主动数据仓库的挑战和解决方案

北京大学计算机系数据库实验室

林子雨

报告内容

- 挑战 #1：支持实时ETL
- 挑战 #2：实时事实表的建模
- 挑战 #3： OLAP查询和变化的数据
- 挑战 #4：可扩展性和查询竞争
- 挑战 #5：实时报警

挑战 #1：支持实时ETL

- 建设任何数据仓库时，最难的部分是数据的抽取、转换、清洗和加载
- 几乎所有的ETL工具和系统，不管是现成的产品还是定制编码的，都是以批处理方式工作
- ETL过程通常需要数据仓库暂时当机，停止对外服务，这时用户就无法访问数据仓库
- 当以实时的方式连续加载数据时，就不可能允许存在系统的当机时间
- 没有当机的情况下对数据仓库进行连续更新，通常与传统的ETL工具和系统的设计理念是相互冲突的

挑战 #1：支持实时ETL

解决方案 1a: 准实时ETL

- 实时ETL问题的最简易的解决方案就是根本不考虑采用真正实时的ETL
- 并不是所有的问题都需要实时的答案，况且，因实时而引起的开销可能超出由实时而带来的收益
- 对于这些应用，只要简单地提高现有的数据加载的频率即可，这种方法可以使数据仓库用户得到比过去更加新颖的数据，同时却不用对现有的数据加载工具、数据模型和报表应用做太大的修改
- 当不需要严格的实时时，准实时是一个比较可行的解决方案

挑战 #1：支持实时ETL

- **解决方案1b：直接流水注入式（direct trickle feed）**
- 把从源系统产生的新数据象水流一样直接注入到数据仓库
- 可以直接在数据库仓库事实表中插入或更新数据
- 也可以把数据插入到实时分区当中的单独的事实表中（见解决方案2b）
- 可扩展性不好，复杂查询和连续插入及更新混在一起进行会严重影响数据库的性能

挑战 #1：支持实时ETL

- **解决方案 1c：流水和跳跃式（trickle & flip）**

- 不是把数据实时地加载到实际的数据仓库表中，而是把数据连续地注入到阶段存储表（staging table）中

- 阶段存储表的结构和数据仓库表的结构相同,阶段存储表中的内容会和事实表周期性地进行交换

- 采用“以视图形式实现的集成的实时分区”这种方法，只需要修改视图的定义，让它包含新的表而不是旧的表

- 建议在处理数据交换的时候，暂时停止OLAP服务，从而保证在进行数据更新的时候，不会收到新的查询

- 当更新周期为每分钟一次到每小时一次之间时，可以采用这种“流水和跳跃”方法。通常情况下，周期为5到10分钟时，系统的性能最好

挑战 #1：支持实时ETL

- **解决方案 1d：外部实时数据缓存**

- 把实时数据存储到一个额外的独立于传统数据仓库的实时数据缓存（RTDC：real time data cache）中，这样可以彻底避免对数据仓库性能的影响，同时不用对现有的数据仓库做出修改

- 实时数据缓存可以是另一个专用的数据库服务器，或者也可以是一个大的数据库系统的单独的实例，专门用来加载、处理和存储实时数据

- 把所有那些需要实时数据的查询定向到实时数据缓存 ，或者把某个查询所需要的实时数据临时地无缝隙地整合到传统的数据仓库中

- 采用实时数据缓存，就不会给现有的数据仓库带来可扩展性和性能问题

- 需要安装和维护一个额外的单独的数据库

挑战 #2：实时事实表的建模

- 数据仓库通常会包含基于时间维的不同粒度的综合数据
- 随着实时数据的引入，可能就需要考虑这些综合数据和实时数据的同步性问题
- 由于数据更新不再以月或周为单位，而是实时更新，那些以前每月更新或是每周更新的度量，其现在的结果可能变得令人难以理解
- 关于建模的主要的问题还在于“把实时数据存储在哪里”，以及“如何把实时数据和数据模型的其余部分连接起来”

挑战 #2：实时事实表的建模

- **解决方案 2a：采用通常方法，直接事实表注入**
- 当采用“直接流水注入式”或者“流水和跳跃式”方式来实现实时ETL时，不需要特殊的数据建模方法
- 从查询工具的角度而言，以这种方式建模得到的实时数据仓库和非实时数据仓库是没有什么区别的
- 使用这种方法时，需要考虑的最主要的事情就是缓存

