MUDD：一个多维数据生成器

当今的商业智能系统由数百个带有磁盘子系统的处理器组成，能够处理多个GB级别的IO带宽。这些系统通常包含TB级别的数据。评估此类系统的数据库系统性能通常需要生成具有明确定义的统计特性的合成数据。为了模拟不同的场景，改变统计属性（包括表格的行数）是很重要的。首先，为了分析大型系统，数据生成器需要能够及时生成数百TB的数据。在本文中，我们提出了一个多维数据生成器MUDD。它最初是为TPC-DS设计的，一个由TPC开发的决策支持benchmark，利用现代多维的处理器架构，包括集群，MUDD能够在数小时内生成高达100TB的文件数据。其新颖的设计在于将数据生成算法与数据分布定义分开，使得用户能够根据个人的需要和不同的场景调整他们的工作负载。

1、简介

本文所描述的工作的起源可以追溯到1992年，和DBGEN的发展，即TPC为他的第一个决策支持benchmark，TPC-D开发的数据生成器。当时开发的生成器和benchmark为早期的决策支持系统提供了一个基本的评价。它依赖于一个相对简单的，第三范式的schema，同时在数据复杂性和扩展性方面的的要求不是很高。作为一个例子，所有的数据都是均匀分布的，并且1TB的数据集就被认为是非常庞大的，这在数据生成器的最初发布版本中是不被支持的。

从那个时候起，因为从早期的benchmarks上得到的经验教训，决策支持系统的重要性和复杂性已经发生了巨大的变化。同时由于现实中的数据倾斜，数据缺乏和数据分布的重要性，催生了新的数据管理技术来解决这些问题。常见的数据量增加了数据级别，甚至普通系统的计算能力也呈指数级增长，从而允许构建更复杂的查询。

TPC已经意识到了范式的变化，一直在开发一个新的决策支持基准:TPC-DS。TPC-DS试图充分利用TPC-H和TPC-R，并在现代决策支持方面更加全面。其数据设计使用具有共享维度和多级层次结构的多个雪花模式。配合使用随机替换的大型复杂查询集，通过更现实的数据维护，它可以提供一个新的benchmark平台。

有一点没有改变，就是需要对新兴系统提供有意义，有代表性的性能评估。为了使这项工作取得成功，它必须能够访问一个数据集，这个数据集在统计上是合理的，有适当的缩放性和有意义的复杂性。生成该数据集的生成器必须是可移植的，可扩展的和可伸缩的。除此外，数据生成平台应该尽可能具有通用性和灵活性，以便使得benchmark和生成器的开发能够独立进行。本文描述的多维数据生成器（MUDD）试图满足这些需求。

本文的其余部分安排如下。在第二部分，我们简要地讨论相关工作。在第三部分，我们给出MUDD架构的详尽的描述，包括我们是如何获得可扩展性，可移植性，可伸缩性的。在第四部分，我们讨论数据分布是如何被定义的和在数据生成中的使用。在第五部分，我们讨论数据生成器的性能。

2、相关工作

数据生成器是每一个数据库系统分析时的核心，因此，在商业和工业领域已经经过了讨论。在格雷等人看来，探索不同的技术来快速为TPC-A生成数十亿个记录的数据库。本文展示了通过划分工作和分配一个处理器分区，来将顺序数据生成转换成并行的数据生成。进一步显示了如何非常有效地生成密集—唯一 —伪随机序列以及如何导出非均匀分布。MUDD引入概念来随机化现实世界的数据。MUDD将数据生成与数据分布定义分开的概念使得用户能够快速更改数据分布来适应个别独立的需求。此外，此外，它还引入了生成层次结构的算法，用于构建多维数据集。APB-1包括用于多维数据的组合数据和查询生成器。其设计允许生成包括历史数据在内的大型数据集到平面文件中。然而，它不会将数据定义从数据生成中分离出来，从而不能对数据设计中的变化做出及时反应。APB-1数据生成器，不能够利用如今的多处理器来并行生成数据。限制了其扩展到非常大的数据集的能力。在早期的winconsin benchmark中，也有对数据生成器的研究。作为容量和压力测试工具的一部分，数据生成器可以从Benchmark Factory和Datatect等不同公司获得。

