树状结构大数据类型的高效支持

陈世敏
中国科学院计算技术研究所，北京 100190

摘要
传统的关系数据模型难以满足大数据应用日益丰富的数据表达和处理的需求，因此实践中涌现了多种非传统的大数据类型。其中，以JSON为代表的树状结构大数据类型被广泛应用，具有重要的理论意义和实用价值。系统介绍了树状结构大数据类型，并探讨如何高效支持树状结构大数据的分析运算。

关键词
树状结构大数据；JSON；赤兔

中图分类号：TP315 文献标识码：A doi: 10.11959/j.issn.2096-0271.2018038

Efficient support of tree-structured data types

CHEN Shimin
Institute of Computing Technology Chinese Academy of Sciences, Beijing 100190, China

Abstract
Traditional relational data model cannot meet the demand of big data applications for expressing and processing wide varieties of data. As a result, a number of non-relational data types have become popular in practice, among which JSON-like tree-structured data types have been widely adopted. Tree-structured data types have important theoretical and practical values. A systematic description of tree-structured data types was provided, and the way to efficiently support data analysis operations on tree-structured data was investigated.

Key words
tree-structured data, JSON, Steed

1 引言

大数据产业是全球高科技竞争的前沿领域。大数据技术的推广应用对国家经济、政治、法治、科技、文化、教育、民生、社会、生态文明、国家安全等方面，都会产生深远的影响。传统的关系数据模型从20世纪70年代出现至今，在商用数据处理方面得到了广泛的应用。但是，关系模型的简单、扁平的二维表结构无法满足各行各业（如社交网络、物联网、医疗生物、金融等）日益丰富的大数据表达和处理的需求。于是，实践中涌现了多种非传统的大数据类型，出现了一批支持非传统数据类型的大数据系统，被统称为NoSQL数据库系统。其中，应用最广泛的是键值对（key-value）数据类型、图（graph）数据类型和以JSON（JavaScript object notation）等为代表的树状结构数据类型（tree-structured data type）。

树状结构数据类型可直观地表达高级程序设计语言中类（class）、结构（struct）等丰富的结构，能够简洁地支持嵌套、多值和缺值，已被广泛应用于社交网络数据服务、Web服务、数据交换格式、分布式系统协议、物联网等，是一种重要的大数据类型。实践中常见的树状结构数据类型有JSON、Protocol Buffers等。JSON是JavaScript语言标准的一个子集，常常作为数据输出和数据交换的类型。Protocol Buffers是Google公司推出的一种数据类型，是实现分布式系统内部通信协议的数据格式，也是Google公司的Dremel[1]和BigQuery[2]等大数据系统的数据类型。

本文将对以JSON为代表的树状结构大数据类型进行深入介绍，首先举例说明树状结构大数据的含义，然后从多个角度说明树状结构大数据的价值和意义，最后结合笔者近期的研究工作，说明树状结构大数据类型的处理和支持。

2 树状结构大数据类型

树状结构大数据类型包括多种新型数据类型，例如JSON、Protocol Buffers、Apache Avro等。这些数据类型的具体表现形式不同，但是它们都可以表达嵌套、多值、缺值等丰富复杂的记录结构，具有相似的结构特点，可以互相转化。它们的记录结构都可以采用语法树来表达，因此将它们统称为树状结构大数据类型。

以JSON为例，一个JSON类型的数据记录如下：

```
{ "geo" :{ "coordinates" :[-7.1,13.4]},
  "retweet_count" :0,
  "user" :{ "status" :28156,
        "favorite" :0,
        "followers" :5740,
        "id" :32
       }
}
```

JSON用花括号、方括号、引号、冒号、逗号等标点符号来表达记录的结构。具体而言，花括号表示一个对象，对象的多个属性以逗号隔开。方括号表示一个数组，数组的多个元素以逗号隔开。一个属性由属性名和属性值组成，两者由冒号隔开，属性名以引号标注，属性值可以是原子的字符串、数值、布尔值等类型，也可以是嵌套的对象或数组。在这个例子中，记录是一个对象，由3个顶层属性geo、retweet_count和user组成。geo属性的属性值是一个嵌套对象，内部只有一个coordinates属性，其属性值是一个数组，数组的每个

