, U1301252),国家科技支撑计划项目(2013BAB06B04),中国华能集团公司总部科技项目(HNKJ13-H17-04),云南省科技计划项目(2014GA007),中央高校基本科研业务费专项资金(2015B22214)资助。

毛莺池(1976—),女,博士,副教授,硕士生导师,CCF 会员,主要研究方向为分布式计算与并行处理、分布式数据管理等,E-mail:maoyingchi@gmail.com;闵 伟(1989—),男,硕士生,主要研究方向为分布式计算与并行处理、数据管理等;接 青(1989—),女,硕士生,主要研究方向为分布计算与并行处理;朱沥沥(1991—),女,硕士生,主要研究方向为分布式数据管理技术。

分。定期 ETL 是将数据源中的数据定期以批处理方式直接导入数据仓库，而实时 ETL 则运用改变数据捕获(CDC)[3]技术将源系统中变化数据加载到动态存储区域。经过查询、更新和删除等操作，当满足系统的触发条件后，这些数据才会以批处理方式载入数据仓库。

数据仓库的存储区域被分为实时数据存储区和历史数据存储区，实时数据查询和存储在实时数据存储区实现。实时数据仓库的更新查询调度模块能够根据用户的需要决定查询和更新等操作序列；数据质量模块在展示数据查询结果时生成一份报告，以便用户知道这些数据内容和新鲜度；查询控制模块实现在动态存储区域进行实时数据查询操作。对于历史数据查询，动态存储区域与数据仓库需要无缝连接，以保证查询结果的精度。

为了克服实时 OLAP(Real-Time OLAP，RTOLAP)查询中的多表连接和分组聚集的性能瓶颈，采用物化策略可以减少 RTOLAP 在动态存储区域中表连接操作访问基表和中间数据结构的次数。首先，RTOLAP 将查询分解为作用在维表和事实表上的子查询，子查询将生成〈key，value〉二元组；然后，扫描事实表一趟，利用事实表的外键映射直接定位相应二元组，完成多表连接或聚集操作。为了提高 RTOLAP 多表连接或聚集性能，本文提出蝇量级物化下的表连接算法 FWMJoin(Fly-Weight Materialization Join)。

本文采用的基于动态镜像的实时数据存取技术在不同的事务更新频率下，查询效率可靠，如果给予动态存储区域更大的内存空间则性能会更好。

本文采用基于 TPC-H 的实时数据仓库测试系统，从 OLAP 响应时间、查询精度方面对动态镜像技术、单镜像、无镜像方案进行比较与分析。实验结果表明，基于动态镜像的实时数据存取技术能有效解决查询竞争问题，提高数据存取效率和查询结果的精度。

2 相关工作

动态镜像是一种实时数据仓库存储预处理技术，主要用于解决数据仓库的查询竞争问题[9]。面对实时数据仓库中的查询与插入的竞争问题，近年来开展了许多研究工作，包括提高数据库的性能、增加外部实时数据缓存、即时(just in time)合并外部数据缓存信息、反向即时数据合并、实时分区、主动分区等。

(1)单独实时数据缓存方法是使用一种与数据仓库分离的外部缓存。外部数据缓存持续更新，数据仓库使用 ETL 工具以批处理模式进行数据更新，所有的实时数据或准实时数据的查询直接定位到外部的数据缓存，从而避免了在数据仓库中的查询竞争问题[7]，但如果数量巨大的复杂查询分析运行在外部实时数据缓存，则同样会发生在数据仓库中出现的查询竞争问题。

(2)简化和限制实时报表方法，需要实时数据的用户只能发出简单的查询要求，限制复杂查询语句。这种方法可以消除查询竞争，但是无法满足用户对复杂查询的要求。

(3)升级硬件，可以为高端的 SMP 数据库系统增加更多的节点或者为数据仓库配备更快的处理器和更大的内存。这种方法能在短期内解决问题，但是增加了成本并且可扩展性低。

(4)反向即时数据合并[4]，就是把所需要的历史数据临时地反向加载到实时数据缓存中，查询在缓存中进行。这种方法可以有效解决查询竞争，但是查询结果的精度却不尽理想。

