分 类 号 密 级

U D C 编 号 10486



**硕 士 专 业 学 位 论 文**

基于多维度聚合金字塔的

时空数据集可视化方案研究

研 究 生 姓 名 ： 周炜轩

学 号 ： 2016286190093

指导教师姓名、职称 ： 宋爱红 吴华意

专业类别（领域） ： 测绘工程

二〇一八年五月

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Thesis for Master Degree | | | | |
|  | | | | |
|  | | | | |
| **Visualization of Massive Spatial-temporal Datasets**  **based on Multi-dimension Aggregation Pyramid Model** | | | | |
|  | Master. Candidate: |  | Zhou Weixuan |  |
| Supervisor: |  | Aihong Song  Prof. Huayi Wu |
| Speciality: |  | Surveying and Mapping Engineering |
|  |  |  |
|  |  |  |
|  | | | | |
| May 23,2018 | | | | |

论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的研究成果。除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

学位论文作者（签名）：

年 月 日

摘要

在大数据时代，数据可视分析是一项非常重要的研究内容，其有助于快速、直观地理解分析大数据蕴含的价值信息。然而，因其海量、时空、高维等特征，大数据可视化存在内存消耗大、渲染延迟高、可视效果差等问题。现有的解决方案主要分两类：一是借助昂贵的高性能硬件提升效率，二是基于预处理结果可视化缩短在线聚合时间。然而，前者成本较高，难以普遍推广；后者存在时空高维支持不够、扩展性较差等问题。

针对上述问题，本文以海量时空点数据为例，采用预处理结果可视化方案，设计并实现了一套高可扩展的分布式可视分析框架。首先，本文借鉴瓦片金字塔模型提出了一种多维度聚合金字塔数据组织模型（Multi-dimension Aggregation Pyramid，MAP），该模型将瓦片金字塔的2D空间层级聚合扩展到时/空/属性多维度，同时支持时间、空间、属性的多维层级聚合。进而，本文以Spark集群作为并行预处理工具，以HBase分布式数据库持久化存储MAP模型数据，实现了预处理能力和存储能力的可扩展。最后，以生成的MAP模型数据为基础，本文设计实现了一套开源的分布式可视化框架（MAP-Vis），分布式处理能力支持下系统能实现亚秒级的查询响应，达到良好的交互式可视化效果。

实验表明：本文提出的基于MAP模型的MAP-Vis可视化框架系统，在时间跨度拉长、数据量增大的情况下，查询时间消耗并未出现线性的增长，依然能够保持较快的响应速度；而随着更多节点的纳入，查询的响应能力和预处理能力得到进一步的提升。

**关键词：**大数据；时空多维；层级聚合；可视化分析；分布式数据库

**ABSTRACT**

For big data, visual data analysis is a very important research topic, which helps to quickly and intuitively analyze and understand the inherent values of big data. However, because of its massive volume, spatio-temporal coordinates and high dimensions, big data visualization poses much challenges, such as large memory consumption, high rendering delay, and poor visual effects. The existing solutions can be divided into two categories: one is to increase the processing efficiency with expensive high-performance hardware, and the second is to shorten the online aggregation time based on the visualization of the pre-processing results. The former needs much higher cost and it is not affordable for normal users, while the latter provides poor scalability for high-dimensional spatio-temporal datasets.

To address these problems, this paper takes massive spatio-temporal point datasets as an example, adopts a pre-processing visualization scheme, and designs and implements a highly scalable distributed visual analysis framework. Firstly, the paper proposes a multi-dimension aggregate pyramid (MAP) model based on the well-known tile pyramid model. This model extends the hierarchical 2D-spatial aggregation of tile pyramids to multiple dimensions of time/space/attribute. It can supports multi-dimensional aggregation of time, space, and attributes. Furthermore, this paper uses the Spark cluster as parallel preprocessing platform and distributed HBase as final storage to store the generated MAP data. This greatly increases the scalability of preprocessing capability and the storage capacity. Finally, with the generated MAP datasets, this paper designs and implements an open-source distributed visualization framework (MAP-Vis). With the distributed processing capabilities, it can achieve sub-second query response and achieve a good interactive visualization effect.

Experiments show that by the multi-dimensional aggregation of the MAP datasets, the query time has very small fluctuation and maintain a fast response speed with the increase of spatio-temporal span and the target data size. When fed with more computing nodes, the query response and preprocessing capabilities have been further improved.

Keywords: big data, multi-dimensional, hierarchical aggregation, distributed database, visual analysis

**目录**

[摘要 I](#_Toc513736410)

[ABSTRACT II](#_Toc513736411)

[第1章 绪论 1](#_Toc513736412)

[1.1. 研究背景与意义 1](#_Toc513736413)

[1.2. 国内外相关研究 2](#_Toc513736414)

[1.2.1. 大数据可视化概述 2](#_Toc513736415)

[1.2.2. 大数据可视化研究的分类 3](#_Toc513736416)

[1.3. 主要研究内容 5](#_Toc513736417)

[1.4. 论文组织结构 7](#_Toc513736418)

[第2章 多维度聚合金字塔（MAP）组织模型 8](#_Toc513736419)

[2.1. 瓦片金字塔模型概述 8](#_Toc513736420)

[2.2. MAP数据组织模型 10](#_Toc513736421)

[2.2.1. MAP模型的基础定义 10](#_Toc513736422)

[2.2.2. MAP模型的构建操作 12](#_Toc513736423)

[2.3. 本章小结 14](#_Toc513736424)

[第3章 基于MAP模型的数据库存储方案 15](#_Toc513736425)

[3.1. 分布式数据库HBase概述 15](#_Toc513736426)

[3.1.1. HBase简介 15](#_Toc513736427)

[3.1.2. HBase协处理器 19](#_Toc513736428)

[3.2. MAP-Vis框架数据存储模型设计 21](#_Toc513736429)

[3.2.1. HBase表结构设计与模型映射 21](#_Toc513736430)

[3.2.2. HBase表主键编码设计 23](#_Toc513736431)

[3.3. 时/空/属查询方法 25](#_Toc513736432)

[3.3.1. 空间查询 26](#_Toc513736433)

[3.3.2. 时间查询 26](#_Toc513736434)

[3.3.3. 属性查询 27](#_Toc513736435)

[3.4. 本章小结 27](#_Toc513736436)

[第4章 海量高维时空数据的预处理 28](#_Toc513736437)

[4.1. Spark分布式计算框架 28](#_Toc513736438)

[4.1.1. Spark概述 28](#_Toc513736439)

[4.1.2. 弹性分布式数据集（RDD） 30](#_Toc513736440)

[4.1.3. Spark的运行机制 32](#_Toc513736441)

[4.2. 高维时空数据预处理 34](#_Toc513736442)

[4.2.1. 总体预处理算法流程 34](#_Toc513736443)

[4.2.2. 具体算子函数解释 36](#_Toc513736444)

[4.3. 实验结果及分析 39](#_Toc513736445)

[4.4. 本章小结 40](#_Toc513736446)

[第5章 基于MAP模型的可视化系统实现 41](#_Toc513736447)

[5.1. 系统总体概述 41](#_Toc513736448)

[5.1.1. 系统设计框架 41](#_Toc513736449)

[5.1.2. 系统开发的具体环境 42](#_Toc513736450)

[5.1.3. 系统主要功能 42](#_Toc513736451)

[5.2. 实验设计与结果分析 44](#_Toc513736452)

[5.3. 本章小结 48](#_Toc513736453)

[第6章 总结与展望 49](#_Toc513736454)

[6.1. 论文总结 49](#_Toc513736455)

[6.2. 展望 49](#_Toc513736456)

[参考文献 51](#_Toc513736457)

[致谢 55](#_Toc513736458)

# 绪论

## 研究背景与意义

如今，随着数据的采集手段成熟与多样化，如个人设备定位、浮动车GPS、物联网、社交媒体等，数据源日益丰富。这些体量巨大、流式生成、类型众多、价值蕴含（5V）的大数据[1]，兼具有时空和高维特征。时空特征，是指大数据都带有空间位置和时间标签或者能体现二者的特征属性；高维特征，是指数据一般会包含多个维度的属性，而这些属性所能反映出的信息规律往往让研究者更感兴趣。其中尤以点数据体现得最为明显，最为典型，例如广泛应用的POI数据等。

对于具备此特征的大数据，可视化无疑是挖掘和分析大数据的一种重要步骤和手段，其能够直白地反映出数据中蕴含的模式和规律。随着数据规模的日渐增大，以往常规的数据可视化方法和工具越来越不适应。以点状数据可视化为例，如果逐点进行可视化，海量的数据点不但容易造成彼此堆叠覆盖，而且对于用户的感知和认知能力来说也难以承受[2]，加之硬件设施负载有限，导致用户体验不佳。现有大数据可视化解决方案，一是采用高性能硬件环境提升可视化效率，二是基于预处理中间结果可视化方案。前者往往硬件成本代价不菲，内外存储空间消耗巨大；后者将结构化的数据（每个数据点可映射成数据表的每一项）进行变换处理，映射为另一种相同或相近内容的易表达的可视化元素，从而形象直观地来显示原本复杂的资源内容，然后通过视图变换将这种可视化结构根据位置、比例、大小等做可视化输出[3]。但这些方案对大数据的时空、高维特征支持不够，尤其难以扩展存储并发挥分布式的优势。所以，如何高效渲染、表达时空大数据的可视化内容，兼顾大数据的高维度信息，使人们能直观地感受其折射的信息内容，同时能充分利用到集群分布式处理的长处，是时空大数据可视化研究的一大难点。

因此，针对上述问题，本文拟以点状数据（如出租车点位数据、签到数据等）为可视化对象，提出一种多维度聚合金字塔数据组织模型，兼顾时空和多维度属性以及地图的层级尺度，设计一套面向点状时空大数据的开源可视化框架，以期解决大数据可视化时扩展性不佳、查询性能低、时空/高维支撑不足的难题，达到快速查询可视化的效果。

## 国内外相关研究

### 大数据可视化概述

互联网、物联网和云计算等信息技术的迅速发展，使其正以极快的速度在渗入人类生活的方方面面，每时每刻都在产生着类型各异的数据，人类社会现已跨入大数据时代[1]。2008年《Nature》杂志以大数据专刊的形式，着重讨论了巨量数据对互联网、环境、经济及生物等领域的影响[4]；2011年，《Science》杂志出版的专刊“Dealing with Data”，探讨了如何借助宝贵的数据资产推动人类社会向前发展[5]；2012年，美国启动了“大数据研究和发展计划”，以期在从大数据中获取知识方面能有所突破；2015年中国国务院也印发了《促进大数据发展行动纲要》，该纲要系统部署了未来大数据的发展工作；因此，大数据发展的重要性不言而喻。

大数据时代，不仅意味可以获取大量的数据信息，更应该发现并理解其中蕴含的信息、知识及价值，因此大数据挖掘分析正是大数据研究领域的重中之重[6]。大数据分析是以强大的支撑平台为基础，在其上搭载运行各类处理工具以及分析算法，从不同层面发现隐藏在大数据中潜在价值的过程[7]。从异构数据源抽取和集成的数据构成了数据分析的原始数据，而如何对这些原始数据进行有效阐释和学习正是大数据分析的核心问题[8]。目前，大数据分析技术主要包括可视化分析、数据挖掘、预测分析、语义分析和数据质量管理[7]，其中可视化分析主要发挥人的主观认知作用，从人的角度出发进行需求理解和分析，是一种强调人机协同交互的、融合人机的认知能力于操作过程的分析方法。

人类是视觉的动物，其吸收的信息超过八成是通过视觉系统从外界获取的。当大数据以可视化图形的方式呈现在我们面前时，我们往往能够以极快的速度理解数据隐藏的信息，并将其转化形成为我们可认知的知识。可视化结合人眼的感知功能和人脑智能，对数据进行交互的可视表达[9]，将抽象的、书面文字难以直接显示或不可见的信息映射为可感知的纹理、符号、图形、颜色等，以提高数据识别效率的同时高效地传递有用信息[9]。数据可视化分析是大数据分析中至关重要的部分，它是指在大数据自动分析挖掘方法的同时，借助支持信息可视化的用户UI以及支持分析过程的人机交互方式，融合计算机的计算能力和人的认知能力，从而可控地对于大规模复杂数据集进行深入的洞察与解析[3]。近年来，通过国内外一些学者在不同领域对大数据可视化研究的不懈努力，取得了不少研究成果。Murthy等为获取飓风对人们生活的影响程度，尝试了对社交媒体数据进行可视化分析[12]；Itoh等综合利用社交数据和交通数据，进行了交通分析与社会媒体分析的可视化集成[13]；马晓亭等设计了一个图书馆大数据可视化分析系统，并进一步阐述了大数据可视化分析在图书馆个性化服务领域的意义[14]；何晓琳等对肿瘤流行病学数据进行可视化分析，并结合肿瘤发病数据、科研文献数据与公众搜索数据对肿瘤流行病学数据进行了多维度分析[15]。然而，这类研究并未完全解决大数据本身特点所带来的严峻挑战。未来在大数据框架下，大数据可视化的研究将主要呈现3大趋势：1）可视化对象不再局限于传统的单一数据来源，正向多来源、多尺度、多维度等的广泛数据方向扩展；2）分析不再是少数专家用户的专利，广泛的不特定用户群体正在参与进来；3）可视化和可视分析在大数据与数据科学的框架下进行[16]。

大数据主流应用的可视化技术主要有文本可视化、网络（图）可视化、时空数据可视化和多维数据可视化技术等，本文研究重点主要面向时空数据可视化和多维数据可视化。

时空大数据是数据中包含地理空间维和时间维的数据[17]。时间与空间是描述一个事物的重要特征。如何合理地选择和布局地图上的可视化元素，从而呈现尽可能多的地理时空信息是可视化的关键[34]。时空数据可视化，即是对可视化对象的时间维与空间维以及相关的属性建立可视化表征，并突出展示与时、空密切相关的模式及规律，其重点是解决时空数据的高维性和实时性[3]。典型方法有将时间事件流与地图相结合的流地图[17]，但当数据规模增大时，传统的流地图面临大量的图元交叉、覆盖等问题，解决此问题可采用大规模图可视化中的边捆绑方法[18]；此外，基于密度计算对时间事件流进行融合处理的密度图技术也能有效解决该问题[19]。另一类主要方法是突破二维平面的局限性，以三维方式直接展现时间、空间及事件的时空立方体(space-time cube)[20]，但这种方式一般构建模型复杂。

一般把超过10个维度称为高维，当时空信息对象拥有较多维度的属性时，一般会因维度灾难而出现展示能力不足的问题，因此，通常将多维数据可视化与时空数据可视化相结合。多维数据是指具有多个维度属性的数据变量，探索多维数据项的分布规律和模式，揭示不同维度属性之间的隐含关系，研究维度组合所能反映的信息，是多维数据分析的目标。多维可视化的基本方法主要包括基于像素、基于几何图形、基于图标、基于层次结构、基于图结构以及混合方法等[21]。