元素是一个数值；retweet_count属性的属性值是一个数值；user属性的属性值是一个嵌套对象，包括status、favorite、followers和id 4个属性，而这4个属性的属性值都是数值。

将上述记录的内部结构表达为一棵树，如**图1**所示。

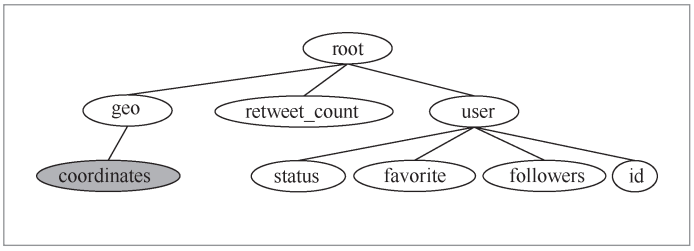

图1 对应于JSON记录实例的语法树

树根代表整个记录。树中的一个节点对应记录中的一个属性，叶子节点对应属性值为原子类型的属性，而非叶子节点则对应属性值是嵌套值的属性。灰色的节点代表多值，即数组的情况。在这个例子中，最高层的3个属性geo、retweet_count和user对应根的3个孩子节点。其中，retweet_count的属性值是原子的数值类型，因此retweet_count节点没有孩子节点，是一个叶子节点。user的属性值是一个嵌套对象，由status、favorite、followers和id 4个属性组成，因此user节点有4个孩子节点。而这4个属性的属性值都是原子类型，因此它们都是叶子节点。geo属性的属性值是一个嵌套对象，包括一个coordinates属性，因此geo节点有一个coordinates孩子节点。coordinates属性的属性值是一个数组，而数组的每个元素是原子类型，用灰色的coordinates节点表达数组，该节点没有孩子，是叶子节点。

JSON、Protocol Buffers、Apache Avro等多种树状结构数据类型都可以把记录结构表达为语法树的形式。可以通过如下递归定义更确切地表达树状数据类型T_{tree}：

$T_{tree} = T_{object}$

$T_{object} = \{key1: T_{value1}, \cdots, keyn: T_{valuen}\}$

$T_{array} = [T_{value}, \cdots, T_{value}]$

$T_{primitive} = string \mid number \mid boolean \mid null$

$T_{value} = T_{primitive} \mid T_{object} \mid T_{array}$

一个树状数据记录可以递归地表达为一棵树，T_{tree}是树根，树根必须是T_{object}，每个T_{object}由属性名key和属性值value组成。除了T_{object}，还定义了数组T_{array}和原子类型$T_{primitive}$。而value类型则可以递归地定义为原子类型、对象类型、数组类型。树状结构数据类型可以表达多层复杂的嵌套，每层嵌套表现为树的一个内部节点，而树的叶子节点是原子类型。

3 实用价值

与关系数据模型相比，树状结构大数据类型的每个记录具有更加灵活丰富的结构，因此在实践中得到了广泛的应用。例如，社交网络数据服务Twitter等输出的数据类型就是JSON，Web 2.0 RESTful架构中推荐的数据交换格式是JSON，许多提供公共数据下载的网站可以使用JSON下载数据，Apache Hadoop、HBase等开源大数据系统中分布式通信协议采用了Protocol Buffers来实现。此外，许多物联网单片机芯片（如Arduino、DragonBoard、BeagleBone等）支持JSON格式的数据输出。大量的原始数据是树状结构数据类型。