(5)实时分区[5]是将实时数据进行数据量均衡的分区，然后分别对各分区数据进行查询导入操作。这种方法有效缓解了查询竞争，但是关于分区的个数和数据量的均衡算法的研究一直未成熟，分区算法随着分区个数增加，其时间复杂度也线性增加，海量数据环境下给系统带来沉重的负担，难以满足实时性的要求[11]。还有主动分区[6]等技术，它们都难以满足用户的需要。

3 动态镜像存取预处理技术

当连续实时地导入数据时，用户的查询操作受到严重影响，包括查询结果的精度和查询效率。因此，实时数据仓库面临实时数据导入和实时数据查询相冲突的问题，即当 ETL 在向实时存储区加载数据时，用户如果正在对实时存储区进行多次查询，这几次查询都将纳入同一个统计结果，那么就要考虑是否将新到来的数据纳入结果中。如果不做任何处理，OLAP 查询结果的精度往往难以满足要求，需要设计一种新的结构来解决此问题。

本文设计一种基于动态镜像的实时数据仓库预存取技术，系统结构如图 1 所示。其包括 3 个部分：

(1)实时数据仓库改进，包括分类 ETL 结构和将数据仓库存储区域分为实时数据存储区和历史数据存储区。

(2)在数据仓库外部构建动态存储区域，动态存储区域由多个数据镜像与基于双重链接的镜像索引组成。

(3)动态镜像管理，包括镜像创建与回收、基于双重链接的镜像索引维护。

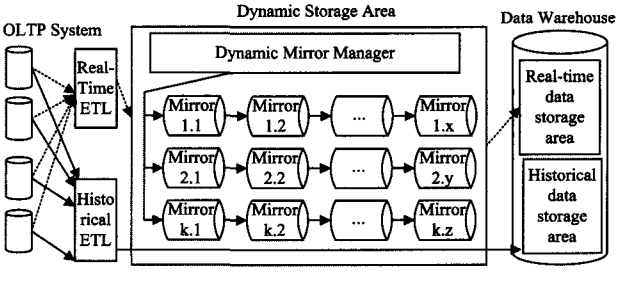

图 1 基于动态镜像的实时数据仓库预存取系统结构

3.1 数据仓库改进

首先是 ETL 分类。对于实时数据，数据在时间有效性范围之外(即失效)将成为历史数据。因此，我们对于实时数据的 ETL 处理同样是分为历史 ETL 和实时 ETL。实时 ETL 通过 CDC 技术捕获 OLTP 系统中查询任务提交时间之后更新的数据并加载到动态存储区域，在动态存储区域对数据进行查询、更新、删除，当满足系统触发条件后再以批处理方式存入到数据仓库中的实时数据存储区；历史 ETL 将 OLTP 系统中查询任务提交时间之前存在的数据以批处理方式存入数据仓库的历史数据存储区域。

其次是数据仓库存储区域分为实时数据存储区和历史数据存储区。历史数据是指由历史 ETL 将 OLTP 系统中的数据处理并存入数据仓库的历史数据存储区的数据；实时数据是指先由实时 ETL 将 OLTP 系统中的数据处理并存入动态存储区域，然后根据触发条件由动态存储区域存入数据仓库的实时数据存储区的数据。

3.2 动态存储区域

动态存储区域由多个数据镜像与基于双重链接的镜像索引组成。镜像是具有相同的逻辑结构和物理结构的数据存储区域，并根据数据查询任务的需求，在动态存储区中动态创建；系统可以将 OLTP 中的实时数据加载至镜像中。

当创建一个镜像时，系统在动态存储区中保存一个相应的镜像文件，用四元组表示：

$$\tau(image\_address, image\_size, data\_id, timestamp)$$

其中，image_address 表示镜像在动态存储区中的首地址，image_size表示镜像分配的存储空间大小，data_id 表示镜像存储的数据源，timestamp 表示数据的时间戳。