3、MUDD的体系架构

3.1 设计目标

MUDD被设计成公开可用的。 MUDD的发展始于schema的确切性质还没有达成一致尚且不知道具体表格的内容什么时候会有详细的阐述。 所以在设计是的定位是达到下面三个目标。可移植性，以确保生成的代码可以移动到尽可能广泛的计算平台范围内，这只需要很少或不需要额外的努力。可伸缩性，以便在schema中简单地引入额外的表格或在已经存在的表格中添加额外的列。可扩展性，支持的平台从单处理器笔记本电脑到服务器级SMP集群的高效使用平台，以及能够生成从单个GB到数百TB的便于统计的可比数据集。

3.2可伸缩性

在开发MUDD的时候，schema及其构成表的精确构成还是不确定的。因此，设计需要允许容易引入额外的表格或能够对现有的表格增加新的列具有新的列。尽管使用面向对象的语言（如C ++甚至是Java）最初看起来因为其可继承和可移植性具有很好的吸引力，但最终使用的是ANSI-C标准。考虑到面向对象方法模块化的优点，数据生成器是围绕每个表格构建的，生成一组以表格为中心的类。

TPC-DS 的schema中的每个表都由ANSI-C 标准的typedef表示，该typedef定义了该表的内部结构。每当新行被填充时，使用两个参数调用关联的数据生成程序，一个指向要填充的结构的指针，另一个指向正在构建的rownumber。一旦数据结构被填充，它就可以移交给为每个表定义的数据输出程序，以产生一个适合加载到DBMS的文件，可以直接加载到DBMS。

3.3可移植性

DBGEN的开发是在1994年完成的。从那时起，代码已经被移植到了广泛的平台上，而对底层算法几乎没有变化，只有在特别的接口部分会需要适当的修改。到了2003年的一月的时候，这个原始的数据生成器已经被成功移植到不少于20个不同的平台上，跨度从UNIX到Windows，从VMS，到MVS，到LINUX。TPC-DS的努力能够充分利用这些工作来确保TPC-DS数据生成器的便携性。通过广泛使用特定于操作系统的特性，避免了一些常见的陷阱。

还有一些其他的移植挑战是在模拟现有C库函数的例程的开发中解决的，以防精确的库函数在平台之间没有可比性的情况下。最重要的例子是在开发一个标准的随机数字生成器器（RNG）的情况下。虽然每一个目标平台提供了RNG的一些形式，来控制和传播它。没有办法提供基准所要求的确保在任何目标平台上都生成相同的数据集而达到随机数字的一致进度。

提供编译环境的最后一步是确保轻松的可移植性。makefile是和数据生成器一起部署的，可以对它重新构建适当的编译和链接时控制。在Makefile中使用了相同的设置来控制构建时间变量，对于每个已知的目标平台，定义了一组适当的编译时间和链接标志和设置，然后通过特定于平台的变量设置进行引用。最终的结果是一个编译环境，通过更改Makefile中的一行，可以将代码从一个已验证的平台移动到下一个平台。

3.4可扩展性

由于数据生成器的初始设计被要求用于生成从1GB到100TB的数据集，这个数据集扩展了五个数量级，所以代码的可扩展性是一个关键的设计要求。可扩展性在两个方面得到解决：算法效率和并行性。源文件已经依赖于平台特定的#defines来隔离与机器相关的代码。

