

# 北京工业大学

## 毕业设计（论文）任务书

题目 一种基于统计推断的交互式大数据探索系统

专业 计算机科学与技术 学号 12070001 姓名 柯伟辰

主要内容、基本要求、主要参考资料等：

主要内容及基本要求：

1. 查询资料，对 Hadoop HDFS 分布式文件存储系统及 Spark 大数据分析框架建立初步的认知；
2. 设计出统计推断模块的核心算法；
3. 分模块设计并实现整个分析系统；
4. 对算法进行准确度分析，对系统进行性能分析，并作适当的改进；

主要参考资料

1. 王松桂，张忠占，程维虎. 概率论与数理统计（第三版）. 科学出版社，2013.
2. Rui ding, Qiang Wang, Yingnong Dang, Qiang Fu, Haidong Zhang, Dongmei Zhang. YADING: Fast Clustering of Large-Scale Time Series Data. In proc of VLDB, 2015
3. Apache Spark: Lightning-Fast Cluster Computing, <http://spark.apache.org/>
4. Apache Hadoop Distributed File System.  
<http://hadoop.apache.org/hdfs/>
5. S. Acharya, P. B. Gibbons, and V. Poosala. Congressional samples for approximate answering of group-by queries. In ACM SIGMOD, May 2000.

完成期限：

指导教师签章：\_\_\_\_\_

专业负责人签章：\_\_\_\_\_

2015 年 11 月 27 日

### 摘要

随着大数据时代的到来，大数据中蕴含的商业价值日益被各家企业所重视。通过对大数据的分析，企业可以获取数据当中隐含的规律，从而更好地服务客户，在制定战略时也能做到有的放矢，最终实现企业经济效益的大幅增长。

然而，分析大数据是一项非常困难的任务。企业决策者们在分析数据时，即使使用成熟的分布式系统进行运算，也不得不忍受漫长的响应时间。这使得决策者的查询效率受到严重限制，从而非常容易遗漏数据当中的规律，错失商业良机。而事实上，与传统数据库技术不同的是，决策者们往往不需要知道精确的查询结果，事实上，他们对于一个“大概准确”的结果就已经满意了。基于这样的现象，本课题旨在利用统计学知识和方法，设计一套能够快速给出大致精确结果的大数据探索系统。

本文的工作和创新性主要有以下几个方面：

首先，构建了一个基于统计推断的交互式大数据分析系统。该系统将传统的 Map-Reduce 方法和采样推断算法结合起来，考虑到“大概准确”的结果就可满足数据工作者的需求，通过牺牲一定的精确度换来非常短的响应时间，使响应时间达到 5 秒以下，从而提高了系统的性能，满足了交互性查询的需要。

其次，本文发现了简单随机采样的一个缺陷，并且针对这个缺陷，提出了使用分层采样替代简单随机采样的方法，取得了良好的实验效果。

第三，推导了基于简单随机抽样和分层采样时，计算 COUNT，SUM 和 AVG 三种聚合函数及其置信区间的公式，并给出了相应的算法设计；

最后，对系统进行了测试，实验结果表明该系统在效率上能够满足交互式探索的需求，并且提出的分层采样算法的准确度相对于简单随机采样有明显优势。

**关键词：**大数据，商业智能，统计推断

## Abstract

With the coming of “The era of big data”, business information underneath the massive data is receiving increasingly attention by companies and enterprises. Companies can discover certain patterns lie in the data with proper analytical methods, which can help them serve their customers better and make targets clearer, and result in the improvement of financial interests essentially.

However, it is a very hard problem to do analytics on big data. Decision makers have to put up with the slow response for their queries on big data, even if they are executed on distributed systems. This would make information in the data unlikely to be fully discovered, and result in the loss of potential business chances. In fact, decision makers do not require accurate answers to the queries: an approximate answer is enough for them. With respect to this phenomena, we designed a system that can give approximate answers of queries in a short time on big data.

Our contributions are:

First, we proposed an interactive big data analyzing system based on statistic inference. We combined sampling with Map Reduce in this system, which sacrifices accuracy for low latency.

Second, we pointed out a case in which random sampling would not work, and suggested that stratified sampling should be adopted in order to solve this case.

Third, we gave formula for estimating COUNT, SUM and AVG functions and their confidence intervals and corresponding algorithms.

Last, we provide evaluation for the entire system, which shows that the system is efficient enough for interactive queries, and stratified sampling yield better accuracy over random sampling.

**Key Words:** Big data, Business intelligence, Statistics inference

## 目录

摘要 .....	1
Abstract .....	2
1. 绪论 .....	5
1.1 研究背景 .....	5
1.1.1 大数据 .....	5
1.1.2 联机分析处理（OLAP） .....	6
1.2 本文的研究目的 .....	7
1.3 保密声明 .....	7
2. 传统方法及其面临的挑战 .....	8
2.1 传统方法 .....	8
2.1.1 OLAP 中的数据模型介绍 .....	8
2.1.2 Map-Reduce 方法 .....	9
2.1.3 数据压缩 .....	10
2.2 传统方法面临的挑战 .....	11
2.2.1 传统的 Map-Reduce 方法面临的挑战 .....	11
2.2.2 传统的数据压缩方法面临的问题 .....	12
2.3 本文提出的采样推断系统 .....	14
3. 验证环节解决方案简述 .....	15
3.1 系统采用的大数据处理框架简述 .....	15
3.2 Hadoop 大数据处理框架 .....	15
3.3 HDFS 分布式存储系统 .....	15
3.4 Spark 分布式大数据计算系统 .....	16
4. 采样推断算法原理 .....	18
4.1 采样推断模块简述 .....	18
4.2 采样推断的统计学原理 .....	18

## 北京工业大学毕业设计（论文）

---

4.3	采样推断算法的问题描述.....	19
4.4	采样推断算法设计 .....	19
4.4.1	Count 聚合函数 .....	19
4.4.2	Avg 聚合函数 .....	20
4.4.3	Sum 聚合函数 .....	24
5.	分层采样算法.....	26
5.1	分层采样综述.....	26
5.2	分层采样算法的设计 .....	27
5.3	对推断算法的修正 .....	27
5.3.1	对 Count 聚合函数的修正 .....	28
5.3.2	对 Sum 聚合函数的修正 .....	28
5.3.3	对 Avg 聚合函数的修正 .....	28
6.	系统设计与实现.....	30
6.1	系统架构设计 .....	30
6.2	服务器部分详细设计 .....	30
6.3	实际系统界面展示 .....	31
7.	实验结果与分析.....	34
7.1	实验环境.....	34
7.2	实验项目介绍.....	34
7.3	性能测试.....	34
7.4	准确度测试.....	35
7.5	实验结论 .....	37
	结论.....	38
	致谢.....	39
	参考文献.....	40

# 1. 绪论

## 1.1 研究背景

### 1.1.1 大数据

如今，我们生活在一个“数据驱动”的时代中，无论是政府机关，学校，科研院所，还是各种企业，商业集团，都在产生大量的数据，比如，在气象、地理等科研领域，无处不在的传感器就在每时每刻产生大量的监测数据；医疗行业中的各种监测仪器也是数据的重要生产者；而在互联网领域，数据的获取更加容易，搜索引擎网站上用户的每一次搜索记录，购物网站中用户的每一次购买行为，对于网站背后的企业来讲都是重要的用户数据。举例来说，Facebook 需要储存，读取以及处理至少 30PB 的用户数据；沃尔玛每小时要处理超过百万次的顾客交易事务，这些事务如果导入到数据库中，估计大小会超过 2.5PB<sup>[1]</sup>；早在 2008 年，Google 每天就要处理 20PB 的数据<sup>[2]</sup>。并且美国互联网数据指出，全世界所有公司的商业数据每 1.2 年将翻一倍，并且有 90% 的数据是最近几年才产生的<sup>[3]</sup>。

所谓“大数据”的概念，就是在这样的背景下产生的。麦肯锡在其报告中指出，“大数据”是指那些大小远远超出传统规模，一般的数据处理软件难以完成存储、管理和分析的数据集。IBM 认为“大数据”有“3V”的特点，即 3 个以 V 字母开头的英文单词，即规模性（Volume）、多样性（Variety）和高速性（Velocity）。规模性指大数据的体积很大，通常来讲计量大数据的单位是 TB 和 PB；多样性指大数据来源广泛，如网络日志、传感器数据、视频、音频、地理位置信息等；高速性指大数据对数据实时处理能力要求极高，否则其价值将大打折扣<sup>[4]</sup>。

大数据的意义并不在于它的体积本身，而在于其中蕴含的信息。最经典的可能就是沃尔玛的“啤酒与尿布”的例子，通过对顾客历史购买记录的分析，沃尔玛发现男性顾客总是同时购买啤酒与尿布，于是该超市将啤酒与尿布放在了货架相近的位置上，结果大大增加了二者的销量；亚马逊有 20% 的销售量是来源于推荐系统的，而推荐系统的建立离不开对海量用户购物数据的分析；除了商业价值以外，大数据对于研究人员也有很大意义。例如微博，Facebook 这样的社交网络的数据对于社会学家来讲意义重大，他们可以在这些数据的基础上分析人类的行为模式、交往方式等<sup>[5]</sup>。

由此可见，获取大数据只是第一步，能否做好对大数据的分析才是真正决定一个企业或组织最后成功与否的能力。近年以来，全世界都在对大数据分析投入大量精力进行研究。美国在 2012 年就开始着手大数据研究，奥巴马在同年投入 2 亿美金在大数据的开发中，更强调大数据会是“未来的石油”。大数据也促进了对数据科学家和信息管理专家的需求。甲骨文，IBM，微软，惠普和戴尔等公司在软件智能管理与数据分析领域已经投入了超

过 150 亿美元。在 2010 年, 该行业本身就已经价值 1000 亿美元, 并且在以每年 10% 的速度增长, 这几乎是整个软件行业增长速度的两倍<sup>[6]</sup>。

### 1.1.2 联机分析处理（OLAP）

在计算机领域中, 联机分析处理（Online Analytical Processing, OLAP）是一种支持对数据集进行快速复杂分析操作的系统, 是现在商业智能当中重要的一环。概括来讲, OLAP 系统可以根据分析人员的要求快速、灵活地进行大数据量的复杂查询处理, 并且以一种直观而易懂的形式将查询结果提供给决策人员, 以便他们准确掌握企业的经营状况, 了解对象的需求, 制定正确的方案。OLAP 系统的经典应用包括在销售领域, 市场领域以及管理领域的汇总报表编写, 预算估计以及走势预测等等<sup>[7]</sup>。

与重视事务的传统数据库不同的是, OLAP 系统侧重于对数据的分析, 其主要作用是支持企业中决策人员和高层管理人员的决策。在实际情况中, 人们往往需要对数据进行多个方面的综合观察, 并且每次观察的指标可能不止一个, 并且还可能需要挖掘这些结果之间的关系。比如, 一个汽车厂商的市场部门经理可能需要从一个汽车销售数据库中了解在过去的几年里, 该公司的 SUV 类型汽车在中国和美国的销售趋势对比情况, 以便于制定接下来针对这两个市场的策略。这就是一个典型的决策问题, 决策者需要的数据总是与一些限制条件有关（比如品牌, 时间, 国家, 地区, 类型等等）, 这些限制条件是多维度的, 用户需要的是在这些维度上对统计指标进行观察, 从而得出结论, 这种功能是传统数据库很难做好的<sup>[8]</sup>。

根据实际的情况以及人们对 OLAP 系统的实际需求, 人们给 OLAP 系统提出了一个更加简单明确的定义, 即“共享多维信息的快速分析系统”。OLAP 主要有以下的一些特点:

- 交互性: 用户对 OLAP 的反应速度有很强的要求, 一般来讲系统应该在 5 秒内对用户的分析要求做出响应。
- 可分析性: OLAP 系统应该能处理任何与该系统逻辑一致的数据, 并且用户通常无需编程就可以对数据进行分析, 同时 OLAP 应该能够提供友好的结果界面给用户, 使用户无需专门知识就能理解分析结果。
- 多维性: OLAP 系统必须提供对多维数据分析的支持, 包括层次维和多层次维的支持。多维分析是分析企业数据最有效的方法, 因此这个特点是 OLAP 系统的灵魂。
- 信息性: 无论数据量有多大, 也无论数据存储在哪里, OLAP 系统都应该能及时获取信息, 并能管理大容量信息。

