分类号 密级

UDC

学 位 论 文

面向视频网站用户行为的数据预处理技术研究与运用

|  |  |  |  |
| --- | --- | --- | --- |
| 作者姓名： | 杨海乐 | | |
| 指导教师： | 韩春燕 副教授 东北大学软件学院 | | |
| 申请学位级别： | 硕 士 | 学科类别： | 工 学 |
| 学科专业名称： |  | | |
| 论文提交日期： |  | 论文答辩日期： |  |
| 学位授予日期： |  | 答辩委员会主席： |  |
| 评阅人： |  | | |

东 北 大 学

年月

##### A Thesis in Computer Science and Technology

**Research and application of data pre-processing technology for video website user behavior data**

By Yang Haile

Supervisor: Associate Professor Han Chunyan

**Northeastern University**

独创性声明

本人声明，所呈交的学位论文是在导师的指导下完成的。论文中取得的研究成果除加以标注和致谢的地方外，不包含其他人己经发表或撰写过的研究成果，也不包括本人为获得其他学位而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名：

日 期：

学位论文版权使用授权书

本学位论文作者和指导教师完全了解东北大学有关保留、使用学位论文的规定：即学校有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人同意东北大学可以将学位论文的全部或部分内容编入有关数据库进行检索、交流。

（如作者和导师不同意网上交流，请在下方签名；否则视为同意。）

学位论文作者签名： 导师签名：

签字日期： 签字日期：

面向视频网站用户行为的数据预处理技术研究与应用

# 摘 要

视频网站用户行为数据具有个别维度基数大、数据量巨大的特点，对于常规的数据预处理方法很难甚至不能在可接受的时间范围内挖掘其中的价值，这对于企业来说几乎是不可接受的。为了解决这一问题，分布式并行计算与空间换时间的思想相结合是一个必然，同时在对处理此场景需求方面，数据预处理方面的Cube物化成为了关键。

MapReduce是目前公认的优秀并行处理框架，对于处理SUM、MAX、MIN等纯代数性质的度量，可利用自上而下并择优选路的算法，可以将IO扫描、CPU计算等资源代价在各个层次上得到偿还，下层结果可以依据上层结果合并得到，进一步优化了计算过程。同时需要解决的问题就是合理分配数据、避免负载不均衡现象、集群内部网络传输过程中所产生的带宽压力、合理划分物化路径与物化路径。对于Distinct Count 等整体性度量来说，自上而下的择路优化方法并不能很好地解决整体向度量的问题，因为整体性度量并不能像SUM、MIN、MAX等纯代数度量可以从顶层的计算结果直接推算其值，为了解决此问题Arnab Nandi等人提出了自下而上的计算方法，通过MR框架Shuffle阶段的排序与捆绑Key来处理整体性度量带来的问题。

为了解决IO频繁操作、网络带宽占用过多、磁盘占用过高等问题，本文使用对维度的不同值进行编码，减少磁盘读操作、磁盘占用、网络带宽占用情况。使用均匀抽样实现Cube物化路径以及物化策略的选择，自动调整Cube物化策略，整体使用自上而下的分区域降维优化算法。并且充分利用MapReduce并行处理框架的能力，借鉴Arnab Nandi等人提出的针对整体性度量计算的方法，提出了一种计算整体性度量的方法，用于解决视频网站用户行为数据分析方面对整体性度量的需求。

最后通过实验验证了系统的有效性，其中优化过的算法在对减少网络带宽、IO读操作以及负载均衡方面都有很好的性能，对于抽样择路也证明了能有效地减少系统计算量，针对整体性度量也证明了算法的有效性。

关键词： 预处理，MapReduce，分布式

**Research and Application of data pre-processing technology for video website user behavior’s data**

# Abstract

**Keywords:**

目 录

[分类号 密级 i](#_Toc463858244)

[独创性声明 I](#_Toc463858245)

[摘 要 I](#_Toc463858246)

[Abstract I](#_Toc463858247)

[第1章 绪 论 1](#_Toc463858248)

[1.1 研究背景 **错误!未定义书签。**](#_Toc463858249)

[1.2 数据预处理系统发展现状 **错误!未定义书签。**](#_Toc463858250)

[1.3 预处理系统研究现状 **错误!未定义书签。**](#_Toc463858251)

[1.3.1 国外研究现状 **错误!未定义书签。**](#_Toc463858252)

[1.3.2 国内研究现状 **错误!未定义书签。**](#_Toc463858253)

[1.3.3 研究趋势 **错误!未定义书签。**](#_Toc463858254)

[1.4 存在的问题与挑战 **错误!未定义书签。**](#_Toc463858255)

[1.5 研究内容和组织结构 1](#_Toc463858256)

[1.5.1 研究内容 **错误!未定义书签。**](#_Toc463858257)

[1.5.2 论文结构 **错误!未定义书签。**](#_Toc463858258)

[1.6 本章小结 1](#_Toc463858259)

[第2章 预处理相关理论与技术 1](#_Toc463858260)

[2.1 MapReduce分布式并行处理框架 1](#_Toc463858261)

[1.1.1 MapReduce简介 1](#_Toc463858262)

[1.1.2 MapReduce任务执行流程 1](#_Toc463858263)

[1.2 Hbase数据库 1](#_Toc463858264)

[2.3 数据清洗技术 1](#_Toc463858266)

[1.2.1 数据清理 1](#_Toc463858267)

[1.2.2 数据集成和变换 1](#_Toc463858268)

[1.2.3 数据归约 1](#_Toc463858269)

[2.4 基于MapReduce的数据立方算法 1](#_Toc463858270)

[1.2.4 基于MapReduce立方体算法概述 1](#_Toc463858271)

[1.2.5 基于MapReduce数据立方算法介绍 1](#_Toc463858272)

[2.5 本章小结 1](#_Toc463858273)

[第3章 基于维值替换的自顶向下数据立方算法 1](#_Toc463858274)

[1.3 自顶向下立方体算法难点 **错误!未定义书签。**](#_Toc463858275)

[1.4 改进后的自顶向下算法 **错误!未定义书签。**](#_Toc463858276)

[1.4.1 维度值快速处理 1](#_Toc463858277)

[1.4.2 抽样规则分析 1](#_Toc463858278)

[1.4.3 择路与物化区域划分 1](#_Toc463858279)

[1.4.4 改进后的快速DataCube算法 **错误!未定义书签。**](#_Toc463858280)

[1.5 实验结果分析 **错误!未定义书签。**](#_Toc463858281)

[1.6 本章小结 1](#_Toc463858282)

[第4章 基于BitMap的Count Distinct度量计算 1](#_Toc463858283)

[1.7 整体性度量计算概述 1](#_Toc463858284)

[1.8 现有算法优缺点 1](#_Toc463858285)

[1.9 基于BitMap的不同值数据立方计算 1](#_Toc463858286)

[1.10 实验结果分析 **错误!未定义书签。**](#_Toc463858287)

[1.10.1 实验环境与数据来源 **错误!未定义书签。**](#_Toc463858288)

[第5章 预处理系统的设计与实现 1](#_Toc463858289)

[5.1 系统需求分析 1](#_Toc463858290)

[5.2 系统总体架构 1](#_Toc463858291)

[5.3 系统软件设计与开发 1](#_Toc463858292)

[5.4 本章小结 1](#_Toc463858293)

[第6章 结论与展望 1](#_Toc463858294)

[6.1 工作总结 1](#_Toc463858295)

[6.2 未来展望 1](#_Toc463858296)

[参考文献 1](#_Toc463858297)

[致 谢 1](#_Toc463858298)