树状结构大数据类型的3个主要特点使其具有广泛的实用价值，具体如下。

（1）丰富的结构

树状结构大数据类型支持嵌套、多

值、缺值等丰富的结构，可以非常方便地表达程序设计语言中“对象”等复杂类型的数据，例如C语言中的struct、C++/Java等语言中的class、Python语言中的dictionary等类型。因此，可以采用树状结构大数据类型自然地对应用程序的内存数据结构进行序列化，便于写入外存和进行网络通信，并保持原始内存数据结构的特征。与之相比，关系数据模型采用简单、扁平的记录结构，记录的每个属性都是原子类型，因此对于应用程序数据结构中的嵌套、多值等情况来说，必须采用特殊的编码，将数据转换为关系数据模型允许的记录格式，才可以完成序列化。可见，树状结构大数据类型丰富的结构可以更好地支持大数据时代多种多样的应用。

（2）灵活的类型

JSON类型不需要事先定义记录的类型就可以直接使用。关系数据库中先要采用create table建立数据表，才可以加载或插入数据记录。而在JSON中，数据记录的结构是在每条记录中定义的。如第2节中的例子，JSON采用标点符号定义每条记录的具体结构。因此，JSON允许灵活地增、删、改记录结构，允许将多种结构的记录放入相同的数据集，这可以简化很多大数据应用领域的数据存储和管理。例如，在数据采集中，经常遇到同种数据可能有很多小的类别的情况，虽然所有数据都有一些公共属性，但是不同类别还包括许多各自特殊的属性。在这种情况下，如果采用关系模型，就需要为每个小类别采用create table建立一个新表，数据存储和后续的数据处理就需要面对大量的关系表，使整个过程十分繁杂。而采用JSON树状结构数据类型，就可以把所有小类别的数据记录都存储在一个数据集中，每个记录允许不同的属性（实际上是允许许多缺值的属性）在一个数据集中对所有数据进行统一的管理，大大简化了数据管理、编程处理的成本。

（3）低廉的成本

XML也可以表达丰富的嵌套、多值结构。实际上，数据库领域一直希望推动XML成为应用数据存储和交换的标准。但是，XML的表达引入了很高的成本，包括DTD类型的定义和解析、XML标签占用的空间等。与之相比，JSON等树状结构数据类型更加简洁轻量。以JSON为例，记录结构采用简单的标点符号来表达，便于人的理解和程序的解析，而且这种表达方式引入的空间开销很小。因此，在实践中JSON等树状结构数据类型已经逐渐取代了XML，成为事实上数据存储和交换的标准。

4 理论意义

树状结构大数据类型不仅具有很强的实用价值，而且具有重要的理论意义，主要表现为以下两个方面。

（1）关系数据模型的第4次变革

在关系模型出现之后，数据库研究领域已经认识到了关系模型表达能力的局限性，曾经3次试图扩展关系数据模型，以支持更加丰富复杂的结构：20世纪80年代后期到20世纪90年代初期的面向对象的数据库（object oriented database）、20世纪90年代中后期的关系对象式数据库（object relational database）和21世纪初的XML[3]。这些尝试取得了一定的学术成果和产业应用，例如关系对象式数据库被PostgreSQL等系统支持，XML形成了国际标准，但是它们都未竟全功，没有得到广泛的接受和使用。树状结构大数据类型实际上可以看作关系数据模型的第4次变革。目前，JSON等类型已经被产业界广泛接受。

产业界的推动与学术界的多次尝试和经验积累，可能引发质变，使这次新的尝试有很大的成功概率，产生能够代替关系数据模型的新数据模型。

（2）一种通用的大数据模型

树状数据结构类型可以表达丰富的结构，包括嵌套、多值、缺值等结构。相对于单个键值对、单个图的顶点、单条图的边、单个图的属性、单条关系型记录等，单条树状结构的数据记录的结构可以更加丰富复杂。因此，实际上很容易把键值对、图的顶点、边、属性和关系型记录写成树状数据结构类型的数据，而且可能有多种不同的写法，从而可能以树状结构大数据类型为基础，实现对其他流行的大数据类型的支持。因此，树状结构大数据类型是一种通用的大数据模型。