根据每个镜像文件中的 data_id，将所有 data_id 相同的镜像构建成一个镜像链表 Link_img，如图 2 所示。镜像链表 Link_img 由链表头节点 img_head 和链表节点 img_node 组成，链表头节点 img_head 由镜像数据源 data_id 与指向链表第一个节点的地址 head_next 组成。在一个镜像链表中，所有镜像的数据源来自同一数据源，数据源 data_id 相同。指向链表第一个节点的地址 head_next 存放第一个镜像首地址 image_address。根据镜像文件内容，链表节点 img_node 由镜像大小 image_size、镜像数据时间戳 timestamp、操作标识符 tag，及指向下一个链表节点的地址 img_next 组成。操作标识符 tag 用于记录当前镜像数据的操作类型，其初始值为 0；若当前镜像内容节点的数据是从源数据库系统 OLTP 导入至动态存储区，则此镜像内容节点的操作标识符置为 0；若当前镜像内容节点的数据需要从动态存储区批量加载至数据仓库的实时数据存储区，则操作标识符置为 1；对于当前镜像而言，若在动态存储区中，不存在来自同一数据源的镜像，则 img_next 设为空；否则，img_next 存放下一个来自同一数据源的镜像的首地址 image_address。

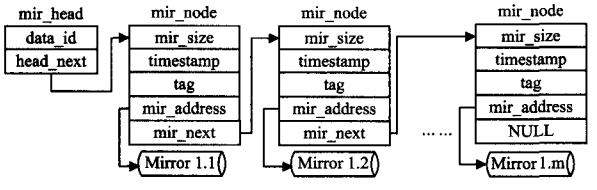

图 2 镜像链表结构

同一个镜像链表中存储了来自同一数据源但是更新时间不同的数据镜像信息；随着系统运行，最近更新数据的时间戳一定大于较早更新数据的时间戳，所以，镜像链表中的节点按其数据时间戳倒序（从大至小）排序。

所有来自同一数据源的镜像构成一个镜像链表，称之为一个镜像桶 bucket，如图 3 所示。其中镜像桶的首地址 bucket_address为链表头节点地址。

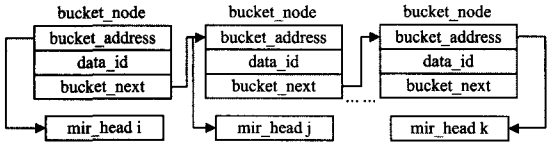

图 3 镜像桶链表结构

在动态存储区中，若存储了 n 个数据源的数据，就有 n 个镜像桶。为了加快镜像数据的查找与定位，将对多个镜像桶采用链表结构，构成一个镜像桶链表 Link_bucket。镜像桶链表 Link_bucket 是一个无链表头节点的链表，仅仅由镜像桶链表节点 bucket_node 构成。

每个镜像桶链表节点 bucket_node 由数据源 data_id、镜像桶的首地址 bucket_address 与指向下一个镜像桶链表节点的地址 bucket_next 组成。其中，数据源 data_id 存放对应的镜像链表的数据源 data_id；镜像桶的首地址 bucket_address 存放对应镜像链表头节点地址；向下一个镜像桶链表节点的地址 bucket_next 存放下一个镜像桶的地址 bucket_address；若动态存储区中不存在任何数据，即不存在任何数据源的镜像桶，则不存在镜像桶链表；在动态存储区中，若只有一个镜像桶，则其 bucket_next 为空；否则，bucket_next 存放下一个 bucket_address。

3.3 动态镜像与查询竞争

通过镜像管理模块可以得知在动态存储区域中哪些镜像用于查询而哪些镜像用于 ETL 的更新数据，这样由多个镜像构成的动态存储区域就可以实现持续数据的实时装载。这种动态镜像的方法不但有效地解决了查询竞争问题，同时也提高了实时数据的查询精度。传统的数据仓库在更新的过程中进行“批”加载[8]，本文通过建立由动态镜像组成的实时存储区使新鲜的数据不断载入，以达到数据的持续更新，从而大大提高了查询结果的精度。