# 绪 论

## 研究背景及意义

随着信息时代的到来，人们在通信与网络等方面体验到了前所未有的便利，越来越多的人在网络、通信等虚拟世界花费更多的时间来获取或者交流信息；人们在享受互联网科技发展给自身带来的好处的同时，也在不断创造海量数据；全球每年数据总量增长速度保持1.5倍左右，预计到2020年在全球范围内数据总量将达到40ZB。在全球数据总量快速增长的大背景下，公司等机构内部的数据也随着互联网等技术的发展，快速的增长起来；海量的数据虽然对公司等机构来说是一笔财富，但是随着企业内部数据量的逐步增长，数据量已经超出了单机所能处理数据的极限；企业迫切需要一种方法去改变现状，由此商业层面的竞争转移到了数据处理方面的竞争，因为大数据处理在很大程度上可以帮助企业做出很多战略性的决策，抢占市场先机；而现实中，企业需要处理TB级数据情况在很多情况下已经是实实在在的需求了，而采用何种方式去有效的处理动辄数TB数据已经成为一个限制数据分析的一个瓶颈。

近年来，全球大数据领域市场逐步扩大，从2012年10亿美元到2016年已经超过了30亿美元，增长了超过3倍，这些数据从背后反映了大数据分析领域已经引起了足够的重视；而现实中，大数据分析已经成为各行业之中与公司之间竞争的焦点；大数据中蕴含着潜在的规律与价值，只有使用有效的手段去挖掘，才能最大化信息财富的优势；例如大数据中所蕴含的数据规律能在很大程度上反映公司目前的经营状况；对于电商网站来说通过大规模数据分析能对商品仓储分布做出合理的规划；在对用户行为数据分析层面，可以合理的预测用户下次点击行为，并以此建立合理的模型向用户推荐合理的潜在的感兴趣的商品。同理，而对于视频网站用户分析过程，用户每一次点击行为都是日后向用户提供更好资源的依据，通过对用户行为倾向的分析，采购符合大众取向的题材视频来提高网站关键指标等；用户行为数据的大数据分析也是视频网站增加流量与广告投放的重要考量，而整体用户汇总的数据有利于公司快速捕捉热点事件，快速的做出合理的商业决策。

而伴随着MapReduce的出现，让人们看到了分布式并行计算在处理大数据上的强大优势，MapReduce结合数据立方体的物化技术看到了在大数据集上快速进行分析的可能性，同时在许多单机无法完成的分析任务重表现出了极好的吞吐性能；但是在实际操作过程中，对于时间跨度很大的场景，公司对数据分析的时间性能要求比较高，而作为批处理方面的优秀框架，MapReduce并不能很好地适应对分析时间有要求的场景；例如对于一个简单的分析问题，例如对于某个视频在一年的用户访问量这样的简单问题，通过MapReduce并行编程模型就会读取全年的数据进行处理；这样处理不但集群会有过高的负载，而且可支持的查询量来说已是很有限的，所以说对于批量的查询来说有些力不从心，而且对于某些查询是重复的。

但是Data Cube的物化技术却给数据预处理方面提出了另一条思路，通过预聚合计算，形成完整的数据立方体，通过将这些结果保存起来，业务分析人员可以从多个角度观察分析数据，可以为维度上卷、钻取、切面、切片等操作提供数据支持，同时重复的查询不会再重复计算，只需要再次查询。所以说数据立方物化可以加快企业数据分析的速度，使其有能力灵活分析时间跨度比较大的数据而不会出现时间消耗无法承受的情况，优势相比于将查询映射成MR任务的方式有着明显的优势；目前在数据立方物化的预处理系统方面，出现了很多基于MapReduce的算法，算法的主要考虑的是如何减少数据立方物化过程中的计算量、带宽、磁盘占用读写。但是也存在一些度量无法满足Data Cube之间聚合的要求。

数据预处理可以帮助企业在数据分析领域加快分析进度，摆脱因数据处理量过大而产生的数据处理时间过长问题；从而企业不但在充分利用原有数据的同时，避免了投入过大的数据分析成本，而且获得多角度快速观察数据的能力，这在一个充满竞争的时代背景下，尤其重要；同时这种数据分析对企业的决策与市场营销手段关系紧密，数据分析上产生的经济效应特别明显；可以说，数据分析的成功是在商业竞争中脱颖而出不可缺少的重要因素，因此对于预处理系统的研究非常具有理论意义与应用价值。

## 国内外研究现状

数据预处理概念不是一个新鲜的话题，真实世界的数据都是需要进一步处理才能为人们所使用，所以数据预处理往往是数据处理的第一步；数据预处理包括很多内容，例如数据清理、数据集成与变换、数据归约。在本文中讨论最多的是属于数据预处理中的数据规约中的重要部分：数据立方聚集。属于数据预处理的数据立方聚集一直是国内外研究的热点领域，从单机算法演进到分布式集群算法，一直是数据预处理与数据分析领域的热点。数据立方计算主要关注点是数据维度上的聚合，从维度和度量视角观察数据，所以计算的核心主要落到了对维度的组合上。虽然说现在主流的数据立方计算算法都是以并行处理为主，但是传统单机数据立方计算算法也有一定的参考价值，很多并行算法都是传统单机立方体算法的一个扩展。所以本章将从传统数据立方算法、并行数据立方算法这两个方面来介绍国内外研究现状。  
 国外在早期就提出了预处理思想，早期研究过程中，由于算法都是基于单机作为研究背景，所以很多经典的数据立方计算算法都是适用于单节点的，主要考虑的优化的方向是大多也是单机适用的，传统数据立方算法主要有BUC、PipeSort、PipeHash等算法，下面对数据立方计算的研究现状以算法的方式做介绍。在1996年，JIM GRAY发表了Data Cube概念的论文[]，在文中提出数据分析应用多是通过许多维度的聚合来展现数据，即是和GROUP BY SQL语句实现的聚合功能相同，并正式讨论了有效的计算方法去得到数据立方例如最小父节点，即是通过某个维度组合来计算其子组合而不是通过重复聚合整个数据集来优化计算Data Cube 锁消耗的时间与资源；同时提出了在关系型数据库中对数据立方进行上卷、下钻等操作；标志着数据立方聚合计算的开始；随后Sunita SaraWagi在Data Cube概念的基础上，提出了针对共享排序、共享分区、缓存结果等多个优化方面的PipeSort与PipeHash算法[]；其中共享排序指的是一般通过排序相同的维度组合值必定相邻，这样避免了重复查找需要组合的元组；其中共享分区指的是在使用Hash类聚合方法时由于Hash表太大导致内存无法容纳，这时需要将Hash表分割成多个区进行操作。而作为自底向上（Bottom-Up）的BUC算法，是由Kevin Beyer等人于1999年提出的解决冰山立方体问题的，一般适用BUC的场景是在查询条件中有类似Count(\*)>X等类似的限制条件，BUC在聚合过程中只要是不满足此类性质的数据集合将会被删除从而减少计算量，这也是算法考虑自顶向下的主要原因。

数据预处理中的数据立方计算算法在并行计算方面也是一个热点研究领域，但是由于受限于并行系统开发的限制，基于并行程序的数据立方计算相对于单机算法来说，出现相对较晚，而由于MapReduce并行框架的出现，使得开发并行程序相对以前来说变得异常简单，再这样的背景下，出现了基于MapReduce程序的高级语言可用于GROUP BY查询例如Pig和Hive。同时也出现了很多基于MapReduce的数据立方优化算法，典型的有Naïve算法、MRDataCube算法、MRLevel算法、基于传统的PipeSort算法的MRPipeLevel算法与MRCube算法等。这些算法在预处理相关理论与技术章节将会详细的介绍，这里将会对算法一些特点进行介绍。Naïve算法是比较简单的一种Data Cube算法，只是计算Data Cube的一种原始算法，应用领域是代数度量，容易引起数据倾斜和巨大的网络带宽开销。而MRDataCube算法是S Lee在2015年提出的对Naïve算法的一种改进，此算法注重Map端的数据立方体顶层聚合来改善数据量与网络带宽，算法总体分为两个阶段；而基于分层MRLevel、MRPipeLevel算法是Suan Lee与2015年提出的基于最小父节点分层优化算法，此算法是单机算法优化思想与并行算法的结合，数据立方计算与新兴并行处理框架的结合给数据立方计算注入了新的活力，在实践或者实验中取得了很好的效果，但是上述所提出的集中数据立方计算方法是针对代数式度量的，或者通过代数度量能得到的度量，对于整体性度量的研究领域，由于并不像代数度量领域那么好处理，所以基于MapReduce的整体性度量的研究成果相对较少。主要是Arnab Nandi等人在2011年提出的MRCube算法，此算法总体使用BUC传统算法思想，将数据区域划分为不同总类实现负载均衡，使用MapReduce框架的自定义Hash实现不同Reducer子任务的整体性度量的可相加性。