### 大数据可视化研究的分类

Godfrey等提出，以可视化查询返回的数据类型划分，大数据交互式可视化类型可以分为：原始数据以及预处理数据[22]。

1. 以原始数据作为返回类型

为了提高查询效率，一般最常用方式是提高计算机性能，需要良好的高性能硬件支持。Google's Big Table/Query (Dremel系统[23]是其商业版本)就属于此类，它是一个基于云平台的大数据分析网络服务，通过融合集群的并行计算、列存储、以及数据压缩等技术，支持billion级记录数据在线查询分析，并实现交互式可视化。BigQuery允许用户上传超大量的数据集，并通过其直接进行交互式分析，无须耗费时间精力建立自己的数据中心。Google表示，BigQuery引擎已具备对70TB以上的未经压缩处理的数据作快速扫描的能力，并且能够立刻得到分析的结果[24]。

MIT研究者Todd Mostak 等开发的大规模并行数据库MapD[25] 利用图形处理单元（GPU）对数据库进行加速处理复杂、实时的时空数据，能以毫秒级来分析处理数十亿行的数据。单节点的情况下，MapD能比集群的CPU数据库快至少2个数量级。2017年MapD项目开放源码，提供了MapD Core开源数据库，其源代码遵循Apache 2许可协议；同时，MapD的可视化库（包括开源的JavaScript库）也开放共享，让用户可以轻松构建基于MapD的WebUI可视化定制应用。该库包含交叉筛选数据、生成图表及管理MapD Core数据库连接等一系列功能，其中图表和交叉筛选库基于流行的Dimensional Charting项目。

这种类型方法可以支持精确的单条数据查询，但缺点是所需硬件资源庞大。且对原始数据进行可视化时，可能会出现覆盖、叠加现象，无法清晰的表达分布信息。

1. 以预处理作为返回类型

最常用的概念即数据立方体Data Cube[26]，现有许多方法都是扩展此概念来建立。数据立方体，是多维数据结构的层级聚合的形式，包含上卷，下钻，切片，切块，旋转等操作[27]，能够提供数据的多维视图，并允许用户预计算从而快速访问汇总数据。然而对于海量的多维数据集，立方体会变得很稀疏，即表示某些属性组合的多个单元是空的，在高维情况下这种冗余尤其明显，进而随着维度增加，内存消耗过大。

为了解决这个问题，许多学者采用各种不同方式减少数据立方体的内存。Dwarf[28,29,30]通过前缀、后缀冗余来压缩数据立方体，减少内存消耗。imMens[2]通过将数据立方体划分为多个子立方体，使得子立方体维度减少，从而减少总的内存消耗，并通过GPU并行处理渲染从而减少查询时间延迟，而其缺点是能够支持的最大维度是四维，且无法自由扩展到更高属性维度。

Nanocubes[31] 扩展了Dwarf思想，提出了一种基于内存的树结构，其添加了不同层级的空间维度，时间维度，以及属性维度，能够支持多维度以及时空的交互式查询。尽管Nanocubes效率很高，且精简了数据立方体，能够在单机上对一定数据量的大数据进行交互式可视化，但由于其是基于内存的紧耦合结构，难以水平扩展到分布式架构。

以上实现都是在单机上，对数据进行聚合从而减少内存，达到交互式可视化的目的。同时，与存在一些基于分布式平台的可视分析平台。例如，VisReduce [32]，是通过分布式数据库来扩展，在线聚合之后再返回查询结果。其扩展了MapReduce算法以及数据压缩方法。因为其是对原始数据在线聚合，随着数据量增大，聚合过程中会消耗更多的时间，使其难以达到交互性可视化。TBVA[33]是通过Spark[50]集群对大数据进行离线预处理，提前生成不同空间层级的切片数据，并计算出各个切片的属性维度的统计值，达到交互式可视化效果。但是，TBVA不支持空间下的时间范围查询，无法分析出数据的时空特征。

在大数据，时空多维度的情景下，已有的单机以及分布式系统中，很难同时满足时空查询以及其他多维度的在线交互式可视化。

## 主要研究内容

本文针对大数据可视分析扩展性较差的问题，以点状的时空大数据为研究对象，设计一套面向此类数据的开源可视化模型及框架。针对点状大数据的时空和高维特性，首先提出多维度聚合金字塔模型能够支持时/空/属性多维的层次聚合；面对海量数据规模，该框架选用分布式NoSQL数据库实现扩展性良好的分布式存储；面对来源各异的原始数据，框架同时提供分布式预处理工具，能够高效地生成符合组织模型的中间预处理结构；最后通过多维联动可视化前端，提供高效、直观的交互式可视化能力[12]。具体的研究内容如下：

1. 多维度聚合金字塔（MAP）数据组织模型

常见的瓦片金字塔模型特点是空间逐层聚合，时空大数据的可视分析需要时/空/属性多维度层级聚合，两者具有一定相似性。因此，需要研究将瓦片金字塔的空间聚合扩展到时空、多维属性的聚合。

本文将研究瓦片金字塔模型空间逐层聚合的思想扩展到时/空/属性多维，并结合Key-Value-Pair的思想，利用传统的层级瓦片来聚合组织时空数据，以属性聚合树的扁平化来组织属性数据，然后通过键值对的形式（以时空单元为Key，属性聚合树为Value）联系二者，使所有维度数据及其聚合结果都能有机的组合在一起，此即本文提出的多维度聚合金字塔模型的主要思想。

1. 基于MAP模型的分布式存储方案

为了提升数据的存储管理扩展能力，本文将选取分布式存储方案。因此，本文进而需要研究如何将按MAP模型组织的中间处理数据存储到分布式存储单元。本文首先选择主要用于存储海量结构化数据的HBase作为数据库，设计相应的存储模型以映射MAP数据模型；同时根据应用场景的需要（主要是时空属的各类查询可视化）设计相应的行键模式和高效的查询方法。

1. 原始海量高维时空数据到MAP模型的快速转化

原始海量高维时空数据到基于MAP模型的分布式存储数据库HBase，中间还需要一个快速转化通道，接着本文将研究这样快速转化通道的实现。从原始数据到符合模型的结构化数据，再导入到HBase中，这一预处理过程本文利用Spark进行操作。由于数据模型以及实际存储模型的关系，Spark应以合适的粒度聚合时空数据，以合适的方式量化不同的属性维度，同时尽量提高预处理速度。

1. 开源的MAP-Vis可视化框架

最后，综合模型、存储方案和预处理为一体，构建开源的分布式的可视化框架（MAP-Vis），这套框架能尽可能地适用于多种数据集。并以此框架为模板，实现一个可视化原型系统，以实验来验证以上模型的可行性，验证该框架对数据扩展性问题的解决能力。

本文的主体技术路线图如图 1‑1所示：



图 1‑1 论文主体技术路线图

## 论文组织结构

本文共分6章，具体的安排如下：

第一章：绪论。本章先阐述了论文的选题背景与研究意义，并介绍了国内外关于大数据可视化的发展境况，然后重点介绍分析了几种主要的大数据可视化解决方案的研究现状及优劣，并对本文的主要研究内容进行了初步的探讨。

第二章：多维度聚合金字塔（MAP）组织模型。本章在介绍瓦片金字塔模型的基础上，引出了MAP金字塔模型的基本概念及主要特点；然后着重阐述本文提出的多维度聚合金字塔模型（MAP）的构成方法，包括其中各组成成分的定义和构建操作的详细叙述。

第三章：基于MAP模型的数据库存储方案。本章首先介绍选用的HBase数据的基本情况和特点以及选型原因；接着阐述映射组织模型的存储模型，包括表结构设计以及具体的行键编码方式；然后是时、空、属性查询在数据库端的具体实现过程。

第四章：海量高维时空数据的预处理。本章首先简单介绍预处理工具Spark的基本概念，然后详细阐述海量高维时空数据的预处理过程。最后设计实施简单的对比实验并分析。

第五章：基于MAP模型的可视化系统实现。本章先介绍基于MAP-Vis可视化框架方案的可视化原型系统的框架结构，以及软硬件环境；然后解释系统具有的各部分功能。在此之上，设计实施了验证模型有效性、预处理与查询扩展性的对比实验并分析。

第六章：总结与展望。本章主要对之前章节的理论模型及实验结果进行整理以及归纳总结，然后回顾全文，指出存在的不足之处以及今后研究中需要改进的地方。

# 多维度聚合金字塔（MAP）组织模型

## 瓦片金字塔模型概述

瓦片地图金字塔模型是一种多分辨率的层次模型，从底层到顶层，金字塔的分辨率越来越低，但表示的地理范围不变。金字塔包含*L*个图层（Layer/Level），每个图层由*M*×*N*个正方形瓦片（Tile）拼接而成，每个瓦片再由2n×2n像素（Pixel）组成（一般默认为256×256），而像素值就是可视化的内容[35]。

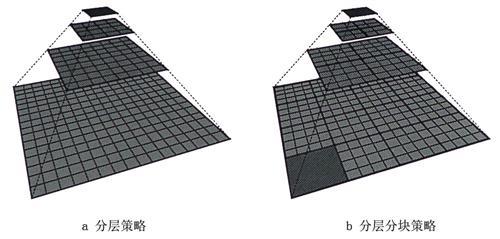


图 2‑1 瓦片金字塔模型

瓦片金字塔模型将由地理空间对应的地图数据或图片，以指定的尺寸大小和格式切割成一块块地图瓦片，通过分层、分块的策略构建多尺度缓存地图库（如图 2‑1所示），提前渲染、切片，存储在地图服务端。这种方法在用户访问时，可直接调取缓存的地图瓦片，避免服务器端实时渲染呈现结果，有效地缩短服务器的地图生成时间和传送时间，缓解服务器压力，减少网络负载和响应时延。

瓦片金字塔模型组织数据的方法，即是采取预处理数据的方案，根据分辨率不同的需求，人为划分若干等级，然后逐级聚合处理好的地图图片。比如分L层的话，以第L层为基础，从地图的原点开始，从左往右，从上往下按照2×2个像素合成一个像素的原则生成新的像素矩阵，然后分块，形成第L-1层，再以此类推，往上不断聚合直到第0层。抽象后的数学模型如下：

1. 像素（Pixel）

像素是瓦片金字塔模型的最小数据单位，金字塔层级中瓦片中的单个像素，可如下表示：

|  |  |  |
| --- | --- | --- |
|  | ***Pixel* =**{*l, x, y, p*} | ( 2‑1) |

其中，*l*为金字塔的层级，*x、y*为该像素对应的空间范围的中心或某一角点坐标，*p*代表在该像素对应的空间范围内的特征采样值，或者下一层级的聚合统计值，如均值，总和等。

1. 瓦片

瓦片金字塔模型中的瓦片，是空间邻近（*n*×*n*邻域）的像素组合结果，默认分辨率为256\*256。可如下表示：

|  |  |  |
| --- | --- | --- |
|  | ***Tile*** = {*l, X, Y,* } | ( 2‑2) |

其中，*l*代表金字塔的层级，*X、Y*为切片的行列号，表示该切片空间范围内的像素集合。

1. 金字塔（Pyramid）

金字塔即各个层级切片共同组合成一个空间多尺度的模型，可如下表示：

|  |  |  |
| --- | --- | --- |
|  | ***Pyramid*** = {} | ( 2‑3) |



图 2‑2 瓦片金字塔模型概念映射图

从上面的数学抽象可以看出，瓦片金字塔（如图 2‑2所示）本质上就是一种2D的空间层级聚合。时空大数据兼具时间和空间，同时包含多维属性，其可视化分析可以借鉴瓦片金字塔模型的思想，将瓦片金字塔的二维空间聚合扩展到时/空/属性的多维聚合。本文提出的多维度聚合金字塔模型，即是尝试在时空数据组织的过程中，以瓦片金字塔模型为基础，结合键值对的方法，融合进时间、多维属性的聚合。时间、空间以时空立方体的DataCube操作进行聚合，多维属性以扁平化的属性聚合树方式进行聚合。

## MAP数据组织模型

多维度聚合金字塔（Multi-dimension Aggregation Pyramid， MAP）模型是在传统的瓦片金字塔模型的基础上，提出的时空多维层级聚合模型。在空间维，其以瓦片金字塔的方式进行逐级聚合；在时间维，其通过预先设定的时间粒度对数据进行划分重组；在属性维，其通过扁平化的属性聚合树进行各维度属性值的聚合。其中，时间、空间维是密切相关的，二者构成的整体组合为一个个时空单元，以时空单元作Key，离散的属性维聚合树为Value，两者以键值对（Key-Value-Pair）的方式进行挂接。在时空聚合的同时，属性维也随之聚合，如此可得到一个包含空间、时间、属性维度的多维时空金字塔模型。



图 2‑3 MAP模型概念图

MAP模型如图 2‑3所示。本文构建的数据模型由于以瓦片金字塔为基础，根据可视化分辨率需求可设置多个层级，逐级聚合而每一层又相互独立，耦合性低；利用扁平化的属性聚合树，使得在增加属性维度时，树的深度不会骤增；加之使用键值对的方式，十分利于分布式的扩展。

### MAP模型的基础定义

构建于瓦片金字塔的MAP模型在一定程度上还是一种金字塔结构，但是在此基础上做了延伸和扩展。传统的瓦片金字塔是2D的结构，而MAP模型融合了时间和属性维，所以原有的Pixel概念拓展为一个多维时空的Cube，每一层不再是一张张简单的瓦片，而是一个个时空单元Cube集合（或高度不为0的时空切片），且每个单元Cube还挂接一棵满载属性的树。对于MAP模型，抽象的概念定义如下：

1. 时空像素（Spatio-Temporal Pixel）



图 2‑4 时空像素示意图

有别于地图瓦片的像素，时空像素是一个高维立方体，是容纳（或者挂载）属性维信息的容器，也是MAP模型的最小数据单位。其延伸了瓦片金字塔像素概念的内涵，除了具有空间范围，还添加了时间维，如图 2‑4所示。在模型中对时空像素定义为如下的五元组：

|  |  |  |
| --- | --- | --- |
|  | ***ST-Pixel =*** {[*l*, *x, y*]*, t, att\_tree*} | ( 2‑4) |

其中，*l*指代金字塔的层级，*x、y*为该像素对应空间范围的中心或某一角点坐标，*t*代表预设的时间粒度范围，*att\_tree*代表属性聚合树，是可视化内容的真正来源。

1. 属性聚合树（Att\_tree）

本模型定义的属性聚合树，是对时空对象的属性各维度单元进行逐维度聚合，从而所形成的树形结构，以广度优先遍历的结果简化实际的存储空间。定义如下：

|  |  |  |
| --- | --- | --- |
|  | ***att\_tree*** = {aall, a1, a2, …, an} | ( 2‑5) |

举例说明，如图 2‑5所示有4条出租车的记录，每条记录有三个属性，分别为载客与否、车辆颜色以及司机性别。图中的4条记录，通过统计聚合得到右侧的属性聚合树结构，再对其做序列化（稀疏存储），可以得到更容易存储的定长一维数组结构。相较于树形结构，这为后续的储存以及处理提供了极大的便利。