3.4 动态镜像管理

动态镜像管理包括镜像创建与回收，及基于双重链接的镜像索引维护。首先动态存储区域空间有限，其次从源系统载入数据到动态存储区域中的镜像，其物理地址不一定是连续的，所以需要有效地分配、收回管理才能保证动态存储区域中数据的查询和导入有条不紊地进行。

3.4.1 动态镜像创建

镜像的创建过程就是将实时数据存入实时存储区域的过程。算法步骤如图 4 所示。

```
INPUT: Input the real-time data: d;
Input the head_pointer hi of img_links, and i=1,2,3…
OUTPUT: Head
1. /* 创建镜像节点 */
2. create * node;
3. Init(node,d);
4. /* 更新镜像链表 */
5. if exists(hi. data_id==d. data_id)
6. imgInsert(node);
7. else createImg(d. data_id);
8. /* 更新桶链表 */
9. update Bucket_Link(d. data_id);
10. return hi;
```

图 4 动态镜像分配算法

该算法的具体操作流程如下。

(1)当有新的数据需要从 OLTP 系统加载至动态存储区时，动态镜像管理模块在动态存储区中分配一块存储空间，创建一个镜像，用于存储新数据；同时，系统在动态存储区中保存一个相应的镜像文件。

(2)动态镜像管理模块采用顺序查找方式，遍历镜像桶链表中的每个镜像链表节点，检查新数据的数据源是否与镜像链表节点的数据源相同，即检查动态存储区中是否存在与新数据属于同一数据源的数据。若存在，则转入(3)；否则转入(9)。

(3)根据镜像桶链表节点的桶镜像首地址,可以查找到来自同一数据源的镜像链表的头节点。

(4)为新创建的镜像,在对应的镜像链表中创建一个新的镜像链表节点,操作标识符 tag 设为 0,指向下一个链表节点的地址设为空。

(5)根据新节点时间戳,将新镜像链表节点插入至相应的镜像链表。在镜像链表中,镜像节点按其数据更新时间戳倒序(从大至小)排序,所以,新镜像链表节点 n 的时间戳最大,将其插入在链表头节点之后。

(6)对于镜像 M 而言,若镜像的数据满足查询条件,或者接收到系统操作指令,需要将数据批量地从动态存储区导入数据仓库的实时数据存储区时,将镜像链表节点的操作标识符 tag 设为 1,同时,相应的数据批量地写入数据仓库的实时数据存储区。

(7)对于镜像 M 而言,若系统接收到数据更新指令,此时,动态镜像管理负责检查同一数据源的镜像数据是否正在从动态存储区导入数据仓库的实时数据存储区。

若存在,正在导入数据的镜像内容节点无法进行更新操作,则动态镜像管理在动态存储区采用步骤(1),分配存储空间,创建镜像,更新所属同一数据源的镜像链表,并接收来自 OLTP 的更新数据。以此类推,若需要将不断更新的数据导入动态存储区,系统不断分配存储空间,创建后续镜像,更新镜像链表,转入(11)。若不存在,则转入步骤(8)。

(8)动态镜像管理模块在动态存储区中分配存储空间,创建镜像,用于存储新数据。同时,系统在动态存储区中保存相应的镜像文件。

(9)为新数据源的镜像创建一个新的镜像链表。链表头节点的数据源 ID 存放镜像文件四元组的数据源;指向链表第一个节点的地址存放新创建的镜像首地址。镜像节点的操作标识符 tag 设为 0,镜像节点的指向下一个链表节点的地址设为空。

(10)根据新创建的镜像链表更新镜像桶链表。在原镜像桶链表的尾部增加一个新镜像桶链表节点。镜像桶链表节点的数据源存放新数据源 ID,镜像桶链表节点的首地址存放新创建的镜像链表的首地址,镜像桶链表节点的指向下一个镜像桶链表节点的地址设为空。

