大规模联机分析处理

黄一夫
复旦大学计算机科学技术学院
中国，上海200433
huangyifu@fudan.edu.cn

Abstract

对极大规模数据集的即席查询和例行分析存在增长性的需求，特别是对于互联网公司。互联网公司的创新强烈依赖于分析每天收集的*TB*级以上的数据。这个任务被称作大规模联机分析处理（*OLAP*）。*OLAP*的传统解决方案是用如*Teradata*一样的并行数据库产品搭建出的数据仓库，但是在极大规模的数据级上通常都是特别昂贵的。除此之外，分析这些数据的很多人员都是经验丰富的过程化程序员，他们往往觉得使用声明式的*SQL*来分析数据不自然。这一点被更加过程化的*map-reduce*编程模型，以及其对应的在商用硬件上的可扩展实现的成功给证明了。*map-reduce*编程范式是在集群上处理大规模数据的现代解决方案。然而，它专注于离线联机分析，太底层并且严格，产生大量的用户代码很难维护和重用。大规模联机分析处理将基于*map-reduce*的系统引入到*OLAP*中，因而同时取得了可扩展性和高性能。在本文中，我调研了最先进的大规模联机分析处理系统，并从总体设计和细节实现进行学习。

1. 介绍

越来越多的组织，其创新都源于对网页爬取，搜索日志和点击流产生的庞大数据集的分析。互联网公司，汝亚马逊，谷歌，微软和雅虎就是极佳的例子。对该类数据的分析，构成了产品改良周期的最初始的一环。例如，开发搜索引擎排序算法的工程师花费大多数他们的时间在分析搜索日志上，用以获取可开发的趋势。

这些数据集的庞大容量表明，应该用高度并行的系统，比如无共享的集群，进行存储和处理。并行数据库产品，比如Teradata，Oracle RAC，Netezza提供了简单的SQL查询接口，隐藏了物理集群的复杂性。然而，这些产品，在互联网级别上是极其昂贵的。除此之外，他们将程序员从偏好的命令式的脚本或代码，转移到了声明式的SQL查询。这让程序员感到不自然，并且过于受限。

作为以上现象的证明，程序员对更加过程化的map-reduce [6]编程模型趋之若鹜。一个map-reduce程序主要实现了在集群机器上的分组聚集操作。程序员提供一个map函数，用于指明分组操作是如何进行的；提供一个reduce函数，用于进行聚集操作。

对于这个模型来说，吸引程序员的地方在于，只有两个高层的声明式原语（map和reduce）来达成并行处理，但是剩下的代码，比如map和reduce函数，能被选择性地写为任何程序语言，而不用担心其并行性。

不幸的是，map-reduce模型有其固有的局限性。它的一个输入，两阶段数据流结构是非常严格的。为了运行含有不同数据流的操作，比如连接或n阶段操作，将产生不优雅的工作流程。并且，客户代码必须被写成甚至特别常用的操作，比如投影和过滤。这些因素导致代码难以重用和维护，以及分析任务的语义将模糊化。而且，map和reduce函数的透明特性将阻碍系统的调优能力。

这个被称作大规模联机分析处理的任务目的在于以高速的响应分析海量的数据集。因此，它将基于MapReduce的系统引入到OLAP中，因而同时获得了可扩展性和高效性。在本文中，我调研了目前最先进的大规模联机分析处理系统，这些系统是由顶尖的大学和公司设计并实现的，比如雅虎，微软，脸书，耶鲁和谷歌。并主要集中于他们的总体设计和细节实现。我将相关的9篇文章分为3组。第一组（Pig [8], SCOPE [5] and Hive [11, 12]）主要将类SQL的查询语言翻译成基于MapReduce的任务。第二组（HadoopDB [9, 1, 2, 3]）将数据库安装到数据节点上。第三组（Dremel [7]）将多层树结构与列存储结合起来。