由于 OLAP 系统是商业智能的重要组成部分, 很多公司也在 OLAP 系统的研发上投入大量精力。该领域当中比较有名的产品有 IBM 的 Cognos, 微软的 Power BI 等等。据统计, 微软, 甲骨文, IBM, SAP 等公司在 OLAP 系统上的总计收入达到了 50 亿美元<sup>[9-11]</sup>。

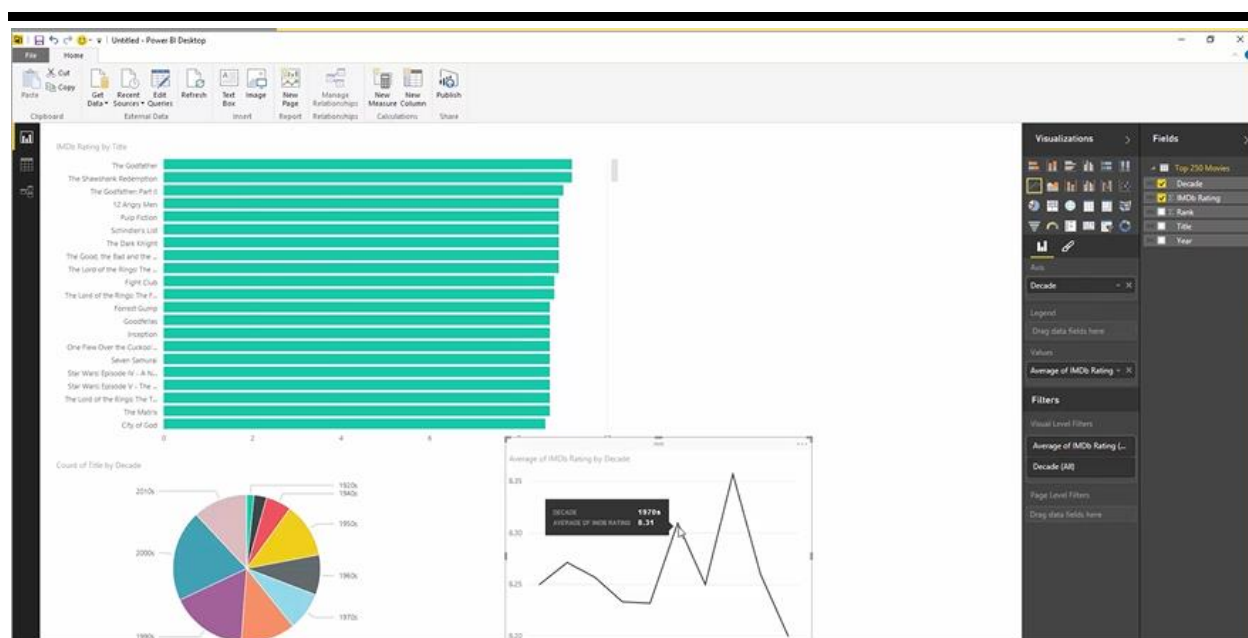


图 1 微软的 Power BI 商业数据分析系统

## 1.2 本文的研究目的

本文旨在设计一套针对大数据的 OLAP 系统，用户可以在对数据经过一定的预处理之后，对数据进行多维度的探索。所谓“探索”，就是一系列类似于 SQL 的操作。用户可以在使用过程中，交互地向应用提出针对数据的询问，并且观察到应用返回的结果。

现有的处理大数据的 OLAP 系统的核心方法主要有两种：**Map-Reduce** 方法和数据压缩方法。在下一章中我们将详细介绍这两种方法，以及这两种方法在本系统面临的情境中不适用的原因。概括来讲，**Map-Reduce** 方法得到的结果十分准确，但是时间消耗太大；数据压缩方法虽然可以快速得到结果，但是要么会带来更大的预处理代价，要么不能保证结果的准确性。因此本文所构建的系统旨在将二者的优点结合起来，首先利用数据压缩方法，使用户能够快速看到一个大致精确的结果，然后用户可以选择其感兴趣的查询使用 **Map-Reduce** 方法进行查询，验证其猜想的结论。这与决策者们的需求是吻合的，决策者们可以首先快速地发现他们感兴趣的查询结果，然后通过大数据上的查询进一步验证其猜想，从而得出结论。前期的数据压缩成果也可以用于各种数据挖掘算法，从而达到自动挖掘出数据当中潜在规律的目的，决策者可以从中选择其感兴趣的结论在大数据上进行验证。

## 1.3 保密声明

该论文是作者在微软亚洲研究院软件分析组实习过程中完成的。整个系统的版权归微软公司所有，部分数据因为涉及机密信息进行了一定的屏蔽。



## 2. 传统方法及其面临的挑战

### 2.1 传统方法

#### 2.1.1 OLAP 中的数据模型介绍

在传统的 OLAP 系统中，数据是以“数据方块”（Data Cube）的形式存储的。每个数据方块的列分为两种：

- 指标（Measure）：指标可以理解为一个数据的“值”，是真正统计的时候需要观察的量。例如，在一份销售表格里面，“销量”就很有可能是指标。
- 维度（Dimension）：维度描述了数据的类别，用户通过维度来筛选数据，选择自己需要观察的方向。例如在一份汽车销售表里面，“国家”，“品牌”，“日期”一般来讲都是维度。

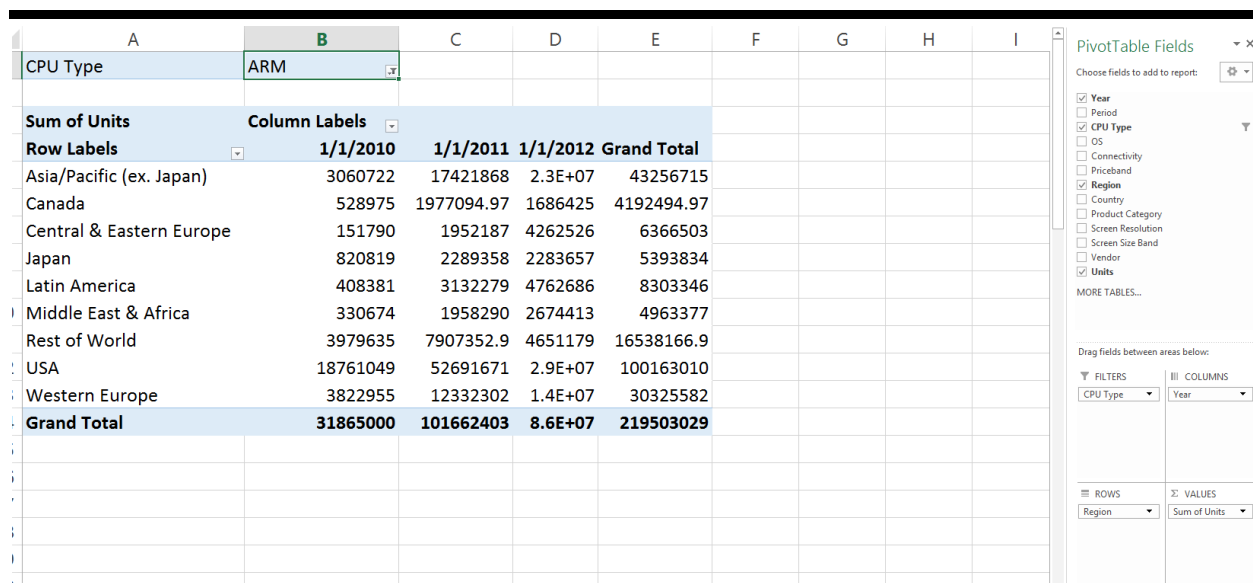
一个典型的 Data Cube 如下表所示：

表 1 一个典型的 Data Cube

年份	品牌	类别	型号	销量
2007	BMW	Compact	BMW-3 Series	142490
2006	Ford	FullSize	Fusion	9704
2007	Ford	Sporty	Mustang	72711
2007	Honda	MidSize	Accord	302014
2008	Honda	SUV	Element	35218

一份数据当中往往有很多个维度，这些维度合并在一起构成了一个维度的向量。用户通常会在一些维度上做筛选，在另一些维度上做分组，并且指定对指标的一种聚合函数，从而观察数据在特定条件下的分布特征。例如，SQL 查询“SELECT SUM(Sales) WHERE Country= ‘USA’ GROUP BY Date”就指定了 USA 作为 Country 维度上的筛选条件，Date 作为分组列，而 Sales 是指标，指定的聚合函数是 SUM，即求和。

一般的 OLAP 软件界面都是矩阵式的，由用户一一指定筛选条件，指标，分组列以及聚合函数。微软 Office 里面的 Excel 套件中有一个“Pivot table”的功能，可以理解为一个简单的 OLAP 系统。



Sum of Units	Column Labels				
Row Labels	1/1/2010	1/1/2011	1/1/2012	Grand Total	
Asia/Pacific (ex. Japan)	3060722	17421868	2.3E+07	43256715	
Canada	528975	1977094.97	1686425	4192494.97	
Central & Eastern Europe	151790	1952187	4262526	6366503	
Japan	820819	2289358	2283657	5393834	
Latin America	408381	3132279	4762686	8303346	
Middle East & Africa	330674	1958290	2674413	4963377	
Rest of World	3979635	7907352.9	4651179	16538166.9	
USA	18761049	52691671	2.9E+07	100163010	
Western Europe	3822955	12332302	1.4E+07	30325582	
<b>Grand Total</b>	<b>31865000</b>	<b>101662403</b>	<b>8.6E+07</b>	<b>219503029</b>	

图 2 Microsoft Excel 当中的 Pivot Table 组件

## 2.1.2 Map-Reduce 方法

Map-Reduce 是 Google 提出的一套针对大数据的处理算法框架，该算法通过多台机器组成的集群实现对大数据的并行计算，从而大大减少了对大数据的运算耗时。

Map-Reduce 会首先将输入数据分割，并且分布式地存储在多台机器上。这样，每台机器在大部分时间只要处理自己本地的数据即可，减少了很大一部分的网络传输数据的时间。

每个 Map-Reduce 程序主要分为 3 个主要步骤：

- **Map:** 在 Map 步骤，程序对数据进行筛选，排序等操作，将原有的一个键值对的列表映射为一个新的键值对的列表。例如“从一个学生列表里面筛选出所有分数大于 20 的学生”就是一种 Map 操作，映射以后产生了一个新的列表，这个列表里面只有分数大于 20 的学生；在 Map 步骤之后，所有的机器都会将结果暂时写到一个地方保存起来。
- **Shuffle:** 所有的机器根据上一步产生的结果，将数据重新分配，使得有相同键值的数据都分配到同一台机器上；
- **Reduce:** 在 Reduce 步骤里，程序会将具有相同键值的数据进行合并，例如“将所有班级相同的学生的分数相加”就是一种 Reduce 操作。

用 Map-Reduce 框架编写的程序是可以高度并行的，这是因为 Map 函数是对原列表中每一个元素进行的独立操作，而这些操作并没有改变原数据，它们只是产生了一系列新的数据，因此也就不存在冲突的问题；而 Reduce 操作因为要对一个列表进行操作，所以它的并行度并没有 Map 那么好，但是在 Map-Reduce 中，每台机器只处理数据的一部分，因此要处理的数据量并不大，所以 Reduce 并不会很大程度上影响整个算法的性能。

### 2.1.3 数据压缩

针对大数据当中数据量巨大，难以存储以及查询时间过长的问題，有文章提出了通过牺牲精度来压缩数据，从而达到减少数据量，进而加快查询速度的方法。

研究人员认为，在 OLAP 系统中决策者们往往不需要完全精确的答案。比如，人们只需要知道“2012 年的总销售额是大概 5 亿美元”，而不需要知道“2012 年的总销售额是 500,011,425.67 美元”。因此，很多有损压缩原数据的方法被提了出来，通过这些方法处理过的数据往往可以直接放到一台机器的内存中，这样就极大地加快了查询速度。同时，这些方法也提供一些手段来估计正确的答案，并且有一些还提供控制误差的手段。

通常的数据压缩方法有下面几种：

#### 2.1.3.1 采样 (Sampling)

采样就是从原数据中随机挑选一些条目来组成一个新的小数据集，之后的查询是在这个小样本数据集上完成的。根据一些统计学的规律，小样本数据集上的结果可以用来估计真实的结果。