总体来说，在数据预处理的数据立方聚集方面的研究是一个研究热点。特别是现在基于MapReduce算法，或者其他比较新兴的技术的研究应用越来越广泛，特别是基于流式计算的数据立方实时处理方面都是国内外研究的热点领域。

## 论文主要工作与研究内容

本文主要对数据预处理系统做了设计与实现，并在几个关键点上进行了研究。基于企业内部的数据处理需求，提出了通过对数据仓库层中的细节数据层（DWD）、轻微汇总数据层（DWS）等抽取数据，建立以维度分析为核心的数据汇总数据立方数据的数据预处理系统，在系统设计与实现中对数据立方完全物化算法进行了优化下面将详细介绍。

在实际数据展现过程中，通常原始数据存储需要保留的时间跨度比较长，所以需要很多存储空间，为此数据预处理在设计时，提出了一套编码替换的方法，利用一般维度的选取基数变化范围不大的特性，使用整数类型进行编码替换，节省宝贵的存储空间资源，有效利用资源。在数据立方物化方面，为了解决数据倾斜、聚合区域选择、不同聚合区域处理流程、任务执行优化方面的问题，提出了Multi-RegionCube算法，此算法总体来说以自顶向下的计算方式物化数据立方，为了高效地进行数据立方物化计算利用抽样方法对大数据量进行抽样，选取数据关键特征，同时抽样的结果也对后续物化区域划分起到了指导作用，同时确定了稠密维度组合，以及特定维度组合下的维度值的均匀分配方法等，加快了数据立方的物化过程。在最后讨论了使用BitMap计算整体性度量Count Distinct问题，提出了使用压缩BitMap数据结构实现对Count Distinct度量的高效计算。

## 论文组织结构

根据研究内容，本文将内容分如下结构：

第一章中介绍了数据预处理系统在实际应用中的研究意义与研究价值，总结了国内外对预处理系统中的数据预处理中的数据立方聚集技术做出了全面的介绍与分析，同时阐述了在本数据预处理中所运用的主要算法和关键技术的研究现状。

第二章主要对系统中所用到的数据立方聚集算法做了详细的分析与介绍，并从总体上将算法进行了分类，从多个角度与评判标准去分析算法的优缺点，并对本预处理系统所用到的关键技术和框架做了介绍。

第三章中首先对数据预处理的总体流程做了介绍，随后提出了本预处理系统中关键部分流程是数据聚集，所以第三章针对数据立方计算设计了相应的算法和流程，其中Multi-RegionCube算法主要针对代数式度量数据立方计算，主要思想是分而治之，对不同特点的数据区域做不同的计算策略。最后讨论了对于整体性度量Count Distinct的数据立方计算方法，讨论了使用BitMap数据结构对Count Distinct数据立方计算的处理流程。

第四章

## 本章小结

# 预处理相关理论与技术

信息化的过程中催生了很多优秀的预处理技术，基于运用场景与处理流程，预处理技术分很多方面，例如数据清洗、数据集成、数据转换等。随着分布式并行处理框架、存储大规模数据的Nosql数据库、海量分布式存储系统的出现，数据预处理技术又有了新的发展，展现出了新的生命力。在本章中，着重强调本系统在设计与实现中用到的技术，例如数据清洗以及数据立方体物化等技术。

## MapReduce分布式并行处理框架

### MapReduce简介

MapReduce是受Lisp等函数式编程语言的Map与Reduce原语启发，由Google公司在2004年向公众公布的分布式并行处理框架，可通过运行于成千上万台商用主机上，提供处理T级或者以上级别的海量数据，通过MapReduce并行程序框架模型，用户只需要实现简单的API接口，就可以使用MapReduce提供的强大的并行处理能力，其主要有Map与Reduce两个可编程的阶段组成，自身拥有很强的容错机制，MapReduce特别适合任务之间相互独立的场景。当前广泛使用的是由Apache开源组织实现的MapReduce开源版本，下面详细介绍MapReduce所具有的特点。

MapReduce本身是Apache Hadoop项目的一部分，其中MapReduce任务的顺利执行依赖另外两个重要组件：HDFS（Hadoop Distributed File System）、YARN。其中HDFS为分布式存储引擎，为MR任务的运行提供存储支撑。而Yarn是新一代MapReduce并行处理执行框架的简称，本身提供对计算机集群资源管控、MR任务调度、任务运行监控等功能，MapReduce执行流程图如图2.1所示。



图2.1 MapReduce工作流

Fig.2.1 MapReduce workflow

### MapReduce任务执行流程

MapReduce任务从用户的角度来说一般只需要实现Mapper与Reducer接口就可以提交给系统运行，而且MR任务处理都是<Key，Value>对的形式出现，必要时根据数据特点，实现自身的partition函数，来实现节点之间的负载均衡。从系统角度来说一个MapReduce任务提交到系统后包含多个阶段，下面将详细介绍：

1. Map阶段，Map阶段在MR体系中职责在于数据读取与封装，经处理过的<Key,Value>形式的输入数据将写入到磁盘中，最终通过集群内部网络到达Reduce任务主机。
2. Combine阶段，Combine严格来说不算一个真正的阶段，只是作为对Map阶段产出结果的一次压缩计算，通常利用的压缩程序时Reduce程序，目的是减少Map产出的中间结果数据占用宝贵的带宽资源。
3. Shuffle和排序阶段，shuffle阶段不是一个独立的过程，在Map与Reduce端甚至在其之间都会发生，shuffle阶段在Map端，通过将<Key，Value>对按partition函数语义分割成小的数据区域，并将各个数据区域局部排序，存入磁盘中，等待Reduce端拉取数据。当Reduce任务开始后，Reduce向Map节点拉取数据，各个Map节点具有相同partition分割属性的数据块将被分发到相同的Reduce节点，并在Reduce段进行排序，合成较大数据块。
4. Reduce阶段，又称规约阶段，相同Key的键值对将会被汇集到某个Reduce进行处理，每个Reduce将产生一个结果文件。

## Hbase数据库

Hbase是Google公布的BigTable项目的开源版本，BigTable是Google为了存储海量结构化数据而设计的类数据库的分布式存储系统。而作为BigTable的开源版本，Hbase已经是Apache Hadoop大数据生态体系的顶级项目，从一开始就被作为Hadoop生态的数据库解决方案来对待。Hbase是为了解决传统关系型数据库不能很好地支持半结构化数据而产生的，并且提供很好的水平扩展性能，相比于传统数据库，Hbase在存储数据级别上有较大幅度的提高。

Hbase适合半结构化数据管理，支持压缩算法插件化，使得对于特定列簇可以指定相应的压缩算法。相比传统关系型数据库来说，Hbase主要优点体现在对几十亿行数据的快速访问上，但是为了获得读写性能，放宽了对ACID的要求，以下从多个方面介绍Hbase。

1. Hbase架构，Hbase集群包括一个Master节点与多个RegionServer节点，并且Hbase集群需要借助Zookeeper提供的分布式锁等服务，Mater节点主要负责负载均衡、分配Region等作用Hbase系统架构图如图所示2.2所示。
2. 面向列簇，一般来说Hbase是面向列簇的数据库，因为在定义一张Hbase表的过程中，列簇是预先定义的，而且是不可更改的，而列簇中的列是可以自定义的。之所以说Hbase是面向列存储的，是因为在HDFS中一个列簇由一个Region负责存储，以达到不同列分区域存储的目的。
3. 物理存储结构，从物理存储层面看，Hbase把一张表划分成了多个区域，从大体来说，一张表将被根据行键分割为多个HRegion，然后Hbase将会对不同的列簇进行分割，此时分割的文件就是HFile文件，而Hbase底层所存储的文件就是HFile文件。