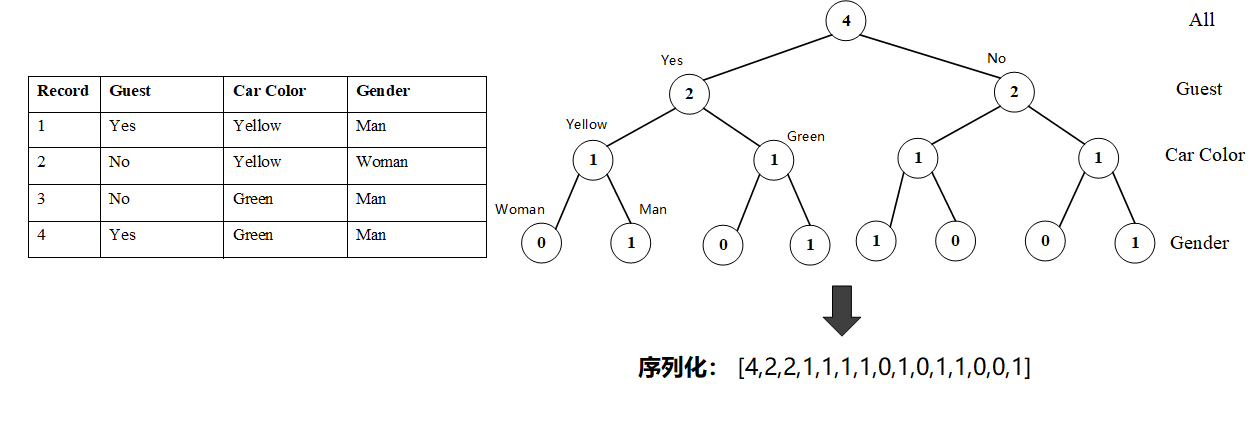


图 2‑5 属性聚合树示意图

1. 时空切片（Spatio-Temporal Tile）

时空切片的概念类比于地图中的瓦片，由时空像素组合而成，而每一片瓦片都带有时间维度，称为时空切片；每一时空切片上也挂载了属性维，累积了时空像素属性维的结果，所形成的整体扩展了2D瓦片的概念，成为一个可实现层级聚合的时空立方体。时空切片是MAP模型的基础显示单位，对它的定义为如下的五元组：

|  |  |  |
| --- | --- | --- |
|  | ***ST-Tile =*** {[*l, X, Y*]*, T, Att\_tree*} | ( 2‑6) |

其中，*l*同样指代金字塔的层级，*X、Y*指代切片的行列号，T代表预设了时间粒度的时间单位，Att\_tree是切片层面的属性聚合树，同样有：

|  |  |  |
| --- | --- | --- |
|  | Att\_tree = {Aall, A1, A2, …, An} | ( 2‑7) |

从公式( 2‑6)以及图 2‑3可以看出，定义的时空切片（或时空立方体）尽管从概念上已经不再是二维的平面了，但由于前端地图可视化是逐切片显示最后的聚合结果，其在概念上与地图瓦片依然有相近的含义。

### MAP模型的构建操作

定义了MAP模型的三个基本概念之后，MAP模型实现还需定义一些对应的动态操作，由这些操作共同完成对多维度聚合金字塔的构建。

1. **extract**操作

该操作负责将每一条原始记录提取到上一节( 2‑4)定义的时空像素单位上，通常一个时空像素上包含不止一条原始记录，其挂载的属性聚合树*att\_tree*已经是统计后的结果。操作公式如下：

|  |  |  |
| --- | --- | --- |
|  | **extract**(*l,* [*Lon, Lat*]*, w*) = ***ST-Pixel*** ([*l*, *x, y*]*, t, att\_tree*) | ( 2‑8) |
|  | **extract** = | ( 2‑9) |

其中需要说明的是，*l*指代层级，[*Lon, Lat*]代表原始记录数据（包括时、空、属性等维度），*w*为定义的切片宽度。

1. **group**操作

该操作分为2步，首先主要是建立时空像素与时空切片（立方体）之间对应的映射关系，即求得像素应隶属的具体切片，而且不止在空间上定位其属于某一切片，在时间上该像素的归属也是确定的。操作公式如下：

|  |  |  |
| --- | --- | --- |
|  | **map**(*x, y*) = | ( 2‑10) |

其中，下标*T*表示每一个对应关系都是有时间限定的，最后得出的结果是找出了每一个像素所属的时空单元。

公式( 2‑10)的作用是将时空像素映射到具体的时空切片上，但这个时候的时空切片还只是一个拟存在的概念，所有时空像素尚未汇聚形成真正的时空切片。group操作的第二步即是对所有映射相同时空切片的时空像素进行归约，操作公式如下：

|  |  |  |
| --- | --- | --- |
|  | **group**(*XT, YT*) = ([*X, Y*]T, count{***ST-Pixel*1**, …, ***ST-Pixel*n**})  = ***ST-Tile*** ([*l, X, Y*]*, T, Att\_tree*) | ( 2‑11) |
|  | ***ST-Tile****.Att\_tree* =***ST-Pixel****.att\_tree* | ( 2‑12) |

从公式( 2‑11)可看出，group操作是以时空切片作为Key来reduce所有像素的过程；其中，时空切片***ST-Tile***的聚合树*Att\_tree*是所有时空像素***ST-Pixel***的*att\_tree*累加的结果，由于在上一节的属性聚合树部分提到，*att\_tree*被处理成一维的定长数组，因此这在计算上也十分简单。

1. **aggregate**操作

以上的2个操作构成了金字塔最底一层的数据时空单元，接下来的工作就是如瓦片金字塔一样逐级往上聚合，生成各级时空切片。aggregate操作即是完成这一过程，操作公式如下：

|  |  |  |
| --- | --- | --- |
|  | **aggregate** (***ST-Tile*** ([*lN*-1*, X, Y*]*, T, Att\_tree*))  =  = ***ST-Tile*** ([*l*N-1*, X, Y*]*, T, Att\_tree*) | ( 2‑13) |

其中，**Ψ**代表一个可选的算子，可选择属性值以何种计算方法（求和、平均、最大最小值等）向上聚合。

## 本章小结

本章在对传统的瓦片金字塔模型做了简要介绍并对其抽象化2D数学模型的基础上，将瓦片金字塔扩展到nD，提出了自己的一套多层级聚合模型—MAP模型来组织点类型大数据，其中对模型的几个基本概念给出了自己的定义，并提供了MAP模型构建的操作方法。

# 基于MAP模型的数据库存储方案

## 分布式数据库HBase概述

创建于2007年2月的HBase项目，是Google BigTable的开源实现，在Hadoop生态里是很重要的一部分[36-39]。其主要用于存储海量的结构化数据，底层即以HDFS作为存储系统[38,39]，是一款分布式、多版本、面向列存储的非关系型数据库，具有可伸缩、随机实时读写、搞高可靠性的特点。在普通硬件环境的基础上，能支撑十亿量级的行和百万量级列的大表[36,37]，适合应用在数据存储量大（PB级），写入性能要求高，数据结构多源，业务简单的场合。本文使用的是相对稳定的HBase-1.2.0版本[39]。

### HBase简介

1. HBase系统架构

HBase集群采用Master/Slave的架构，由Zookeeper服务器、主服务器Master、Region服务器（RegionServer）以及客户端（Client）等将部分组成[38,39,41-48]，如图 3‑1所示。

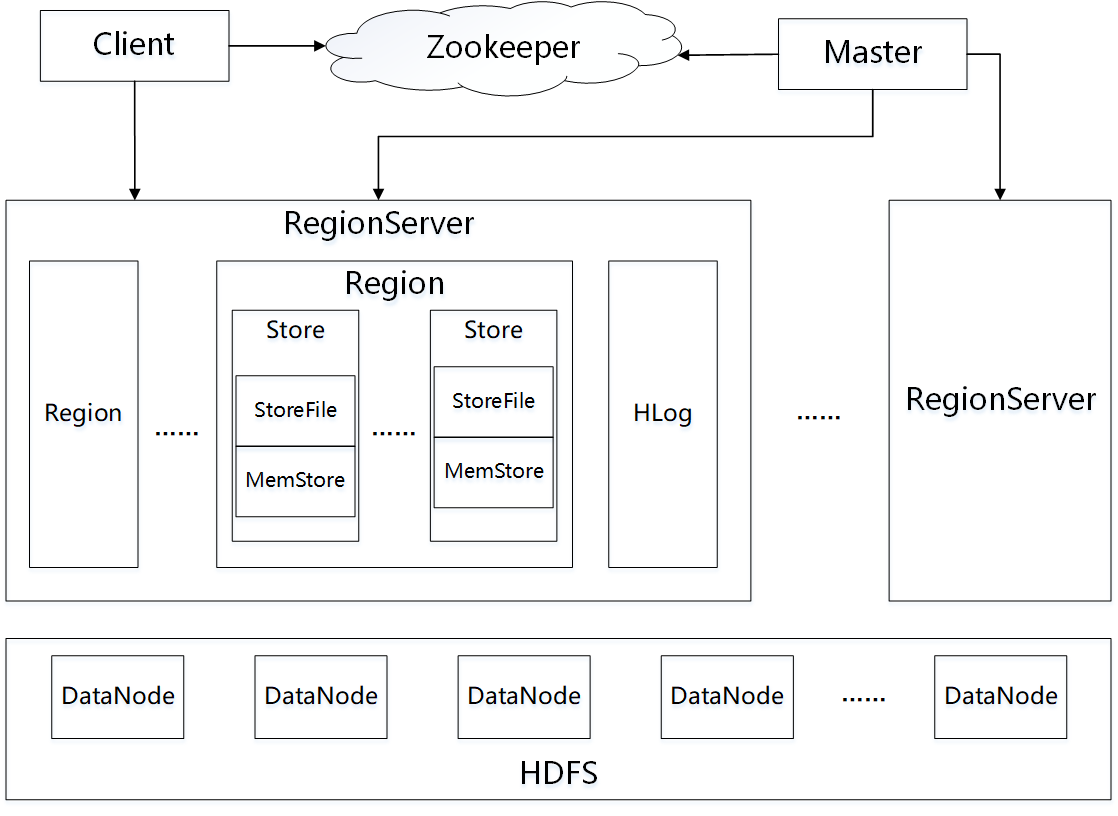


图 3‑1 HBase系统架构图

Zookeeper为整个集群提供协调服务，维护集群Master的失效重启且保证其永远只有一个；并实时对各RegionServer节点的状态作监控，同时将其具体的状态信息向Master报备；Zookeeper还保存着-ROOT-表和.META.表的地址（每次查询发起时寻找对应Region的入口），以及每个HBase表的元数据信息，如模式设计等。

Master是HBase集群的管理节点，一方面负责在RegionServer宕机时将原来的Region重新分配至其他节点，以调整负载均衡；同时管理着用户的CRUD（建表、读取、更新、删除）操作。

RegionServer作为集群的核心部件，上面分布有Master分配的若干Region和一个HLog日志文件，由节点单独维护；同时处理来自Client的读写请求，在HDFS上进行相应的IO操作；此外，负责切分因为数据行增多或数据量增加而超过阈值的Region。

Client为用户的数据读写请求提供入口，通过远程调用协议（RPC）与Master进行通信，获取Region的节点信息，然后在RegionServer上做读写类的操作[43]。

当用户发起数据读取请求时，Client首先访问Zookeeper获得-ROOT-的位置，接着访问-ROOT-表获取.META.的位置信息，然后访问.META.表得知用户请求的表所对应的Region位置信息，最后Client直接访问对应的Region获取用户数据。这样一个三层的查询结构，尽管需要多次网络操作，但 Client包含了cache缓存，可以提高查询效率；且当得知数据所在的RegionServer后直接访问相应节点的方式，跳过主节点，使得Master的负载较低，也不会成为集群查询的瓶颈。如此的架构设计，使得HBase有着相当出色的负载均衡和容错的性能。

1. HBase数据与存储模型[39]

在数据模型上，HBase采取与Google BigTable相同的数据模型，以表的形式存储数据，表具体的逻辑视图如表 3‑1所示。HBase底层存储采用Key-Value形式存储，表拥有一个大的映射关系，其中行键（RowKey）、列族（Column Family）、列限定符（Column Qualifier）和时间戳（TimeStamp）共同组成了Key，而Value由这4维坐标唯一确定。行键是每行数据记录的唯一标识，类似于关系型数据库的主键，按照字典序（byte order）排列；列族是列的集合（列族的设置在构建数据的语义边界或局部边界有一定的帮助），列都以列族为前缀（Column Family:Qualifier），使用表前需要事先定义好列族，但二者均支持动态扩展，以及独立检索。对数据的每一次修改操作均有一个64位整型的时间戳与之对应，即每一行数据对象可有多个时间版本，以Timestamp索引，在写入RegionServer节点时系统自动赋值，并按照时间顺序降序组织，每次数据访问时优先取最新的版本（时间戳的版本管理方法有2种：要么限制数量，只保存最新的n个版本，要么限制时间，只保留最近时间段内的所有版本）。所有的单元格（Cell）数据均以字节码的形式存储，没有类型之分。在表 3‑1中，对于类似如下一些空白的单元值，实际的物理存储上没有为其开辟空间，如果请求了空白值，会返回NULL，因此，HBase表也可以设计得非常稀疏。

表 3‑1 HBase逻辑表结构

|  |  |  |  |
| --- | --- | --- | --- |
| **Rowkey** | **Timestamp** | **Column Family** | |
| Qualifier1 | Qualifier2 |
| Key1 | T3 | Value11 | Value21 |
| T2 |  | Value22 |
| T1 | Value12 | Value23 |
| Key2 | T4 | Value13 |  |

在存储模型上，HBase的存储、权限控制和检索都基于列族。每张表由若干个Region组成（一个Region包含了一定行键范围内的数据），每个Region当数据行数达到设定阈值后，一个Region会如细胞分裂般拆分为2个，而这些Region都分散地存储在不同的RegionServer节点上，但一个Region内部的数据只会存储在同一节点。Region是HBase中分布式负载均衡的原子单位，但不是存储的原子单元，一个Region会根据该表列族的个数产生对应的Store，每个Store负责管理一个列族，每个Store都有1个MemStore，以及若干StoreFile，这取决于改该列族上的数据量，甚至可以没有StoreFile文件。如图 3‑2所示。



图 3‑2 HBase存储模型

MemStore位于RegionServer服务器的主内存中，当写数据时，会首先写入到MemStore中，并在MemStore中进行排序，当数据累计达到某一阈值后，创建新的MemStore，并将原来的MemStore添加到刷新队列里，等待MemStore中的内容Flush到硬盘里，形成一个StoreFile文件。同时，为记录此前的数据变更已持久化到硬盘，还需在Zookeeper里作备份。这样设计的原因是：HBase的数据实际是存储在HDFS上的，而为了检索优化的考虑，必须满足HBase的行键按照排序存储的需求，但HDFS上的数据又不可随意修改，换言之，即必须排好序再作持久化，这就产生了排序与写数据的矛盾。为克服这个困难，HBase将数据的写入过程做了一个中转，先把新数据缓存到MemStore，利用内存的高速读写能力做好排序工作，然后再持久化到HDFS。

StoreFile是只读文件，一旦创建无法修改，所以HBase的更新操作没有删除改写，而是一个将新数据以一个个StoreFile文件不断append至数据库中的过程。当StoreFile文件数量超过一定阈值后，会自发地触发一次合并（compact）操作，对相同RowKey的数据项作合并，生成一个更大的StoreFile。当然StoreFile也不会无限扩大，如果StoreFile的文件大小达到阈值时，又会平均地分裂（split）该StoreFile文件为两个新的StoreFile。因为排序工作已经在内存中完成了，StoreFile中的数据都是有序的，而且StoreFile包含了内存索引，所以合并效率特别高。HBase即以这样的（compact-split）机制，控制着StoreFile数量与大小之间的平衡。

表 3‑2 HBase数据模型与物理存储文件映射表

|  |  |
| --- | --- |
| **数据模型** | **物理存储模型** |
| 行键 | Region |
| 列族 | Store |
| Cell单元值 | StoreFile/HFile |
| --- | MemStore |
| --- | HLog |