例如我们有一个 10 个元素的数据集：{3, 4, 5, 6, 9, 10, 12, 13, 15, 19}，我们希望求这 10 个元素的和，如果事先我们对其采样，获得一个 5 个元素的样本 {10, 5, 9, 5, 15}，那么我们就可以通过计算这个样本的和： $Sum_{sample} = 10 + 5 + 9 + 5 + 15 = 44$  来估计真正的和：

$$Sum_{estimate} = \frac{Sum_{sample}}{SampleRate} = \frac{44}{0.5} = 88 \quad (2-1)$$

事实上，真正的原数据的和是 96，这个例子中样本的和很好地估计了原数据的总和。当然，采样统计的理论基础不是这么简单的，本文将在第 4 部分深入探讨采样的统计学原理。

#### 2.1.3.2 直方图 (Histograms)

直方图原本是一种统计学上的图表，用于统计数据落在不同区段内的频数，但它也可以用来估计和原数据相关的统计量。例如，我们如果有如下数据：{ 1.61, 1.72, 2.23, 2.33, 2.71, 2.90, 3.41, 4.21, 4.70, 4.82, 4.85, 4.91 }，画出该数据的直方图，其大致如下所示：

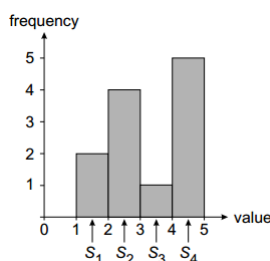


图 3 一维直方图示例

图中，横轴代表值，纵轴代表原数据落在相应值区段中的频数， $S_1, S_2, S_3, S_4$  分别代表分出来的 4 个区间。例如，原数据中落在 1-2 之间的值有 2 个，所以  $S_1$  区间的条长度为 2。

现在，如果用户询问“大于等于 1.1，且小于等于 4.5 的值在原数据里有多少个？”我们就可以利用这幅直方图来回答了。我们会发现这个询问区间段完全覆盖了  $S_2$  和  $S_3$  两个区间，因此我们可以将  $S_2$  和  $S_3$  的频数都加起来；然而  $S_1$  和  $S_4$  部分与查询区间部分相交，因此不能全都加起来。在直方图中有一个基本假设，就是“平均分配假设”，该假设认为在每一个小区段中，数据都是均匀分布的。因此，查询区间与  $S_1$  区间的相交部分占整个  $S_1$  区间的比例为  $\frac{2.0-1.1}{2.0-1.0} = 0.9$ ，所以我们认为  $S_1$  区间中大约有  $2 \times 0.9 = 1.8$  个符合条件的记录；同样的，查询区间占  $S_2$  区间的比例为  $\frac{5.0-4.5}{5.0-4.0} = 0.5$ ，因此我们认为  $S_4$  中符合条件的记录条数为  $5 \times 0.5 = 2.5$ ，那么该查询的估计答案为  $1.8 + 4 + 1 + 2.5 = 9.3$ 。实际上，正确的答案是 8。这个简单的例子说明了直方图是如何估测原数据的统计量的。

近年以来，人们也提出了很多改进的直方图算法，例如动态地决定分区间的长度，多维上的直方图等等。

### 2.1.3.3 小波分析 (Wavelets)

小波分析最初是应用于数字信号处理领域的，但是近几年也有人将其应用到数据压缩上。小波分析的基本原理是在原数据上执行小波变换，从而将原数据压缩成一系列的“小波系数” (wavelet coefficients)。在生成小波系数的时候，算法会将临近的值相近的记录合并为同一条记录。执行查询的时候，通过这些小波系数来还原出原先的数据，从而回答询问。

## 2.2 传统方法面临的挑战

### 2.2.1 传统的 Map-Reduce 方法面临的挑战

在商业智能的应用场景当中，数据分析师面临的是一份完全未经处理的数据，这样的数据可能是很高维度的，也是没有任何线索的。数据分析师想要从这份数据当中获取有用的信息的话，就必定要对数据进行“探索”，就是说他要对数据进行不定向的分析。与那种批处理式的查询不同，这种探索式的查询是完全随机的，数据分析师可以任意地指定筛选条件，分组列，要查看的指标等等。而且由于他对数据内部隐含的知识一无所知，所以他需要很快地知道一个查询的结果，这样他才能断定这个结果他是否感兴趣，因此对于这种应用场景来说：

- 必须支持对数据的随机提问
- 必须能在可交互的时间内给出答案，通常该时间是 3-5 秒。

而传统的 Map-Reduce 方法是针对大数据的处理方法，其处理时间是以小时计的。即使是处理很小规模的数据，由于该算法也需要对数据进行分割，执行 Map 函数和 Reduce 函数，所以其速度也是分钟级别的，这可能还没有直接执行相应的查询快。相比于单个小数据的处理速度，Map-Reduce 解决的是大数据是否能在一个可以容忍的时间之内计算的问题。因此，我们很难想象，一个数据分析师提出一个查询之后要等好几个小时才能看到结果，而且他完全不知道这个结果对他是否有意义；如果他等来的结果并不能使他发现任何有用的结论的话，这个时间就白白浪费了。

并且，正如上文所说，对于数据分析师而言，他并不需要完全精确的数据，因此 Map-Reduce 花费如此长的时间计算出的精确结果，可能还不如一个花 3 秒钟估计出的，和真正答案相差 10% 的数据有意义。所以，我们不能仅仅使用 Map-Reduce 作为我们处理大数据的算法。

### 2.2.2 传统的数据压缩方法面临的问题

在上文中，我们介绍了几种数据压缩的方法，它们都可以极大减小原数据的比例，同时还能够估计原数据的某些统计量，比如和、平均数等等。可惜的是，以上的每一种方法都有其不能解决的问题。

#### 2.2.2.1 采样面临的问题

采样面临的最严重的问题是丢组问题。

丢组问题指的是如果在分组列（即进行 GROUP BY 操作的列）上，各个值的分布严重不均的时候，样本可能会丢失其中的一些组别。举例来讲，考虑下面的数据集：

表 2 一张数据分布不均的表

ID	城市	年龄	薪水
1	New York	22	50,000
2	Ann Arbor	25	120,222
3	New York	23	73,240
4	New York	67	34,342
5	New York	34	96,034
6	Ann Arbor	55	73,920

如果我们从该表中采样 1/3 的数据，即 2 条数据，我们很有可能采样到的两条数据全都是 New York 的；从而，如果有用户提问“每个城市的平均薪资是多少？”的时候，该数据提供的答案就会彻底漏掉 Ann Arbor 的数据，因为在样本中，根本就不知道原数据中有 Ann Arbor 这个城市存在。这个问题由于每条记录有自己的指标值，所以并不能忽视，因为有可能被漏掉的维度上，其指标值可能很大。比如在这个数据集中，Ann Arbor 的平

均薪水是 97,071，而 New York 的平均薪资是 63,404，Ann Arbor 的平均薪资实际上比 New York 还大的。如果在估计当中漏掉这么重要的信息的话，很容易使用户发生误判。

#### 2.2.2.2 直方图面临的问题

直方图面临的问题在于在高维度下，其计算复杂度非常高。

正如上文所说，直方图的一个基本假设是落在每一个区间内的值视为均匀分布的。那么，一个很直接的想法就是尽可能通过合理划分区间，使得每个区间内的值真的接近均匀分布。由于最后压缩后的数据实际上保存的就是区间以及每个区间内的频数，所以划分的区间多少就决定了压缩之后的数据体积。这样，求这个直方图的问题实际上就变成了一个组合优化问题：给定最大能划分的区间数  $M$ ，求这  $M$  个区间，使得每个区间之内的数据尽可能平均分布。这个平均分布可以用 SSE (Sum Square Error, 均方误差) 等指标衡量。我们不妨称这个问题为最佳直方图问题。

在一维情况下，求最佳直方图是不困难的，只要利用动态规划算法，就可以在  $O(NM)$  的复杂度下解决，其中  $N$  是原数据的数量。在此基础上，如果不追求全局最优解，那么利用贪心的近似算法，可以在  $O(N)$  的时间复杂度之内解决问题。但是，在高维情况下，哪怕是二维，最佳直方图问题也已经被证明是一个 NP-Hard 问题，也就是说不存在多项式级别的算法可以求出一个高维数据的最佳直方图。虽然在此基础上，人们提出了种种改进的高维直方图，例如 MHIST, STHoles, HBH, GHBH 等等，这些直方图无一例外地对分割数据的方式提出了限制，例如 GHBH 要求每一个新的区间必须将恰好一个现有区间分成两份，并且分割点只能在区间边缘的  $k$  分位点上等等。但是，这仍然改变不了其复杂度过高的问题：即使是复杂度最低的 GHBH，其复杂度也达到了  $O(dNM)$ ，其中  $d$  是数据的维数。在大数据中，通常  $N$  会非常大，其数量级在十亿以上； $M$  是由压缩之后的数据大小限制决定的，通常在十万左右，这个复杂度是完全不可接受的。

#### 2.2.2.3 小波变换面临的问题

小波变换面临的最大问题就是在数据稀疏的时候，其空间复杂度非常高，并且有可能引起额外的 I/O 消耗。

所谓“数据稀疏”，指的是数据点的数量相对于整个空间的体积非常少。例如，考虑如下三维的例子：

表 3 一张数据稀疏的表

A 列	B 列	C 列	值
A1	B1	C1	5
A2	B2	C1	3
A1	B1	C2	8

这个表中，A 列，B 列，C 列各自有 2 种可能的取值，从而整个维度空间总共有  $2^3 = 8$  种可能的数据点。但是，在这个数据集中，数据点只有 3 个，这就是一种稀疏的数据分布。

与上面两种方法不同的是，小波变换不能直接在这种关系数据模型上操作，也就是说，小波变换必须将不存在的数据点认为值为 0 才能工作。上面的两种方法对于不存在的数据点是不敏感的——它们不需要显式地将这些数据点赋值为 0，但是小波变换需要。这带来的后果就是在高维数据稀疏的情况下，小波变换往往需要多次从硬盘读取文件的一部分，进行处理以后再写回硬盘。这样就引起了很大的磁盘 I/O 消耗，对于稀疏的数据来说非常浪费。而事实上，大数据的维数往往很高，而在高维的情况下，原始数据十分倾向于稀疏分布，小波变换并不适用于处理这样的数据。

### 2.3 本文提出的采样推断系统

由上文可见，无论是现有的能直接处理大数据的 Map-Reduce 方法，还是各种数据压缩方法，都不能满足我们这套系统的需求。单独使用 Map-Reduce 方法，得到的结果十分准确，但是时间消耗太大；单独使用数据压缩方法，虽然可以快速得到结果，但是要么会带来更大的预处理代价，要么不能保证结果的准确性。

因此，我们提出将两种方法结合起来：先对原数据进行采样，获取原数据的一份样本，之后用户的探索式查询实际上是在这份样本上做的。这样，我们可以保证在交互级别的时间内给出答案；之后，如果用户对探索的结果感兴趣，可以要求在大数据上进行验证，这样用户就可以有目标地执行大数据查询操作，节省了很大一部分盲目探索的时间。

综合起来，本文所构建的系统主要有以下的特点：

- 支持交互式地探索大数据，在交互环节内的询问能做到快速响应，同时支持在大数据上进行查询；
- 采用了一种改进的基于采样的数据压缩方法，能够在一定程度上解决上文所述的丢失组别的问题；
- 支持三种聚合函数：COUNT，SUM 以及 AVG，并且在大数据上能够给出估计真实值的置信区间。

## 3. 验证环节解决方案简述

### 3.1 系统采用的大数据处理框架简述

验证环节的模块需要处理的是一个针对大数据的类似 SQL 的查询，针对这样需求的解决方案已经相当成熟。本系统选用的是 Hadoop 分布式处理框架，其中大数据存储部分使用 HDFS 完成，对大数据的查询部分使用 Spark 完成。

### 3.2 Hadoop 大数据处理框架

Apache Hadoop 是一个开源的框架，用于大数据的分布式存储以及处理。值得注意的是，Hadoop 不是一个单独的软件，而是一整个大数据处理工具生态系统的统称。现在 Hadoop 生态当中已经有了各种各样的工具，例如 HDFS，MapReduce，YARN，ZooKeeper，Spark 等等。下图是一张目前 Hadoop 框架常用的工具的层次结构示意图。