## 数据清洗技术

企业在实际生产中，往往产生很多数据，这些数据并不是完全可信的，甚至有些在逻辑层面就不应该存在或者直接为null值，这些存在问题的数据严重影响了对数据进行处理的准确性，严重时会使处理结果失去意义。数据清洗技术不是新的概念，随着数据预处理技术的兴起，数据清洗一直是一个热门话题，也是数据与处理过程中无法忽略的一个重点环节。

### 数据清理

数据清理是在原数据的基础上，根据一定规则来填充当前值或者修改有问题的数据。在数据清理大致可以分为三类：空缺值、噪声数据、不一致数据，以下将分别对这三种数据清理方式做详细介绍。

#### 空缺值

企业收集的数据有一些数据属性是缺失的，比如对于视频网站数据来说，可能用户关闭了cookie等原因，用户名属性是空值，比如在数据处理底层由于某些处理程序设计不完善，导致某些属性的值出现了null值。对于空缺值目前有很多解决方式，直接丢弃空缺值是最简单的方式，但是对于属性null值比例起伏过大时，简单的丢弃数据条目的方式就显得不可取。人工补全也是处理空缺值的一种手段，但是随着数据量剧增的背景下，这种方式显得格外无力。

对于上述方式来说，都不能较好的适应真实的生产环境，而第三种方式是将某一个属性的空缺值以此属性所有值的平均值来填补，此方式比起前两个方式有一定的进步。而据此进一步的优化是取同属于一类的属性记录的平均值作为空缺值，这样使得填补的空缺值更加平滑，在大部分实际情况下更接近真实曲线。最后一种方式是采用回归、贝叶斯形式化、判定树等手段，使用最多的信息进行推测缺失元组最有可能属于哪一类。这种方式使得结果更加忠于现实，是最常用的数据清理方式。

#### 噪声数据

噪声数据是由于处理属性值时的偏差或者随机错误造成的，解决此类数据问题常用的方式有四种大类如下。

1. 分箱，分箱方法自身有很多种分类，总体思想是根据相邻数据值平滑自身数据，分箱规则取值规则可能是取平均值、边界值等，是一种离散化的应用。
2. 聚类，聚类分析可以运用到很多方面，通过对数据进行相似度分析来实现数据簇划分，可以聚类在噪声数据中实现噪声数据归类取值，典型的聚类算法包括K-Means、ID3等算法。
3. 计算机和人工检查结合，人可以识别并定义噪声数据模式，而机器则可以快速判断与筛选。
4. 回归，回归清除噪音是指找到数据服从的回归函数，用回归函数指导数据清理，清除噪声数据。

#### 不一致数据

数据不一致在数据处理中经常会出现，大多是由于属性称谓不统一，多条数据相互矛盾造成的，这类数据需要有统一的例程来纠正属性值。在正式应用中，不一致的数据将会在数据集成与变换过程中得到处理，如下节所述。

### 数据集成和变换

数据集成与数据变换是两个数据预处理的过程，数据集成关注点是将多个分散的数据源合并成统一的数据源，多个数据源往往使用不同的技术存储数据，而数据变换关注点是在变换过程中如何实现将数据规范化、利于数据分析化、消除数据本身缺陷等。

无论是公司合并整合数据资源，还是应用内部整合，都需要数据集成的支持，而数据集成所面临的挑战有很多，数据集成在对内部人员要求方面讲，需要设计人员通晓公司所有整合数据的详情，对于集成过程中数据源正确对接就需要实体识别，而另一个具有挑战性的问题是如何判断冗余，对于给定的A、B属性，两者之间的相关性可用下列公式2.1表达。

|  |  |
| --- | --- |
|  | 公式2.1 |

其中，n是数据条目，分别为数据源标准差，、是均值，最后需要注意的是处理检测数据冲突，即是不一致数据的处理，一般合并在一起的数据源对相同概念将采用统一的称谓和度量标准，母线企业数据集成使用最多的是建立数据仓库。

数据变换所涉及的内容比较多，总体来说包括数据清理部分与数据整合部分，数据清理部分在这里不再赘述。数据整合部分包括数据规范化、特征构造等。数据规范化在上述中已经说明，特征构造是指在原数据集中并没有需要的某些维度，通过构造而产生新的利于数据分析与挖掘的数据维度或属性。

### 数据归约

企业内部为了方便进行数据分析所建立的数据库一般很难在其上直接做数据分析，因为数据量庞大，时间成本过高，这就需要一种方法去压缩或者说浓缩数据，使得处理后的数据基本不丢失分析属性的完整性且比原始数据小得多，可在可承受的时间成本上完成分析，这类方法有去除不相关属性或者针对分析目的预计算相关维度合并相关度量结果来压缩元组等，数据规约就是来处理这种需求的方法。

数据规约包括很多种，Data Cube聚集、维度规约、编码压缩、离散化和概念分层产生。维度规约就是去除与分析关系不大或者已有替代维的维度，编码压缩就是利用编码技术将属性值使用特定的编码进行替换，达到减少数据空间占用的目的，在这里不再赘述。Data Cube聚集技术其中一种是通过物化所有的维度组合，将计算得到的Data Cube数据分层次存储。例如由于业务分析需要，将维度地区、时间维度进行物化，业务关注重心是销售额度量，其中地区（包括国家、地区）与时间（包括年、月、日）具有层次的维度。在业务分析之前，Data Cube将会计算各个维度组合所产生的度量值，这里度量值将是销售额。其中每日某地区的立方单元将是数据立方中的最小立方。通过Data Cube切片、切块等动作，充分支持业务分析。

## 基于MapReduce的数据立方算法

本节主要对基于MapReduce的数据立方体计算方法做了详细的介绍与分析，包括自顶向下与自底向上的数据立方算法，分析了列举算法的有缺点，其中为了方便后续内容的叙述，首先对数据立方计算的相关概念进行了介绍和定义。

### 基于MapReduce立方体算法概述

随着数据立方计算的原数据逐渐增大，导致传统单机Data Cube算法表现出了极大的局限性，而基于MapReduce的数据立方体计算由于其吞吐量大，算法并行框架易于使用等特点，成为了近几年研究的热点，在基于MapReduce计算数据立方体算法中，由于处理的数据量巨大，导致逐层分摊计算消耗成为了比较理想的方法，而对于分层计算算法来说，主要解决的问题是网络带宽占用、物化路径选择、IO读写、数据负载均衡、整体性度量计算等问题。完全物化Data Cube 虽然可以支持快速的查询分析操作，但是需要在计算和存储方面付出代价。在这里为方便后续介绍，阐述相关数据立方概念。

Cubeid，一个Cubeid是指特定维度组合，例如对于数据立方维度有country、time、product三个维度，则其中<country，time，\*>就是一个cubeid，实际含义是观察数据的一种视角与关注点。数据立方晶格，数据立方晶格是具有层次概念的由所有可能的cubeid组成的概念结构，代表了所有可能的维度组合或者数据观察视角。对于上述三个维度组成的立方晶格如图所示



图2.3 数据立方晶格

Fig.2.3 data cube lattice

最小父cubeid，对于cubeid来说，其父cubeid可能有多个，对于分析数据来说，分析的维度组合需要有度量去量化，例如分析在某个商品的总销量，销量就是分析中的度量，而这个度量的计算，可以由多个父cubeid聚合得到，而消耗最小资源而得到的就是最小父cubeid。

典型的一个*n*维立方体来说，由立方体的逐层排列组合得到，即是自顶向下的底i层具有的cubeid个数为，设最终得到的数据条目上限为g，总的cubeid个数为s，则：

|  |  |
| --- | --- |
|  | 公式1.1 |
|  | 公式1.2 |

由此可见得到的可能数据量相比原始数据来说可能特别大，下面将从几个典型的MR算法介绍各类算法框架。

### 基于MapReduce数据立方算法介绍

MRNaive算法是一个比较经典也是比较简单的数据立方计算方法，它将数据立方计算任务简单的映射成一次MapReduce任务，Maper阶段一个元组将会产生所有的cubeid对应的中间元组，在Reducer阶段统一的去聚合生成所有的cubeid，产生最终的Data Cube，例如对于一个拥有*n*个维度的Data Cube，每一条数据记录，将会产生个<*Key*,*Value*>对。此算法优点是算法结构简单，一次MR任务提交减少了任务分配所占用的时间，同时一次读取原数据极大地降低了反复读取、存储结果数据或者中间数据所产生的IO消耗，对于基数以及cubeid所产生的组合基数不大的场景效果极为有效。但是缺点也显而易见，任务在中间过程中，很可能占用过多的网络带宽，而且容易在MR任务中的shuffle和排序过程中导致时间成本过高。Naïve算法伪代码如下。