树状结构大数据类型具有很强的表达能力，其难点在于如何高效地支持树状大数据类型的存储和运算，以支持其丰富、灵活的结构。

5 现有的树状结构数据处理系统

现有的树状结构数据处理系统主要有以下3种。

（1）扩展关系型数据库系统

主流的关系数据库系统Oracle、Microsoft SQL Server、IBM DB2和开源数据库系统PostgreSQL等都扩展了对JSON的支持。基本思路是将整个JSON记录以文本或者二进制格式存放在关系表的单个属性中，提供内置函数，支持JSON的解析和访问，从而可以在SQL语句中动态地解析JSON记录，提取JSON属性值，并用于SQL查询[4]，这也是SQL/JSON工作组的基本解决方案。但是，这种解决方案对数据分析的支持较差。数据分析操作通常只关心JSON记录的少量属性，存储和读取整条JSON记录会导致大量不必要的I/O访问。而且，每次执行SQL查询语句，都要动态地解析JSON记录，引入很大的性能开销。

（2）行式树状结构数据处理系统

以MongoDB为代表的文档存储（document store）系统支持JSON的行式存储和处理。MongoDB是通过C/C++实现的，采用二进制的BSON格式存储JSON记录。对于JSON的属性名，BSON仍然存储其字符串；而对于JSON的原子属性值，BSON采用二进制存储。MongoDB提供一组JavaScript编程界面，可以执行与SQL查询功能相当的操作。和扩展关系型数据库系统相似，由于采用行式存储，数据分析操作会导致大量的I/O开销。此外，在访问JSON嵌套结构时，MongoDB需要在每个嵌套层次进行字符串比较，搜索对应的属性名，性能代价较大。

（3）列式树状结构数据处理系统

Google Dremel提出了Protocol Buffers数据的列式存储编码方式[1]。Apache Parquet是Dremel的开源实现，支持Parquet格式的文件存取和访问。与Apache Hive结合，就可以将数据存放在Parquet列式文件中，并利用Hive实现基于MapReduce的SQL查询，对大规模的树状数据进行分析。由于采用了列式存储，Parquet可以有效地避免读取不相关属性的I/O操作。但其基于MapReduce和Java的实现影响了查询的效率。

上述3种系统都采用完全通用的设计，为了支持树状结构数据类型可能出现的丰富、复杂的嵌套和多值结构，引入了复杂的算法。例如，为了把多个分别存储的数据列组装还原成原始记录，Dremel的组装算法要建立一个有限状态自动机，根据自动机和列式文件中的特殊编码完成组装。除

了上述讨论每种系统各自的性能问题外，这种完全通用的解决方案本身也存在相对高昂的代价。

6 树状大数据的高效支持：赤兔

笔者设计实现了一个树状结构数据管理系统——赤兔（system for tree structured data, Steed）[5]。Steed是采用C/C++语言实现的，支持通用的树状结构数据存储和类似SQL的查询处理，包括选择、投影、连接、分组、聚集、排序等多种运算。Steed同时支持行式和列式的树状结构数据存储，以适应不同类型应用的需要。系统能够自动提取JSON的语法树，从而有效地压缩了对属性名的存储。**图2**展示了Steed的系统结构，主要包括数据解析模块、数据存储模块和查询执行模块。数据解析模块读取并解析文本的JSON或Protocol Buffers数据，将其转化为行式或者列式的二进制格式，存储在数据存储模块中。数据存储模块存储行式或列式二进制数据，支持两种格式数据的相互转换。查询执行模块支持类SQL的查询，支持行式和列式数据的查询处理。整个查询执行采用传统关系型数据库中查询树的方式实现。