2. 第一组：Pig，SCOPE和Hive

2.1. Pig

Apache Pig是一个分析大规模数据集的平台，由一个表示数据分析程序的高层语言，以及评估这些程序的基础设施组成。Pig程序的显著特性是他们的结构有义务于实质的并行化，这将使得他们能处理大规模的数据集。就目前而言，Pig的基础设施层次由一个产生Map-Reduce序列程序的编译器组成，这已经有了大规模的并行实现（比如Hadoop的子项目）。Pig的语言层次目前由一个叫做Pig Latin的文本语言组成，其拥有

如下的关键属性：

易于编程。实现简单，“尴尬并行”的数据分析任务的并行执行是容易的。复杂的任务由多个相关的数据转换组成，他们显式地编码为数据流序列，使得他们易于写，理解和维护。

利于优化。任务的编码方式允许系统自动化地优化他们的执行，允许用户专注于语义而不是效率。

可延伸性。用户能创建他们自己的函数来做特定目的的处理。

来自雅虎的作者描述了一种叫Pig Latin的新的语言，该语言被设计于填补声明式的SQL和底层的过程式的map-reduce之间的鸿沟。同时被称为Pig的相关系统也被实现，它能将Pig Latin编译成可以在Hadoop上运行的物理计划。Hadoop是map-reduce的一种开源实现。他们给出了一些雅虎工程师使用Pig的例子，相对于直接使用Hadoop来说，他们运行在Pig上的数据分析任务极大地减少了开发和执行的所需时间。他们特报告了一种与Pig相结合的新颖的调式环境，这将产生更高的生产力提升。Pig是一个开源的Apache孵化项目，可被获取用于广泛用途。

Pig的首要设计目标是为有经验的程序员提供极大规模数据集的即席查询能力。结果，从传统的数据库和SQL的角度来看，Pig Latin有非常多耀眼的特征。这里，我们将描述Pig的特征，以及其背后的原理。

数据流语言。在Pig Latin中，用户可以指定许多步的序列，每步只由一个高层的数据转换指定。这在文体上与SQL的方法有差别，SQL的用户指定一系列的声明式的约束，这些约束集合起来定义了结果。虽然SQL的方法对非程序员和小数据集来说是好的，但是有经验的必须操纵大数据集的程序员更喜欢Pig Latin的方法。

易上手与交互操作性。Pig是设计来支持即席数据查询的。如果一个用户拥有一个数据文件，如来自于搜索引擎日志的转储，他能直接上数据集上运行Pig Latin查询。他所需要的只是提供一个函数，用于给予Pig将文件内容解析成元组的能力。而不需要在查询之前，运行一个耗时的数据导入，这是在传统的数据库管理系统里非常常见的。类似地，一个Pig程序的输出能根据用户的选择进行格式化，根据用户提供的函数，能将元组转换为一个比特序列。因此，将Pig分析会话的输出用到之后的应用程序中，是很容易的。比如放到像Excel这样的可视化表格应用程序中。

嵌套的数据模型。程序员经常用嵌套的数据结构进行思考。从另外一方面看，数据库，只允许平面的表，比如列一样的原子域，除非要违背第一范式（1NF）的设计。Pig Latin有一个灵活的，完全嵌套的数据模型，并且允许复杂的，非原子的数据类型，比如说集合，映射和表中一些列组成的元组。

用户定义函数的提升。为了适应专门的数据处理任务，Pig Latin对用户定义函数（UDFs）拥有丰富的支持。Pig Latin里处理的主要方面包括分组，过滤，连接和单个元组处理，都能使用UDFs进行自定义。Pig Latin里UDFs的输入和输出服从我们灵活，完全嵌套的数据结构。作为结果，一个Pig Latin使用的UDF能用一个非原子的参数作为输入，并能输出非原子的值。这种灵活性是非常有用的。