算法效率在任何实用程序中都很重要。在任何一个执行过程中，一个将被要求产生多达一万亿个实体的模块将变得至关重要。MUDD采用了许多标准技术来确保其代码路径在其偶尔冗长的执行过程中的效率。(例如，大量使用向量例程和其他可重用的缓冲区，以避免不必要地调用malloc()和free()等等)。然而，对于这个范围的应用程序，可扩展性的真正关键是实现并行性。使用MUDD，没有必要以线性，单内核的方式生成数据。它可以使用最适合计算和IO资源的并行程度来生成数据。这个功能的实现很简单。

首先，RNG必须在所有目标平台上通用。MUDD通过提供可移植的RNG来完成这一任务，而不是依赖特定操作系统提供的例程。接下来，无论比例因子如何，对于特定表内的每一行，对RNG的调用次数必须是确定的。对于大多数列类型和数据域，这是一个简单的任务，因为大多数常见的值（即数字类型）可以通过对RNG的单个调用来生成。在调用次数可以变化的情况下，如在基于简化语法的英文文本产生中，必须确定每行调用次数的上限。如果对于RNG的调用是有条件的(“对于有周末时间的40%商店，从这些值中选择一个 ，否则S\_WKND\_ HOURS就是NULL”)，不管这个条件的输出是怎么样的，必须做出对RNG的调用。虽然这会导致对RNG的额外调用，但它将恢复每行调用量度的确定性性质。给定表格的行生成例程的最后一个操作是为了确保在这一行内已经进行了所需的调用次数，如有必要，额外地调用RNG。

有了这两个因素，对于构建特定表的一系列协作生成器的任何成员，决定要生成的行的数量是很容易的，并且，在任何数据生成器的工作之前，将RNG初始化成前一个生成器对前一行生成的值。MUDD使用64位线性同余随机数生成器。因此，有可能编写有效地“跳转”到RNG的生成值序列中的任何特定点的例程。如果这是不可能的话，那么并行的大部分好处将会丢失，因为在开始分区之前，每个并行生成任务将需要依次生成所有先前的随机数。

可以使任意一组计算和IO资源饱和（请参见下面的性能部分）。此外，由于并行执行中的每个生成过程是完全独立的，因此不需要集中协调或其他的IPC开销。数据生成可以使用一个单一的机器或者多个机器的集合。它可以一次完成，或者数据集可以随着时间和计算资源的允许而分阶段建立。每个配置的结果数据集都是一样的。

4、Data population/Distribution

4.1Domain Scaling versus Dataset Scaling

扩展数据集可以从两个方面来做。在一种情况下，数据集里元组的数量被扩展，但基本的数值集(域）保持不变。这可以类比成一个商业系统，其中商店的数量保持不变，但每年的交易量增加。在另一种情况下，元组的数量保持不变，但用于生成它们的域被扩展。例如，零售数据集中可能引入了一种新型商店，或者数据集可能涵盖更长的历史时期。显然，在数据集中这两种类型的扩展都可以被认为是正确的，通常一个测试将采用这两种方法来增长数据集。MUDD选择主要使用data set scaling而不是domain scaling。通过保持domains不变，增加被管理的数据的量，可以实现对数据库管理系统更有意义的全系统的测试。当然，由于在测试一个数据库管理系统的关键部分是其能够智能地管理和联结joins的能力。而且由于主键和外键的关系必须按定义使用data scaling，所以在这种情况下结果也算是两种方法的混合。

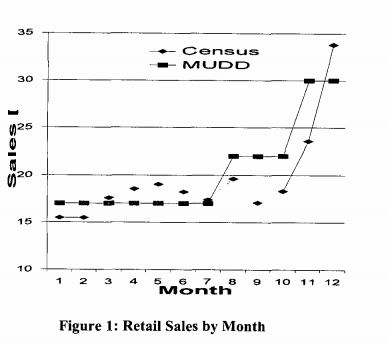
4.2合成数据中的可比性区域

因为MUDD是在数据库管理系统决策支持方法的性能评估中使用的，它需要生成可用于适合benchmark查询集的数据。一个数据集为什么非常适合这种性能评估的工作的精确本质，超出了本论文的范围。但是，它要求的一个方面将会被提及。