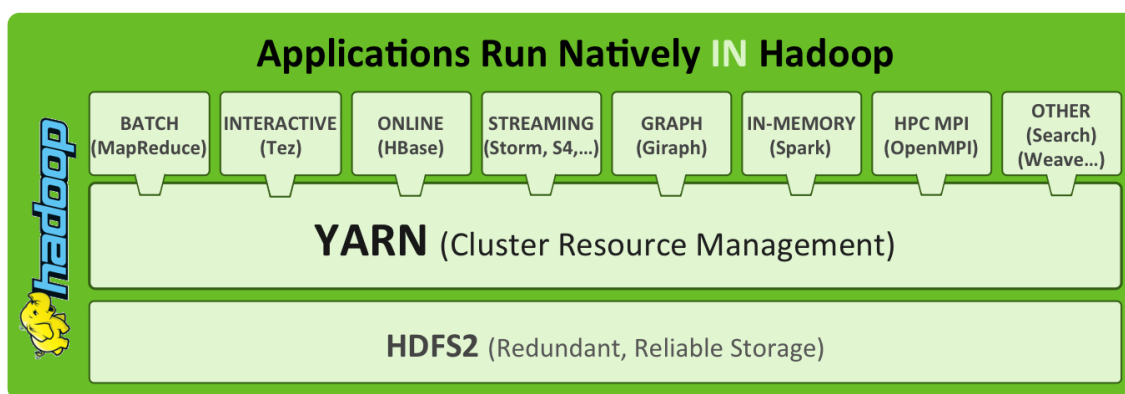


图 4 Hadoop 生态圈主要组成部分示意图

在本系统当中，我们主要使用 HDFS 和 Spark 两个组件来管理、分析大数据。

### 3.3 HDFS 分布式存储系统

HDFS (Hadoop Distributed File System, Hadoop 分布式文件系统) 是用于分布式存储大容量数据的一套文件系统，是 Hadoop 框架的数据管理层，负责大规模的分布式文件存储，同时 HDFS 具有容灾性好，成本低廉等优点。

一个 HDFS 集群主要分为两部分：NameNode 和 DataNode。NameNode 负责管理整个集群文件的元数据，例如存储位置，权限等等；DataNode 负责存储文件块本身。

当文件被上传至 HDFS 集群的时候，其内容会被分块存储，一个块通常是 128MB 大小。每个文件块会有多个副本，并且多个副本会被放到不同的 DataNode 上。在每个



DataNode 上，这些文件块会被保存在本地的文件系统中。NameNode 会监视这些文件块，如果有 DataNode 崩溃导致文件块丢失的话，NameNode 就会再创建一份这一块的副本。

HDFS 的使用是十分简单的，它可以使用命令行访问，其中命令行的指令与 Linux 的文件系统指令非常相似，例如下面就是一些简单的文件操作命令：

表 4 HDFS 的基本命令行操作

命令	作用
<code>hdfs dfs -put path/to/file.txt hdfs://namenode/target/path</code>	将 file.txt 上传到 HDFS
<code>hdfs dfs -ls hdfs://namenode/target/path</code>	列出 namenode/target/path 下的目录与文件
<code>hdfs dfs -rm hdfs://namenode/target/path/file.txt</code>	从 HDFS 上删除 file.txt

除了命令行参数以外，HDFS 还提供了一套 Java 与 Scala 的 API，可以在程序中动态访问、处理文件。

在本系统中，我们直接使用 HDFS 作为文件系统。

## 3.4 Spark 分布式大数据计算系统

Apache Spark 是一个开源的集群运算框架，最初由加州大学伯克利分校的 AMPLab 开发，其在 Hadoop 生态系统中的地位与 MapReduce 相当，但是它的运行速度比 MapReduce 要快得多。这主要是因为 Spark 使用了内存内运算技术，能在数据尚未写入磁盘的时候就在内存当中进行运算。实验表明，Spark 在内存运行程序的速度能比 MapReduce 快上 100 倍。并且，Spark 支持用户把数据加载到集群的内存中，然后多次对其执行查询，非常适合机器学习等算法。

部署 Spark 的时候需要指定集群资源管理软件和文件系统，Spark 支持独立模式，Hadoop YARN 和 Apache Mesos 三种集群资源管理软件，文件系统上支持 HDFS，Cassandra 等文件系统。

Spark 的核心数据结构是 RDD（Resilient Distributed Dataset，弹性分布式数据集），RDD 是支持并行操作，有容错机制的数据集合。RDD 可以直接用外部系统的数据创建，比如直接导入 HDFS 上的数据源；也可以由现有的 RDD 通过转换操作而创建。RDD 有一套 Scala、Java 等语言的 API，编程起来十分简单。

在 Spark 核心之上，Spark 还有一些应用于特定场合的组件，对本系统而言最重要的是 Spark SQL。Spark SQL 是 Spark 核心之上的组件，它引入了一种新的抽象数据结构 DataFrame 用于表示结构化的数据。Spark SQL 提供了一套 DSL（Domain Specific

Language，特定领域语言）用于操作 DataFrame，这些 DSL 提供了和标准 SQL 相似的功能。Spark SQL 也支持直接使用 SQL 语句查询。

在本系统中，我们采用 Spark SQL 作为大数据查询的引擎。

## 4. 采样推断算法原理

### 4.1 采样推断模块简述

采样推断模块需要解决的问题主要是在获取一份原数据的样本，并且在样本上执行了某个查询之后，如何根据样本的信息推断出该查询在原数据的正确答案，并且给出置信区间这样一个问题。在采样推断模块的设计中，核心任务就是根据统计学当中的基本原理，推导出利用样本统计量估计总体统计量的公式。

### 4.2 采样推断的统计学原理

采样推断模块主要基于统计学中的中心极限定理进行算法设计。中心极限定理是 De Moivre 在 18 世纪首先提出的一组定理，这些定理说明，大量相互独立的随机变量，其均值的分布以正态分布为极限。这组定理是数理统计学和误差分析的理论基础，指出了大量随机变量之和近似服从正态分布的条件。

**定理 4.1**（独立同分布的中心极限定理）：设随机变量  $X_1, X_2, \dots, X_n, \dots$  相互独立，且服从同一分布，并且具有期望和方差： $E(X_i) = \mu, \text{Var}(X_i) = \sigma^2 > 0, i = 1, 2, \dots$ ，那么随机变量

$$Y_n = \frac{\sum_{i=1}^n X_i - n\mu}{\sqrt{n}\sigma} \quad (4-1)$$

的分布函数  $F_n(x)$  收敛到标准正态分布  $N(0,1)$  的分布函数，即对任意实数  $x$  满足

$$\lim_{n \rightarrow \infty} F_n(x) = \lim_{n \rightarrow \infty} P\{Y_n \leq x\} = \phi(x) \quad (4-2)$$

其中， $\phi(x)$  是标准正态分布  $N(0,1)$  的分布函数，即  $\phi(x) = \int_{-\infty}^x \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt$ 。

该定理表明，随机变量序列  $Y_1, Y_2, \dots, Y_n, \dots$  的分布函数序列  $\{F_n(x)\}$  的极限是  $\phi(x)$ 。

注意到  $E(\sum_{i=1}^n X_i) = n\mu$ ， $\text{Var}(\sum_{i=1}^n X_i) = n\sigma^2$ ，上式可改写为

$$Y_n = \frac{\sum_{i=1}^n X_i - E(\sum_{i=1}^n X_i)}{\sqrt{\text{Var}(\sum_{i=1}^n X_i)}} \quad (4-3)$$

从而 $Y_n$ 的期望是 0，方差是 1， $Y_n$ 是 $\sum_{i=1}^n X_i$ 的标准化的随机变量。由定理 1 知，当  $n$  很大时， $Y_n$ 近似服从于标准正态分布  $N(0, 1)$ ，从而当  $n$  很大时， $\sum_{i=1}^n X_i$ 近似服从于正态分布  $N(n\mu, n\sigma^2)$ 。由于 $X_i$ 的分布往往是未知的，所以 $\sum_{i=1}^n X_i$ 的精确分布往往是难以求得的。这时，只要  $n$  很大，就能通过正态分布来近似 $\sum_{i=1}^n X_i$ 的分布。

### 4.3 采样推断算法的问题描述

给定一个条数为  $n$  的样本数据集，样本数据集中的数据共分为  $G$  组，每条记录都有自己的权值；给定置信系数  $\alpha$ ，求条数为  $N$  的总体数据中，对于每个组别  $G_i$  的某种统计量  $A_i$  的估计上下界  $[u_l, u_r]$ ，使得  $P(u_l \leq A_i \leq u_r) \geq \alpha$ 。统计量  $A_i$  可以为：

- $A_i = \text{Count}(G_i)$ ，即第  $i$  组记录的出现条数。
- $A_i = \text{Sum}(G_i)$ ，即第  $i$  组记录的权值之和；
- $A_i = \text{Avg}(G_i)$ ，即第  $i$  组记录的权值的算术平均数；

### 4.4 采样推断算法设计

#### 4.4.1 Count 聚合函数

**定理 4.2**（Count 真实值的分布）设在小数据中，第  $i$  组数据出现了  $n_i$  条，那么大数据中第  $i$  组数据的条数  $N_i$  服从二项分布，即

$$N_i \sim B(N, p_i) \quad (4-4)$$

其中  $p_i = \frac{n_i}{n}$  为估计的第  $i$  组的出现概率。

证明：我们将大数据的  $N$  条数据视为从一个无穷总体当中做  $N$  次随机抽样所得的一个样本，其中  $p_i$  是第  $i$  组记录被抽中的概率，那么显然第  $i$  组的出现条数服从于二项分布  $B(N, p_i)$ ；由于  $p_i$  无法直接从大数据中获得，利用最大似然估计，我们可以从大数据的一个样本中统计第  $i$  组的频数  $n_i$ ，从而用频率  $\frac{n_i}{n}$  估计概率  $p_i$ 。

当我们已经求得  $N_i$  的分布之后，利用二分查找的方法就很容易求得上下界  $[u_l, u_r]$  了。若  $p_i$  很小，也可以将二项分布近似为泊松分布或正态分布，从而利用查表等方法更加简单地求得置信区间。

聚合函数为 Count 时，算法伪代码如下：

算法 4.1: 由样本估计总体 Count 聚合函数真实值

输入: 大数据条数  $N$ , 样本  $s$ , 置信度  $\alpha$

输出: 第  $i$  组的置信区间  $[u_l^i, u_r^i]$

算法:

对  $s$  中的每一组  $i$ :

统计  $i$  在  $s$  中的数量  $n_i$ ;

计算概率  $p_i = \frac{n_i}{s.size()}$ , 得到概率密度函数  $B(N, p_i)$ ;

利用二分查找或查表法计算得到  $u_l^i, u_r^i$ 。

#### 4.4.2 Avg 聚合函数

**定理 4.3** (样本中每组 Count 的分布函数): 设在大数据中, 第  $i$  组数据的出现概率为  $p_i$ , 那么在样本中, 第  $i$  组数据出现的数量  $n_i$  服从于二项分布  $B(n, p_i)$ 。

该定理的证明与定理 4.2 非常类似, 我们将采样过程视为从大数据中抽取  $n$  条记录。由于总体数量  $N$  非常大, 可以视为无穷总体, 那么采样这种无放回抽样也可近似为有放回抽样处理。那么第  $i$  组数据出现的数量  $n_i$  就服从于二项分布  $B(n, p_i)$ 。

定理 4.3 是十分重要的定理, 因为要想利用中心极限定理来估计总体的统计量的话, 必须要有“样本数量”这个参数。而值得注意的是, 在本系统面临的情境中, 每一组的样本数量实际上是一个随机变量, 也就是该定理所指出的服从于二项分布。我们不能简单地将样本中第  $i$  组的出现条数认为就是中心极限定理中所指出的样本数量。这一点在接下来的推导中也将得到更加深入的阐释; 事实上我们会发现, 将每组的样本数量视为随机变量, 和用样本当中的出现次数直接去估计样本数量, 最后得到的分布函数是不一样的。但是, 我们可以用  $\frac{n_i}{n}$  去估计第  $i$  组在大数据中的出现频率  $p_i$ 。