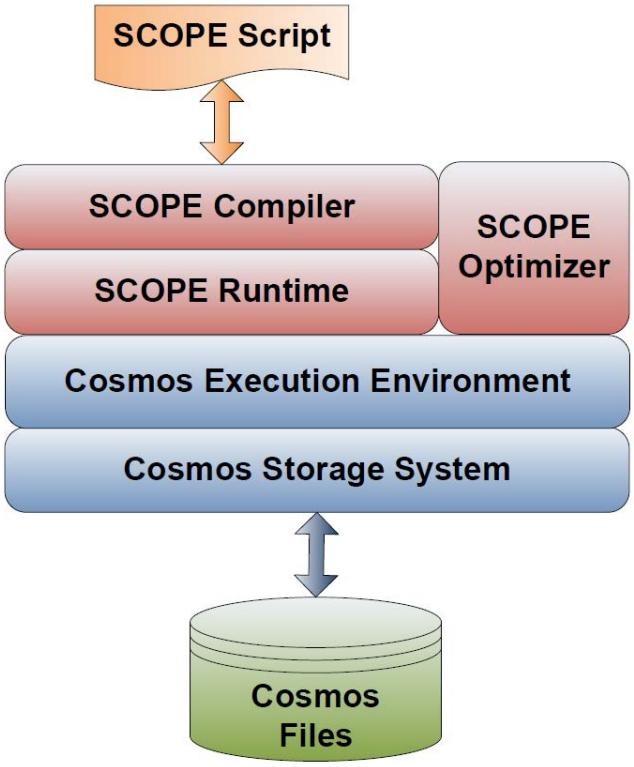

Figure 1. Cosmos软件层

并行度需求。因为Pig Latin用于处理互联网级的数据，因此不考虑并行化的评估是没有道理的。作为结果，我们只在Pig Latin中包含了一小部分的精心挑选的易于并行化的原语。不能高效并行执行的语言原语，如非等连接，相关子查询等被有意排除掉了。这样的操作当然，能通过编写UDFs进行实现。然而，因为语言没有提过这些操作的显式原语，用户能知道他们的程序有多高效，以及它们是否会被并行化。

调试环境。在任何语言里，得到一个正确的数据处理语言程序经常将需要很多次迭代，因为起初的迭代中经常有用户引入的错误处理。对于Pig将要处理的数据级，一次迭代将花费数分钟或小时（甚至使用了大规模的并行处理）。因而，这常用的运行-调试-运行循环将变得缓慢且低效。与Pig一起的还有一个新颖的交互式调试环境，它能产生一个简单的数据表例子来阐释用户程序每一步的输出。这样例数据被精心挑选，因而与真实的数据足够类似，从而能解释程序每一步的语义。而且，样例数据能随着程序的进化而自动地改变。

2.2. SCOPE

如图 1所示的SCOPE由微软开发。在这片论文中，作者提出了一种新的脚本语言，SCOPE（并行执行

的结构化计算优化），目的在于用户微软开发中的大规模数据分析。许多用户熟悉关系数据以及SQL。为了适应于新的运行环境，SCOPE有意地在这之上进行了简化。熟悉SQL的用户需要较少或不需要培训使用SCOPE。如同SQL，数据被建模成行和列的集合。所有的行集合都有良好定义的模式。SCOPE运行时提供了许多标准物理算子的实现，减少了用户重复实现相同功能的时间。SCOPE已被用于微软内部多种数据分析和数据挖掘应用。

SCOPE是一种声明式的语言。它允许用户将注意力放在解决问题所需的数据转换上，而隐藏了底层平台实现细节的复杂性。SCOPE编译器和优化器负责产生一个高效的执行计划，这样的计划拥有最小的开销。

SCOPE是高度可延伸的。用户能容易地定义他们自己的函数，并实现他们自己的算子版本：提取器（从文件中解析和构造数据行），处理器（以行为单位进行处理），规约器（以组为单位进行处理），和结合器（从两个输入上结合行数据）。这种灵活性极大地延伸了该语言的能力范围，允许用户解决那些不能被很容易地表示为SQL语句的问题。