算法2.1 Naïve算法描述

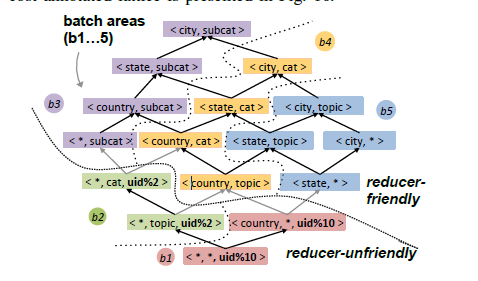
Algorithm2.1 Naïve algorithm description

|  |
| --- |
| 输入：原始数据集  输出：数据立方  MAP(e)  1 #e 是数据条目，C是数据立方晶格，R是Cubeid  2 let C be the Cube Lattice  3 for each Cubeid U in C  4 do k = U(e);  5 EMIT k ⇒ e  REDUCE/COMBINE(k, {e1, e2, ...})  1 let M be the measure function  2 EMIT K ⇒ M({e1, e2, ...}) |

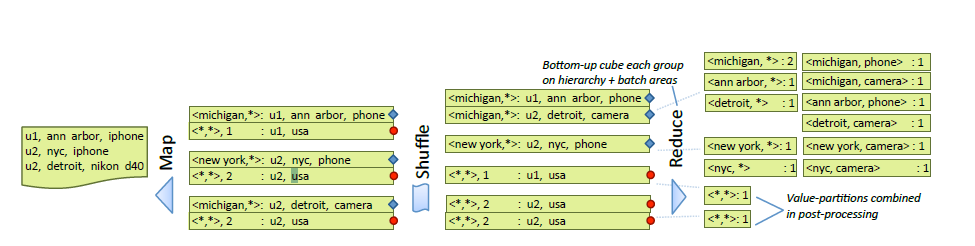
MRDataCube算法是由S Lee等人提出的针对朴素算法的改进，其算法将数据立方计算分为两个大的阶段MRSpread阶段和MRAssemble阶段；在MRSpread阶段，一个或者多个Maper task接收原始数据记录，简单的将各维度值拼接成键值对的Key，经MR阶段的Combiner聚合同一Maper task相同Key值的度量值，例如对于SUM度量，数据单元(*a*1, *b*1: 1）与（a1，b1；3）将被Combiner阶段合并成一条数据单元（a1，b1；4）；然后各Maper task产生的相同Key将被发往MRAssemble阶段的Reducer阶段等待处理对于a1，b1这样的属性发往Reducer形式为（cell；m1,m2,….）在Reducer阶段，相同Key值的度量值将被计算得到最终的顶层Cubeid的最终结果；然后根据顶层cubeid计算的结果，在Reducer阶段根据合并的顶层cubeid每一条记录，生成各个子cubeid。在MRAssemble阶段的输入时MRSpread阶段的输出，即是未合并的部分cubeid，这一阶段过程比较简单，只需要简单合并相同Key值就可以得到最终的完整cubeid。整个算法的数据处理过程如下：

相对于MRNaive算法，Arnab Nandi等人提出的MRLevel算法将整个Data Cube计算过程按数据立方晶格分层划分，从上层开始，每一层发起一次MapReduce任务和MRLevel任务，MR任务负责计算每层聚合值，MRLevel来确认最小父cubeid，每层输入数据来源于上层的处理结果，从而直接利用最小父cubeid的计算结果去做聚合。对于拥有ABC三个维度的Data Cube来说，使用MRLevel所产生的物化路径与层次如下图所示（右图默认是最小父cubeid）。在MRLevel基础上Arnab Nandi等人提出了与PipeSort相结合的MRPipeLevel，思想是利用传统PipeSort算法与MapReduce计算框架相结合，通过将整个数据晶格划分成多颗子树，每一个子树分配一个MR任务，并，通过cubeid共享前缀规则和跨子树共享排序来组成执行树。如图一个拥有三个维度的Data Cube的MRPipeLevel计算流程如下图所示：，MRPipeLevel算法优点是通过划分子树，可以增加程序的并行度，减少MR任务的提交次数，并且利用MR框架自动排序的优点很好的适应了PipeSort。同时继承MRLevel算法的优点，但是缺点是当维度增加时，子树划分可能过多，算法执行过程变得复杂，反而影响了算法的效率。

上述算法只是对代数度量进行了优化，而对于整体性度量来说，Arnab Nandi提出的MRCube算法第一次解决了MR框架下的整体性度量存在的问题，分析了整体性度量与代数式度量区别，MRCube提出为了解决数据倾斜问题，将数据立方晶格分割成多个Batch区域，提出了判断cubeid，此*cubeid*的维度值组是否是reducer-friendly、reducer-unfriendly的方法；通过抽样检验，每个Batch区域使用reducer-friendly与reducer-unfriendly进行标记，Reducer-unfriendly将使用MapReduce提供的partition函数被分成更多的reducer上进行处理，分区处理示意图如下。



MRCube总体采用BUC即自下而上的计算方式，这样对于不符合构建Cube的数据会被及时清除，有利于节省计算资源。对于整体性度量来说，以Count distinct 为例，MRCube将会使用需要计算count distinct的度量（在原文中也是维度）来hash到reducer，这样保证了相同的度量值会被分发给同一个reducer任务，所有reducer处理后的结果是可以直接相加的，MRCube处理流程如。对于MRCube算法，其缺点是对于增量Cube的情况并没有涉及在算法中，而且分区方法没有一个具体的过程，对于只考虑包含多个cubeid的Batch区域来说，通过简单的Hash分配给更多的reducer并不能很好的解决负载均衡问题。



## 本章小结

在本章中，首先对本文所用到的开源技术做了一个简单的介绍，接着对当前常用的数据清洗技术做了一个总结，接着在第三小节重点介绍了与本文后续联系紧密的数据立方体算法，并对其相关概念作了介绍，然后对基于MapReduce的此类算法要解决的主要问题进行了分析；最后对现有常用的基于MapReduce数据立方体计算算法逐个进行了介绍，并分析了算法执行过程，研究了算法各自的有缺点。

# 预处理系统关键技术

## 数据预处理系统概述

### 预处理系统流程

数据预处理主要考虑如何对原数据进行初步过滤或者错误值纠正等数据清洗一系列过程，主要保证数据在进行聚合之前的数据准确性，避免由不合理的数据对聚合结果造成较大的误差；对于经过数据预处理后的原数据，最关键的一步是对数据进行聚合预处理操作，这其中包括物化数据立方算法的设计、为缩减结果所占存储设计的维度编码方式等；经过聚合后的数据存放在基于Nosql的Hbase分布式数据库中，以供不同的开发人员使用；数据来源与处理流程如图3.1所示；在本小节中主要对数据清洗、数据聚合、数据立方存储三个方面做简单的流程介绍，所涉及到的数据如图3.1所示。

图3.1 数据流向

Fig.3.1 data flow

在本系统数据预处理流程中，数据清洗阶段是整个流程开始的前期准备，数据清洗阶段数据输入来源于企业内数据仓库中的基础数据层（DWB）区域与明细数据层（DWD）区域；对于数据仓库，基础数据层存放的数据主要来源于基础源数据层，存放的数据是对源数据区域（SDF）对不同业务方向进行的抽取，且主题广泛。汇总数据层（DWS）是对原有明细数据层的轻度聚合的结果。而本系统所提到的数据清洗流程主要应用于明细数据层与汇总数据层，主要对不同的维度值采用不同的清洗策略，对数据进行归一化标准化处理，特别是空缺值的填充部分，这一处理过程在系统实现章节会有详细的介绍。

