3.1.2目录

为了保持一个逻辑模式和物理模式之间的正确的映射,促进系统的其余部分的优化,Sinew仔细记录属性名称、类型和存储的方法(物理或虚拟列)。这个元数据保存在一个目录,信息记录如下:

•被存储的键

•已经从数据导出的键类型信息

•每个键出现的次数

•是否物理或虚拟列

•“脏”标识(3.1.4节讨论)

事实上,这目录分为两个部分。第一部分,如图4中的示例所示，包含一个全局列表，其属性为出现在任何文件中的id、名称、类型三元组。这个全局列表辅助数据序列化(在4.1节中描述)，作为字典,把每个属性映射到一个ID,从而，在一个特定的属性需要从存储层引用的时候，提供一个紧凑的键的表示这个属性。第二部分的目录(参见图4(b))对每个表维持一个目录(而不是全局范围地跨表),并包含上面提到的其他信息。

根据这些信息,Sinew能够识别的逻辑模式和当前的物理模式,使查询能够有处理指向虚拟列的引用转化为能与底层RDBMS模式相匹配的语句。统计(唯一的键的数目和每个键的出现次数),由模式分析程序使用(在第3.1.3部分中进行了描述),动态地决定对哪些列实现化或留在列库。下面我们将讨论这种交互。

3.1.3模式分析程序

为了适应不断变化的数据模型和查询模式,模式分析程序定期评估当前目录中定义的存储模式,以决定适当的物理和虚拟列的分布。选择哪些列进行实例化的主要目标是使得实例化的整体系统花销和相关的系统的维护表的开销最小化,使系统的性能最大化。

经常出现的属性和,带有明显不同于RDBMS优化器的默认假设的基数的属性是好的实例化的候选项。一个简单的阈值用于这两种情况下。出现频率高于第一阈值的属性，或带有差异高于第二阈值的基数的属性，被实例化为实体列,其余属性作为虚拟列。

作为新数据加载到系统,密度和基数的特征列可能会改变。因此,模式分析程序也会检查已经实例化的列是否仍高于阈值。如果不是,他们会被标记为去实例化。

3.1.4列实例化程序

列实例化程序负责维护动态的物理模式，通过从列库到物理列移动数据（或者相反)。我们的目标是其作为一个后台运行进程，只有当系统中有闲置资源可用时才运行。实现这一目标关键需求是列实例化程序的渐进进程,其他查询运行时可以停止,当查询完成和资源空闲时可以恢复。因此,我们的设计不要求整体的实例化,一些键的值可能存在于列库，而另一些存在于相应的物理列。我们称这样的一个脏列,在这种情况下,在目录下的脏标识必须被设置。已经设置了脏位,查询访问键时必须检查物理列和库列(这是通过合并函数实现的，见3.2.2节)。

列实例化程序按如下工作。每当模式分析器决定把一个虚拟列成一个物理列，反之亦然,在数据移动之前，它把对应的脏标识设置为真。实例化程序周期性地轮询目录中标识为脏的列,对这样的列,它检查目录是否列现在应该是物理或虚拟(这决定了数据移动的方向)。然后,它逐行循环遍历和任何行,它发现列库中的数据,它应该是在一个物理列(或者相反),它对那一行(并且只有那一行)执行一个原子更新，把值移动到正确的位置。实例化程序和加载程序不允许同时运行(我们通过目录中的锁存器实现),所以当迭代结束,它可以保证现在所有的数据都是在正确的位置。然后把脏标识设置为假,这个过程就完成了。

正如上面提到的,这个设计的重要特征是,虽然每个行更新是原子性的,整个实例化过程不是。在任何时候,实例化程序被停止并进行查询处理。由于需要添加合并函数查询处理，这些查询会略慢于对非脏列的查询。精确的减缓程度依赖于底层的数据库系统是如何实现合并的。在我们基于PostgreSQL的实现(见第五节),我们观察到合并查询10%的最大减速。对于受限于磁盘带宽的工作负载，我们没有观察到减缓。

3.2用户层

3.2.1加载器

批量加载是在两个步骤完成,序列化和插入。在第一步中,加载程序解析每个文档,以确保它的语法有效,然后序列化成4.1节中描述的格式。序列化发生的时候,加载器聚合出现在数据集的关于键的形式、类型和稀疏度的信息并将这些信息添加到目录中。更确切地说,每一个要加载的键-值对,加载程序推断数据类型并在目录中查找键和类型(我们称之为一个属性的组合)，获取其属性ID。如果属性不存在于目录,序列化器将它插入目录,获得一个新生成的ID，并序列化键-值对，与其余的数据一起存入列库。因此,虚拟列在加载时自动创建新的键，在序列化过程中。将一个新属性添加到模式的成本只是在序列化中它第一次出现在数据集时，将新属性插入目录的成本(用户的感知不到的成本)。

在插入中,所有的序列化数据放入列库。然后Sinew在目录中将所有受影响的列的脏标识设置为真,当列实例化程序发现脏标识并实列化新加载的数据，它们的数据最终被转移到适当的物理列。这个设计决定是出于希望保持尽可能的模块化系统组件。总是加载到列库,加载程序不需要知道物理模式,并且不需要与模式分析程序和系统的列实现组件交互。

3.2.2查询重写器

Sinew的混合存储方案需要,把面向用户、逻辑模式的查询转变与底层物理模式相匹配的查询。因此,Sinew的查询重写器将查询发送到存储层执行前修改查询。具体地说,将给定的查询转换成一个抽象语法树,重写器使所有引用目录中的信息的列有效化。任何无效的引用列,无论是因为其引用了虚拟列或是引用了“脏”的实体列，都会被重写。例如,给定的查询:

SELECT url, owner

FROM webrequests

WHERE ip IS NOT NULL;

对虚拟列的引用,“owner,”将由一个函数转换,这个函数基于系统选择的序列化格式，从列库提取键(在4.1节我们提供实例实现一个这样的功能):

SELECT url, extract\_key\_txt(data, ‘owner’)

FROM webrequests

WHERE ip IS NOT NULL;

在“owner”是脏的(即没有完全实例化)情况下,列引用将被转换为物理列和键提取的SQL合并:

SELECT url, COALESCE(owner, extract\_key\_txt(data, ‘owner’))

FROM webrequests

WHERE ip IS NOT NULL;

除了所需的键,提取函数包含了类型参数(上面是把“text”作为参数的语法糖衣),它是基于呈现在原始的语义的类型约束，由查询重写器动态地确定的查询。然后提取函数只作用于那些正确的类型的值。这种行为使Sinew优雅地处理情况相同键值对应于多个类型而不是抛出类型不匹配的异常(例如：如果该值是一个预计为整数的函数参数),它将有选择性地提取整数值,对字符串、布尔值、或其他类型的值则返回NULL。另一方面,在通常情况下,预期属性的类型不能由查询语义确定(比如：投影的情况下),函数将返回向下转换为字符串类型的值。