SCOPE提供了类似于SQL里视图的功能。这个特征极大地提升了模块化和可重用性。这也能被用于敏感数据的限制性访问。

SCOPE支持使用传统的嵌套SQL表达式或一系列的简单数据转换编写程序。程序员更加爱偏好后者的风格，因为他们习惯于将计算想作一系列的步骤。

SCOPE脚本语言类似于SQL语言，但同时也包含C#表达式。这样的设计选择提供了许多优势。它的类SQL性质减少了用户的学习曲线，并能将已有SQL重用到SCOPE中。SCOPE表达式能使用C#库。自定义的C#类能计算以标量或整个行集为输入的函数。SCOPE脚本由命令序列组成。为了一些辅助的命令，一般的命令是将一个或多个行集作为输入，在数据上进行一些操作，然后输出一个行集的数据转换算子。每个行集有一个定义良好的模式，它的所有数据都能遵守。默认的，一个命令将之前命令所产生的结果行集作为输入。SCOPE命令也能使用命名的输入，以及用户用赋值符号命名一个命令的输出。一个命令的输出能被之后的命令使用一次或多次。命名的输入/输出能使得用户编写多步的脚本，这种风格是受程序员欢迎的。SCOPE支持许多数据类型，包括int，long，double，float，DateTime，string，bool以及他们对应的空值。SCOPE使用C#语义来描述空值，这于SQL的空值语义不同。空值与空值相等。空值不等于任何非空值。空值排序在前。SCOPE中的聚集函数忽略空值。脚本编写者能将所有的算子视为完全串行的；将脚本映射到高效的并行执行计划，这个过程完全按被SCOPE的编译器和优化器完成。

2.3. Hive

Apache Hive数据仓库软件用于查询和管理存储在分布式存储中的大数据集。Hive使用一种类似SQL的语言HiveQL，来将结构投影到数据上，并查询数据。于

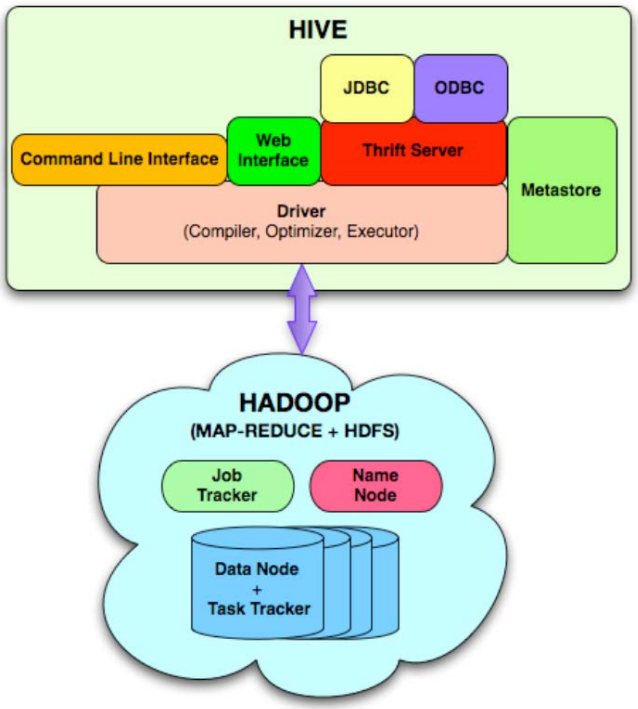

Figure 2. Hive系统架构

此同时，该语言也允许传统的map/reduce程序员将他们自定义的mappers和reducers整合到系统里，从而解决了不方便表示为HiveQL逻辑的计算。

如图 2所示，Hive由脸书公司开发。在这篇文章中，作者提出了Hive系统，一个建立在Hadoop之上的开源数据仓库解决方案。Hive支持将查询表示为类SQL的语言——HiveQL，它将被编译成mapreduce任务从而在Hadoop上执行。除此之外，HiveQL允许用户插入自定义的map-reduce脚本到查询语言里。该语言包括一个类型系统，其支持包含初始类型的表，像数组，映射和它们的嵌套组合。底层的IO库能被延伸成自定义的查询数据格式。Hive也包括一个系统目录——Metastore——其包括模式和统计数据，对数据探索，查询优化和查询编译产生作用。在Facebook，Hive仓库包括数以万计的表，存储了超过700TB的数据，这些数据被用于每月200个用户以上的报表和即席查询。