（1）简单路径及其优化

通过对现实的树状结构数据进行分

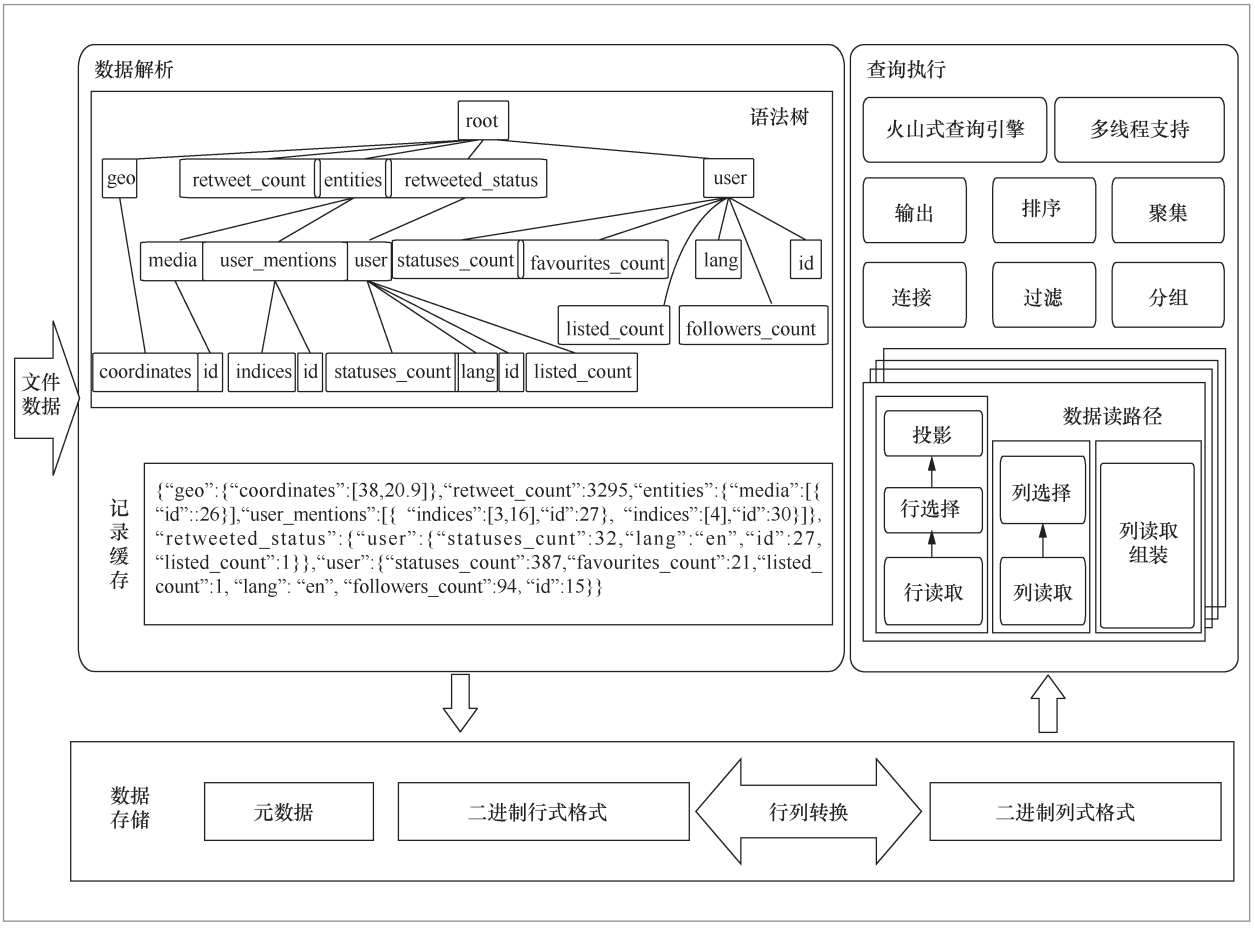

图 2 Steed 系统结构

析，提出了一种频繁子模式——简单路径（simple path）。在树状结构数据的语法树中，存在许多从树根到叶子的路径。如果在根到叶子的路径上存在很多的多值（数组）节点，那么其存储和处理就要考虑很多可能出现的情况，相对复杂。相反，当路径中不存在多值节点或仅存在一个多值节点时，就有可能进行简化的处理。这种一条包含最多一个多值节点的从树根到叶子的路径就是简单路径。笔者分析了现实的树状结构数据的结构，发现简单路径大量存在。例如，Twitter数据的语法树中包含203个叶子节点，其中195个叶子节点对应的路径是简单的，即96%的路径是简单路径。此外，还分析了Yahoo、IMDB、Sina Weibo等多种表述性状态传递（representational state transfer，REST）服务数据，发现超过99%的路径是简单的。其中，Sina Weibo提供104种不同的调用服务，这些服务数据对应的语法树中67%的路径不包含多值节点，32%的路径包含一个多值节点，只有1%的路径是包含两个到多个多值节点的复杂路径。笔者分析了Apache Hadoop中所有449种非空的基于Protocol Buffers的通信协议，发现97%的路径是简单路径。在其他多类现实数据集中，都观察到相似的现象：绝大多数路径是简单路径。针对简单路径，Steed优化了列式存储、列式组装和内存行式格式。

（2）MongoDB+Steed

MongoDB是对JSON文件进行存储和处理的大数据系统。MongoDB是采用C/C++语言实现的，包括Mongod和Mongos两个主要部分。其中，Mongod负责单机的数据管理和运算，而Mongos在Mongod的基础上实现了分布式处理，提供了数据划分、备份、分布式执行等功能。