除了StoreFile和MemStore，数据还以HLog的格式存储在HDFS上（如图 3‑1所示），以防在系统出错或宕机时，如果某个节点掉线了，其MemStore的内存数据就不会丢失。HLog的原理与MemStore相似，让每个RegionServer都有一个日志对象，当数据写入时，同时在MemStore和HLog中刷写一份相同的数据，HLog定期作数据持久化，然后覆盖已存在的数据。当RegionServer节点掉线了，Master主机会向Zookeeper发消息，Master会先处理HLog，将不同Region的HLog数据进行划分，放到对应的Region目录下，然后重新分配失效的Region。分配到Region的节点在加载Region的过程中，如果有HLog，就会读取其数据写入MemStore中，然后刷新到StoreFile，完成数据恢复。HBase的数据模型与实际物理存储模型之间的对应关系大致如表 3‑2所示。

### HBase协处理器

HBase与传统的关系型数据库相比，其写入性能高了一个数量级，但其查询性能要低一个数量级。通常访问HBase数据的方式是，使用Scan方法做全表扫描或者Get方法直接获取，根据需要可以使用Filter过滤掉冗余的部分数据，最后在获取到的数据上进行业务运算。但如果查询结果非常庞大，对如此大体量的数据做传输势必造成极大的网络延迟，而且在客户端还要进行复杂的运算，对服务器的内存和CPU也是巨大的挑战，这样的过程十分浪费时间。除此之外，0.92版本之前的HBase还存在一个问题，即很难进行简单的排序、求和、计数等操作。因此为了应对这样的困境，协处理器机制应运而生。

HBase的协处理器（Coprocessor）发端于Jeff Dean的一篇关于Google BigTable的Coprocessor特性的演讲[39,44]，其主要思想即是把复杂的处理环节，分发到各个RegionServer节点上，使得大部分计算可以在数据库端处理完成，从而提高了处理的效率；加之服务器只返回处理后的结果给客户端，客户端只需进行较小的部分计算，这既减小网络传输的损耗和耗时，也缓解了客户端的压力，在性能方面也是很大的提升。

相比于Google BigTable的协处理器，同样将计算过程转移到存储节点，HBase Coprocessor最大的区别在于它是一套在Master和RegionServer进程内运行的框架，可在运行期间动态加载，然后执行相应的功能。HBase Coprocessor的运行原理非常类似于MapReduce的分析处理组件，但极大简化了MapReduce的处理模型。MapReduce的加载在初始化阶段耗时长，且每次运行又要重新加载，难以满足线上请求的即时性；而Coprocessor偏轻量级，能动态加载，加载速度快。

HBase Coprocessor有2种不同的类型，分别为Observer（观察者）和Endpoint（终端）。Observer，是在某一特定事件（如Get或Put数据）发生前或发生后触发，执行相应的回调函数。其位于客户端和HBase数据库之间，在二者发生请求调用和响应返回的过程中，修改数据访问，适用于统计行数或构建辅助索引等操作，类似于设计模式中的Observer模式和关系型数据库中的触发器。以RegionObserver为例，调用过程如图 3‑3所示：



图 3‑3 Observer调用原理图

Endpoint把用户自定义的功能代码加载到服务器端，当执行时可以通过客户端的RPC调用触发。通过扩展了Hbase的RPC协议，将数据库端代码的执行结果返回给客户端作进一步处理，非常适合利用Endpoint来实现求和或求均值等聚合计算，类似于关系型数据库的存储过程。以Endpoint统计行数为例，调用过程如图 3‑4所示。

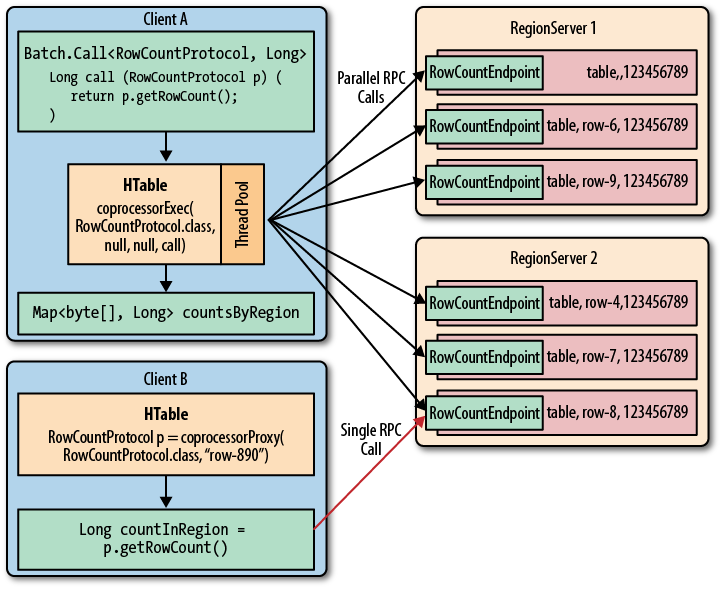


图 3‑4 Endpoint调用原理图

HBase的Coprocessor机制把任意计算逻辑推到托管数据的HBase节点（RegionServer）上，且执行代码是跨所有RegionServer并行运行的。这个特性把HBase集群从水平扩展存储系统，转变为高效的、分布式的数据存储和数据处理系统。基于Coprocessor的功能扩展开发是HBase开源社区的一大热点，本框架即是借助于Coprocessor将大量的聚合任务放在了服务端，在一定程度上以提高查询可视化的效率，下文3.3部分将详细进行阐述。

## MAP-Vis框架数据存储模型设计

### HBase表结构设计与模型映射

根据第二章2.2节的MAP模型描述，结合HBase的数据存储模型特征，本框架对存入HBase的数据设计了如下的表结构，如表 3‑3所示：

表 3‑3 MAP-Vis框架表结构示意图

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Rowkey** | **Heatmap** | | | | | | **Sum** | | | | | |
| 20150101 | … | 201501 | … | 2015 | … | 20150101 | … | 201501 | … | 2015 | … |
| Tile1+cate1 | a |  | b |  | c |  | A |  | B |  | C |  | |
| Tile1+cate2 |  |  |  |  |  |  |  |  |  |  |  |  | |
| … |  |  |  |  |  |  |  |  |  |  |  |  | |
| Tile2+cate1 |  |  |  |  |  |  |  |  |  |  |  |  | |
| … |  |  |  |  |  |  |  |  |  |  |  |  | |

其中，横向上我们以“切片号+属性节点”的编码方式作为每一行的Rowkey，纵向上有2个列族，分别为Heatmap列族和Sum列族。Heatmap列族负责存储连续的热图数据，即每个cell存储每张切片256\*256个像素值的数组，亦可以理解为可视化显示的每一张图片；Heatmap列族下按照时间设置列（Qualified），每一个列为最小的时间粒度（天），并且额外预处理每一个月、每一年的聚合结果，也即有a1 + … + an = b，b1 + … + b12 = c。

Sum列族负责存储离散的属性聚合值，其底下的列名与Heatmap的列名一致。Sum列族下的每个cell都是对应Heatmap列族的同名列的像素聚合值，也就是说，a的所有像素值累加等于A，B、C同理。

本框架的表结构设计，没有把时间维度加入Rowkey，而是把既定的属性维度加入到Rowkey中，让时间做横向的扩展，属性做纵向的扩展，每一个切片对应固定的若干行键（或一个Rowkey组），然后根据时间范围在横向上做聚合，如此设计的好处如下：

1. Rowkey的数目大大减少

如果Rowkey包含时间维度的话，随着数据的增加，Rowkey的数目递增会非常快。比方说，一层有1000个瓦片，一年的数据就是365\*1000个Rowkey；而本框架的表结构设计，每层只要1000个key就可以了，且随着新的数据加入进来，新方案可通过水平增加列族的方式扩展（HBase能支持几十万的列族），1000个key（在不增加属性维度的情况下）数目不会变化，但Rowkey包含时间维度的方式则会变成n\*365\*1000个。即便加入属性维度，Rowkey的数目也不会急剧增加，因为一般来说，可视化能利用的不过十几个属性维度。

1. 聚合时叠加次数大大减少

如果仅以天的粒度组织数据，聚合操作需要一天天地叠加，当查询的时间范围是1年时，需要叠加365次；现在由于在时间跨度上，本框架提前预处理出月的聚合、年的聚合，当查询的时间范围跨月时，直接读取月聚合的value值，1年只需叠加12次，这无疑大大降低了计算工作。而且，时间跨度拉的更大时，这种设计理论上也能够保证查询的响应效率不会呈线性递增，而是保持比较平稳的响应时间。

1. 表结构设计基本可以映射MAP模型

横向上，以行键上编排对应空间的切片，纵向上，以列族和列划分时间，这样表的每一个cell就都可以对应为一个时空单元，而且是对应到某个属性的时空单元，而像素的概念被包含在每一个cell里。具体的映射如表 3‑4所示：

表 3‑4 表结构与MAP模型的具体映射

|  |  |
| --- | --- |
| **MAP模型** | **HBase表存储** |
| 金字塔 | 表 |
| 层级 | 行键Rowkey |
| 时空切片/立方体 | 每一行数据 |
| 时空像素 | Cell或者包含于Cell内的更小单元 |
| 属性聚合树 | 若干行键（Rowkey组） |

不得不说，实际的存储模式并不能严格地与MAP模型的抽象概念一一对应，在具体的表结构设计时还是做了些许改动，但总体上遵循了MAP模型的层级聚合思想，并且从实际操作上的易行性考虑，添加了年、月的聚合，为查询响应提高提供了保障。

### HBase表主键编码设计

在HBase表中，行键的设计很大程度上决定了该表的数据处理性能[44]，这主要是因为，HBase在主键上设计了类B+树的索引，支持基于主键的数据快速查询；再者，各个Region基于行键提供一个区间服务（每个Region有起始行键startkey和终止行键endkey），即能够依据行键快速定位数据所在的Region位置故而，设计一个好的有效的行键至为重要。

HBase的行键设计一般遵循长度控制和唯一性2个原则[44]。文献[50]建议行键长度设计在10至100个字节之间，原因是：如果行键太长，内存的有效利用率会很低，系统无法缓存更多的数据，检索效率也不高；同理，过长的行键也必将导致底层文件系统的存储效率大打折扣；而且，目前的操作系统以64位为主，内存以8字节对齐，将行键设计成8字节的整数倍，可以充分发挥操作系统字节对齐的最佳存储特性。而为了保持行键的唯一性，即能以行键唯一标识一行数据，行键长度也不宜过短。

行键的设计还需满足2个均匀，即数量分布均匀以及读写分布均匀。数量分布均匀要求数据尽可能均衡地分布在各个Region上；读写分布均匀要求用户的读写请求能均衡地摊到每个Region上，因为HBase的Master主节点只管理集群Region的分配，具体的读写操作都有各个Region独立负责，读写分布均匀就是要避免过多的IO被分到一个Region这种情况。

以纽约出租车数据集为例，如图 3‑5所示，行键前缀的数字表示层级。0层到15层这一部分数据，几乎都分布于cu29节点的一个Region上，而16到17两层数据则分布在另外3个节点，这样的行键设计就会导致一种结果，即当在0-14层进行查询时，（由于本框架在空间查询上是异步查询的，3.3节将详细阐述）所有切片的查询请求最终都会发往cu29节点，使得查询造成阻塞，也就是导致了所谓的热点问题，数据分布不均匀。所以这是一种很糟糕的行键设计方法，应该避免。

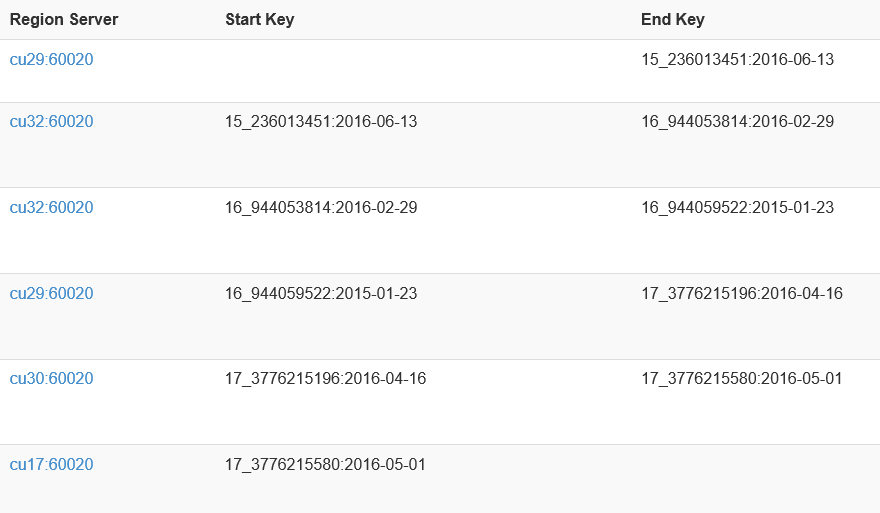


图 3‑5 HBase数据Region分布示意图

本文针对HBase数据库的特点以及MAP模型的要求，提出了适合本可视化框架的表行键设计模式。

根据上一节的表结构设计，行键以“切片号+属性”的方式进行编码，并且行键里不包含时间维度，下面将对具体的编码方式进一步作描述。首先是将每一层级的每张切片计算得出其所在层级的四叉树编码，然后每一个属性节点对应做一个类别代号，比方说20个类别，以0-19分别代表。接着用MD5对四叉树编码和层级作加密，起到哈希的作用，防止数据瓦片在存放时过于集中，最后将加密结果与类别代号直接拼接组成一个字符串，中间以“：”隔开，即“MD5(quadCode : level ): cateNode”，作为HBase表的行键，其具体的设计生成过程如图 3‑6所示：

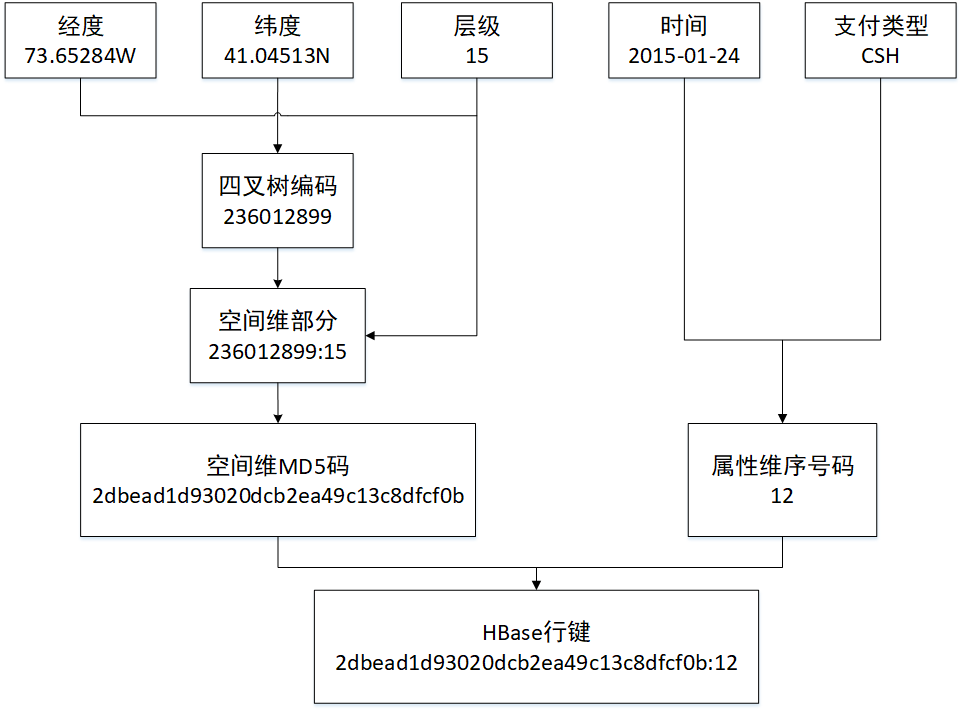


图 3‑6 HBase行键编码设计过程

这样的行键设计具有以下几个特点：

1. 前半部分以“四叉树编码+层级”的方式，保证每一层的每张切片的编码唯一性；四叉树编码在前，层级在后，然后使用MD5对其进行编码，在很大程度上能确保同一层的数据不会被分到一个Region或一个节点上，实现读写的均匀分布。
2. 适应本框架的数据查询。因为本框架要支持时、空、属的查询，空间查询时与属性的关系不大，而时、属查询对属性的要求较高，这样的编码设计方式能够保证不同切片间行键的“低耦合性”，适合支持空间查询，而在单个切片的行键里，行键的“高内聚性”（因为属性节点的编码特点，使得检索数据时能通过Scan扫描一次性获取），适合支持时间、属性查询。（具体的三类查询过程在3.3节会具体阐述）