**定理 4.4** (Avg 真实值的近似分布性质): 当样本容量足够大的时候, 每组的 Avg 真实值近似服从于正态分布。

证明: 设  $\bar{V}_i$  是大数据中第  $i$  组权值的均值,  $\sigma_i^2$  是大数据中第  $i$  组权值的方差,  $n_i$  为给定的第  $i$  组在样本当中的出现次数, 那么假设样本容量足够大, 根据中心极限定理, 有如下结论成立:

$$Avg_i \sim N\left(\bar{V}_i, \frac{\sigma_i^2}{n_i}\right) \quad (4-5)$$

注意到  $Avg_i$  是一个随机变量, 因为当我们进行多次采样的时候,  $Avg_i$  每次都是有可能不同的。我们用  $p(Avg_i)$  表示  $Avg_i$  的概率密度函数, 我们已经求出的是给定  $n_i$  情况下的  $p(Avg_i)$ , 即

$$p(Avg_i | n_i) = N\left(\bar{V}_i, \frac{\sigma_i^2}{n_i}\right) \quad (4-6)$$

注意到对于任意随机变量  $x, n$ ,  $p(x) = \sum_n p(x|n) \cdot p(n)$ , 类似地, 我们可以得出

$$p(Avg_i) = \sum_{n_i} p(Avg_i|n_i) \cdot p(n_i) \quad (4-7)$$

根据定理 4.3，联立（4-6）式，将上式右边的两项等量代换，得到

$$p(Avg_i) = \sum_{n_i} N\left(\bar{V}_i, \frac{\sigma_i^2}{n_i}\right) \cdot B(n, p_i) \quad (4-8)$$

我们用  $\frac{n_i}{n}$  来估计  $p_i$ ，那么此式就会变成

$$p(Avg_i) = \sum_{n_i} N\left(\bar{V}_i, \frac{\sigma_i^2}{n_i}\right) \cdot B(n, \frac{n_i}{n}) \quad (4-9)$$

观察这个式子，我们会发现，当  $n_i$  给定的时候， $B(n, \frac{n_i}{n})$  是一个常数，那么右边可以看做是  $(n_i + 1)$  个（方差不同的）正态分布的和。而我们知道，正态分布具有如下性质：

- 一个正态分布与一个常数相乘，其结果仍是正态分布；
- 两个正态分布相加，其结果仍是正态分布。

因此， $p(Avg_i)$  仍然是一个正态分布，证明完毕。

为表征方便，在下面的推导中我们将  $Avg_i$  记为随机变量  $z$ ，不妨设  $z \sim N(\mu_z, \sigma_z^2)$ ，下面我们来求这两个参数。

首先让我们来求期望  $\mu_z$ 。我们可以做如下推导：

$$\mu_z = \int_{-\infty}^{+\infty} zp(z)dz = \int_{-\infty}^{+\infty} z \cdot \sum_{n_i} p(z|n_i)p(n_i)dz = \int_{-\infty}^{+\infty} z \cdot \sum_{n_i} p(z|n_i)B(n, p_i)dz$$

交换积分和求和的次序，我们得到

$$\mu_z = \sum_{n_i} p(n_i) \int_{-\infty}^{+\infty} zp(z|n_i) dz$$

注意到  $\int_{-\infty}^{+\infty} zp(z|n_i) dz$  实际上就是给定  $n_i$  时随机变量  $z$  的期望，我们记这个期望为  $n_i$  的函数，即  $\mu_z(n_i)$ 。根据（4-4）式， $\mu_z(n_i) = \bar{V}_i$ ，因此我们可以得出

$$\mu_z = \sum_{n_i=0}^n p(n_i)\mu_z(n_i) = \sum_{n_i=0}^n p(n_i)\bar{V}_i$$

由于 $\overline{v_i}$ 不好直接从大数据中求得，根据最大似然估计，我们使用小数据中第  $i$  组的平均值 $\overline{v_i}$ 来估计 $\overline{v_i}$ 。因此我们有：

$$\mu_z = \sum_{n_i=0}^n p(n_i) \overline{v_i}$$

由于 $\overline{v_i}$ 是常数，所以可得

$$\mu_z = \overline{v_i} \sum_{n_i=0}^n p(n_i) = \overline{v_i} \quad (4-10)$$

至此我们求得了 Avg 概率密度函数的期望，该期望就是小数据中第  $i$  组的权值的平均数。

下面我们来求方差 $\sigma_z^2$ 。

$$\begin{aligned} \sigma_z^2 &= \int_{-\infty}^{+\infty} p(z) \cdot (z - \mu_z)^2 dz \\ &= \int_{-\infty}^{+\infty} z^2 p(z) dz + \mu_z^2 \int_{-\infty}^{+\infty} p(z) dz - 2\mu_z \int_{-\infty}^{+\infty} zp(z) \cdot dz \end{aligned}$$

我们知道 $\int_{-\infty}^{+\infty} zp(z) dz = \mu_z$ ，将其代入上式，化简可得

$$\sigma_z^2 = \int_{-\infty}^{+\infty} z^2 p(z) dz - \mu_z^2$$

继续推导可得

$$\begin{aligned} \sigma_z^2 &= \int_{-\infty}^{+\infty} z^2 p(z) dz - \mu_z^2 = \int_{-\infty}^{+\infty} z^2 \cdot \sum_{n_i=0}^s p(z|n_i) p(n_i) dz - \mu_z^2 \\ &= \sum_{n_i=0}^s p(n_i) \int_{-\infty}^{+\infty} z^2 p(z|n_i) dz - \mu_z^2 \end{aligned}$$

注意到 $\int_{-\infty}^{+\infty} z^2 p(z|n_i) dz$ 实际上是给定 $n_i$ 时随机变量 $z^2$ 的期望：

$$\int_{-\infty}^{+\infty} z^2 p(z|n_i) dz = E_{n_i}(z^2)$$

而根据方差的性质，我们可以做如下变形：

$$E_{n_i}(z^2) = E_{n_i}(z)^2 + Var_{n_i}(z) = \mu_z(n_i)^2 + \sigma_z^2(n_i)$$

我们前面已经知道  $\mu_z(n_i) = \bar{V}_i \approx \bar{v}_i$ 。

根据（4-1）式，我们可以知道  $\sigma_z^2(n_i) = \frac{\sigma_i^2}{n_i}$ ，而  $\sigma_i^2$  是难以在大数据中计算的。同样，我们可以用最大似然的方法，利用样本方差  $S_i^2$  来估计总体方差。这样，我们就得到：

$$\int_{-\infty}^{+\infty} z^2 p(z|n_i) dz = u_z(n_i)^2 + \frac{S_i^2}{n_i}$$

所以我们会得到：

$$\begin{aligned} \sigma_z^2 &= \sum_{n_i'=0}^s p(n_i) \int_{-\infty}^{+\infty} z^2 p(z|n_i) dz - \mu_z^2 = \sum_{n_i=0}^n p(n_i) (u_z(n_i)^2 + \sigma_z^2(n_i)) - \mu_z^2 \\ &= \sum_{n_i=0}^n p(n_i) \left( \bar{v}_i^2 + \frac{S_i^2}{n_i} \right) - \mu_z^2 \end{aligned}$$

而  $\mu_z$  我们刚刚求得，根据（4-10）式：

$$\mu_z = \bar{v}_i$$

另外，根据定理 4.3，我们也有：

$$p(n_i) = B(n, p_i)$$

由这些我们就可以得到最终的表达式：

$$\sigma_z^2 = \sum_{n_i=0}^n B(n, p_i) \left( \bar{v}_i^2 + \frac{S_i^2}{n_i} \right) - \bar{v}_i^2 \quad (4-11)$$

至此，我们就导出了  $\sigma_z^2$  的表达式。

由这个表达式，我们也可以看到，将  $n_i$  视为常数与视为随机变量，虽然得到的结果都是正态分布且期望相同，但是视为随机变量的时候，方差要比视为常数时大。

下面是求 Avg 聚合函数的置信区间的算法伪代码：



算法 4.2: 由样本估计总体 Avg 聚合函数真实值

输入: 大数据条数  $N$ , 样本  $s$ , 置信度  $\alpha$

输出: 第  $i$  组的置信区间  $[u_l^i, u_r^i]$

算法:

对  $s$  中的每一组  $i$ :

求第  $i$  组的权值平均数  $\overline{v_i}$ , 并记  $\mu_i = \overline{v_i}$ ;

$\sigma_i^2 = 0$ ;

for  $n_i = 0$  to  $n$ :

$p_i = \frac{n_i}{n}$ ;

根据 (4-11) 式求和累加  $\sigma_i^2$ ;

end

$Avg_i = N(\mu_i, \sigma_i^2)$

查表得到  $u_l^i, u_r^i$ 。

#### 4.4.3 Sum 聚合函数

Sum 聚合函数与 Avg 聚合函数在本质上是相同的, 因此我们可以借用很多上一小节当中的结论来设计 Sum 函数的估计方法。

我们知道,  $Sum_i = N_i * Avg_i$ , 其中  $N_i$  是大数据中第  $i$  组的条目数, 而在上一节中我们已经得出

$$Avg_i \sim N(\mu_z, \sigma_z^2)$$

所以我们可以得到

$$Sum_i \sim N_i * N(\mu_z, \sigma_z^2) = N(N_i \mu_z, N_i^2 \sigma_z^2) \quad (4-12)$$

该方程中除了  $N_i$ , 其他参数我们都已经在上一节中导出了。而  $N_i$  我们可以通过以下公式估计:

$$N_i = n_i * \frac{N}{n} \quad (4-13)$$

因此, 我们可以很容易地写出求 Sum 的伪代码:

算法 4.3: 由样本估计总体 Sum 聚合函数真实值

输入: 大数据条数  $N$ , 样本  $s$ , 置信度  $\alpha$

输出: 第  $i$  组的置信区间  $[u_l^i, u_r^i]$

算法:

对  $s$  中的每一组  $i$ :

    利用算法 4.4.2 中的相关步骤, 求出  $\mu_z$  和  $\sigma_z^2$ ;

    求出该组在  $s$  中所占条数  $n_i$ , 估计出大数据中所占的条数  $N_i$ ;

    利用 (4-12) 式得出 Sum 的概率密度函数;

    查表得到  $u_l^i, u_r^i$ 。

## 5. 分层采样算法

### 5.1 分层采样综述

在上一节中，我们推导了使用一般的随机采样时估计大数据中某些统计量的计算方法，然而这种简单的随机采样对于一些数据效果会打很大的折扣，其中最重要的问题就是丢组的问题。回顾我们在论文第二部分当中举的例子：

表 5 一张数据分布不均的表

ID	城市	年龄	薪水
1	New York	22	50,000
2	Ann Arbor	25	120,222
3	New York	23	73,240
4	New York	67	34,342
5	New York	34	96,034
6	Ann Arbor	55	73,920

正如第二部分当中所讲的，一般的随机采样很容易漏掉数量很少的组，比如这里的“Ann Arbor”。这就会导致在样本当中根本就不会有“Ann Arbor”这样的记录出现，也就无法对这一组的数据进行估测，从而对估计结果带来很大的影响。

为了解决这一问题，我们可以使用分层采样的方法。所谓分层采样，就是在每组当中按比例进行随机抽样，这样就可以保证不丢失任何一个组别。例如，对于上面的这个例子，我们如果以“城市”为组，以 1/2 采样率进行采样的话，就应该从城市为“New York”的 4 条记录中随机抽出两条，再从城市为“Ann Arbor”的两条记录中随机抽取一条，而不是像简单随机抽样一样，从 6 条记录里面随机抽取 3 条。

分层采样方法的原理是非常简单的，但是使用分层采样算法时，要解决的最大的问题就是以哪一列或哪几列分组。通过上面的例子我们可以看出，如果一个分层采样的样本是以某个列组合  $S$  分组的话，那么询问任何  $S$  当中的列或者列组合时，该样本都能保证不丢组。同时，这份分层采样的样本也能处理关于  $S$  之外的列的询问，只是不丢组的性质就不能保证了。

事实上，目前业界最新的 BlinkDB 数据库系统进行数据压缩的原理就是分层采样[19]。BlinkDB 建立在已知对某个表的一些历史查询数据的情景上，该系统会通过历史数据的分析，找出用户最感兴趣的一个或多个列组合，并且将这些列组合分别认为是分层采样当中的组，然后关于这些组进行分层采样。但是在我们的情境中并没有历史数据，因此我们