目前，MongoDB以行式BSON记录作为数据的存储格式，对大规模数据分析的运算可能引起大量额外的I/O操作。而Steed支持JSON的二进制列式存储，可以很好地支持大规模的数据分析运算。因此，将Steed作为MongoDB的存储引擎，就有可能使MongoDB对JSON数据进行列式I/O操作，从而大大提升大数据分析的效率。此外，目前Steed的实现是一个单机的数据库系统，而MongoDB在分布式处理方面有较强的能力，因此将Steed作为MongoDB的存储引擎，就可以自然地利用MongoDB的分布式处理能力，形成一个能够支持多机分布式存储和运算的树状结构大数据系统。MongoDB已经在实践中得到了广泛应用，在最流行的数据库引擎排名中名列第5位。若MongoDB+Steed仍然采用MongoDB的前端界面和编程语言，就有可能更容易地被广大MongoDB用户接受，因此笔者将MongoDB的后端WiredTiger存储引擎替换为Steed，使用列式存储读取数据，并转化为BSON，使用MongoDB现有的内存处理。在具体实现中，需要把上层查询运算的相关信息发送到下层的Steed存储引擎。只有这样，Steed才可能得知有哪些属性列参与了当前的查询运算，从而可以采用列式的访问读取这些列的相关信息，而不是访问全部的列，达到减少I/O开销、提升效率的目的。

（3）性能比较

Steed和现有的树状结构数据处理系统PostgresQL、MongoDB、Hive+Parquet的性能对比如**图3**所示。Steed Row和Steed Column分别采用行式和列式数据存储。实验是在单台联想ThinkCentre M8500t工作站上运行的，工作站配有一个3.4 GHz 4核的Intel Core i7-4770处理器、16 GB内存和一个7 200 r/min的SATA硬盘。在Twitter数据上运行多种查询操作，**图3**中从左到右依次是选择一