挑战 #2：实时事实表的建模

- **解决方案 2b：单独的实时分区**
- 建模实时数据的一种方法是，把实时数据存储在一个单独的数据仓库事实表中
- 根据事实表的类型，许多支持表分区的查询工具，就可以自动从实时分区取得其所需的实时数据
- 对于不适用这种方式的事实表，可以为其建立额外的事实和属性，并指向实时数据表，查询工具就可以从历史数据钻取到实时数据
- 从查询工具配置和管理的角度而言，这种数据建模方法实现起来是最复杂的

注：Ralph Kimball在他的2002年2月发表的文章《Real time Partitions》中详细阐述了实时分区的问题

挑战 #2：实时事实表的建模

- **解决方案 2c：集成的实时视图**

- 把实时数据和历史数据存储在不同的表中，但是采用相同的表结构。然后，通过使用数据库视图，把历史数据和实时数据表结合在一起，使得对于查询工具而言，只有一个统一的逻辑视图

- 这种方法可以避免很多由于实时分区而引起的问题，因为查询工具和终端用户不需要进行两个表的连接操作

- 这种方法和直接事实表注入方法很相似，除了数据是被注入到更小的表中

挑战 #2：实时事实表的建模

- **解决方案 2d：外部实时数据缓存时的建模**

- 当采用一个外部实时数据缓存时，就不需要特殊的数据建模

- 外部数据缓存数据库使用和数据仓库相同的数据模型，但是只包含需要实时数据的表

- 如果采用和访问数据仓库不一样的方式来访问外部缓存（比如某个特定的OLAP工程），就需要在缓存中增加额外的表，比如查询表

- 当出于查询和分析考虑而需要历史数据和实时数据实现无缝集成时，外部实时缓存是很有用的

- 从数据建模的角度而言，这种方法的优点和直接事实表注入方式和采用视图的集成方式一样，这种方法还有一个额外的优点，那就是消除了性能问题

挑战 #3：OLAP查询和变化的数据

- OLAP和查询工具被设计成在静态的、不发生变化的历史数据上进行操作

- 由于假设所使用的数据不发生变化，OLAP和查询工具在设计时不会考虑到如何保证他们的结果不会被同时发生的数据更新所影响。在某些情况下，这可能导致不一致和混乱的查询结果

- 关系型OLAP工具对这种问题就特别敏感，因为他们使用多段SQL（multi-pass SQL）来执行最简单的数据分析操作

挑战 #3：OLAP查询和变化的数据

```
0:00 create table TEMP1 (
        Category_Id LONG, DOLLARSALES DOUBLE)

0:01 insert into TEMP1
     select      a11.[Category_Id] AS Category_Id,
        sum(a11.[Tot_Dollar_Sales]) as DOLLARSALES
     from        [YR_CATEGORY_SLS] a11
     group by    a11.[Category_Id]

0:05 create table TEMP2 ( ALLPRODUCTSD  DOUBLE)

0:06 insert into TEMP2
     select sum(a11.[Tot_Dollar_Sales]) as
     ALLPRODUCTSD
     from        [YR_CATEGORY_SLS] a11

0:08 select distinct pa1.[Category_Id] AS Category_Id,
        a11.[Category_Desc] AS Category_Desc,
        pa1.[DOLLARSALES] as DOLLARSALES,
        (pa1.[DOLLARSALES] / pa2.[ALLPRODUCTSD])
          as DOLLARSALESC
     from        [TEMP1] pa1,
                 [TEMP2] pa2,
                [LU_CATEGORY] a11
      where      pa1.[Category_Id] =
      a11.[Category_Id]

0:09 drop table TEMP1
0:10 drop table TEMP2
```

Category	Metrics Dollar Sales	Dollar Sales Contribution to All Products Abs.
Electronics	$ 39,915.00	19.7%
Food	$ 10,938.00	5.4%
Gifts	$ 36,362.00	18.0%
Health & Beauty	$ 11,707.00	5.8%
Household	$ 88,774.00	43.8%
Kid's Korner	$ 5,502.00	2.7%
Travel	$ 9,289.00	4.6%
Total	**$ 202,487.00**	**100.0%**

Figure 1: Sales by Category with Percent Contribution

挑战 #3：OLAP查询和变化的数据

- 对于大多数关系型OLAP系统，上面的这个例子都是非常典型的
- 大多数用户经历的查询响应时间都是从几秒到几个小时。对于一个设计合理的系统而言，可以接受的响应延迟从15秒到2-5分钟
- 典型的OLAP查询通常包含多个代码段，对于很多报表而言，包含10-50段SQL语句是很常见的
- 第一个问题是，当一个查询执行了1分钟才返回查询结果时，那么该结果就很难说是实时的
- 第二个问题是，需要多个SQL代码段来执行OLAP报表和分析操作，任何实时数据仓库都会面临刚才上面讲述的内部不一致性问题