在数据聚合节点，即是数据立方计算阶段，是本章节的重点。基于MapReduce的并行计算物化算法的设计也是预处理系统的一个重要环节，在本章以下内容将会介绍对清洗后的数据的处理流程以及数据立方物化算法的设计；其中介绍的Multi-RegionCube算法包括维度值编码、数据抽样、数据立方晶格物化区域划分与计算。其中Multi-RegionCube是针对代数度量设计的算法；对于Count Distinct整体性度量，在本章的1.5小节会讨论如何处理Count Distinct度量，物化流程将在后续小节详细讲解。

结果存储是对数据立方结果的存放过程，本系统采用Hbase数据库存储数据立方结果，主要考虑的是Hbase属于Nosql数据库，对数据能提供快速查询而且相对容量巨大，可以满足对数据查询的要求，存储阶段在系统设计与实现章节将会设计，在这里不再赘述。

### 关键问题定义

本文预处理系统主要满足业务人员快速分析的业务需求，而为支撑此需求，采用系统总体思想是空间换时间，因而系统面临的问题是如何快速聚合数据，构建数据立方，此问题包含两个子问题，其一是对仅包含代数式度量数据进行聚合计算，其二是对包含Count Distinct度量的数据进行聚合问题。对于第一个子问题，以下章节将通过介绍Multi-RegionCube算法来解决，对于第二个子问题一下章节将会对Count Distinct度量提出基于BitMap的解决方案。

#### 计算代数性度量存在问题

数据立方计算存在很多种分类，但从大体上分为部分物化计算例如冰山立方体、完全立方体物化。实现查询分析对时间的高要求，这里讨论对数据进行完全立方体计算，完全立方体计算具有如下问题。

1. 数据倾斜

数据立方在进行计算过程容易出现Key值分布不均匀，框架本身的Hash策略容易造成过多键值对块分配给相同Reduce处理模块，如果不加以解决，容易导致单个Reduce任务节点在处理并对落到本节点的键值对块做归并与处理时间过长，任务时间消耗取决于耗时最多的Reduce子任务，从而延长整个任务的执行时间。

1. 空间占用

完全数据立方与原数据相比，磁盘占用随着选取分析维度增加，呈指数级递增；对结果数据立方的存储压缩也是一个重要问题，其中基于MapReduce的并行处理框架数据立方体算法在处理过程中，会产生过多中间数据，这意味着磁盘IO的重复读写，网络带宽的重复占用，过多的中间数据会影响Map、Reduce端排序时间，甚至可能导致崩溃。

1. 计算效率

数据立方计算过程中，对于代数式度量和部分整体性度量来说，子维度组合聚合存在重复计算可能，因为子维度组合可以由其父类维度组合聚合得到，而父维度组合的确定也是其中一个需要解决的问题，所以BUC等自底向上的数据立方算法，可能处理完全立方体时会消耗更多的计算资源。

#### 计算整体性度量存在问题

计算数据立方过程中，代数式度量由于其简单累加性，可以高效累加计算，得到数据立方结果，但对于Count Distinct整体性度量来说，却不能累加得到，而阻碍整体性度量的最大因素是需要保留度量的不同值列表，并且快速去重，第一个子问题导致了中间结果所占存储空间过大，第二个子问题导致对CPU资源的需求过大。

## Multi-RegionCube算法

### 算法总体流程与定义

针对上一小节提出的数据立方计算所产生的问题，本节提出的对数据立方计算流程采取首先对源数据进行编码处理，对数据进行编码处理主要解决中间结果占用与网络带宽占用问题，减少最终完全数据立方磁盘占用，方便后续对数据相关操作；采用随机抽样确定数据分布情况，抽样目的是确定物化区域性质，确定最小父cubeid，减少重复计算；针对不同的物化区域采用不同的计算与负载均衡策略，总体采用自顶向下逐层计算策略。对于整体性度量的数据列，使用整数编码是为了计算Count Distinct，关于Count Distinct度量计算将会在下节讨论；在叙述算法流程之前，对以下需要应用的概念定义如下；对于算法总体流程如图3.2所示。



图3.2 数据预处理流程

Fig.3.1 Process of data pre-processing

数据倾斜包括某一cubeid所包含的待计算元组过多或者在一个cubeid中的某些数据分组过大的问题。数据倾斜问题将导致待计算的元组过多的分配给同一个Reduce任务，此Reducer任务运行同比其他Reducer任务消耗过多的时间，等待将导致整体MapReduc任务运行时间过高。

物化区域划分直接关系到MR任务的个数，以及整体计算效能的高低；对于同一cubeid可以提供聚合复用的父cubeid可能有多个，这时父cubeid的选取就会变得异常重要，计算相同cubeid，选取不同的父cubeid可能会产生比较大的性能差异。例如维度A是基数比较大的维度，而维度B基数小，在计算cubeid(\*，\*，C)的时候应选择cubeid（\*，B，C）作为聚合（\*，\*，C）的数据输入，如果选择（A，\*，C）作为输入数据，则将会导致计算资源有较高消耗，所以说使用合适的方法确定合并使用同一MR任务区域的确定以及选择合适的父cubeid十分重要。

作为支撑业务分析的Data cube可能涉及的数据量巨大，时间跨度较长；当处理此类数据时，如果不采用替换真实值来压缩数据cube的话，将会导致巨大的空间占用，例如一个基数为万级以上的维度，不做编码或者替换的话，面对几十亿甚至更多的场景，将可能多占用几倍甚至几十倍的空间。特别是拥有类似整体性度量例如count distinct类似的保存结果，存储压缩手段更是必须的。而在计算Data Cube过程中要产生的中间数据也可以使用编码技术去压缩从而缩减占用的宝贵带宽资源。以上总结了基于MapReduce预处理数据产生Data Cube所存在的问题，下面将介绍为了加快Data Cube计算过程，减少计算开销，均衡数据负载而提出的新的应用算法。

定义1 Cube模型，Cube模型定义了数据立方计算包含的维度列及度量列，以及维度与度量的数据类型与度量的类型，例如SUM，MIN等。

定义2 维度基数，维度基数指的是一个维度实际包含的不同真实值的个数，例如对于VID维度来说，其维度值有包含A、B、B、C则VID维度的基数是3。

定义3 Cubeid值组，cubeid值组是指实际元组或者由元组生成的属于不同cubeid的实际元组，例如对于cubeid<CNAME，CITY>是cubeid，而作为属于cubeid元组（livetv，武汉）属于<CNAME,CITY>实际值组。

定义4局部立方体，设数据立方cubeid全部集合为U，某一cubeid集合为G，若，则G即为局部立方体，且局部立方体是指由部分cubeid组成的多维数据结构，本身具有不完整性，是部分物化数据立方的结果。

定义5维度值组，维度值组是指属于某一cubeid的由此cubeid包含的各维度所属值组成的元组。

### 维度值编码

为了获取原始数据中各个维度值的不同值，算法首先聚合相同维度相同属性值的数据；对于企业数据来说，一般会有很多基础表来满足上层计算的要求，但是一般基础表包含的信息会很多，所以在算法处理过程中只会处理cube模型中定义到的需要编码的维度和度量作为Map阶段输出的中间结果值，其中维度以“维度名+值”作为Key，以入列标记作为Value，对于Reducer阶段将会简单丢弃相同key下的value值中包含入列标记的key值序列,Combiner阶段逻辑和Reduce相同；下面将会以一组数据来说明具体数据流，如下图所示。入列标记是方便后期对不存在的维度值进行插入现有序列的标记，用于区分维度值是否存在于列表中，对于Count Distinct的度量来说，是为了标记BitMap上的下标位，方便cube合并。

算法3.1 数据值去重算法

Algorithm3.1 Distinct Value Algorithm

|  |
| --- |
| *MAP(e)*  1 #e是数据元组，U是需要编码列  2 *for* *each* *column in U*  3 do  4 *k.key=column.name+column.value(e);*  5 *k.value = exist\_flag;*  6 *emit(k);*  7 *end do*  *REDUCE/COMBINE(k, {f1, f2, ...})*  1 *for all f in {f1,f2,….}*  2 *do* |
| 3 *if f1.exist\_flag = true*  *4 then*  *5 exit;*  *6 end then*  7 *end do*  8 *emit(k,padding);* |