非合成数据通常包含不连续性，但它们很少位于数据样本的一致或可重复部分。虽然可以生成数据来模拟这些行为（请参阅下面的“在合成生成器中使用真实数据”），对于查询性能评估来说，如果忽略到此行为，用处会多一点。查询工作负载需要从一个查询接着下一个查询提供足够的可变性，从而确保对于查询的响应是从DBMS中派生来的—即不是简单地从查询的最后一个相同的执行中检索。与此同时，它们需要确保经过两次基准测试执行的查询集可以提供对被测DBMS的比较—即他们代表了相同数量的工作。本质上，SQL查询基准测试引入了另一个随机性级别，除了基础数据之外，还需要对其进行控制以保证正在测试的查询执行之间的可比性。

举一个例子，考虑一整年的零售可能性。研究表明，在年末假期销售的比例很高，通常最后两个月的销售额可以占到年销售额的30%左右(大部分是在12月下旬的时间)。显然，这种数据倾斜是数据集中的一个重要特征，需要由数据生成器来捕获。然而，数据的使用也需要被考虑。如果精确地捕捉到年底的突破，那么很可能不会有两天产生完全相同的销量。在我们查询的例子中，那么就不可能确保基于这个关键范围内的日期（或者全年其他地方）的查询在被测系统上产生可比较的负载量或活动量。留给数据集/基准测试设计人员的选项是根据基准中基于该范围的日期来移除查询，或者找到一些方法来确保他们和彼此具有可比性。

鉴于在数据集中数据倾斜的重要性，并且在这个特定的数据集中这种特殊的偏差是主要的，省略是个不好的选择。从而替代的是，这个数据集需要调整，引入可比性区域—实质上是分布中的flat spot—可以被用来提供数据的最终用户要求的变化性和可比性。



4.3在合成生成器中真实数据的用处

合成数据生成器面临着固有的挑战。如果数据太综合(例如，完全一致的分布)，由于没有捕获真实数据集的“有趣“的性质，它有着被拒绝的风险。相反的情况下，如果它使用从事务或装置收集的数据（“即从现实世界中”），那么对研究人员和基准来说，风险变地很小或者将没有价值，因为既不能产生可比较的工作负载结果，也不能解决有趣的假设性问题。MUDD试图找到这两者的适当组合。对于它的大部分数据，它使用传统的合成分布，产生均匀分布的整数，或使用高斯分布的字选择。然而，对于一些关键的分布来说，数据生成器依赖来自现实世界的数据来生成更真实的数据集。

列的第一个类依赖于真实数据。无论是人名还是列名，MUDD依赖1990年人口普查的数据来确定允许值的范围。结果是一个名字组合的填充，它是静态的但是很庞大，并在数据集内增加了“可识别性”。

基于现实数据的处理地址的第二个类。数据生成器有两个问题。一方面，地址域必须符合在地理数据中已经被定义好的层次结构，即正确地显示国家、州、城市等层次，如果不能完全正确的话，甚至到了邮政编码和电话号码需要合理的地步。同时，该领域必须随着数据集的扩展而增长。MUDD的解决方案是结合已知的分布（美国地质调查局的城镇名单和普通城镇名称列表，以及县，州，邮政编码和电话区号之间的适当关联）。最终的结果是一个地址生成器，虽然不是真的，但是非常现实。同时，数据显示了可扩展的地理位置; 小数据集只限于一两个国家，而大数据集包含越来越多的国家，而网络销售渠道则排在第一位。

4.4数据总量中的MDD issues问题

4.4.1层次结构

TPC-DS schema结构都显示简单的单一继承关系。那就是，对于层次结构中给定的级别，确切地说总是只有一个父级。有了这个保证，和给定的层次结构中的每一个级别的基数，这个生成器变得非常直截了当。