数据模型。Hive里的数据被组织为：表——类似于关系数据库里的表。每个表拥有一个对应的HDFS目录。表中的数据被序列化并存储在这个目录下的文件里。用户能将表和底层数据的序列化格式联系起来。Hive提供内建的序列化格式，它使用了压缩和惰性去序列化。用户也能为新的数据格式添加支持，比如定义称为SerDe的序列化和去序列化的方法，由Java编写。每个表的序列化格式存储在系统目录里，并在查询编译和执行阶段被自动地使用。Hive也支持存储在HDFS，NFS或本地目录下的外部表。分区——每个表能有一个或多个分区，这决定了数据在表

目录下的子目录里的数据分布。假设对表T的数据是存储在目录/wh/T下的。如果T在列ds和ctry进行了分区，然后拥有特定ds值20090101和ctry值US的数据，将被存储在/wh/T/ds=20090101/ctry=US目录下的文件里。桶——每个分区下的数据又根据表中列的哈希值被分配到不同的桶里。每个桶存储在分区目录下的一个文件里。

查询语言。Hive提供称为HiveQL的类SQL的查询语言，它支持select，project，join，aggregate，union all和子查询。HiveQL支持用数据定义（DDL）语句来创建特定序列化格式的表，分区，桶列。用户能从外部外部源中加载数据，并通过查询数据操作（DML）语句导入到Hive表中。HiveQL目前不支持更新和删除已有表里的行。HiveQL支持多表插入，用户能使用一个HiveQL语句完成相同数据上的多个查询。Hive通过共享输入数据的扫描来优化这些查询。HiveQL也是非常有延伸性的。它支持由Java实现的用户定义列转换（UDF）和聚集（UADF）函数。另外，用户能通过一个简单的基于行的流接口，比如从标准输入读行，将行写出到标准输出，嵌入由任何语言编写的mapreduce脚本。这种灵活性由将行转换为字符串的代价完成。

Hive架构。Hive的主要结构分为：外部接口——Hive同时提供了命令行（CLI）和浏览器UI，以及应用程序编程接口（API）如JDBC和ODBC。Hive Thrift服务器提供了一个分厂简单的客户端API用于执行HiveQL语句。Thrift是一个跨语言服务的框架，服务器端由一门语言编写（如Java），同时也支持客户端由其他语言编写。由不同语言生成的Thrift Hive客户端被用作建立共同的驱动器，如JDBC（java），ODBC（C++）和由php，perl和python等编写的脚本驱动器。Metastore是系统的目录。Hive的其他所有部分都与metastore进行交互。启动器管理了HiveQL语句的编译，优化，执行生命周期。当从thrift server或其他接口接受到HiveQL语句，它建立一个会话用来几轮执行时间，输出行数等统计数据。当接收到HiveQL语句被驱动器接受后，编译器将被调用。编译器将语句翻译为由一个有向无环图组成的mapreduce任务。驱动器将根据这个有向无环图的拓扑顺序，提交单独的任务到执行引擎。Hive目前使用Hadoop作为它的执行引擎。

3. Group II: HadoopDB

HadoopDB由如图 3所示，其被耶鲁开发。

关于在这样的环境中数据分析的技术来说，有两种趋势。并行数据库的支持者认为，并行数据库强调的性能和效率，使得其非常适用于如此场景。另一方面，其他人任务基于MapReduce的系统更加合适，因为它们的极佳的可扩展性，容错性和处理非结构化数据的灵活性。在这片论文中，作者探索了建设混合系统来结合两项技术优点的可能性；他们建立的原型系统在效率上接近并行数据库，仍保留基