挑战 #3：OLAP查询和变化的数据

- **解决方案 3a：使用准实时方法**

- 解决这个问题的最简单的方法就是首先避免产生这个问题

- 只有当数据变化得太快，以至于在多段查询开始执行时的数据和执行结束时的数据不一致，才会出现报表的不一致性问题

- 如果OLAP服务器被设计成不会在数据仓库加载或跳越时发起新的查询，那么使用准实时的ETL方法（解决方案1a），或者采用具备相对较长周期的流水跳跃式（解决方案1c），可以使该问题得到缓解

- 这种方法的缺点是，数据并不是真正实时的，而是有一定的延迟

- 还必须采取措施保证在数据仓库加载或跳跃期间，OLAP服务器可以被正确地停止

挑战 #3：OLAP查询和变化的数据

- **解决方案 3b：针对实时数据的风险缓解**
- 最简单的方法是禁止用户对实时数据执行非常复杂的查询
- 另一个替代方法是，在单独的分区维护一份实时数据的快照，为复杂查询提供服务，快照的更新不必太频繁，但是，这又增加了安装、维护和用户培训的复杂性

挑战 #3：OLAP查询和变化的数据

- **解决方案 3c：使用额外的实时数据缓存**

- 通过把实时数据和历史数据分开存储，报表就不会出现内部不一致性问题

- 采用即时数据合并处理（将在下面的解决方案4d中阐述），仍然可以实现历史数据和实时数据的无缝集成，为报表提供服务

- 采用反向的即时数据合并（把历史数据合并到基于内存的实时数据缓存中），就可以解决报表的延迟问题（该方法将在解决方案4e中进行讨论）

挑战 #4：可扩展性和查询竞争

- 当实施实时数据仓库解决方案时，组织面临的最重要的问题就是查询竞争和可扩展性问题

- 数据仓库和交易系统是分离开来的，因为在数据仓库上运行的复杂查询无法在频繁更新的环境下取得良好性能

- 通常数据仓库和OLAP解决方案的可扩展性是“查询涉及的数据量和并发用户数”的直接函数。在给定数据量的情况下，系统的用户数和响应时间成正比。用户数量的增多会延长查询的响应时间

- 在实时系统中，连续数据更新和加载也会进一步增加系统资源的消耗。不幸的是，连续数据加载所消耗的额外的系统资源并不简单地等同于在系统中增加一个或两个并发查询用户，原因就在于数据插入操作和典型的OLAP的SELECT语句的冲突

- 连续数据插入和复杂SELECT操作的冲突将会严重限制系统的可扩展性。很快，连续数据加载进程就会被阻塞，平常本应快速完成的查询也将出现很大的延迟。

挑战 #4：可扩展性和查询竞争

- **解决方案 4a：简化和限制实时报表**

- 需要实时报表的用户可能只发出很简单的查询要求，如果可以把报表限制在很简单的且快速的单段SQL查询，那么，许多关系数据库本身就可以处理这种冲突

- 数据仓库中最复杂的查询都会在较长的时间跨度内访问数据。如果使这些查询只基于不发生变化的静态的历史数据，那么就可以消除和实时数据的冲突

- 另一个需要考虑的很重要的方面就是，到底有哪些用户需要访问实时数据

- 许多可能对实时数据感兴趣的用户，可以通过通知机制来很好地满足他们的要求

挑战 #4：可扩展性和查询竞争

- **解决方案 4b：采用更高级的数据库系统硬件配置**
- 可以选择升级硬件来解决可扩展性问题
- 可以为高端的SMP数据库系统增加更多的节点，或者可以为数据仓库配备更快的处理器和更大的内存
- 这种方法可以解决短期的可扩展性问题，但并非解决问题的良策
- 实时查询冲突不仅和可用的系统资源相关，更和数据库系统的基本设计有关

挑战 #4：可扩展性和查询竞争

- **解决方案 4c：单独分离的实时数据缓存**
- 使用外部实时数据缓存（见解决方案1d）解决可扩展性和查询冲突问题
- 该方法把所有实时数据加载和查询活动重定向到一个独立的专门设计用来存储实时数据的数据库
- 把所有的实时活动都放在一个单独的实时数据缓存上执行，就不会给已有的数据仓库增加任何开销
- 缺点是，把实时数据的存储和数据仓库分离开来，报表工具就无法把实时数据和历史数据有效结合起来
- 如果在实时数据缓存上运行很复杂的报表，那么，就可能出现前面所论述的报表的内部不一致性、数据冲突和可扩展性等问题