由于MR并行处理框架并不适合处理为不同值增量编码问题，所在在这一阶段将在单节点上执行，算法将会在cube第一次建立时为每一个维度列和count distinct度量列维护一颗tried树，而保存tried树的目的是为替换原数据中的原始值节省磁盘空间。选择tried树的原因是tried树形结构可以将字符串比较的时间复杂度控制到，可以高度压缩原始字符串，缓解内存压力，其中n为tried树高度。对于不同的编码维度将会保留单独的维度tried树文件和Value-Index文件，Value-Index文件中记录的是属于此维度的值所对应的下标关系，便于反向查询和整体性度量BitMap文件的构建。

数据替换是作为Data Cube处理的预处理环节，使用整形值代替字符串类型度量和维度值，方便压缩数据存储所占用的空间；这一环节将启动一次MapReduce任务来处理替换。首先Map任务将会把trie树文件读入内存，查询相应维度值和countdistinct度量对应的下标索引,使用下标索引替换真实值。

### 抽样规则分析

大数据背景下的数据特征很难通过处理整个大数据来获得，特别是统计*Cubeid*值组的数据条目、分析维度组合值分布是稠密的还是均匀的等，这些内容只能通过数据抽样的方式了解大数据的数据特征以及分布；数据抽样方法是确定数据分布特征的关键，对于*Data* *Cube*物化计算来说，数据抽样用来划分合并处理的*cubeid*组，确定稀疏*cubeid*，确定父*cubeid*等；所以说数据抽样方法的选取对于减少*Data* *Cube*物化计算有很大的影响；下面将从维度组值稀疏性和*cubeid*内部数据特征的抽样上来做分析。

抽样规则首先要保证抽样数据与整体数据的大致数据特征，即是两者之间的相似性即首先需要确定样本容量。而抽样规则所确立的方法需要解决查找超大规模*cubeid*问题，以及*cubeid*内部稀疏程度问题，对于第一个问题查找超大规模*cubeid*问题，本文基于霍夫丁不等式来确定对于大数据数据集情况下，来确定抽样数据中观察到的属于某一*cubeid*的概率小于某一值的置信度。设数据集总量为*S*，样本空间为*N*，某一*Cubeid*为*C*，将一系列抽样过程看作是否属于*C*的伯努利实验，即是，数据在样本空间*N*中属于*C*的期望为，其中指的是若为超大*cubeid*所设定的阀值。设*X*指代样本数据集落入*C*的数据实际总数，则对于任意>0，根据霍夫丁不等式得，其中。

|  |  |
| --- | --- |
|  | 公式3.1 |

对于任何0<δ<1，0<1-δ<1,同上式等价变换如下形式：

|  |  |
| --- | --- |
|  | 公式3.2 |

根据[]得其实际应用较多的不等式：

|  |  |
| --- | --- |
|  | 公式3.3 |

将u带入得

|  |  |
| --- | --- |
|  | 公式3.4 |

令由此可知=0.1，则，取=0.0001S，则意味着如果某一cubeid包含多于个时，抽样显示此cubeid的在样本空间中出现的次数小于的概率小于0.01，霍夫丁边界即可得到此结论只要保证此时的样本数量大于万分之一，当样本中目标出现的个数小于，目标实际多于时的概率小于0.01，可以检测到超出的cubeid，设此cubeid组合为U，从而为自顶向下聚合计算寻找合适的父cubeid。而且以上抽样利用朴素Data Cube立方算法可以轻易地得到结果。也可以将此类推倒推广到探测具体的维度值组合，利用MRCube算法中采用的以Reducer处理能力的40%作为阀值进行检查规模比较大的cubeid值组，设得到值组G，用于作为避免数据倾斜的依据。

对于真实案例的包含五钟维度的数据立方，如VTYPE（视频类型）、VID（视频ID）、PTYPE（播放类型）、CITY（所在城市），PAY（付费类型），通过抽样得到的父cubeid完整物化的数据立方晶格如图3.2所示。



图3.3 数据立方晶格与层级

Fig.3.3 data cube lattice and cubeid level

经过抽样统计，一个cubeid确定的父cubeid可能由于抽样父cubeid大小相同有多个，可能的初步数据立方树如图3.3所示。



图3.4 确认父cubeid的数据立方晶格

Fig.3.4 data cube lattice with definite parent

由上节可知，维度值经过替换后读取的是整型值，整型值可以使用*hash*方法将不同的维度组编码值划分为[0,）…..[,NW]数据*Hash*段，其中*Nw*指的是某个维度组设定Hash长度，在抽样过程中即是样本总量，k是分段数，从而分别计算落入段中的count数来确定某一特定维度组数据特征的分布。这样做是因为在视频网站数据的处理过程中，一般分析的任务或者建立的cube模型很多时候只涉及很少的大基数维度，而一般成熟的cube业务模型为了防止完全物化数据立方占用过多存储空间，一般只包含的一个基数相对较大的维度，所以说确定维度值组的Key值分布可以防止稠密的数据范围落入相同的reduce任务处理中，在实际业务中能在一定程度上解决数据负载均衡问题，值段数k的选取可以参考在中的由每个Reduce节点所配置的资源而定的处理元组的能力来选取。



图3.5维度值组稀疏度

Fig.3.5 parse degree of dimension combination

通过以上抽样，为物化数据立方提供了数据处理提供了cubeid规模估计，数据值组合过大的cubeid值组为以下物化路径选取与物化cubeid区域的划分提供了依据。

### 物化区域划分与计算

本节主要介绍通过上一节抽样分析的结果，进行MR任务区域的划分问题，本节将通过一个用于生产中的实例cube模型来介绍对于一个数据立方晶格在此算法中如何进行划分物化区域的；为了方便叙述本节的cube模型维度包括VTYPE（视频类型）、VID（视频ID）、PTYPE（播放类型）、CITY（所在城市），PAY（付费类型）五个维度来介绍物化区域划分，对于度量为了方便介绍，度量取PLAY（播放次数），度量类型为SUM。通过上一节抽样分析，Multi-RegionCube算法将数据立方晶格的*cubeid*分为高基高量、稀疏、普通三类区域。

Multi-RegionCube算法将使用自顶向下的逐层算法计算出前两层cubeid聚合结果，按第二层cubeid，将确定最小父cuebid的晶格分为N个区域，其中N为维度个数，这里为避免重复计算充分利用最小父亲cubeid规则，规定划分出的区域，未计算的cubeid如果在抽样中全部表现为稀疏cubeid，则此区域为稀疏区域，或者对于包含维度W1、W2、W3…..Wn，设，若S是一个较小值，则cubeid也为稀疏型。对于高基数区域来说，即是包含高基数维度，且元组数超过设定的阀值的cuebid组成的，这样的cuebid在MapReduce并行框架容易造成数据倾斜，则会使用在抽样过程中得到的结果进行数据倾斜处理，其中层级与元组数的关系在实际生产环境的观察的对应关系如图3.4所示。



图3.6层级与元组数的关系

Fig.3.6 the relation of level and tuples’ amount

对于抽样过程中得到的三个区域划分如图3.7所示，从第二层开始划分有利于普通区域、高基数区域进行利用最小父cubeid优化原则减少重复计算。而且第二层Cubeid的个数为个，即为n个，层级与元组数之间的关系如图3.6所示，所以与逐层降维聚合比，稀疏区域聚合所承载的数据量不会过多可以一次计算稀疏区域内所有cuebid。