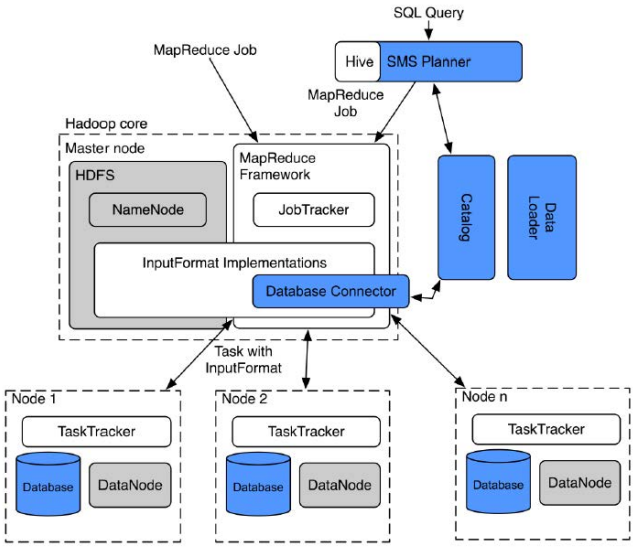

Figure 3. HadoopDB架构

于MapReduce系统的可扩展性，容错性和灵活性。

性能。性能是商业数据库系统用于将自己与其他解决方案区分的主要特性，因而其市场营销资料中经常声称一个特定的解决方案比其竞争对手快多少倍。在数量，质量和系统分析的深度上能造成数量级的差异。高性能系统优势也能节省成本。升级到一个更快的软件产品能延缓公司进行昂贵的硬件革新，或者当应用程序持续扩展时防止购买而外的计算节点。在公有的云计算平台上，定价是结构化的，用多少付多少，因而供应商价格将随必须存储，网络带宽和计算能力线性增长。因此，如果数据分析软件A比数据分析软件B需要高一个数量级的计算节点来做相同的任务，产品A将会比B多花费一个数量级。高效的软件对底线有直接的影响。

容错性。分析数据负载下的容错性与事务负载情景下的容错性在度量上是不同的。对于事务性负载，一个容错的数据库管理系统能从错误中恢复，并不丢失任何最近提交事务的数据或更新。在分布式数据库的情景下，甚至面临工作节点失效的情况，事务仍旧能提交，负载仍旧能继续。对于分析负载下的只读查询，没有写事务需要提交，节点失败更新丢失的情况不存在。因此，一个容错的分析数据库管理系统为，如果一个查询处理的节点失效了，而不必重启该查询。给定已证明的使用廉价的，不可靠的商用硬件来建立一个无共享的集群的操作收益和资源消耗节约，以及数据中心里极低端硬件的趋势，节点失效的概率在查询处理期间急剧增长。当扩展到大规模时，这个问题变得更严重：越大数量的数据需要被分析查询所读取，越多的节点需要参与到查询处理里来。这进一步增加了查询执行期间之上一个节点失效的概率。比如说Google，对于每个分析任务，平均有1.2个失效。如果每次节点失效后查询都需要重启，那么长的，复杂的查询很难完成。

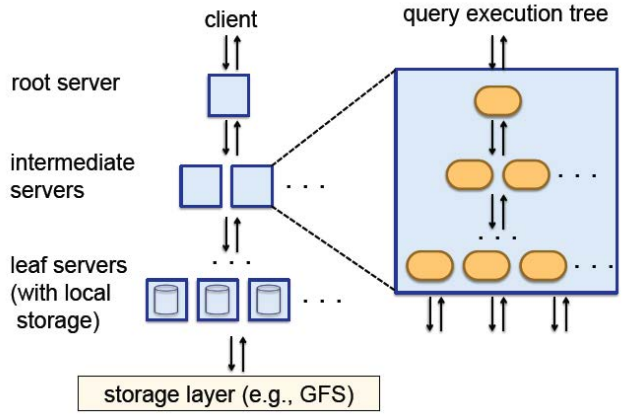

Figure 4. 系统架构和服务器节点里的执行

在异构环境中的运行能力。正如上面描述的，越来越多的节点参与到查询执行中来是一个很强的趋势。在成百上千的计算节点上获得同质性能几乎是不可能的，甚至假设每个节点都在相同的硬件或者相同的虚拟机上运行。部分失效不会造成完全的节点失效，但是将降低硬件性能，特别是规模变大以后。单个节点磁盘碎片和软件配置错误也能造成一些节点上的性能下降。并发查询（或一些情况下的，并发进程）进一步降低了集群性能的同质性。在虚拟机上，字啊相同物理机上的不同的虚拟机并发的活动将造成2-4%的性能差别。如果执行一个查询的工作量需要相等地分配在无共享集群的所有节点上，这样存在风险，完成查询的时间将近似等于完成分配给其任务的最缓慢的计算节点。性能下降的节点将对整个查询时间有不成比例的影响。被设计来在一个异质环境中运行的系统必须采取合适的度量来防止这种情况的发生。