## 时/空/属查询方法

由于本框架的存储模型设计，每一张切片的不同分类、不同时间数据分布于不同行、不同列，而最终的可视化效果是一个聚合的结果，为了减少查询获取的数据太多导致网络延迟，本框架利用HBase的协处理器（Coprocessor）机制，在数据库端即做聚合的操作，只把聚合计算后的结果返回服务端。

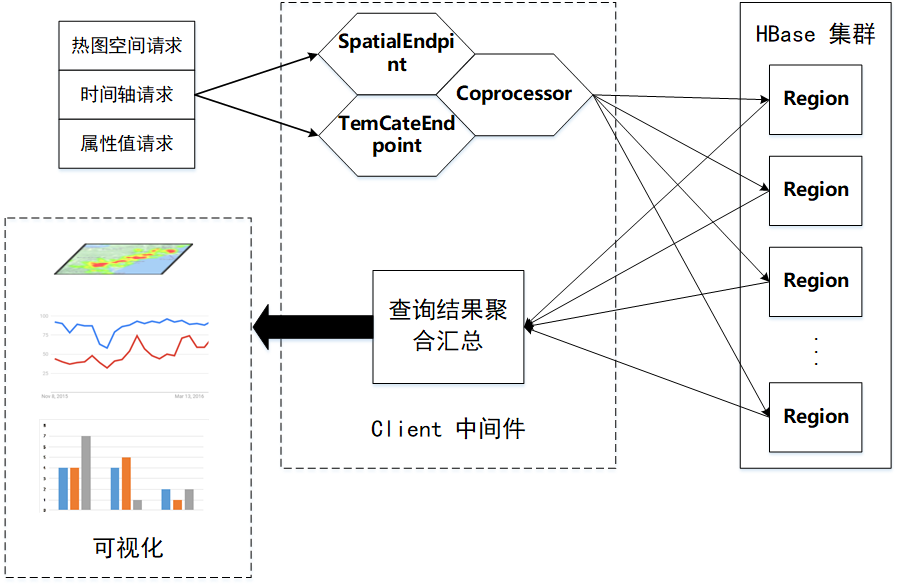


图 3‑7 MAP-Vis查询示意图

这部分“查询-获取-聚合-返回”的过程，主要由2部分构成：HBase协处理器和client中间件。协处理器有2个，都是使用的Endpoint终端类型，一个是空间查询聚合的SpatialEndpoint，一个是时间属性查询聚合的TemCateEndpoint（由于时间、属性获取的都是单值的数据，故将二者编码到一个coprocessor中）。Client中间件的作用是解析服务端发来的请求，然后调用HBase表上对应的coprocessor，再将各Region聚合计算的结果做汇总，打包成json格式返回给服务端。以下分三类查询进行详细阐述，大致过程如图 3‑7所示。

### 空间查询

本框架所谓的空间查询，指的是整个地图图幅范围内，在一定时间区间里所查询显示的数据内容。

空间查询采用的是异步请求，即每一张切片单独向服务端发送查询请求。当服务端接收到查询请求操作时，调用client中间件的空间查询函数；空间查询函数根据切片对应的RowKey以及查询时间范围的上下限，直接调用对应表上相应Region的协处理器做聚合计算。

在client中间件上，Endpoint协处理器的调用机制分2种：一是指定触发特定的 Region 上的Endpoint，在单个 Region 上进行处理并返回一定结果；二是指定触发一定范围内的若干 Region 上的Endpoint（包括所有Region）并发执行，然后在client中间件上对结果进行汇总处理。对于第一种情况，尽管Region的名字可以随时改变，但其起止行键的范围是不变的，Endpoint即利用Rowkey判断对应的Region。即使查询的Rowkey不存在于数据库中，但若其包含于某Rowkey区间内，亦可定位到对应的Region上。

本框架采用前者，即根据RowKey指定 Region，只调用对应Region上的协处理器。这样的好处有2个：一是我们的行键设计就是一个切片对应一行，这样数据一定在某个Region上，其他Region查找不会有结果，能够极大减少其他Region上扫描数据的资源；另一个是只调用切片自身数据所在的Region，将其他Region让给别的切片请求，而且我们设计的RowKey已经将数据尽可能分布在每个Region，可以大大减少其他切片请求等待的时间。

发到协处理器上的数据请求，依据时间范围，计算得出该段时间内对应的时间节点，在横向上聚合这些时间节点的值（即每一时刻的数据，或者说一张图片）。由于我们在天的粒度上有提前聚合了月和年的数据，所以针对时间跨度超过一个月或一年的请求，可以直接聚合月或年的数据，提高查询响应时间。

最后聚合结果返回至client中间件，由中间件提交给服务端。

### 时间查询

本框架所谓的时间查询，指的是在查询的时间范围内，所有的时间节点（如每一天）对应的整个地图图幅内的数据累加值，所形成的时间曲线。

时间查询由于是要计算整个地图图幅的总值，所以没有运用异步请求，而是将当前地图图幅的范围以及时间区间发往服务端，在后台解析成每个切片对应的RowKey，组成RowKey序列与时间一同传给client中间件，调用时间查询函数。

时间函数将RowKey序列拆分，同样采用调用单个Region的协处理器的方法。协处理器根据时间范围，利用HBase的过滤器在sum列族上扫描获取对应时间段内每一个时间节点的数据，然后返回给client中间件。每一个RowKey都循环这么操作，最后在client中间件统计整个图幅各个时间节点的总和值，提交至服务端，进行时间轴曲线的绘制。

### 属性查询

本框架所谓的属性查询，指的是空间上以当前所处的地图范围为限，时间上以查询的时间范围为限，在这个时空区间内，每个类别属性的各自数据总和值，以柱状图方式显示。

从一定程度上讲，属性查询与时间查询很类似，都在于对表的sum列族做操作，所以属性查询的协处理器在时间的协处理器基础上做加工。以纽约出租车的数据为例，属性查询的类别共有2个大类，21个小类，大类分别是：day\_of\_week表示一周里的每一天（Mon.、Tue.、Web.、Thur.、Fri.、Sat.、Sun.），以及payment表示乘客的支付方式（CreditCard、Cash、Other）。

属性查询的接收参数与时间查询相同，在扫描出相应时间范围的数据后，统计0-20个类别每一行的总和，然后返回给中间件，在中间件上进一步统计切片序列的每一个类别节点的计数总和。最后打包成json格式，与时间查询的结果一起提交回服务端进行前端绘制。

## 本章小结

本章介绍了分布式NoSQL数据库HBase和其协处理器（Coprocessor）机制，在此基础上详细阐述了第二章MAP模型在数据库中的具体映射，以及表结构的设计，给出了表中的行键RowKey的设计方案，并简述其特点；最后介绍了本可视化框架在时、空、属查询时的具体过程。

# 海量高维时空数据的预处理

预处理海量的高维时空大数据，将数据转化为本框架需要的结构化数据，对计算能力和方法有很高的要求。当前，用的最多的是分布式的并行计算思想，即先把任务分解成若干个子任务，分配给多个计算单元同时处理，最后汇总计算结果。Spark因为兼有Hadoop平台的优点，又简化了计算模型，大大降低了I/O消耗，加之基于内存进行数据运算，因此本文选用Spark作为本框架的预处理计算平台。

## Spark分布式计算框架

Spark是 UC Berkeley Amp Lab 开发的继 Hadoop之后的新一代大数据分布式处理框架, 基于Scala语言, 通过将计算的中间结果数据持久地存储在内存中减少磁盘 I/O, 非常适合迭代式算法与交互式数据分析[51,55]。

### Spark概述

Spark基于内存计算，是一套可扩展的开源集群计算系统，接口丰富，提供了Scala、Java、Python等接口；除了支持常见的MapReduce范式，还包含了特定场景下的计算库：Steaming、GraphX、MLlib、SQL等，整合了流式处理、图计算、机器学习等功能模块，一站式地解决大规模数据处理遇到的困难。由于Spark在内存中进行计算处理操作，更快速响应查询，很大程度上弥补了Hadoop MapReduce的缺陷，即在网络上传输大量数据和作磁盘读写所带来的效率低下，并能低延迟地返回请求结果；另外Spark侧重于通过数据处理机制的优化来实现批处理的运行速度提升，相同的算法在Spark下的运行速度可以比Hadoop快10-100倍，具备了流式处理的能力[52,54]。

Spark不单单利用内存计算的处理方式来大幅优化性能，与磁盘读写相关的任务处理速度的提升也很大。Spark优先对任务整体进行的解读实现了整体的优化与完善，它会创建有向无环图（DAG），能够描述数据、所要执行的全部操作和数据与操作之间的关系，通过这种方法，处理器能够实现任务间更合理的协调。

Spark生态圈[53]也称为BDAS（Berkeley Data Analysis Stack），它是一个集成了Algorithms、Machines、People的大数据应用平台。Spark生态圈（如图 4‑1所示）以Spark Core为核心，主体的基础功能与Hadoop别无二致，均从持久化层（HDFS、Amazon S3等）读取数据，以MapReduce的模式进行分布式计算，然后利用资源管理器（如MESS、YARN或自带的Standalone）做任务的调度，以支撑构建于平台上的Application。除了主要的基础部分，生态圈还提供了很多应对具体场景的功能组件，如面向实时处理的Spark Streaming、用于当下流行的机器学习的MLlib/MLbase库、满足即时查询的Spark SQL、作图计算的GraphX和进行数学计算的SparkR等等。



图 4‑1 Spark生态系统

1. 资源管理器

Spark可运行在各种资源管理器上，并通过其现全局计算资源的分配、管理和回收。自带的Standalone独立模式是部署Spark集群最简单的方法，但要与别的分布式应用共享集群的话，常用的如Apache社区开发的Yarn和U.C.Berkeley开发的Mesos是应用最普遍的2款管理器。

1. 分布式文件系统（持久化层）

Spark利用分布式文件系统实现大规模数据的存储，保证读写效率和存储的可靠性。Tachyon是基于内存的分布式文件系统，读写操作不经过硬盘。HDFS是基于Google File System架构的开源分布式文件系统，具有高效、可容错等特点。通常将大量的冷数据文件存储在HDFS上，Tachyon里则存放频繁调用的热数据文件。

1. 分布式计算框架

Spark基于MapReduce思想，实现基于内存的分布式计算框架。相比于Hadoop，其更适合处理迭代计算。

1. 数据查询工具

Spark SQL从Spark 1.0开始引入，是用以操作结构化数据的程序包。通过构建一个DataFrame的抽象模型，Spark SQL可以使用SQL语句或者Hive的HQL语句进行数据查询。Spark SQL还支持多种数据源，集成多种程序语言，使得同时运行SQL查询和复杂的数据分析变得非常轻松。

1. 流式处理框架

针对日志更新或浮动车GPS点位等实时数据流，Spark提供了Spark Streaming作为流式计算的组件。Spark Streaming从消息队列中获取数据，然后将计算任务分解为一系列小任务/作业，并以批处理的方式在单位时间里计算每一个小作业。

1. 机器学习库

MLlib是Spark中提供的有关机器学习的程序库，其中包含了很多种ML算法，如分类、回归、聚类、协同过滤等，并且支持模型评估、数据导入等额外功能；MLlib还提供了一些更底层的机器学习原语，包括一个通用的梯度下降优化算法。所有方法均设计为可在集群上轻松伸缩的架构。

1. 图计算框架

GraphX是用来操作图的（如社交中的关系网）的程序库，可以进行并行的图计算。目前，GraphX实现了PageRank、最大连通图社区发现等常用的图算法，广泛应用于社交网络、地图聚类分析等领域。

1. 数据分析工具

SparkR提供基于R语言的数据统计分析功能，能够良好的对接R语言lib库，使得R摆脱单机处理的命运，能够作为job运行于Spark集群上，扩展了R的数据处理能力，常用于金融和统计等领域。

总的来讲，相对于其他几种分布式计算框架，Spark优势明显，其在处理小数据集时，能达到亚秒级延迟，在应对大数据集时，比基于Hadoop的方案也要更快，更高效；而且提供了除Map和Reduce之外更多的数据操作类型，如filter、flatMap、groupByKey等（下一节会详细提到），通用性更强；在执行处理时，支持通过checkpoint实现容错处理；丰富的访问接口，方便开发者使用。尽管，Spark运行时相当耗费集群资源，同一时间下如果集群在跑其他应用程序，甚至还受到相互影响，但在多元化的应用环境下，Spark无疑是最佳的选择。

### 弹性分布式数据集（RDD）

Spark为了实现在内存中执行所有数据处理工作，提出了一种核心的数据结构抽象­——弹性分布式数据集（Resilient Distributed Dataset，简称RDD）[52]。

Spark平台主要使用RDD存储数据，RDD是分布式内存的一个抽象概念，是分布在集群中的只读对象集合，里面的成员被水平切割成若干小数据块，分布于集群的多个节点上并建立分区，然后在内存中的多个计算上提供一种高度受限的共享。RDD不仅支持基于数据集的应用，还具有容错、局部计算调度和可扩展等特性。

RDD只能通过在其他的RDD执行确定的转换操作（如map、join和groupby）而创建，即RDD可视为Spark的一个对象，只能由另外一个RDD产生，这样的限制使得实现容错的开销很低。在RDD上执行不同操作后会产生新的RDD，而新的RDD能够依据依赖列表（LineAge）追溯到最早的RDD，并且向上追溯到所存储的数据，这使得Spark在每个中间过程中不必执行磁盘读写的条件下实现容错。

RDD的操作分为Transformation和Action两种，前者接受一个RDD并返回一个RDD，后者接受一个RDD但返回结果不是RDD[51,56]。Transformation采用惰性调用机制，即一个数据集只记录了从父数据集成的方法，并未进行真正的操作，这种调用链表称之为血缘，而Action操作会直接根据之前的记录的操作进行计算。