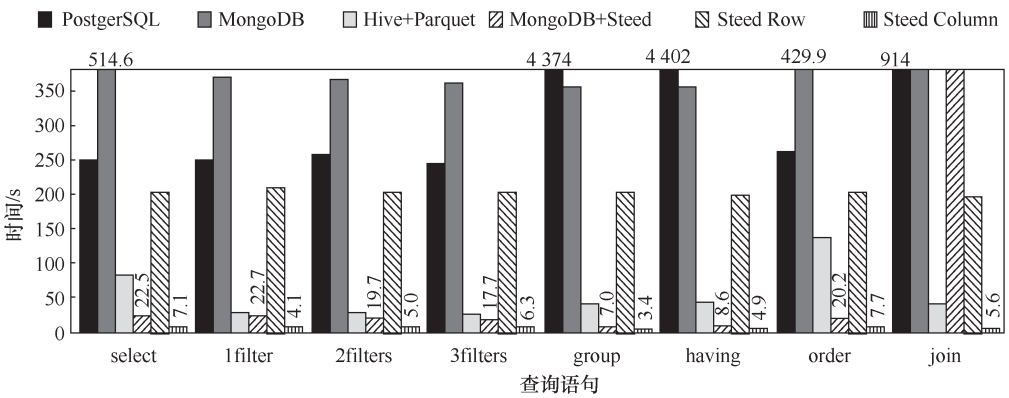

图 3 Steed 和现有树状结构数据处理系统的性能对比

个属性(select)、使用1~3个条件过滤数据集(1filter、2filters、3filters)、分组统计(group)、在分组统计的结果上进一步过滤(having)、排序(order)、连接(join)。总体而言,Steed Column性能在所有系统中最优,比Hive+Parquet提高4.1~17.8倍,比MongoDB提高55.9~105.2倍,比PostgreSQL提高33.8~1 294倍。MongoDB+Steed采用了列式存储,比采用行式存储的原始MongoDB性能提升16~51倍。把BSON格式改变为Steed的内存结构,节省了对字符串属性名的比较,采用了更高效的查询处理实现,Steed Column比MongoDB+Steed性能提升1.8~5.5倍。

7 结束语

传统的关系数据模型难以满足大数据应用日益丰富的大数据表达和处理需求,因此实践中涌现了多种非传统的大数据类型。其中,以JSON为代表的树状结构大数据类型是一种重要的大数据类型。本文从数据模型、实用价值和理论意义等方面介绍了树状结构大数据类型,探讨了树状结构大数据类型的高效处理,设计实现了一个树状结构数据管理系统——Steed系统,支持通用的树状结构数据存储和类似SQL的查询处理。通过分析现实数据,提出一种频繁子模式——简单路径,简单路径在实际数据中大量存在。针对简单路径,Steed优化了外存存储、列组装算法和内存行式结构。与现有系统PostgreSQL、MongoDB、Hive+Parquet相比,Steed对数据分析类的操作普遍有10~1 000倍的性能提升。

参考文献:

[1] MELNIK S, GUBAREV A, LONG J J, et al. Dremel: interactive analysis of web-scale datasets[J]. PVLDB, 2010, 3(1): 330-339.

[2] Google. An inside look at google bigquery[R]. 2016.

[3] CAREY J M. Beyond rows and columns: Is the fourth time the charm[C]// IEEE International Conference on Data Engineering (ICDE), May 16-20, 2016, Helsinki, Finland. New Jersey: IEEE Press, 2016.

[4] LIU Z H, HAMMERSCHMIDT B, MCMAHON D. Json data management: supporting schema-less development

in rdbms[C]// 2014 ACM SIGMOD International Conference on Management of Data, June 22-27, 2014, Snowbird, USA. New York: ACM Press, 2014.

[5] WANG Z Y, CHEN S M. Exploiting common patterns for tree-structured data[C]// 2017 ACM SIGMOD International Conference on Management of Data, May 14-19, 2017, Chicago, USA. New York: ACM Press, 2017.

作者简介

陈世敏(1973-),男,中国科学院计算技术研究所研究员,中国计算机学会大数据专家委员会委员和数据库专家委员会委员,分别于1997年和1999年获得清华大学计算机系学士和硕士学位,于2005年在美国卡耐基梅隆大学获得计算机科学博士学位。曾担任期刊《PVLDB》2017年的副主编,国际会议ICDE 2018、ICDCS 2016、CIKM 2014的PC领域主席。主要研究方向为数据库系统和大数据处理。

收稿日期: 2018-05-15