灵活的查询接口。有多种面向客户的商务智能工具，与数据库软件协调工作，并在可视化，查询生成，结果报表化和高级数据分析上进行辅助。这些工具对于分析数据管理非常重要，因为商业分析员一般不擅长技术，从而对直接与数据库软件进行交互感到不舒适。商务智能工具一般使用ODBD或者JDBC连接数据库，而想要与这些工具协同工作的数据库必须通过这些接口接收SQL查询。理想地，数据分析系统应该拥有一种鲁棒性的机制，允许用户编写用户自定义函数（UDFs），并且使用UDFs的查询应该自动化地在无共享的集群上并行。这样，SQL和非SQL接口语言都是合理的。

4. Group III: Dremel

如图 4所示，Dremel由Google开发。Dremel是一个可扩展的，交互的即席查询系统，用于分析只读的嵌套数据。通过结合多层次的执行树和列式数据存储，能在万亿行的表上以数秒的时间完成聚集查询。该系统可扩展至数以千记的CPUs和PB级的数据，在Google内部拥有数以千记的用户。在这片论文中，

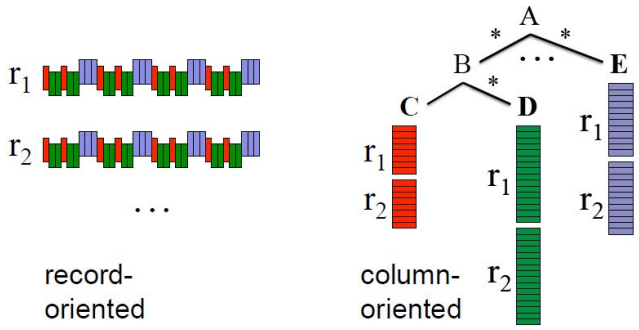

Figure 5. 记录级对列存储的嵌套数据

DocId

value	r	d
10	0	0
20	0	0

Name.Url

value	r	d
http://A	0	2
http://B	1	2
NULL	1	1
http://C	0	2

Links.Forward

value	r	d
20	0	2
40	1	2
60	1	2
80	0	2

Links.Backward

value	r	d
NULL	0	1
10	0	2
30	1	2

Name.Language.Code

value	r	d
en-us	0	2
en	2	2
NULL	1	1
en-gb	1	2
NULL	0	1

Name.Language.Country

value	r	d
us	0	3
NULL	2	2
NULL	1	1
gb	1	3
NULL	0	1

Figure 6. 简单数据的列条纹表示，展示重复级别（r）和定义级别（d）

作者描述了Dremel的架构和实现，并解释了其如何弥补了基于MapReduce的计算。我们为嵌套的记录提出了一种新颖的列式数据表示，并讨论了在数千节点上的系统实验。

数据模型。数据模型来源于分布式系统的场景（‘Protocol Buffers’），广泛地用于Google内部，并有相应的开源实现可使用。数据模型基于强类型的嵌套记录。嵌套的数据模型以一种Google内部的平台无关的，可延伸机制的，序列话的结构数据作为支持。代码生成工具为如C++或Java的程序语言产生绑定。通过使用一个标准的二进制记录表示，其中域值按记录里出现的次序顺序地表示，从而达成了跨语言的互操作性。这方面，一个由Java编写的MR程序能读取由C++库产生的数据源。因而，如果记录存储为列表示，快速地组装它们对于与MR和其他数据工具进行互操作是非常重要的。