图 4‑2 RDD的Transformation和Action操作

由于惰性机制的优点，可以pipeline的形式将一系列RDD操作串联起来，单独放在一个节点上运行，减少中间数据的产生，使逻辑显得单一而简单；同时，减少了由于多次RDD转换而进行数据同步的时间消耗，也避免了像MapReduce那样将太过复杂的逻辑注入单个map/reduce过程的情况。

常用的Transformation和Action操作如表 4‑1和表 4‑2所示：

表 4‑1 常用Transformation操作

|  |  |
| --- | --- |
| **Transformation操作** | **说明** |
| map(func) | 对数据集中的每个元素经过func函数转换后形成一个新的数据集 |
| filter(func) | 过滤，选取能够让func函数返回值为true的元素，构成一个新的数据集 |
| flatMap(func) | 类似于map，但是每一个输入元素，会被映射为0到多个输出元素 |
| mapPartition(func) | 对RDD的每个分区执行func函数 |
| sample(withReplacement,  num, seed) | 根据给定的随机种子seed，随机抽样出数量为num的数据集 |
| groupByKey | 当数据为(K, Y)键值对时，按照K进行分组，返回一个元素类型为(K, Iterable<V>)的数据集 |
| reduceByKey(func) | 当元素为(K, Y)键值对时，先按照K进行分组，然后通过func函数聚合V值，返回一个元素类型为(K, V)的数据集。其中func函数的类型必须为(V, V)=>V |

表 4‑2 常用Action操作

|  |  |
| --- | --- |
| **Action操作** | **说明** |
| reduce(func) | 通过函数func聚集数据集中的所有元素，func函数接受2个参数，返回一个值。这个函数必须是关联性的，确保可以被正确并发执行 |
| collect() | 以数组的形式，返回数据集的所有元素到Driver中。为防止Driver溢出，通常需要控制返回数据集的大小 |
| count(func) | 返回数据集中元素的个数 |
| saveAsTextFile(path) | 将数据集的元素作为文本文件，保存于文件系统（HDFS或其他所支持的文件系统）中 |
| saveAsSequenceFile(path) | 将数据集的元素以sequencefile的方式，保存于文件系统中 |
| saveAsObjectFile(path) | 将数据集以Java序列化的方式写入到一个简单的格式中，可通过SparkContext对象的objectfile()方法加载 |

Spark在任务执行时将数据加载到内存，和需要把计算结果最终写入到持久层设备时才与其进行交互，过程中的运算结果都临时放在内存中。而对于计算的过程中某个中间结果需要重复应用的情况，RDD还提供了一种持久化机制，可以把某个步骤的结果存储在内存或磁盘中，这样可以避免一个数据集多次使用时的重复计算问题。

### Spark的运行机制

Spark集群节点分为主节点（Master）和从节点（Slave），Master管理和调度整个集群，Slave（分为2类，Driver节点和Worker节点）执行集群的作业。每个Spark Application的完成都会经过Master主节点、集群资源管理器（Cluster Manager）、控制作业的Driver节点、执行作业的Worker节点和更小的执行单元以及客户端Client等这几个组成部分。

一个Spark Application被提交到集群后，首先主节点Master会启动一个Driver节点，这类节点是application逻辑上的起点，它会分析主函数中RDD创建、存储和转换等操作，使用DAGScheduler将任务组织构建成一个DAG（有向无环图），并划分成不同的Stage，然后由TaskScheduler通过ActorSystem将任务提交给Cluster Manager，Cluster Manager向Master主节点为各个Executor（执行器）进程申请必要的计算核心、内存等资源进行有序的计算。

Master节点收到请求信息后，向注册过的Worker节点发送启动Executor进程的命令，并在Worker节点启动相应进程后，通知Driver节点进行任务的分配；Driver节点在从文件系统读取数据并创建RDD时，可以指定RDD一个partitons参数（代表在该数据集被划分成的分区的数量），对这些分区进行处理的任务会形成不同的task（任务），Driver节点将task分配给各个Executor（执行器）进行处理，由Executor进程独立完成任务的计算，并将结果返回给Driver节点。在进行Transformation操作时由内建的任务调度器来记录各个操作，并用DAG管理各个任务之间的依赖关系。当进行Action操作时，任务调度器按照DAG把一系列的操作组织成任务，分配给各个Executor进行有序地处理。待全部任务执行完成后，Driver节点通知客户端程序运行结束，完成一次Spark应用的执行[54-57]。Spark数据处理的具体流程如图 4‑3所示：



图 4‑3 Spark数据处理运行流程图

相对于Hadoop框架，Spark对每个task的运算均在内存中完成，中间结果直接用于下个步骤的计算，不存入文件系统，从而节约了大量的I/O操作，具有更高的效率。

## 高维时空数据预处理

在MAP-Vis可视化框架中主要进行可视化的数据，一般都是海量、高维、涉及时空因素的。对于这部分数据，框架的预处理工具的主要任务是：依据MAP组织模型的思路指导，并按照HBase的存储模型设计，将原始数据转化为符合框架可视化格式要求的结构化数据，然后导入设计好的HBase数据表。

### 总体预处理算法流程

因为以瓦片金字塔的构建思想为蓝本，数据自下而上不断融合简化，所以，预处理工作只要完成最底下一层的数据时空切片，往上的每一层都可以不断聚合迭代生成。因此，在预处理之前，需要先确定空间最大层级*l*n，以及时间粒度（本预处理方 法以“天”为最小时间粒度）。

在读取原始数据后，获取每条记录的空间信息、时间信息，以及其他维度信息（*lon，lat，time，categories*），其他维度信息按照2.2.1节的属性聚合树概念，见公式( 2‑5），可得到属性维度的序列化数组；利用空间的对应关系，调用Bin函数，根据（*lon，lat*）可以得到其在最大层级*l*n中的*Pixel*的空间范围，时间time按定义的时间粒度离散化得到其相应的时间段；遍历所有数据则得，不同时间段*tk*下，各个属性节点*cat*的*Pixel*；然后，将同一时间点，相同属性节点，且对应于同一切片数据的*Pixel*进行聚合，即最终得到最大层级*l*n中各属性节点下各个时间段的热图切片数据；最后利用金字塔模型层级之间的关系，即*l*n-1层级的一个切片数据等于其对应空间范围的*l*n层级的四个子切片数据的聚合累加，累加操作后便可得到*l*n-1层级的聚合数据，通过此关系不断聚合，便可以逐层得到*l*n-2, *l*n-3, …, 1层级的切片数据。这个过程的预处理伪代码如表 4‑3所示：

表 4‑3 预处理算法伪代码

|  |
| --- |
| **Algorithm 1** Preprocessing for the system |
| **Input:** Original record, Rm(*lng, lat, time, categories*), m=1, 2, …, N;  **Input:** The maximum map level, *l*n;  **Input:** The time granularity;  **Output:** *l*n(*i*=0, 1, 2, …, n), hierarchical aggregation result;  1: **for** each record R*i* in Origin data **do**  2: cate = FlatTree(categories)  3: *t*k = tBin(time)  4: Pixel*ik* = Extract(*lng, lat, l*n*, cate, t*k)  5: **end for**  6: **for** each Pixel*ik* in the same Tile with Tile*jk* **do**  7: Map Pixel*ik* to Tile*jk*  8: **end for**  9: **for** each Tile*jk* with cate = cat*all*, cat*1*, …, cat*s* **do**  10: Key-Value Tile*ik* to Tile*jks*  11: **end for**  12: Tileln ← All(Tilejks)  13: **for** *i* = *l*n-1, *l*n-2, …, 1 **do**  14: Tile*li* = Aggregate(Tile*li*+1)  15: **end for**  16: **return** Tile*li*, *i* = 0, 1, 2, …, n; |

简言之，预处理的大体步骤如下：

1. 首先需要输入参数，包括预处理的材料、空间的最大层级*l*n，以及时间粒度（天）；
2. 提取原始记录的地理信息，时间信息，以及其他属性维度信息，得到每条记录对应的Pixel，对应模型的extract操作；
3. 利用Spark的Reduce操作，聚合属于同一时间t内，同一瓦片内的像素，对应模型group操作的第一步map；
4. 由于每个像素的属性是序列化的之后的数组结构，每一个下标索引即代表一种属性，我们对切片的四叉树编码，时间序列t，以及下标索引组合做为Key，其像素以及对应值作为Value，生成键值对。其所代表的意义即是在什么时间、什么地点、发生了什么属性的事件，并最终得到最大层级*l*n中各属性节点下各个时间段的热图聚合数据，对应模型group操作的第二步，表 4‑3中换用用Key-Value表示；
5. 根据金字塔逐层聚合的关系，即第*l*n层四个子切片数据聚合成相同空间范围的第*l*n-1层级的一个切片数据，即表 4‑3中的Aggregate，对应模型的aggregate操作。通过此关系，不断聚合，便可以逐层得到*l*n-2，*l*n-3，…1层级的切片数据，构建成多维度的聚合金字塔。

### 具体算子函数解释

预处理工作通过Spark SQL组件以DataFrame的形式读取原始数据的文本[51,52,58]，挑选其中的上车经度pickup\_longitude、纬度pickup\_latitude、上车时间pickup\_datetime以及乘客支付方式payment\_type作为预处理的主要字段，以此为基础的数据集开始执行RDD操作，如图 4‑4所示。



图 4‑4 Spark预处理基本路线图

图中，矩形框表示的是原始数据集和预处理过程中的若干中间数据集RDD，菱形框表示执行的RDD操作算子。当把数据从原始集通过Spark SQL的方式读入结构化的DataFrame后，首先用map算子执行Pixel2Tile函数，将每一条原始数据记录计算得出对应的具体瓦片，然后用flatMap算子扁平化，再调用reduceByKey算子执行group2Tile函数，统计同一个Key的像素之个数。我们以天为最小的时间粒度，在天的基础上调用day2Month聚合成月，调用month2Year聚合成年，完成最底层的热图数据的预处理工作并调saveAsTextFile操作持久化到硬盘。

在聚合成热图数据的tile\_day、tile\_month及tile\_year的基础上，用map算子执行cat2All函数，然后reduceByKey统计每天、每月、每年的每个属性的热图聚合值，形成最底层的天、月、年的属性值，然后作持久化。最后以底层的热图数据和属性数据，执行Tile2Parent函数实现逐层向上聚合，并持久化存储每一层的结构化数据。

以下对预处理过程中算子执行的主要的3个函数（具体对应MAP模型中的3个操作）作具体的解释：

1. Pixel2Tile函数

该函数的输入参数包括原始记录的经纬度、层级、日期时间以及支付方式类型，计算得出该点位对应于某层级的切片（瓦片）号，以及在该切片内的像素坐标。lnglat\_to\_quad将经纬度转四叉树编码，quadkey\_to\_num将四叉树编码转为数值形式；由于切片tile默认是256\*256的分辨率，在层级1的某个切片（256\*256像素）的某一个像素所能代表的地理范围，就是与之对应的第8级的某个切片的地理范围，所以点位对应的瓦片中的像素位置offset就是这么求得的。timeStamp把时间离散化到天的单位，catCode根据日期时间和支付类型映射出属性点序号。具体代码如表 4‑4所示：

表 4‑4 Pixel2Tile函数主要代码

|  |
| --- |
| def Pixel2Tile(lng, lat, zoom, time, payment):  result = []  try:  quadkey\_pixel = lnglat\_to\_quad(float(lng), float(lat), zoom)  pixel = mercantile.quadkey\_to\_tile(quadkey\_pixel)  tile = mercantile.tile(\*mercantile.ul(pixel.x, pixel.y, pixel.z) + (pixel.z - 8,))  tile\_num = quadkey\_to\_num(mercantile.quadkey(tile))  origin\_x = tile.x \* math.pow(2,8)  origin\_y = tile.y \* math.pow(2,8)  pixel\_x = int(pixel.x - origin\_x)  pixel\_y = int(pixel.y - origin\_y)  offset = int(pixel\_y\*math.pow(2,8)+pixel\_x)  day = timeStamp(time)  cat = catCode(time,payment)  st\_tile\_D = str(tile\_num) + ":" + str(zoom-8) + ":"+str(cat) +":" + str(day)  one = [ {offset:1} ,1 ]  result.append((st\_tile\_D,one))  except:  return None  return result |

1. Tile2Parent函数

该函数的输入参数是*l*n层的子切片数据，每一个切片对应其上一层级*l*n-1的父切片，会对应在四个象限0、1、2、3中的一个；每4个子切片先缩小一倍重采样之后，再根据其在0、1、2、3四个象限编码位置，换算到父切片的像素中去，其像素对应的位置即是切片缩小一倍重采样之后的像素加上象限位置的偏移所得。具体代码如表 4‑5所示：

表 4‑5 Tile2Parent函数主要代码

|  |
| --- |
| def Tile2Parent(st\_tile):  childId, child = st\_tile  parent = [{},0]  zoom = childId.split(":")[1]  quad\_num = childId.split(":")[0]  cat = childId.split(":")[2]  tStamp = childId.split(":")[3]  parentNum = int(quad\_num) / 4  parentId = str(parentNum)+":"+str(int(zoom)-1) + ":"+cat+ ":" + tStamp  parent\_quad\_position = int(quad\_num) % 4  x\_offset = 0  y\_offset= 0  if parent\_quad\_position in [1, 3]:  x\_offset = 128  if parent\_quad\_position in [2, 3]:  y\_offset = 128  for old\_pixel, count in child[0].items():  old\_x = (old\_pixel) % 256  old\_y = (old\_pixel) / 256  new\_x = (x\_offset + old\_x / 2)  new\_y = (y\_offset + old\_y / 2)  new\_pixel = int(new\_y \* (math.pow(2, 8)) + new\_x)  if new\_pixel in parent[0]:  parent[0][new\_pixel] += child[0].pop(old\_pixel)  else:  parent[0][new\_pixel] = child[0].pop(old\_pixel)  parent[1] = child[1]  return (parentId,parent) |

1. group2Tile函数

该函数由reduceByKey算子调用，其作用域是key-value类型的键值对，并且是只对每个key的value进行处理，如果含有多个key的话，那么就对多个values进行处理[51]。group2Tile对相同key的集合（像素集合或者属性集合），做运算，Counter先分别统计每个集合中的像素点或属性点值，然后二者叠加，相同key的集合两两执行此操作。具体代码如表 4‑6所示：

表 4‑6 group2Tile函数主要代码

|  |
| --- |
| def group2Tile(pixels\_1, pixels\_2):  pixels\_1 = list(pixels\_1)  pixels\_2 = list(pixels\_2)  A = pixels\_1[0]  B = pixels\_2[0]  countA = Counter(A)  countB = Counter(B)  result = dict(countA + countB)  sum = pixels\_1[1] + pixels\_2[1]  return [result, sum] |

## 实验结果及分析

基于上述的Spark预处理工具，我们对三个数据集进行预处理实验，检验实际的处理效率，如表 4‑7所示（taxi—纽约出租车数据，crime—芝加哥犯罪数据，brightkite—全球推特签到数据）：

表 ‑7 不同数据集Spark预处理效率



表中从左往右显示的是每个数据集的原始数据大小、数据记录数、处理后存入表内的数据大小和行数、预处理过程的内存消耗以及处理时间。其中，签到数据由于是全球范围的，比芝加哥犯罪数据的空间范围大，切片数更多，所以出现原始数据少但处理结果和内存消耗大于犯罪数据的情况。由上表可知，该预处理工具对数据集的处理能力还有待提高，但基本控制在可以接受的时间范围内。