对于分层表中任何给定的行，行号唯一确定该行在每个层级中的位置。需要为层次结构中的某部分上的表生成适当的连接字段，这个是一个简单的模板函数序列，基于已知的特定级别的基数，总高度，目标层次结构。

4.4.2 数据倾斜到数据稀疏

也许多维数据集中最显著的特征是空间概念, 事实表中特定行引用的每个维度至少在架构上是独立的。也就是说，特定行的各个维度的值之间没有必要存在相互依赖关系。结合任何给定维度的广泛领域的可能性，维度空间可以是巨大的。这样做的直接副作用是这个可能空间的大部分将是空的。例如，一个零售数据库可能包含产品的尺寸，颜色和购买日期的这些维度。所有尺寸和颜色组合产品都不可能在任何一天都有被选购到—就像这些属性的特定组合几乎不会在任何一个特定的日子里被购买一样。只有在真正的购买数量增加的情况下，才有可能覆盖整个维度空间。对于数据生成器，这个有两个有意义的影响。

首先，数据集需要通过确保相对于正在生成的数据集的足够大的domain来对朝稀疏数据转换的趋势做出解释。考虑到在一个普遍的多维schema中维度的数目，这几乎不能算是一个问题。

其次，数据集必须同时考虑在大多数数据集中存在的数据聚集和数据倾斜问题（例如，周六会比周一有更多的购物量），并且确保数据它自己是可以预测的和可重复的行为。这里的关键是提供一个强大且灵活的方法，通过它来定义可以被引入到数据中的数据倾斜。当它们被应用在给定的事实表共享的多个维度时，他们的分布将会通过傅里叶转换来组合，这个将会放大任何给定的分布倾斜，并在数据集中产生期望的短暂性和“突发性”。

4.4.3SCD

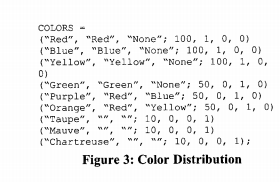
除了基于事务的事实表和基于属性的维度表，一个典型的多维度系统包括一些维度，这些维度的数据在整个系统的生命周期中演变。它们被称为缓慢变化的维度（SCD），它们捕捉数据集的历史演变。作为一个例子，考虑一个跟踪数月或数年的零售销售的系统。在那一个时期，基础产品线，价格结构，销售区域地理—几乎事务上下文的每一部分都有可能改变。为了可能得到有意义的纵向分析，数据集的用户能够重现上下文场景是非常重要的，将旧定价模式的销售与使用的新定价模式的销售进行比较。虽然使用多维的数据仓库，可以有很多方法来解决这种类型的数据，一种常见的技术是包含版本信息，在多维表中，通常以开始日期和结束日期的形式。然后查询可以限定应该使用维度条目的哪个修订版来探查事实表。

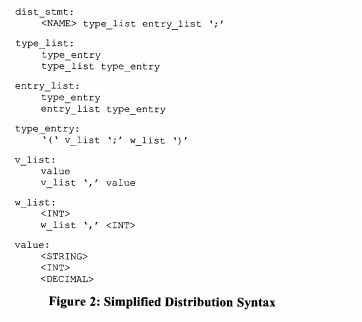
在生成的数据集中重新创建此行为。MUDD有将一个表识别成一个SCD的能力，将那些在生成数据的日期范围内修改的表中的那些列与那些静态的列相隔离，并且将所需的日期和schema的日期单独进行同步。

4.5捕获领域分布

在MUDD中，领域和分布功能的核心是定义值的任意分布。除了定义这些值本身之外，MUDD依赖这些值或相关值的相对频率(当相同的值集以不同的方式使用时，这可能会有所不同)。所有这些信息都包含在一个ASCII文件中，所以在benchmark的实验和微调的时候，可以改变数值和权重，而不需要改变数据生成器本身。分布定义的格式在图2中总结，结果是一个任意复杂度和解析度的阶段函数—这个的瓶颈更多的在于数据源和创造力的函数，然后才是计算能力。在图3中通常的颜色名被组织成类中。通过选择适合的权重集，有可能在整个色谱中，产生一个权重分布，或者随机地从给定地类中选择一种颜色。