无法从历史数据当中学习用户究竟是对哪些列感兴趣。我们也不能以列的全集为组，否则虽然能够在全体上保证不丢组，但是采样率会很高，几乎达不到数据压缩的目的。不过，BlinkDB 在其论文中指出，对于某张高维数据表，绝大部分的查询所涉及到的列都是固定的几列。他们调查了 Facebook 一周的查询历史记录，发现有 90% 的查询都落在了 20% 的列组合上，并且这些列组合即使随着时间的推移也是相对稳定的。因此，我们完全可以让用户自己给定自己相对感兴趣的列，这些列数并不会很多，然后我们以这些列为组进行分组采样。即使用户对于自己的选择在一段时间之内不甚满意，也可以重新指定另外的一些列重新进行采样，因为分层采样的复杂度虽然略高于一般随机采样，但是相对于其他数据压缩方法也非常低，因此对原数据进行多份采样也是可以接受的。

## 5.2 分层采样算法的设计

如上节所述，分层采样的首要任务是要指定一些列，将这些列视为组。指定列的工作在本系统中由用户完成。

其次是确定每组的采样率。虽然标准的分层采样是组内的采样率固定为总体的采样率，但是因为我们做的是统计推断，如果一个组的条目数过少的话，会影响推断的准确度。因此我们做如下设计：

- 若某一组预定的采样条数（即大数据条数 $\times$ 采样率）小于给定的阈值  $K$ ，那么该组固定采样  $K$  条，不足  $K$  条则全部保留；
- 否则，按采样率导出的样本条数进行采样。

## 5.3 对推断算法的修正

值得注意的是，采用这种分层采样的时候，就不是无偏采样而是有偏采样了，因为每组的采样率很可能是不同的。因此，我们必须做一定的修正，否则在估测的时候会对结果造成影响。

为了叙述方便，避免混淆，我们做如下约定：一个样本当中被视为分组标准的列组合以  $G$  表示，这个列组合的集合称为“预设列集合”， $g=1,2,3,\dots$  代表预设列集合当中的每一个组，称为“预设组”。这主要是为了和询问当中的分组区别开。

例如，下面是一个示例样本，这份样本是关于 A 列和 B 列采样的，那么 A 和 B 就叫做“预设列集合”，即  $G=\{A,B\}$ ；(A1, B1), (A1, B2), (A2, B1), (A2, B2) 分别就是四个“预设组”。在这份样本上，同时也可以处理关于 C 列的询问。

表 6 分层采样示例

A 列	B 列	C 列	值
A1	B1	C1	10
A1	B1	C2	20

A 列	B 列	C 列	值
A1	B2	C1	15
A2	B1	C3	30
A2	B2	C1	20

我们应当注意到，一条很重要的性质是对于每一个预设组  $g$ ，组内的抽样过程仍然是简单随机抽样过程。在本小节中，我们会对之前部分的定理做一些修正。

### 5.3.1 对 Count 聚合函数的修正

在本节中，我们主要修正定理 4.4.1.1 当中对第  $i$  组数据在大数据中的出现概率  $p_i$  的估计方法，其余推导部分完全同 4.4.1.1。

设在小数据中，第  $i$  个查询组的数据在第  $g$  个预设组中出现了  $n_{gi}$  条，大数据条目数为  $N$ ，同时已知第  $g$  个预设组的采样率为  $rate_g$ ，那么我们可以估计第  $i$  组在大数据中的出现概率为

$$p_i = \sum_g \frac{n_{gi}}{N} \cdot \frac{1}{rate_g} \quad (5-1)$$

这实际上就是对每个预设组内第  $i$  组出现概率的一种加权。

### 5.3.2 对 Sum 聚合函数的修正

设  $Sum_{gi}$  为第  $g$  个预设组中第  $i$  组的权值之和，我们可以注意到有如下的定理成立：

$$Sum_i = \sum_g Sum_{gi} \quad (5-2)$$

而每个预设组内的过程就是简单随机采样过程。我们在 4.4.3 中已经知道， $Sum_{gi}$  服从于正态分布，而根据正态分布的性质，两个正态分布之和还是正态分布，因此  $Sum_i$  也服从于正态分布。根据期望和方差的性质，由于随机变量  $Sum_{gi}$  互相独立，我们很容易求出  $Sum_i$  的期望和方差：

$$\mu_i = \sum_g \mu_{gi} \quad (5-3)$$

$$\sigma_i^2 = \sum_g \sigma_{gi}^2 \quad (5-4)$$

而每个预设组内的期望和方差的计算方法我们已经在 4.4.3 中推导过，这里就不再重复了。

### 5.3.3 对 Avg 聚合函数的修正

在上一节中我们已经推导出了修正好的 Sum 聚合函数的公式，我们知道

$$Sum_i \sim N(\mu_i, \sigma_i^2) \quad (5-5)$$

那么我们可以得出

$$Avg_i = \frac{Sum_i}{N_i} \sim N\left(\frac{\mu_i}{N_i}, \frac{\sigma_i^2}{N_i^2}\right) \quad (5-6)$$

我们只需用如下方法估计 $N_i$ :

$$N_i = \sum_g \frac{n_{gi}}{rate_i} \quad (5-7)$$

即可得出 $Avg_i$ 的分布，从而得到置信区间。

## 6. 系统设计与实现

### 6.1 系统架构设计

本系统主要有 3 个部分构成：前端界面，后端服务器，以及负责运算和存储的集群。系统整体架构图如下所示：

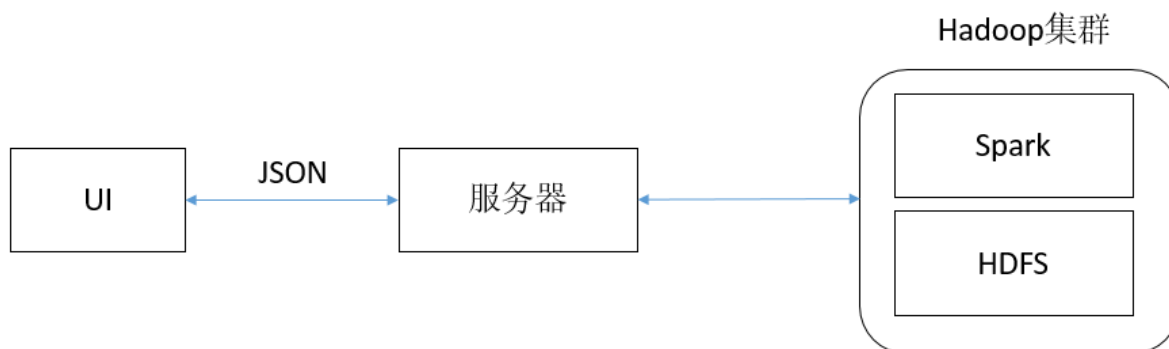


图 5 系统结构示意图

其中，UI 部分使用 C#实现，服务器部分使用 Scala 和 Java 混合实现，UI 和服务端之间通过 JSON 通信。本文的主要工作集中在后端，即服务器端和 Hadoop 集群这两个部分上。UI 部分不是本文的讨论重点。

### 6.2 服务器部分详细设计

服务器部分主要需要支持的功能如下：

- 能够新增以及删除数据集；
- 能够对已经导入的数据集的样本进行查询操作；
- 能够对已经导入的数据集进行大数据查询操作。

服务器部分主要的系统结构如下所示：

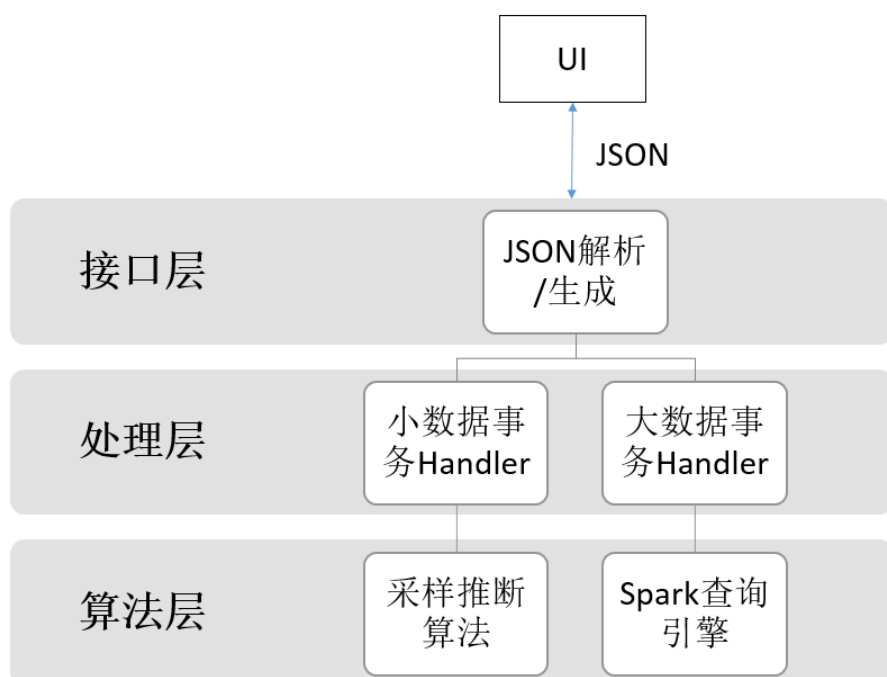


图 6 服务器部分结构示意图

如图，UI 发来的 JSON 请求将首先被接口层的 JSON 解析器解析，之后根据其请求类型交给相应的 Handler。其中，小数据事务 Handler 主要负责调用推断算法模块，处理针对采样进行的统计推断请求；大数据事务 Handler 主要负责调用 Spark 引擎处理新建采样请求和大数据验证请求。处理完成后再经由接口层将结果序列化为 JSON 返回 UI。