(11)动态镜像创建完毕,基于双重链表的镜像索引也完成相应的更新。

3.4.2 动态镜像回收

(1)对于镜像 M 而言,当系统将镜像的数据批量地从动态存储区导入数据仓库的实时数据存储区完成后,实时数据仓库系统将发送反馈信息至动态镜像管理模块。

(2)动态镜像管理模块根据收到的反馈信息,释放导入镜像数据所占用的存储空间。

(3)动态镜像管理模块根据镜像的数据源 ID 和镜像桶链表节点的首地址,定位到同一数据源的镜像链表中对应的镜像链表节点,将其节点从镜像链表中删除。

(4)若删除镜像链表节点后,其所属镜像链表的节点个数为 0,仅存在链表头节点时,则动态镜像管理块将此镜像链表所对应的镜像桶链表节点从镜像桶链表中删除,并释放其占用的存储空间。

3.4.3 查询请求处理

本文采用时间戳匹配法处理实时数据仓库查询请求,请求过程如下:

(1)若实时数据仓库系统已运行,根据用户发出的查询请求,镜像管理模块首先获取请求的时间戳和 data_id。

(2)然后根据 data_id 在动态存储区中查找相应的镜像,如果没有,则返回空值,并将查询请求发送至数据仓库处理;否则,在对应 data_id 的镜像链表中顺序查找,转(3)。

(3)在相应的 data_id 镜像链表中,按时间戳从大到小查找。若时间戳失效,则返回空值,并将查询请求发送至数据仓库处理;否则,读取与请求最匹配的镜像节点数据,并将结果返回至用户。

镜像链表中镜像按时间戳从大到小有序存储。若存在某一节点的时间戳小于或等于请求的时间戳,且其前一个节点的时间戳大于请求的时间戳,则说明此节点数据按时间戳匹配原则,最佳满足用户请求,所以,返回此节点的数据至用户。图 5 显示了在镜像链表中处理用户查询请求的实例。

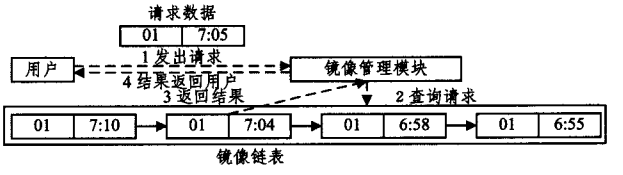

图 5 数据查找实例

3.5 查询竞争分析

从分析可知,查询竞争严重影响查询效率和查询结果的精度,动态镜像是一种实时数据仓库存储预处理技术,用于解决数据仓库的查询竞争问题[9]。当一条数据载入数据仓库后,首先判断数据是否为实时数据,如果是,则被实时 ETL 捕获,经过一系列清洗、过滤操作后载入动态存储区域。为了说明动态镜像技术是如何解决查询竞争问题的,本文假设所有新捕获的数据都是同源数据,即数据具有相同 data_id,且动态存储区域已经存在该数据的同源镜像链表。当数据第一次加载至动态存储区时,为其建立相应的数据镜像,命名为 $M1$,同时插入相应的镜像链表中。当镜像 $M1$ 中的数据时间戳失效时,将其导入数据仓库中。后续同源的数据再加载至动态存储区时,系统也建立相应的数据镜像,命名 $M2$、$M3$,并插入镜像链表中,以此类推。镜像 $M1$ 数据由于实时性失效,被载入数据仓库后,为用户提供历史数据的查询分析,而存放于镜像 $M2$、镜像 $M3$ 中的实时数据加载至动态存储区后,为用户提供实时数据查询分析功能。通过时间戳比较匹配方法,来满足用户实时性要求,提高查询结果的精度。通过合理、高效的数据分配和管理,基于动态镜像的数据存取技术可以有效地解决查询竞争问题。

3.6 时间复杂度分析

对于传统关系数据仓库而言,在数据更新阶段,实时数据以批处理模式加载至存储区时,由于查询竞争,会导致数据更新丢失,影响数据查询精度[10]。本文提出的动态存储区可以动态地为实时数据分配镜像存储空间,缓解查询竞争压力。时间复杂度作为衡量算法质量的重要指标,关系到算法的执行效率。下面对动态镜像创建、回收过程进行时间复杂度分析。