MUDD工具集中还包含一个简单的编译器，将ASCII域表示转换为二进制格式。这个加速了在数据生成期间的使用，它通过防止无意义地修改域定义，有助于确保在不同站点使用的分布定义之间的可比性。

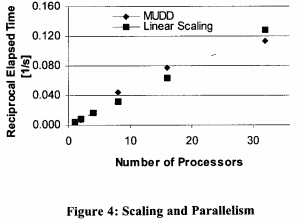


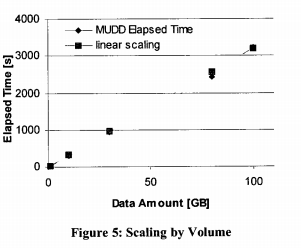


5、性能

MUDD已经发展到几乎能够在任何计算平台上快速产生大量数据，从shared-nothing到shared-everything，从单处理器到数百个地计算机集群。在本节中，我们将展示MUDD如何如何随着数据量和处理器的数量而扩展的。

图4展示了MUDD是如何随着处理器的数量的扩展而扩展的。它显示了生成82GB数据所耗用的时间，MUDD的实例数量从1到32不等。该系统使用一个shared-everything架构。由于MUDD的实例之间没有交流，这个实验的结果和shared-nothing系统上得到的是差不多的。该图显示MUDD与进程数量成线性比例关系。一个MUDD实例需要大约250秒来产生82GB，而32个实例需要大约8.8秒。在第二个实验中，我们展示了MUDD是如何与数据量进行扩展的。MUDD产生的数据量由一个比例因子决定，它以GB的级别表示原始数据。该比例因子可以是1至100000的任意一个数字。图五展示了比例因子在1GB至100GB期间的耗费时间。就像图中所显示的那样，消耗的时间和数据量成线性关系。要产生100GB的数据，MUDD需要消耗3228s，产生1GB的数据，需要消耗32s。





6.将来的工作

MUDD很多地方都可以改进和扩展，其中的一些改变将会简化更大数据集的生成。例如，现有的代码除了提供平台内部的并行性以外，可以扩展到对多个协作生成数据平台的支持。其它可能的变化将会更侧重结果数据的使用。在两个重要领域，工作已经开始进行。MUDD正在扩展到以支持TPC-DS工作负载的ETL阶段所需的3NF数据的生成。除此之外，它正在被集成到一个相关的查询生成工具中，以提供生成任意数据集的能力，并将其用于可比较的动态生成的查询压力测试。

未来工作的另一个领域是改进数据生成器本身的架构。这可能包括进一步使用现实世界的数据源，以增加现实主义和更迅速地适应新出现的数据趋势和波动。当前方法（对此已经开始一些工作）的示例修改是在固有的schema定义（即，SQL DDL）和数据生成例程之间提供更紧密的链接。这个的目标是在被填充的当前数据库schema和数据生成器之间提供更大程度的分离度，从而在运行的时候不会引起sql dialect编译的麻烦。

7、结论

本文描述了在构建MUDD（一种多维数据生成器）时所面临的挑战，及做出的决策的概述。我们引入了一种架构来将数据分布和实际的数据生成算法分离，即一个基本的范例来构建灵活的数据生成器。一种数据分布格式被引入以便于定义数据分布来模拟任何情况。有了这个设计，人们可以方便地将实际数据整合到一个合成数据生成器中。MUDD的设计实际上已经实现了，能够用于新的决策支持基准TPC-DS中。我们已经描述了在MUDD中如何实现可扩展性、可移植性、可伸缩性。在本论文中展现的测试中，表明了MUDD能够随着数据量、处理器的数量线性扩展，并且能够快速产生数百TB级别的数据。