挑战 #4：可扩展性和查询竞争

- **解决方案 4d：从外部数据缓存的即时信息合并**

- 我们前面所描述的多种解决方案其实始终无法满足某一类的应用需求，这类应用需求具备一个或一个以上的下面列出的特征：
  - 需要真正的实时数据（而不是准实时数据）
  - 包含快速变化的数据（每秒10-1000个交易）
  - 需要满足10个、100个甚至1000个并发用户的访问
  - 结合历史数据和实时数据进行分析
  - 包含多段、复杂的分析型OLAP查询

- 实时数据被保存在外部的实时数据缓存中，历史数据保存在数据仓库中，这两种数据会根据需要被整合到一起呈现给应用

- 可以通过JIM（just-in-time information merging ）机制实现历史数据和实时数据的合并

挑战 #4：可扩展性和查询竞争

- **解决方案 4e：反向即时数据合并**

- 对于那些主要使用实时数据而很少使用历史数据的查询而言，RJIM非常有用

- 所需要的数据需要从数据仓库临时地反向加载到实时数据缓存，查询就在实时数据缓存上运行

- 这种机制只有当实时数据缓存所在的RDBMS完全支持SQL时才可以实现，对于那些没有支持许多SQL函数的内存数据库来说，无法实现这种机制

- 对于一个智能的RJIM来说，它会在数据仓库上做尽量多的分析，从而得到所需要的最合适的具备很高粒度的综合数据，然后再加载到实时数据缓存中，这样可以减少数据传递量

挑战 #5：实时报警

- 许多和数据仓库相关的报警应用，主要是在晚上完成数据仓库的数据加载以后，采用以电子邮件的形式通知用户
- 实时数据的可用性，使得数据仓库报警应用更加吸引用户，因为用户此时不需要等到晚上的数据加载完成以后才能得到报警，而是实时地得到报警
- 现有产品以周期方式或以事件触发方式进行运作，因此，他们或者可以每隔几分钟或几小时发出报警，也可以被外部事件触发报警
- 当报警被频繁触发时（而不是每个晚上一次），就必须有一个机制来保证，当某一个报警被触发并发送给用户以后，在接来下来的每个报警周期内，同一个报警不会被反复发送给用户

挑战 #5：实时报警

- **解决方案 5a：n分钟的周期调度**

- 在不增加复杂的实时数据流监测解决方案的前提下，有一种方法可以近似实现实时报警，那就是周期性地使用数据仓库报警工具包，可以把周期设置为1、5、15或30分钟

- 对于一个以准实时方式加载数据的数据仓库来说，所要做的事情只是在数据更新结束以后触发警报

挑战 #5：实时报警

- **解决方案 5b：真正的实时数据监测和触发**

- 对于真正的实时数据报警，需要引入触发系统，因为现有的数据仓库报警系统不能监测实时数据流

- 这些系统相当复杂，复杂度取决于用户的数量、报警的条件和到达数据的数量，还需要大量的硬件资源，尤其是内存

挑战 #5：实时报警

- **解决方案 5c；实时报警门槛管理**

- 不管采用准实时的批周期方式还是采用实时的触发系统来生成报警，有一点都很重要，那就是，对用户的报警门槛值进行正确管理

- 采用静态的门槛值定义，对于那些每晚更新一次或每月更新一次的系统来说，不会发生频繁地重复报警问题

- 对于更加频繁更新的系统来说，就可能会发生频繁地重复报警问题

参考文献

- [1] Neil Raden.Real Time: Get Real, Part II. June 30, 2003 . http://www.intelligententerprise.com/030630/611warehouse1_1.jhtml?_requestid=1132505
- [2] Ralph Kimball. Real time Partitions Feb, 2002. http://www.intelligententerprise.com/020201
- [3] Neil Raden. Raden.Real Time: Get Real. June 17, 2003 . http://www.intelligententerprise.com/030617/610warehouse1_1.jhtml;jsessionid=RQKS3PEOAEEXYQSNDLPSKH0CJUNN2JVN
- [4]Langseth, J., "Real-Time Data Warehousing: Challenges and Solutions", DSSResources.COM, 02/08/2004.