(1)动态镜像创建的时间复杂度是由数据导入的频率决定的。最坏情况为数据查询过程中不断有更新数据导入,则时间复杂度为 $O(n^2)$;最好情况为数据查询操作恰好在数据

导入完成后进行，则时间复杂度为 $O(n)$。

(2)动态镜像回收过程是在操作完成后，回收数据所在的镜像桶空间，时间复杂度为 $O(n^2)$。

本文利用空间换取时间思想，以内存空间为代价，减少数据 I/O 开销，提高查询效率和查询结果的精度。

4 实时 OLAP 的查询优化算法——FWMJoin

OLAP 上的查询分析操作一般表现为在事实表和维表之间连接操作的基础上对结果集进行分组聚集等操作。因此，表连接和分组聚集操作是关系联机分析处理(Relational OLAP，ROLAP)性能的两个重要决定因素。为了提高实时 OLAP 进行基于多表连接查询的数据分析性能，期望查询执行算法能够采用一趟扫描机制完成表连接、过滤与聚集等操作。下面给出本文提出的蝇量级物化下的表连接算法 FWMJoin(Fly-Weight Materialization Join)。本文提出的蝇量级物化连接算法 FWMJoin 是 LWMJoin[13] 算法的改进。LWMJoin 是针对星型模型的数据仓库，基于轻量级物化而提出来的表连接算法，它分为 3 个阶段：首先，将维表上的谓词条件作用在相应维表上，产生三元组〈surrogate，key，value〉，抽取符合过滤条件的元组构建哈希表；然后，顺序扫描事实表，利用事实表外键映射函数定位相应维表的三元组；最后，根据过滤三元组确定是否访问事实表相应位置上的元组构建结果集。如果三元组中的 key 值存在为 false，那么该元组应舍弃；否则，将 key 值与事实表度量属性拼接成一条元组，并将该元组以流水线方式输入到下一个操作。

FWMJoin 对 LWMJoin 的改进如下：将 invisible join 中由元组关键字构建的哈希表替换为 Chord 环。通过结合 DHT(Distributed Hash Table)技术，将维表中符合过滤条件的元组的关键字所在的内存地址的哈希值作为节点标识符(Node ID)，并对元组的关键字进行散列运算，产生一个唯一的资源标识符(Object ID)，且该资源将存储在节点标识符与之相等或相近的节点上。使用事实表外键作为关键字(key)进行查找。Chord 与其他结构相比较，具有负载均衡、鲁棒性高、可扩展和命名灵活等特点。

5 实验与分析

5.1 实验环境

作为一种优化实时数据仓库性能的手段，动态存储区域下的动态镜像实时数据仓库的存取预处理技术必须与实时数据仓库结合使用。本文实验部分将动态镜像技术与基于多级缓存(multi-cache)的实时数据仓库结合。

影响计算机系统性能的因素是多种多样的，加上其应用环境也特别复杂，基本不可能使用一个准确、固定的公式来表达，常用衡量的方式是指通用的基准程序。TPC-H 是一种基准，用来衡量数据库系统在硬件、软件方面联机分析处理的性能以及对 DSS 支持的能力[12]。这一基准是由事务处理性能委员会(TPC)在 1999 年公布的。通过测量给定的数据库单位时间内能够处理事务的数量来测试数据库性能。

本文选用来自 Sean Lahman[14] 网站的结构化数据。Sean Lahman 网站详细记载了从 1871 年到 2014 年美国职棒大联盟(Major League Baseball)各支球队和各个运动员的比赛数据。本文在 Hive 中建立 16GB 空间的实时数据仓库，约 350 万条记录。

5.2 实验分析

(1)统一设置动态存储区域大小为 128MB，无镜像方案没有动态存储区域，不设置。从图 6 中可以看出，使用动态镜像技术在事务更新频率为 1.0s/次的情况下，这是测试基准最低开销，响应时间增长率都趋于 0。当事务更新频率设置为最高值时，每次间隔 0.1s 需处理一个事务，OLAP 响应时间也达到最大，动态镜像技术相比于单镜像提升约 1 倍，与不采用镜像技术相比，性能提升接近 3 倍。随着事务发生频率增大，本文提出的动态镜像技术响应时间变长，但即使在事务成倍增加的情况下，使用动态镜像技术的响应时间的增长还是可以控制在 50%以内。这种效率是完全可以接受的，同时也在用户满意的范围。