### 6.3 实际系统界面展示

以下是一些典型的查询结果界面截图，这些查询是在一个汽车销售数据的样本上进行的：

如下是一个普通的在样本上的查询，该查询以“Year”列分组，查询每年的销售量综合 Sales：



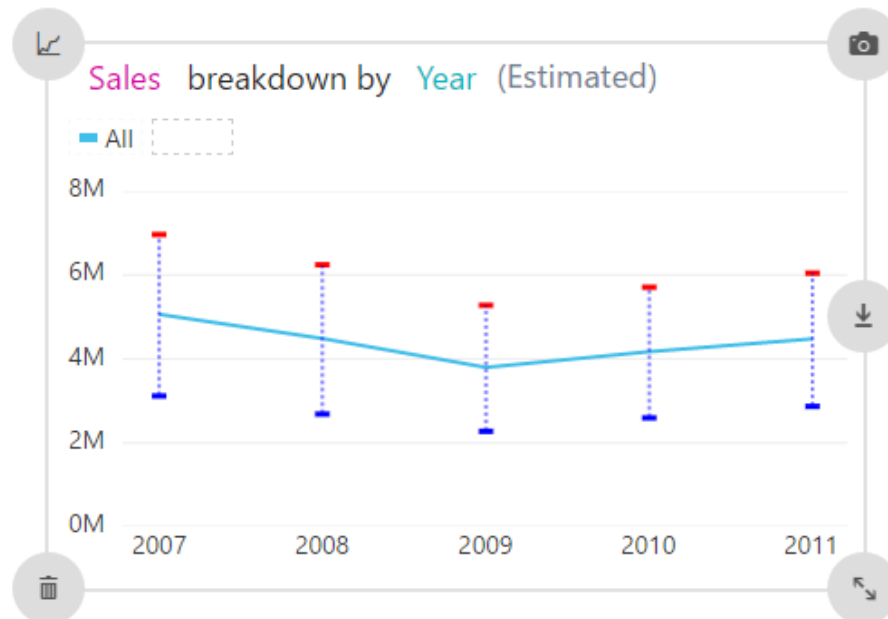


图 7 一个在样本上查询的界面显示

图中的红点为上界的指示标志，蓝点为下界的指示标志。中间的虚线范围就是系统估计的真实值所在区间。

也可以查看每个数据点的取值：

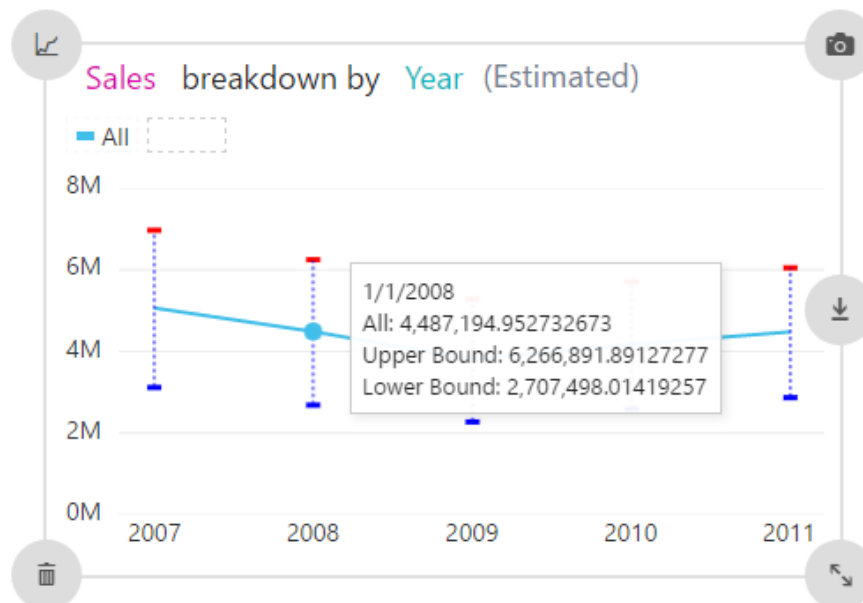


图 8 一个在样本上查询的详细数字显示

我们可以看到系统对 2008 年的销售量估计在 270 万到 626 万之间。

点击右边的按钮，可以调用 Spark 在原始数据上做查询：

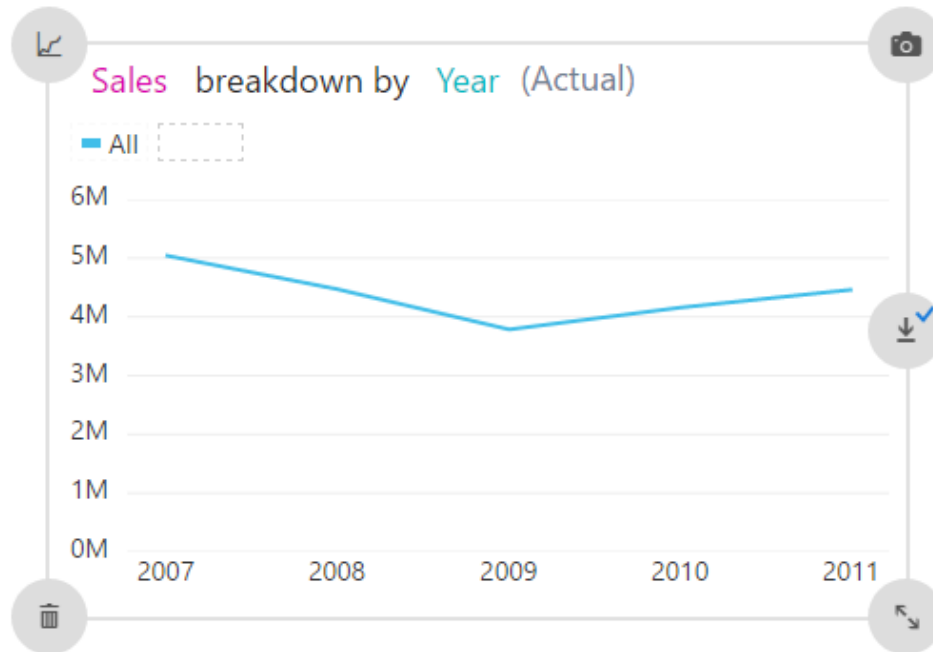


图 9 调用 Spark 的查询界面

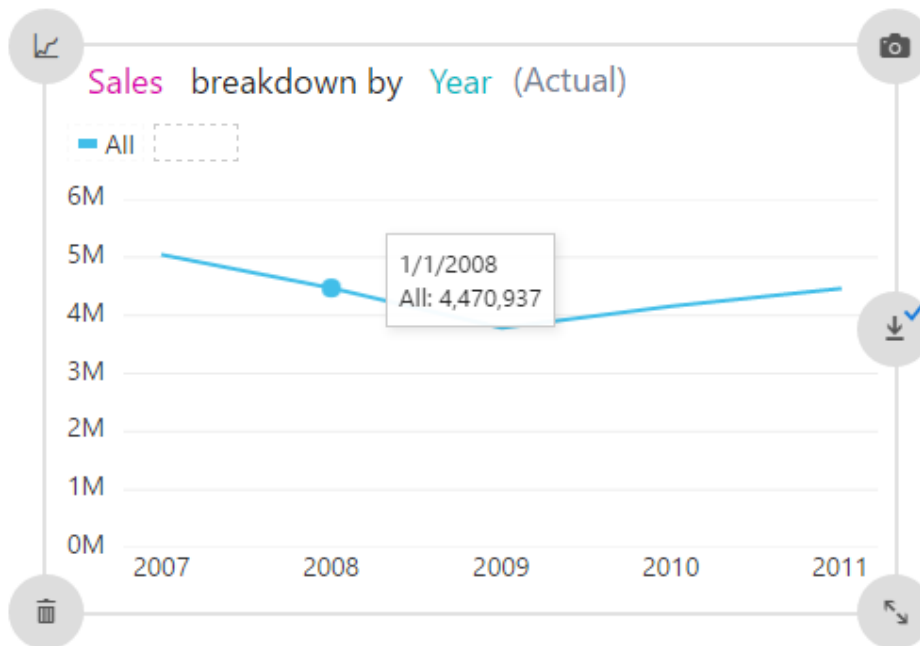


图 10 调用 Spark 的真实数据显示

此时的图片显示的就是在大数据上的真实结果，真实结果为 447 万，落在之前提示的区间内。

## 7. 实验结果与分析

### 7.1 实验环境

实验采用的软硬件环境如下所示。

界面端：使用 Chrome 浏览器，版本号为 49.0.2623.

服务器端：

- 操作系统：Windows Server 2012 R2 64 位
- CPU：Intel Xeon E5-2650 v2，主频 2.60GHz
- 内存：256GB

所使用的 Hadoop 集群：

- 机器数：6 台，每台机器的配置与服务器端相同；
- 使用的 Hadoop 版本：2.6.0
- 使用的 Spark 版本：1.5.1

### 7.2 实验项目介绍

评价本系统的实验主要有如下两项：

- 性能测试：本系统设计的主要目的就是给予用户快速的反馈，因此查询速度是评价本系统的一项重要指标。这里的查询速度既指统计推断算法在小数据上的运行速度，也指在大数据上进行验证时的速度。同时，由于我们做了一些数据的压缩，那么实际保存的小数据体积也是值得关注的评价指标。

- 准确度测试：统计推断部分为本系统的核心，因此其准确度对于衡量本系统的可靠性非常重要。在本部分我们也将对比一般的随机采样和分组采样准确度的差别。

### 7.3 性能测试

本部分测试系统在小数据和大数据上执行查询的速度，以及比较它们的差异。

在本部分中我们选择了一份实际数据，大小约 10GB，表中有约 1 亿行，12 列，其中 11 列为 Dimension，1 列为 Measure。执行采样时，指定采样条数为 10 万条。

实验方法为用一个预先定义好的查询集合，分别对该数据的样本进行查询以及调用 Spark 对原数据查询，同时记录查询总共用的时间。查询集合中，所有的查询都是简单 SQL 查询（只包含聚合函数，WHERE 语句以及 GROUP BY 语句），查询集合当中包含 10 条这样的查询。

实验结果如下：

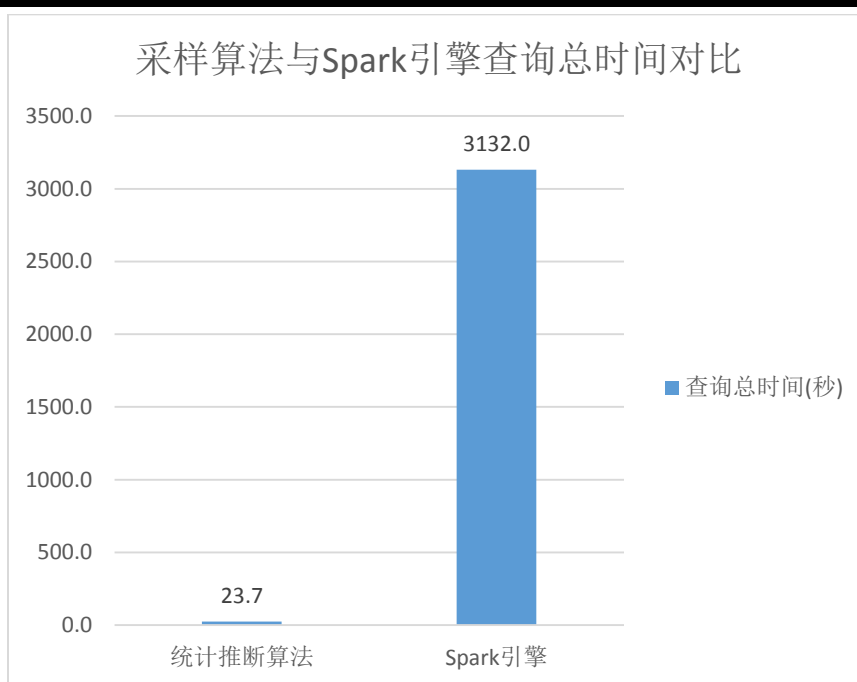


图 11 统计推断算法与直接利用 Spark 查询的耗时对比

由图可见，统计推断算法在小数据上执行查询的速度相当快，平均每条查询仅需要 2.3 秒，用户几乎无需等待，就可以立刻观察到结果；而 Spark 引擎对于每条查询都要耗费平均 5 分钟左右，远远超出了用户能忍耐的时间。所以，使用采样推断的方法，能够显著减少用户等待的时间，满足用户“交互”探索数据的需求。

## 7.4 准确度测试

在本部分中我们将测试统计推断算法的准确度，也将比较分层采样算法和简单随机采样算法的准确度。

本部分中我们采用与上部分完全相同的测试数据，并分别用分层采样算法和简单随机抽样算法制作该数据的两份样本。分层采样的样本中，指定了 3 列作为预设列集合。值得注意的是，这 3 个预设列集合中，有一列的取值分布非常不均匀：该列只有两种不同的取值，其中一种的条目数占到了总条目数的 80% 以上。

本部分的查询集合共包含 100 条查询，其中 80% 的查询所涉及的列都在预设列集合中，其余的 20% 涉及的列是任意的。与上一部分相同，这些查询都是简单 SQL 查询，但是因为 GROUP BY 语句实际上等价于一组 WHERE 语句的集合，因此本部分中的 SQL 查询均不含 GROUP BY 语句。

另外，在本部分的实验中，我们设定置信度  $\alpha = 90\%$ 。

衡量准确度的指标有以下两项：

- 正确率。正确率的定义如下：对于每一个查询，若真实值落在所估测的置信区间以内，则判定为正确，否则为错误。正确率的计算方法是

$$Accuracy = \frac{n_{correct}}{n_{all}}$$

即正确的查询数量与总查询数量之比。理论上讲，这个比率应当与置信度 $\alpha$ 接近。

- 相对误差。相对误差的计算方法如下：

$$error = \min\left(\frac{|v_{actual} - v_{upperbound}|}{v_{actual}}, \frac{|v_{actual} - v_{lowerbound}|}{v_{actual}}\right)$$