## 本章小结

本章在对Spark计算框架做了简要介绍（包括Spark架构、核心组件RDD及具体处理流程）后，详细阐述了MAP-Vis可视化框架的预处理工具的处理方法和运作过程，解释了处理过程中几个主要的函数代码，并给出基本的预处理效率结果，从实验的结果可以看出其大体处于可以接受的范围内。

# 基于MAP模型的可视化系统实现

## 系统总体概述

### 系统设计框架

为验证本文提出的MAP模型和可视化框架的可行性和有效性，实现了一套纽约出租车时空数据可视化原型系统。如图 5‑1所示，本文系统采用B/S架构，系统架构分为客户端，中间件，以及后台集群三部分组成。其中，预处理及存储模块建构于Linux高性能的后台集群之上，其主要负责原始数据的多维金字塔模型建立以及预处理数据的分布式存储；中间件包含一个数据访问接口，其主要解析用户的查询请求，并从数据库端抽取相应数据作简单的统计操作，然后付诸可视化；客户端主要负责对从后台获取的热图数据、时序数据和属性数据进行可视化。

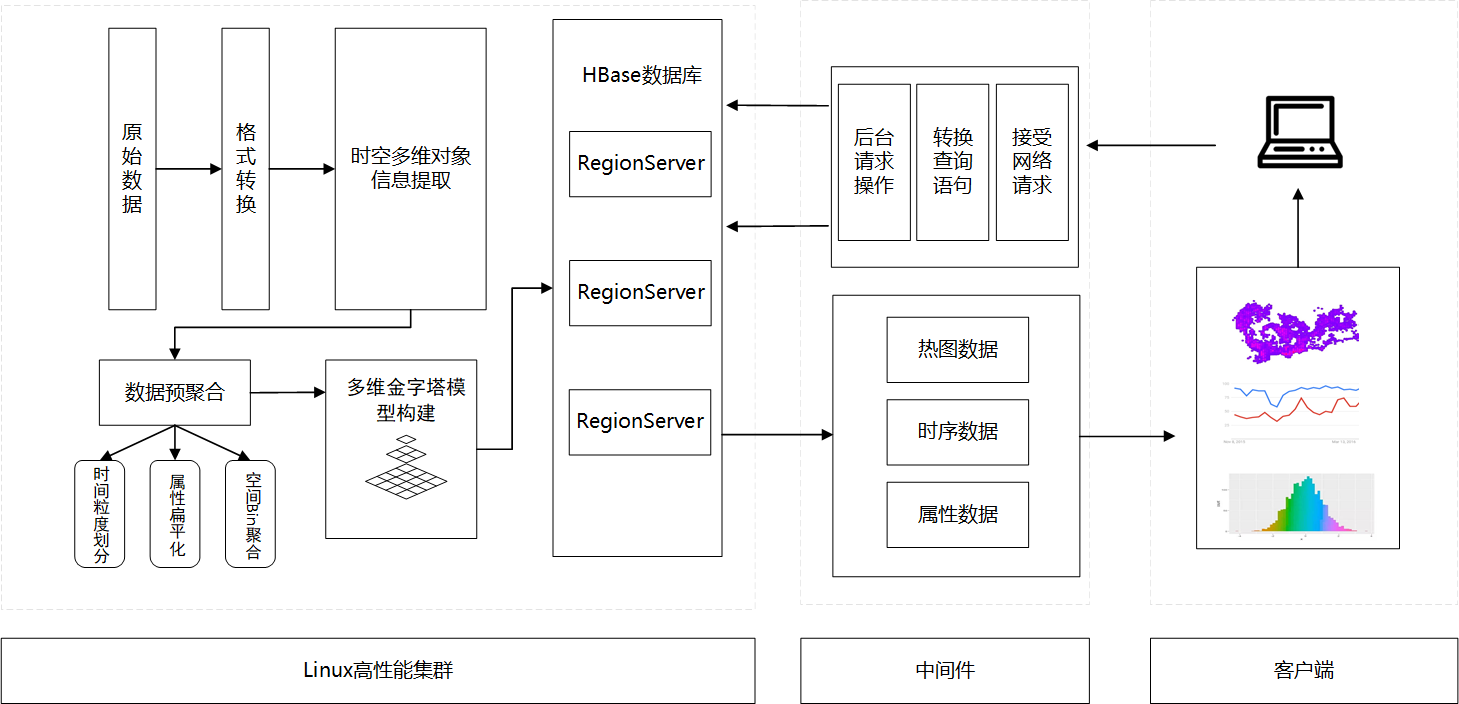


图 5‑1 系统框架图

其中，服务端由数据预处理以及HBase数据库存储模块组成。其中数据预处理由Spark并行工具完成，首先对原始数据进行格式转换，并提取数据对象时空，以及其他维度的属性，并对数据进行时间粒度的划分，属性的序列化，然后进行map、group以及aggregate三个操作，最终构建时空多维属性的金字塔模型，并按照3.1介绍的存储映射方式存于HBase数据库。

中间件包含一个数据访问接口，其主要解析用户的查询请求，并转换为HBase的查询语句，从HBase数据库端抽取相应的可视化数据并最终返回到前端绘制。

客户端负责发送请求，并接受相应的可视化数据，并将热图数据，时序数据，维度属性结果显示在同一个平台，其主要能实现地图多尺度的热图数据查询，时间范围以及维度的联动查询。

### 系统开发的具体环境

本文可视化系统的开发软硬件环境，以及用以实验的数据集如下：

1. 系统开发的软件环境如表 5‑1所示：

表 5‑1 系统开发软件环境

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **开发平台** | **数据库** | **开发语言** | **集群环境** | **部署环境** |
| Windows10教育版64bit | HBase 1.2.0-CDH5.10.0 | Java  Python  HTML  CSS | Hadoop 2.6.0  Spark 1.6.0 | CentOS 6.2 64bit |

1. 系统运行的主要硬件环境

本可视化系统使用了6台12核的浪潮刀片服务器，配置情况如图 5‑2所示：



图 5‑2 服务器集群配置情况图

1. 可视化系统使用的数据集

纽约市2014年、2015年以及2016年上半年共30个月的出租车（yellow car）原始数据，数据体量约54.5G。

### 系统主要功能

如图 5‑3所示，是MAP-Vis框架下支持的出租车数据可视化原型系统的前端显示界面，主要包括空间地图部分、时间轴部分、属性柱状图部分以及数据集选择切换部分，其具有的功能包括：



图 5‑3 纽约出租车数据可视化原型系统前端显示界面

1. 空间地图数据查询与可视化

出租车的原始数据经过预处理，存储在后台的集群数据库中，在对其可视化之前需要发出查询请求。空间地图模块主要发送空间查询请求，当用户进行交互操作（包括地图的放大、缩小及平移）导致地图范围发生变化时，地图的每个瓦片（切片）都会异步地独立发送其空间范围及此时的时间轴的时间区间到服务端，到后台作编码处理，再发往client中间件作查询。返回的都是聚合好的对应瓦片像素值的json包，再付之前端渲染。如表 5‑2所示是一个具体的返回json包示例：

表 5‑2 空间查询返回json示例

|  |
| --- |
| {17731: 1, 17423: 489, 32810: 1, 2099: 1, 2100: 1, 2112: 1, 3137: 1, 64579: 1, 2117: 1, 4166: 1, 11337: 1, 14413: 1, 20561: 3, 20562: 2, 18515: 64, 27732: 1, 29783: 1, 30808: 5, 39001: 1, 53094:5790, 55144: 1003, 1856: 2, 1857: 235, 16738: 2, 2959: 1, 11155: 206, 6980: 1, 6469: 1, 27553: 344, 1089: 1, 18766: 1, 12251: 1, 5090: 1, 26598: 1, 35822: 1, 37874: 1, 34131: 1, 10230: 1, 10235: 2508, 27477: 1} |

1. 时间轴数据查询与可视化

时间轴模块主要发送时间查询请求，显示的是每一时间点（如每一天）当前地图整个空间范围的可视化值的总和。当用户滑动滚轮改变时间轴的时间区间范围，系统都会触发一次请求，将当前地图的整个空间范围以及改变后的时间区间发往服务端，在后台作解析瓦片及编码处理，再发往client中间件作查询。返回的是对应时间轴每一天的数值的json包，再付之渲染。如表 5‑3所示是一个具体的返回json包示例：

表 5‑3 时间查询返回json示例

|  |
| --- |
| {"time\_series":  {"2015-03-11":368,"2015-03-10":350,"2015-03-08":307,"2015-03-07":370,"2015-03-06":380,"2015-03-05":331,"2015-03-04":331,"2015-03-03":351,"2015-04-15":379,"2015-03-02":327,"2015-03-01":255,"2015-04-13":347,"2015-03-09":315,"2015-03-31":419,"2015-03-30":401,"2015-03-29":250,"2015-03-28":352,"2015-03-27":377,"2015-03-26":366,"2015-03-25":321,"2015-03-24":338,"2015-03-23":298,"2015-03-22":260,"2015-03-21":304,"2015-03-20":354 }} |

1. 属性柱状图数据查询与可视化

属性柱状图模块并不具体发送请求至服务端，只作属性数据的可视化显示，本文实现的系统目前只纳入支付方式的属性维度。由于属性维度是编码在行键里的，在作时间查询时，会顺便统计每个属性在当前时间范围内，地图的整个空间范围里的可视化值之和，数据也是以json包返回。如表 5‑4所示是一个具体的返回json包示例：

表 5‑4 属性查询返回json示例

|  |
| --- |
| {"categories":{"0":[2122,1873,7],"1":[2180,1624,9],"2":[2183,1737,9],"3":[2441,1976,7],"4":[2356,1939,7],"5":[2058,1777,7],"6":[1893,1570,12]}} |

1. 数据集显示切换

本系统除了能够可视化纽约出租车数据外，系统还预处理了芝加哥犯罪数据和全球签到数据，可以选择数据集内容切换显示，功能与以上3点相同。

## 实验设计与结果分析

本实验以纽约曼哈顿区的出租车Yellow Car数据为可视化对象，该数据集记录了2014年至2016年6月份共30个月的出租车点位等信息，大小共计约54.5G。实验分别从MAP模型的有效性、数据扩展性、预处理的集群规模扩展性3个角度测试了MAP-Vis的数据可视化性能。集群测试的基本环境共有7个节点（其中一个主节点Master），集群的节点配置如表 5‑5所示：

表 5‑5 集群节点配置情况

|  |  |
| --- | --- |
| **属性** | **配置信息** |
| CPU | 2路 12 Core Intel(R) Xeon(R) CPU E5-2620 2.00GHz |
| Memory | 32G |
| Network | BandWidth 1Gbps |
| OS | CentOS 6.2 64bit |
| JVM Version | JVM 1.8.0 |
| HBase Version | HBase 1.2.0 |
| Spark Version | Spark1.6.0 |
| Zookeeper Version | Zookeeper 3.4.5 |

1. MAP模型有效性实验

该实验以查询获取的数据量作为自变量，观察查询的响应时间变化，其中通过改变查询的时间跨度来进而控制数据量的多少，实验结果如表 5‑6所示：

表 5‑6 模型有效性实验结果表

|  |  |
| --- | --- |
| **时间跨度/月** | **平均查询响应时间/ms** |
| 3 | 974 |
| 6 | 1223 |
| 9 | 1056 |
| 12 | 1204 |
| 15 | 1428 |
| 18 | 1114 |
| 21 | 1268 |
| 24 | 1131 |
| 27 | 1129 |
| 30 | 1264 |

其中，平均查询响应时间是从请求发起到最后渲染完毕进行计时的。中间可能包括一些网络延迟的因素，但总体可以看出其响应延迟稳定在1.1-1.2s之间；如果摒除网络方面的影响，基本能够保证亚秒级的响应，基本达到流畅的交互可视化体验。变化趋势如图 5‑4所示：

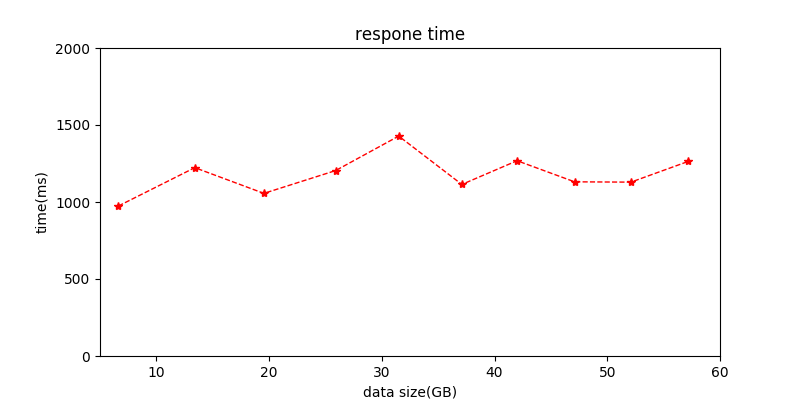


图 5‑4 模型有效性实验响应时间变化趋势

图中横坐标代表的是不同时间跨度下包含的数据容量大小。从图中也可以看出，随着查询的数据规模的扩大，响应时间曲线基本保持平稳，未呈明显的线性增长，可见MAP模型及相应的存储模型在组织数据并作可视化时，能够很好地适应大规模的时空数据集。

1. 数据扩展性实验

该实验以HBase集群的Region数目作为自变量，观察查询响应时间的变化。因为HBase的查询是直接访问到每个Region上的，所以集群规模的大小是会对响应时间造成影响的。实验结果如图 5‑5所示：

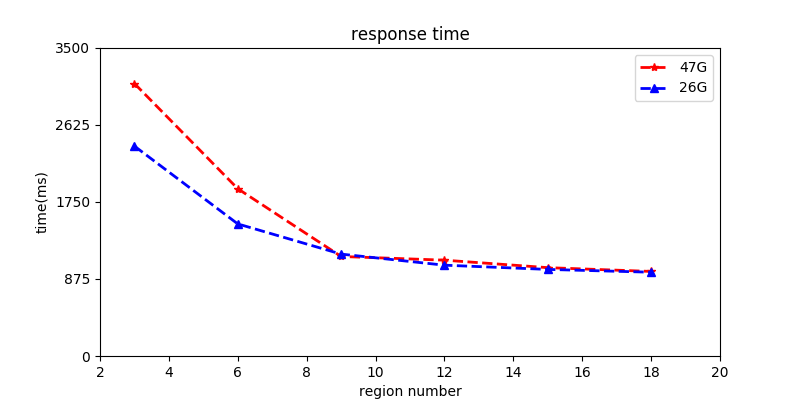


图 5‑5 数据扩展性实验响应时间变化趋势

图中，蓝线和红线分别对应的是查询1年与2年的数据量的变化趋势。从图中可以看出，随着集群Region数目的增加，数据存储得更加分散，单个Region被访问变得没有那么频繁，减少了很多请求访问的排队时间，从而提高了整体的响应时间。换言之，如果条件允许，本框架完全可以通过增加节点，增加Region的数量来提高查询的效率，充分利用集群分布式的优势，进而改善交互的可视化体验。

1. 预处理集群的规模扩展性实验

该实验以可视化框架的Spark预处理集群为实验对象，以集群的计算节点数作为自变量，观察预处理时间的变化，预处理的材料是上述2年半的纽约出租车数据，处理时间统计不包括导入HBase数据库的时间。实验结果如表 5‑7所示：