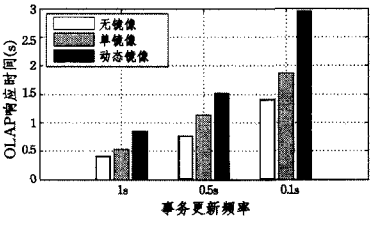

图 6 不同事务更新频率下的 OLAP 响应时间

(2)统一设置事务更新频率为 0.5s/次。如图 7 所示，在条件相同的情况下，动态存储区域的空间越小，OLAP 的响应时间则越长。当动态存储区域为 32MB 时，对于 OLAP 查询操作，单镜像与动态镜像方案响应时间相当，动态镜像方案略占优势。但是，当存储空间较大时，对于 OLAP 操作，动态镜像和单镜像方案的查询响应时间明显占优。动态镜像方案的查询响应时间比无镜像方案快 1.8 倍，比单镜像方案提升约 16%。

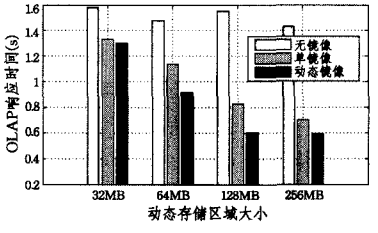

图 7 不同动态存储区域大小下的 OLAP 响应时间

(3)设置事务更新频率为 0.5s/次，动态存储区域统一为 128MB 空间。从图 8 中可以看出动态镜像技术、单镜像技术及无镜像 3 种情况在不同的事务更新频率下的实时数据查询结果精度。当事务更新频率较低时，3 种方案的查询精度较为接近；当事务更新频率设置为 0.1s/次的最高频率时，动态镜像技术的查询结果精度达到 85%，单镜像技术的精度为 73%，而不采用镜像技术的精度为 48%。本文提出的体系结构随着事务发生频率增大，平均精度达到 89.3%。

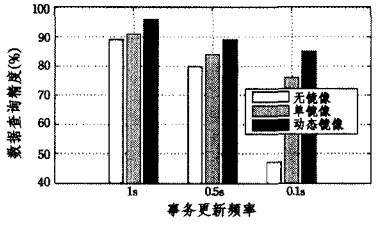

图 8 不同事务更新频率下的实时数据查询精度

实验结果表明，当实时数据被不断加载时，本文采用的基于动态镜像的实时数据存取技术在不同的事务更新频率下，查询效率可靠，如果给予动态存储区域更大的内存空间则性能会更好。

结束语 本文提出的基于动态镜像的实时数据仓库预存取技术是一种空间换时间的技术。通过在实时数据仓库外部创建一个由多个镜像组成的实时存储区域，有效地解决了实时数据仓库的查询竞争问题，并且提升了实时数据查询结果的精度。另一方面，在海量数据的背景下，如何进一步提高实时数据查询的精度，以及保证实时数据新鲜度，将在后续的研究中展开。

参 考 文 献

[1] Mohammad R, Klanmehr K, Alhajj R, et al. Data warehouse architecture and design[C]// Proc. of 2008 IEEE Int'l Conf. on Information Reuse and Integration. 2008:58-63

[2] Vassiliadis P, Simitsis A. Near real time ETL[M]// New Trends in Data Warehousing and Data Analysis. 2009:1-31

[3] White C. Intelligent business strategies: Real-time data warehousing heats up[D]. DM Review, 2012

[4] 徐俊刚 裴莹. 数据 ETL 研究综述[J]. 计算机科学, 2011, 38(4)
Xu J, Pei Y, Overview of data extraction, transaction and loading [J]. Computer Science, 2011, 38(4)

[5] Heman S, Zukowski M, Nes N J, et al. Positional update handling in column stores[C]// SIGMOD. 2010:543-554