即取实际值与上下界的差当中较小的一个，与实际值的比值。

实验结果如下：

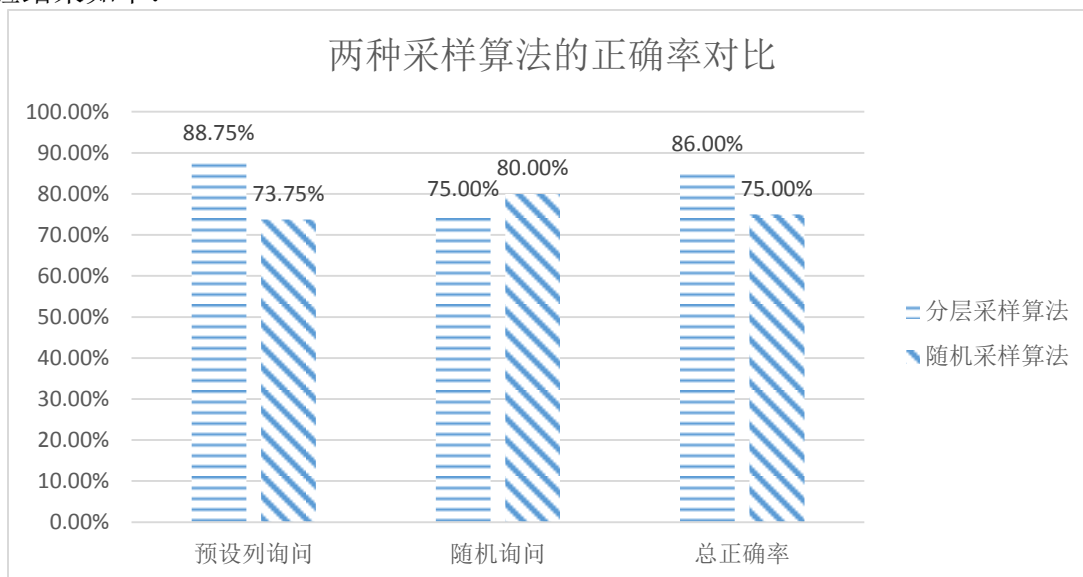


图 12 分层采样和随机采样的正确率对比

由图可见，对于预设列询问，分层采样算法的准确度明显好于随机采样，在随机询问上二者差异不明显，总体来看，分层采样算法的准确度要比随机采样高很多。

我们再来比较两种算法的相对误差：

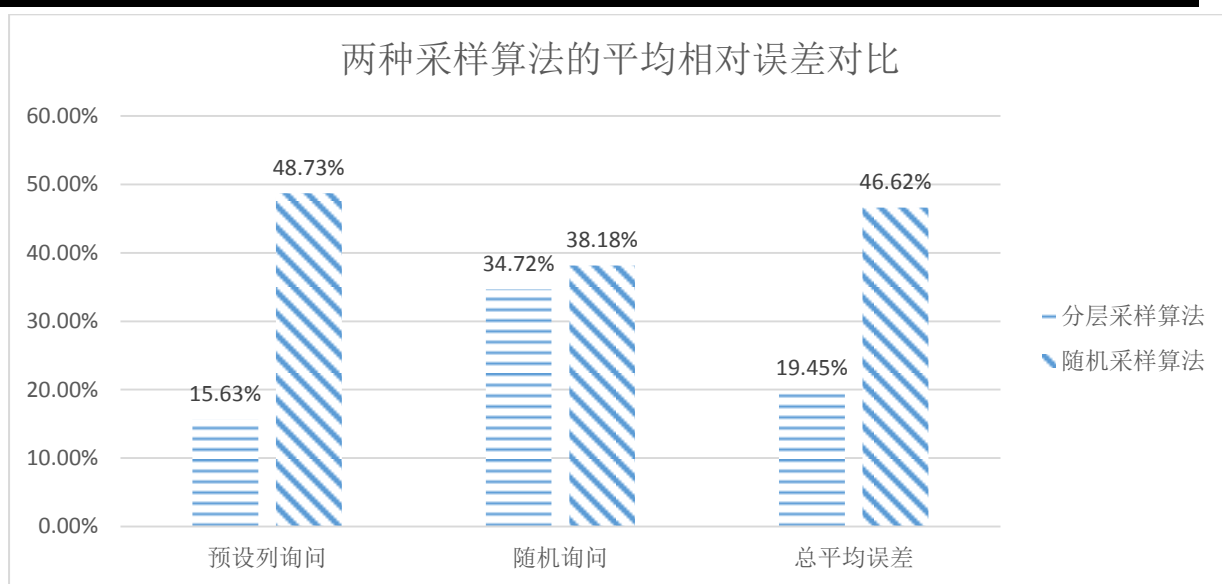


图 13 分层采样与随机采样的平均相对误差对比

由图可知，两种采样算法的平均相对误差在预设列询问上相差极大。其最主要的原因就是简单随机采样算法在预设列上的丢组问题，这个问题导致其对于小类根本无法准确估计，以至于产生了 100% 的相对错误。在随机询问上，两种算法差别不大，但是建立在这种情景上时，分层采样算法明显优于简单随机采样算法。

## 7.5 实验结论

由以上的实验，可得出以下结论：

- 采样推断的思想相对于利用分布式系统直接查询大数据，能够节省大量的查询时间，做到交互式回答询问。观察实验数据我们可以发现，在采样上回答查询的速度比在大数据上直接查询要快 100 倍以上，这不仅是因为查询设计的数据量小，算法复杂度低，也是因为小数据可以直接放到单机内存中，避免了大数据查询时硬盘和网络 I/O 消耗的时间以及数据分割、任务调度等复杂流程带来的损耗。
- 在查询只涉及预设列的时候，分层采样算法的准确率和相对误差都要明显优于随机采样算法；其余情况下，二者差别不大。这说明，只要能够适当地选取预设列，和实际的查询集合吻合度较高的话，采样分层采样算法就能获得令人满意的准确度。这也证明了前文中的观点，即随机采样算法对丢组问题无能为力，但是分层采样算法可以在一定程度上解决这个问题，带来准确度的提升。

# 结论

随着大数据时代的到来，大数据中蕴含的商业价值日益被各家企业所重视。通过对大数据的分析，企业可以获取数据当中隐含的规律，从而更好地服务客户，在制定战略时也能做到有的放矢，最终实现的是企业经济效益的大幅增长。然而，分析大数据是一项非常困难的任务，数据分析师们在研究数据时，不得不忍受漫长的查询响应时间，在庞大的、高维的数据当中想要获取有用的信息，对他们来说如同大海捞针一般困难。

基于这样的现象，本文在借鉴业界各项研究成果的基础之上，进行了一定的改良和创新。同时，本文也根据本文的应用场景，提出了适合这种应用场景的设计与实现。本文旨在通过这些创新性的设计和改进，实现一种通用的，能够处理大数据的交互式查询系统。

在上述文章的基础上，我们可以给出以下结论：

首先，本文构建的基于统计推断的交互式大数据分析系统成功将传统的 **Map-Reduce** 方法和采样推断算法结合起来，通过对精确度的一些牺牲，换来非常短的查询响应时间，其平均查询响应速度在 5 秒以下，满足了数据工作者对大数据的探索需求。

其次，本文使用的分层采样算法能够解决随机采样算法当中的丢组缺陷，其准确率相比简单随机采样有较大幅度的提升。

第三，本文在理论上推导了基于简单随机抽样和分层采样时，计算 **COUNT**，**SUM** 和 **AVG** 三种聚合函数及其置信区间的公式；

最后，本文对系统进行了测试，实验结果表明该系统在效率上能够满足交互式探索的需求，并且提出的分层采样算法的准确度相对于简单随机采样有明显优势。

但是，本文的研究内容尚有继续挖掘的空间。例如，在目前的系统中，我们是让用户自己选择预设列组合的，但是有没有办法让系统自己识别出较为重要的列，依据这些列进行分组呢？如果我们能够研究出一套这样的学习算法，毫无疑问可以让系统的准确度更上一层楼，更加改善用户的使用体验。

另外，本文的分层采样算法只能解决在 **Dimension** 上的取值不均问题，但是如果是在 **Measure** 上的取值不均，就只能通过增加采样率的方法来提高准确度。而提高采样率是要带来存储空间和查询时间的额外代价的。有没有办法可以在尽量不提高采样率的前提下，解决数值不均带来的准确度下降的问题呢？这也是本文的未来工作之一。

以上两点只是相关领域未来工作的两个示例，如果能够解决的话，我们的这套系统会更加智能、准确。

### 致谢

大学本科的四年时间转瞬即逝，在此完成学业之际，我要向这四年中给予过我无私帮助，关注、鼓励、支持、教导过我的人表示衷心的感谢。

我想感谢的是我的指导教师蒋宗礼老师。蒋老师深厚的学术积累，严谨的学术作风，都给我留下了十分深刻的印象。接触蒋老师期间，蒋老师多次指导我认真做事，对我严格要求，并时常与我探讨专业领域相关问题。与蒋老师的交流总是使我受益匪浅，蒋老师的教导我也将铭记于心。

以及，我想特别感谢微软亚洲研究院的导师林庆维研究员和楼建光博士，是他们给了我机会到这所业界最领先的、学术界与工业界交汇的机构参与实习，大四这一年的实习当中我的收获甚至超过过往三年的总和。我与各所学校最出色的同学们坐在一起讨论问题，接触到第一手的数据，努力思考解决工业界最迫切需要解决的问题。经过这一年的实习，我的眼界得到了极大的开阔，思维方式也得到了极大的充实，同时还认识了很多在计算机领域中有热情、有实力以及有天赋的同学们。我也时时刻刻为这里的研究员们的专业素养和学术造诣所折服。在这一年的时间里，是我的两位导师帮助我适应这里的环境，指导我遇到问题时的思维方法，耐心与我讨论，指出我之前学习过程中的不足。两位导师的指导与鼓励将成为未来我职业生涯中重要的指路标。我愿意再一次感谢两位导师给予我在微软亚洲研究院实习的机会，以及在实习过程中对我的指导和鼓励，这一年的时光将使我受益一生。

然后，我想感谢高红雨老师，四年以来一直不遗余力地为我的学习与研究创造条件，指导我的学业道路与未来职业规划，在生活上也给予我无私的关怀。如果没有高老师的支持，我根本无法想象今天自己能取得的成绩。同时，高老师学识渊博却谦虚有礼，实事求是的品质深刻地影响着我。

之后，感谢大学中所有指导过我的任课老师，学生工作部门的老师，我的大学同学们，我也想在此一并致谢。感谢老师们帮我奠定这门学科的基础，给予我生活上和学术上的支持；感谢同学们四年以来的陪伴，以及对我的包容和鼓励。

最后，我想首先感谢我的父母，是他们支持我读完大学，我能够顺利走完大学四年，离不开他们在背后默默的付出。他们努力工作的身影一直以来都是我前进的动力。

在北京工业大学的学习生活马上就要结束了。在这里度过的四年时光将使我终身难忘。



## 参考文献

- [1] 严霄凤, 张德馨. 大数据研究[J]. 计算机技术与发展, 2013, 4(32): 4.
- [2] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.
- [3] Mayer-Schönberger V, Cukier K. Big data: A revolution that will transform how we live, work, and think[M]. Houghton Mifflin Harcourt, 2013.
- [4] Cukier K. Data, data everywhere: A special report on managing information[M]. Economist Newspaper, 2010.
- [5] Big Data [EB/OL]. Wikipedia. 2013. [http://en.wikipedia.org/wiki/Big\\_data](http://en.wikipedia.org/wiki/Big_data)
- [6] Cukier K. Data, data everywhere: A special report on managing information[M]. Economist Newspaper, 2010.
- [7] 喻钢, 周定康. 联机分析处理 (OLAP) 技术的研究[J]. 计算机应用, 2001, 21(11): 80-81.
- [8] 张忠平, 李荣, 郭丽丽. 联机分析处理的综述和分析[J]. 计算机应用研究, 2003, 20(8): 10-13.
- [9] IBM OLAP [EB/OL]. IBM. 2016. <http://www-01.ibm.com/software/analytics/rte/an/olap/>
- [10] Power BI [EB/OL]. Microsoft. 2016. <https://powerbi.microsoft.com>
- [11] Pendse N, Creeth R. The OLAP report[J]. Business Intelligence, 1995.
- [12] Online analytical processing [EB/OL]. Wikipedia. 2016. [https://en.wikipedia.org/wiki/Online\\_analytical\\_processing](https://en.wikipedia.org/wiki/Online_analytical_processing)
- [13] Spark [EB/OL]. Apache. 2016. <http://spark.apache.org/>
- [14] Hadoop [EB/OL]. Apache. 2016. <http://hadoop.apache.org/>
- [15] Cormode G, Garofalakis M, Haas P J, et al. Synopses for massive data: Samples, histograms, wavelets, sketches[J]. Foundations and Trends in Databases, 2012, 4(1-3): 1-294.
- [16] Sirangelo C. Approximate Query Answering on Multi-dimensional Data[D]. PhD Thesis, University of Calabria, 2005.
- [17] 王松桂, 张忠占, 程维虎. 概率论与数理统计（第三版）[M]. 科学出版社, 2013: 146-156.
- [18] Ding R, Wang Q, Dang Y, et al. Yading: Fast clustering of large-scale time series data[J]. Proceedings of the VLDB Endowment, 2015, 8(5): 473-484.
- [19] Agarwal S, Mozafari B, Panda A, et al. BlinkDB: queries with bounded errors and bounded response times on very large data[C]//Proceedings of the 8th ACM European Conference on Computer Systems. ACM, 2013: 29-42.