表 5‑7 预处理集群扩展性实验结果表

|  |  |
| --- | --- |
| **计算节点数** | **预处理时间/min** |
| 4 | 366 |
| 5 | 288 |
| 6 | 234 |
| 7 | 204 |
| 8 | 168 |
| 9 | 162 |

通过引入更多的计算节点，使得Spark集群有更多的Worker节点和Executor进程分摊task，预处理的工作必然加速，实验结果也是可期的。预处理时间的变化趋势如图 5‑6所示：

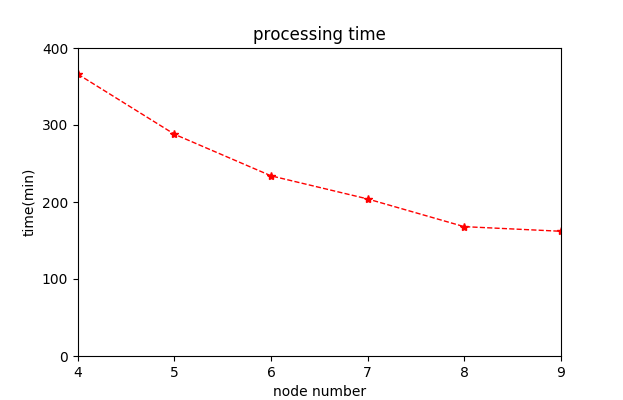


图 5‑6 Spark集群规模扩展的预处理时间变化趋势

由图可知，节点的增加使得预处理时间有明显的下降趋势，在未来可以利用这一点构建大规模的集群架构，以提高预处理效率。

## 本章小结

本章从实现系统的架构、开发环境以及系统功能等方面介绍了用于展示的纽约出租车数据可视化原型系统，以证明本文提出的可视化框架的可行性。同时，设计了3个实验：MAP模型有效性、数据扩展性和集群规模扩展性实验。通过这3个实验表明，在MAP模型及MAP-Vis可视化框架支撑下的可视化原型系统在查询效率上基本能够达到亚秒级的响应，且不随查询数据的增大而出现响应曲线的线性上升；而且可以通过增加节点的方式，进一步提高系统的查询效率和预处理效率。

# 总结与展望

## 论文总结

本文从时空大数据高效可视化以及分析挖掘的需求着眼，重点针对可视化的扩展性问题，力求以分布式的方式提升可视分析数据规模及预处理能力，同时保证高效的查询响应效率以及交互可视化效果。

本文的主要工作总结如下：

1. 本文以传统的瓦片金字塔模型作为切入点，提出了层级聚合时、空、属性多维度的数据组织模型——多维度聚合金字塔（MAP）模型。该模型延伸了瓦片金字塔的空间聚合方式，将之从2D的聚合扩展到nD的聚合，极大程度上适应了大数据的时空和高维特性。该模型定义了3个基本概念（时空像素、属性聚合树和时空切片），用于作为数据的组织单位，并阐释了组织成MAP模型结构的3个操作：extract、group及aggregate，同时考虑到空间多尺度问题，并兼顾时间和其他多维的属性，多维度的属性值可随时空的聚合变化而汇聚分解。从数学模型上较完备地定义了原始数据到可视化的中间结构数据的转换过程，为后续的实际处理工作提供了理论依据。
2. 以提出的MAP模型为基础，设计实现了一套基于该模型的开源可视化框架（MAP-Vis），其中以HBase作为数据库分布式存储，以Spark集群作为预处理工具。存储上首先设计了MAP模型数据到分布式数据库的映射方案，以及对应的HBase库表结构及键值编码，满足海量存储需求及上层的查询读取。在预处理上设计了原始数据到MAP模型数据的映射方案及处理流程，以Spark集群分布式方式运行，完成从原始数据到处理、再到入库的过程，尽可能缩短预处理的时间。
3. 为验证MAP模型和开源MAP-Vis可视化框架的可行性和有效性，本文以纽约出租车数据为例，进行了查询效率、分布式处理能力多项实验。实验结果表明，本系统能支持在亚秒级对查询结果的快速获取，在大数据的条件下，仍能达到良好的交互式可视化效果。

## 展望

尽管本文的研究工作取得了一定的进展，提出的MAP-Vis可视化框架在可视化效果以及整体交互上比较流畅，其查询响应和预处理效率可以满足实际应用需求。但现有的实现方法和技术，仍旧存在进步的空间，下一步的工作主要包括以下几点：

1. 本框架对点类型的数据支持能力较强，但是可视化的元素还可以包括线类型元素、面类型元素、影像等，应该考虑尽可能兼容其他数据类型的需求。
2. 可视化UI中，查询、交互的方式较简单，针对矩形查询（拉框查询）和多边形查询的情况，MAP-Vis对此应进行研究并提出改进方案，以丰富可视化的查询形式。
3. 在当地图分辨率到一定的等级，可以考虑可视化显示原始的点位数据，而非预处理数据，这就要求原始数据必须予以保留（存放在HDFS上），可能需要对MAP模型做进一步的扩展或修改。
4. 单纯的可视化展示不能完全发掘大数据的规律与价值，尝试加入数据分析模块，使MAP-Vis的框架功能更完整。

参考文献

1. 李德仁,马军,邵振峰.论时空大数据及其应用[J].卫星应用,2015(9):7-11.
2. Liu Zhicheng, Biye Jiang, Jeffrey Heer. imMens: Real-time Visual Querying of Big Data[C]. Eurographics Conference on Visualization (EuroVis),2013,32(3).
3. 任磊,杜一,马帅,张小龙,戴国忠.大数据可视化分析综述[J]. 软件学报,2014,5(9):1909-1936.
4. Cory D. Big data: Welcome to the petacentre[J]. Nature, 2008, 455(7209).
5. JRO, BJM, PSM. Challenges and opportunities of open data in ecology[J]. Science, 2011, 331(6018).
6. Labrinidis A, Jagadish H V. Challenges and opportunities with big data[J]. Proc. VLDB Endow., 2012, 5(12): 2032-2033.
7. 陶雪娇,胡晓峰,刘洋. 大数据研究综述[J]. 系统仿真学报, 2013, (S1): 142-146.
8. 程学旗,靳小龙,王元卓, et al. 大数据系统和分析技术综述[J]. 软件学报, 2014, (09): 1889-1908.
9. 高志鹏,牛琨,刘杰. 面向大数据的分析技术[J]. 北京邮电大学学报, 2015, (03): 1-12.
10. T M. Visualization analysis and design[J]. Wiley Interdisciplinary Reviews Computational Statistics, 2015: 2 ( 4) : 387-403.
11. J C D H G. The Visualization Handbook[J]. New York: Academic Press, 2004: 76-85.
12. Murthy D, Gross A, Mcgarry M. Visual Social Media and Big Data. Interpreting Instagram Images Posted on Twitter. Digital Culture & Society. 2016: 113.
13. Itoh M, Yokoyama D, Toyoda M, et al. Visual fusion of mega-city big data: An application to traffic and tweets data analysis of Metro passengers[C]. 2014 IEEE International Conference on Big Data (Big Data), 2014: 431-440.
14. 马晓亭. 图书馆大数据可视化分析系统的设计与实现[J]. 图书馆学研究, 2015, (10): 37-41+36.
15. 何晓琳,钱庆,张泽. 肿瘤流行病学数据可视化分析[J]. 中华医学图书情报杂志, 2016, (01): 73-80.
16. 崔迪,郭小燕,陈为. 大数据可视化的挑战与最新进展[J]. 计算机应用, 2017, (07): 2044-2049+2056.
17. Tobler W R. Experiments In Migration Mapping By Computer[J]. The American Cartographer, 1987, 14(2): 155-163.
18. Buchin K, Speckmann B, Verbeek K. Flow Map Layout via Spiral Trees[J]. IEEE Transactions on Visualization and Computer Graphics, 2011, 17(12): 2536-2544.
19. Scheepens R, Willems N, Wetering H V D, et al. Composite Density Maps for Multivariate Trajectories[J]. IEEE Transactions on Visualization and Computer Graphics, 2011, 17(12): 2518-2527.
20. Peuquet D J, Kraak M-J. Geobrowsing: Creative Thinking and Knowledge Discovery Using Geographic Visualization[J]. Information Visualization, 2002, 1(1): 80-91.
21. Keim D A, Kriegel H P. Visualization techniques for mining large databases: a comparison[J]. IEEE Transactions on Knowledge and Data Engineering, 1996, 8(6): 923-938.
22. P. Godfrey, J. Gryz, and P. Lasek. Interactive visualization of large data sets. Technical Report EECS-2015-03, York University, March 2015.
23. S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar,M. Tolton, and T. Vassilakis, “Dremel: Interactive analysis of web-scale datasets,” Proc. Very Large Data Bases Endowment,vol. 3, no. 1-2, pp. 330–339, 2010.
24. Google BigQuery[EB/OL]. https://www.bimeanalytics.com/data-connectors/big-query.
25. MapD[EB/OL]. https://www.mapd.com/platform/core.
26. Gray J, Bosworth A, Layman A, Pirahesh H. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In: Su SYW, ed. Proc. of the 12th Int’l Conf. on Data Engineering. New Orleans: IEEE Computer Society, 1996, 152~159.
27. Jack孟.数据立方体---维度与OLAP[EB/OL].[2014-12-10].https://www.cnblogs.com/mq0036/p/4155832.html.
28. Yannis Sismanis, Nick Roussopoulos. The Dwarf Data Cube Eliminates the High Dimensionality Curse[J]. UMIACS-TR,2003,120.
29. Yannis Sismanis, Antonios Deligiannakis, Nick Roussopoulos, Yannis Kotidis. Dwarf: Shrinking the PetaCube. In Proceedings of ACM International Conference on Management of Data(SIGMOD), pages 464-475, 2002.
30. 冷芳玲. 支持高效查询的数据立方构建技术研究[D]. 沈阳:东北大学,2008.
31. Lauro Lins, James T. Klosowski, and Carlos Scheidegger. Nanocubes for Real-Time Exploration of Spatiotemporal Datasets[J]. IEEE Transactions on Visualization and Computer Graphics, 2013,12(19):2456-2465.
32. J. Im, F. G. Villegas, and M. J. McGuffin, “Visreduce: Fast and responsive incremental information visualization of large datasets,” in Proc. IEEE Int. Conf. Big Data, 2013, pp. 25–32.
33. D. Cheng, P. Schretlen, N. Kronenfeld, N. Bozowsky, and W. Wright, “Tile based visual analytics for twitter big data exploratory analysis”, IEEE International Conference on Big Data, 2013.
34. 王瑞松. 大数据环境下时空多维数据可视化研究[D]. 杭州:浙江大学,2016.
35. 殷君茹,侯瑞霞,唐小明,罗鹏. 基于瓦片金字塔模型的海量空间数据快速分发方法[J]. 吉林大学学报(理学版),2015,11(53):1269-1274.
36. 张延辉,孟鑫,李立松.HBase企业应用开发实战[M]. 北京:机械工业出版社,2014.
37. Nick Dimiduk, Amandeep Khurana. HBase实战[M]. 北京:人民邮电出版社,2013.
38. Lars George. HBase权威指南[M]. 北京:人民邮电出版社,2013.
39. HBase[EB/OL].http://en.wikipedia.org/wiki/NoSQL.
40. 雷军,刘绍辉,冯宏华,何炎祥. HBase针对移动互联网业务的改进与优化[J]. 小型微型计算机系统,2017,38(5):1158-1163.
41. 孟鑫淼. 基于HBase的空间数据云存储研究[D]. 开封:河南大学,2016.
42. 李冬. 面向实时交通流数据的HBase辅助索引技术研究和实现[D]. 北京:北方工业大学，2017.
43. 陆婷. 基于HBase的交通流数据实时存储系统的设计与实现[D]. 北京:北方工业大学,2016.
44. 丁琛. 基于HBase的空间数据分布式存储和并行查询算法研究[D]. 南京:南京师范大学,2014.
45. 付文静. 基于HBase的大数据存储查询技术研究[D]. 成都:电子科技大学,2015.
46. 卓海艺. 基于HBase的海量数据实时查询系统设计与实现[D]. 北京:北京邮电大学,2013.
47. 秦恒. HBase查询及索引机制研究[D]. 重庆:重庆邮电大学,2016.
48. 廖一陈. HBase多条件复杂查询的实现方法研究[D]. 北京:北京交通大学,2017.
49. 崔晨,郑林江,韩凤萍,何牧君. 基于内存的HBase二级索引设计[J]. 计算机应用,2018.
50. 基于HBase的并行计算架构之rowkey设计篇[WB/OL]. http://xdataopen.blog.51cto.com/4219560/1117864.
51. Spark[EB/OL]. <https://en.wikipedia.org/wiki/Apache_Spark>.
52. Holden Karau, Andy Konwinski, Patrick Wendell, Matei Zaharia. Spark快速大数据分析[M]. 北京:人民邮电出版社,2015.
53. 郭景瞻. Spark入门实战系列--1.Spark及其生态圈简介[EB/OL]. [2015-08-04]. https://www.cnblogs.com/shishanyuan/archive/2015/08/04/4700615.html.
54. 宋晓悦. 基于Spark的城市轨道交通客流分析平台研究[D]. 北京:北京交通大学,2017.
55. 贾昀腾. 基于Spark的出租车轨迹组织方法研究[D]. 南京:南京师范大学,2017.
56. 王茄力. 基于Spark的混合推荐系统[D]. 合肥:中国科学技术大学,2017.
57. 孙科. 基于Spark的机器学习应用框架研究与实现[D]. 上海:上海交通大学,2015.
58. 张磊, 朱锋, 钟华. 基于Spark的交互式数据预处理系统[J]. 计算机系统应用,2016,25(11):84-89.

致谢

在论文完成之际，谨向我给予帮助的老师、同学、朋友表示衷心的感谢。

首先，衷心感谢导师吴华意教授。回顾在硕士两年的学习中，吴老师不仅提供了优越的学习环境，而且其严谨的治学态度和对我的严格要求促进我进步。两年来，我在学习和科研上所取得的每一点成绩、每一点进步无不浸透着导师的心血。在此谨向尊敬的导师致以最诚挚的谢意。

其次，我要感谢关雪峰老师，在我论文的开题与写作当中给予的帮助与支持。更重要的是，对我的学习生活悉心指导。关老师渊博的知识、勤奋的精神以及对学生高度负责任的态度人令我难忘，激励着我上进。

同时，还要感谢在我论文写作过程中给予指导意见的成波师兄、李真强师兄以及谢冲同学。在我论文开题与实验设计、论文写作当中，2位师兄给了我莫大的帮助与意见，尤其是谢冲同学，这份毕设是我们共同努力的成果，凝结了我们的心血，正是有了师兄和同门的督促与指导、帮助与扶持，才有了本毕业论文的完成。

感谢我的室友康泽慧、倪文强和胡锦程两年的陪伴，在生活上给予了我莫大的帮助，你们的存在，才有温馨的325宿舍。

感谢我的父母亲人，多年来对我的关爱与资助，你们的期待与关心是我不断前进的动力。

最后，我要感谢我自己，感谢自己本科时期的努力，才有了保送研究生的资格，拥有了继续深造的机会；感谢自己研究生两年里勤俭节约，不依靠家里的资助完成了学业；感谢自己的努力，有了进入华为工作的机会；感谢自己的坚持，羽毛球技术有了长进；感谢自己，今后我将继续努力。