[6] Kuo T W, Kao Y T, Kuo C F. Two-version based concurrency control and recovery in real-time client/server databases[J]. IEEE Transaction on Computer, 2003, 52(4):506-524

[7] Stonebraker M, Abadi D J, Batkin A, et al. C-store: a column-oriented DBMS[C]// VLDB. 2005:553-564

[8] Langseth J. Real-time data warehousing: challenges and solutions [OL]. http://dssresources.com/papers/features/langseth/langsth02082004.html

[9] Ankorion I. Change Data Capture-Efficient ETL for Real-Time BI[J]. Article published in DM Review Magazine, January 2005

[10] Italiano I C, Ferreira J E. Synchronization Options for Data Warehouse Designs[J]. IEEE Computer Magazine, 2006, 8(4): 167-172

[11] Lin Z, Yang D, Song G, et al. Dealing with Query Contention Issue in Real-time Data Warehouses by Dynamic Multi-level Caches[C]// Seventh International Conference on Computer and Information Technology. 2012

[12] TPC-H decision support bench mark, Transaction Processing Council[OL]. http://www.tpc.com

[13] 朱阅岸，张延松，周烜，等. 一个基于三元组存储的列式 OLAP 查询执行引擎[J]. 软件学报, 2014, 25(4):753-767
Zhu Y A, Zhang Y S, Zhou X, et al. Column-Oriented query execution engine for OLAP based on triplet[J]. Journal of Software, 2014, 25(4):753-767

[14] Lahman S. The Lahman Baseball database[OL]. http://www.baseball.com

(上接第 97 页)

区通讯和调度等服务；其次，有效区分了应用软件执行所消耗的时间和与 IMA 平台交互所消耗的时间；最后，通过故障注入模拟了应用软件执行过程中平台所出现的相应故障，使得可以分析故障处理对实时性的影响，并开展了成功场景和失败场景的差异分析。为了验证本文研究的有效性，在与工业合作伙伴的合作过程中使用了一个真实的工业项目进行案例研究。结果表明，本文研究成果可有效地把 UML 设计模型转换为仿真模型，并支持开展相应的仿真实验。目前，该方法仅适用于采用 ARINC653 标准的机载嵌入式软件系统。本文后续研究将进一步把 IMA 物理平台带入到仿真环境，支持运行平台硬件在环的半实物仿真实验。

参 考 文 献

[1] Airlines Electronic Engineering Committee[S]. Avionics Application Software Standard Interface. Aeronautical Radio, 1997

[2] Wilson A, Preyssler T. Incremental Certification and Integrated Modular Avionics[J]. Aerospace and Electronic Systems Magazine, IEEE, 2009, 24(11):10-15

[3] Parkinson P, Kinnan L. Safety-critical software development for integrated modular avionics[J]. Embedded System Engineering, 2003, 11(7):40-41

[4] Jouault F, Allilaire F, Bézivin J, et al. ATL: a QVT-like transformation language[C]// Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications. ACM, 2006:719-720

[5] 罗国庆. VxWorks 与嵌入式开发[M]. 北京：中国机械工业出版社，2003

[6] http://www.windriver.com/products/platforms/safety_critical_arinc_653/index.html

[7] OMG. QVT Query/views/Transformations RFP, OMG Document ad[OL]. http://www.omg.org/cgi-bin/doc? ad/2002-4-10

[8] Van Amstel M, Bosems S, Kurtev I, et al. Performance in model transformations: experiments with ATL and QVT[M]// Theory and Practice of Model Transformations. Springer Berlin Heidelberg, 2011:198-212

[9] OMG. Unified Modeling Language Specification, Version 2.0, OMG Document formal [OL]. http://www.omg.org/spec/UML/2.0

[10] Vanderperren Y, Dehaene W. From UML/SysML to Matlab/Simulink: current state and future perspectives[C]// Proceedings of the Conference on Design, Automation and Test in Europe(DATE 2006). Munich, Germany, 2006:93-93

[11] 耿素云，屈婉玲. 离散数学[M]. 北京：清华大学出版社，2004