嵌套列存储。值单独不能表示记录的结构。给定一个重复域的两个值，我们不知道值在什么“级别”上重复（例如，是否这些值来自于两个不同的记录，或一个相同记录的两个重复值）。类似的，给定一个丢失的可选域，我们不知道哪个封闭性的记录被显式都定义了。因此，我们引入了重复和定义级别，定义

如下。之前我们展示了记录结构在列格式上的编码。我们迎接的下一个挑战是，如何高效地产生带有重复和定义级别的列条纹。算法递归如记录结构，计算出每个域值的级别。正如之前阐述的，甚至加入域值丢失时，重复和定义级别需要被计算。Google内部的许多数据集都是稀疏的；一个模式拥有数以千记的域不太常见，给定数据中一般仅一百域。因此，我们尝试尽可能低消耗地产生丢失域。为了产生列条纹，我们创建一个域写者的树，它的结构与模式中的域结构匹配。主要的思想是仅当它们有自己的数据时，更新域写者，并不尝试扩散父亲的状态到树下，除非非常需要。为了做到这点，孩子写者从他们的父亲那集成级别。一旦新值被加入了，孩子写者同步它们的父亲的级别。

5. 未来工作

对于未来工作，我们能进一步将内存计算技术引入到大规模联机分析处理中来，以进一步增强系统的处理性能。内存数据库（IMDB；也称内存数据库系统或MMDB或常驻内存数据库）是一种数据库管理系统，优先使用内存作为计算机数据内存。它与使用磁盘存储机制的数据库管理系统想对应。内存数据库比磁盘优化的数据库更快，因为其内部的优化算法更简单，并执行更少的CPU指令。在内存中存储数据消除了查询数据所产生的寻址时间，这相对于磁盘，提供了更快，更可预测的性能。当应用程序的响应时间非常重要，比如那些运行电信网络设备和移动广告网络，一般使用内存数据库。IMDBs得到许多牵引力，特别是在数据分析领域，开始于20世纪中叶，主要是源于更加廉价的RAM。随着引入非易失性随机存储内存技术，内存数据库将能高速运行，并在电源失效后依旧保持数据。特别地，考虑整合列存储的内存数据库MonetDB [4]和行存储的内存数据库VoltDB [10]到基于Hadoop的系统中。增加基准数据集的数据量来找内存数据库的瓶颈。被使用并行数据库中的优化技术例如分区来缓解内存数据的瓶颈。

References

[1] A. Abouzeid, K. Bajda-Pawlikowski, D. J. Abadi, A. Rasin, and A. Silberschatz. Hadoopdb: An architectural hybrid of mapreduce and DBMS technologies for analytical workloads. *PVLDB*, 2(1):922–933, 2009.

[2] A. Abouzied, K. Bajda-Pawlikowski, J. Huang, D. J. Abadi, and A. Silberschatz. Hadoopdb in action: building real world applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*, pages 1111–1114, 2010.

[3] K. Bajda-Pawlikowski, D. J. Abadi, A. Silberschatz, and E. Paulson. Efficient processing of data warehousing queries in a split execution environment. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, pages 1165–1176, 2011.

[4] P. A. Boncz, M. L. Kersten, and S. Manegold. Breaking the memory wall in monetdb. *Commun. ACM*, 51(12):77–85, 2008.

[5] R. Chaiken, B. Jenkins, P. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. SCOPE: easy and efficient parallel processing of massive data sets. *PVLDB*, 1(2):1265–1276, 2008.

[6] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004*, pages 137–150, 2004.

[7] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. Dremel: Interactive analysis of web-scale datasets. *PVLDB*, 3(1):330–339, 2010.

[8] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 1099–1110, 2008.

[9] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*, pages 165–178, 2009.

[10] M. Stonebraker and A. Weisberg. The voltdb main memory DBMS. *IEEE Data Eng. Bull.*, 36(2):21–27, 2013.

[11] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive - A warehousing solution over a map-reduce framework. *PVLDB*, 2(2):1626–1629, 2009.

[12] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Anthony, H. Liu, and R. Murthy. Hive - a petabyte scale data warehouse using hadoop. In *Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*, pages 996–1005, 2010.