图3.7划分区域后的数据立方晶格

Fig.3.7 data cube lattice with partitioning area

稀疏型是指在使用cubeid在计算过程中数据量合并率比较高，例如对于cubeid<CITY，PAY>来说，CITY不同值数量保持在百级设为n，而PAY维度的不同值个数分为三个分别为0（试看）、1（付费看）、2（免费看），后续可能会增加不同含义的值，但是可以预见的是PAY的不同值总数m值很小，所以在构建数据立方过程中，最终此cubeid构成的合并元组数为m×n。可以看出对于此cubeid为稀疏型cubeid。对于稀疏型cubeid区域，其数据特点是经过一次数据立方体计算后，稀疏型cubeid数据量相对父cubeid来说数量缩减比较可观，所以Multi-Region算法采用maper端预先合并构建局部*cube*。对于此类型cubeid区域，本算法采用顶层cubeid作为输入，算法由顶层*cubeid*的输入的每个元组分化出符合稀疏cubeid的元组形式，组成新的维度组合值，发送到Reducer端进行聚合操作，经过聚合操作形成的数据立方只能算得上是部分cube，因为此cube只包含部分cubeid。对于稀疏类型cubeid的数据流处理方式如图3.8所示。



图3.8 计算数据立方晶格稀疏区域

Fig.3.8 the way to calculate spare area of data cube lattice

对于稀疏类型数据处理来说，算法充分利用了Map端Combine端做局部合并，Combine在MapReduce并行框架下是用于计算减少网络带宽占用的函数，它定义了对相同Key值的Value进行操作的逻辑，一般来说Combine与Reduce阶段的逻辑相同，只不过Reduce定义了对全局Key值相同的Value的处理逻辑，算法正是利用这一点，在Map端对每一条元组产出尽可能的属于稀疏区域cubeid的元组，在本地利用Combine阶段压缩值组，减少MR任务提交次数，因为MR任务在分发和调度本身将花销额外的资源，所以可以减少不必要的资源开销。稀疏类型区域的cubeid在计算过程中的具体数据流的例子如图3.9所示。



图3.9稀疏区域数据流

Fig.3.9 parse area dataflow

对于稀疏区域的算法描述如下。

算法3.1 Multi-RegionCube对稀疏区域算法描述

Algorithm3.1 Multi-RegionCube for spare area

|  |
| --- |
| *MAP(e)*  1 #e是数据元组，C是数据立方晶格，R是Cubeid  2 #S是稀疏区域集合、任务包含的稀疏Cubeid集合N  2 *for* *each* *e in SecondLevelCubeid*  3 do  4 *for each Cubeid Q in N*  7 *do*  8 *k.key=Q.nameDir+Q.Value(e);*  9 *k.value = e.Measure();*  10 *emit(k);*  11 *end do*  12 *end do*  *REDUCE/COMBINE(k, {e1, e2, ...})*  1 #M设为度量方法  2 *for all e in {e1,e2,….}*  3 *do* |
| 4  *for each in M*  5 *do*  6  7 *end do*  8 *end do*  9 *emit();* |

对于高基区域和普通类型区域数据立方体计算来说，由于不能像稀疏区域那样，在Maper端可以高效聚合元组，聚合后的结果对比Map端处理后的输入数据来说往往很小，所以适合上述策略。但是普通区域与高基区域由于不具备上述条件，所以算法尽量利用最小父cubeid原则减少计算量，采用自顶向下逐层计算的计算方式，对于高基数维度组合采用对维度组合进行hash到一个范围区间内，将此数据范围分割成等长数据段，对数据段使用贪心策略近似相等的合理分配到k个Reduce节点，对于非稀疏区域的稀疏cuebid采用Combine阶段压缩处理。此阶段数据处理如算法3.2所示。

算法3.2 Multi-RegionCube对非稀疏区域算法描述

Algorithm3.1 Multi-RegionCube for no-spare area

|  |
| --- |
| *MAP(e)*  1 *for* *each* *e in lastLevelResult*  2 do  3 *for each tuple in e.generatNextLevel()*  4 *do*  *5 if tuple.cubeid ;*  *6 then*  *7 key = tuple.cubeidName+tuple.cubeidValue;*  *8 value = measure(tuple);*  *9 emit(key,value);*  *10 end then*  11 *end do*  12 *end do*  *PARTITIONER(k,v)*  1 *if k*  *2 then*  3 *index* = *calculate(k);*  4 *return* *index*;  5.*end* *then*  *COMBINE(k, {e1, e2, ...})*  1 *if e.cubeid !parseCubeid*  2 *then exit;*  *3 end then*  *4 for all e in {e1,e2,….}*  *5 do*  6  *for each in M*  7 *do*  8  9 *end do*  10 *end do*  11 *emit();*  *REDUCE(k, {e1, e2, ...})*  1 *for all e in {e1,e2,….}*  2 *do* |
| 3  *for each in M*  4 *do*  5  6 *end do*  7 *end do*  8 *emit();* |

## Count Distinct度量计算

### 整体性度量计算概述

Data Cube分析是多维数据分析的一个强有力的工具，它允许用户快速分析大量数据，但是现存的Data Cube计算技术存在两个比较大的限制，导致Data Cube面对大数据分析场景，有些不尽人意。第一个是大多数Data Cube算法是基于单节点运算环境的，这一点在MapReduce环境下情况有了比较好的改善，而另一个限制来自于度量计算本身，对于MapReduce框架来说，它可以通过自顶向下避免重复计算高效完成聚合任务，例如SUM度量，对于cubeid为(A, B,\*)的SUM度量可以由cubeid为（A,B,C）的SUM简单累加得到。而整体性度量因其不能像代数度量那样简单的由顶层父cubeid聚合得到，所以一直是Data Cube计算的难点。

代数度量之所以计算存在难点，因为从直观上来看，代数度量所得到的值不能由父cubeid直接得到，而只能从原始cubeid本身数据聚合筛选得到，而对于Count Distinct整体度量来说，在视频网站中经常用到去重计数，则需要在每次计算*cubeid*时保留每个元组的不同维度值列表，如果元组数较多，则存储的中间值将会使内存过载；这是因为处理过程中每个元组产生的中间数据s的上限为Count Distinct （维度w）个，所以如果不对数据进行压缩处理，无论是对于中间数据传输还是空间占用来说都是无法容忍的，同时数据需要有快速标记某一维度值是否已经存在的特性，同时数据不应占用过多存储空间。

### 现有计算方法优缺点

Arnab Nandi提出的MRCube算法属于自底向上（bottom to up）的数据立方体算法，为了解决数据倾斜问题，采用了将数据晶格划分成reducer-unfriendly与reducer-friendly区域的方法同时在数据均衡上提出了两种值分区和批处理区域甄别方法，用于有效的利用MapReduce框架合理分发数据和平衡计算负载，同时在算法中MRCube第一次讨论了整体性度量（Holistic）与部分整体性度量划分规则，并提出了通过自上而下（BUC）的计算整体性度量的方法，在计算整体性度量的方法时，MRCube方法通过值哈希实现同一度量的相同值被分发到相同的Reduce节点，实现部分整体性度量的可加性。

但是MRCube在计算过程中实现的是自顶向上的计算方式，数据需要被多次访问才能求得度量，相对自顶向下类的计算方式在重复磁盘读写上有明显的差别，虽然基于BUC的算法在处理具有Having条件的限制性数据立方计算方面，但是在本系统中为了取得良好的查询性能，建立的是完全数据立方体，而且MRCube不能利用共享父cubeid来计算代数式度量所带来的优点，这对于计算靠近底层的Cubeid时相对于自顶向下的算法产生更高的开销。

在实际应用中，MRCube算法在一次计算不同值的过程中确实可以通过值分区方法，实现例如Count Distinct度量的相加操作，但是对于与日俱增的数据量来说不容易实现增量计算，特别对于规模庞大的数据进行值分区操作，将面对巨大的IO操作，而且MRCube对于生成数据立方后的增量计算并没有涉及，对于例如*Count* *Distinct*度量来说只是计算出了度量本身，而再次增量计算时将会再次使用时必须重新由原数据进行再次计算。所以针对Count Distinct度量本文提出使用BitMap来计算Count Distinct的方法，实现增量计算Count Distinct 度量，使用BitMap数据结构本身可以压缩每条元组所必须添加的Count Distinct度量值列表，在后续小节将介绍使用压缩方法的BitMap数据结构，这里不再详细介绍。

### 基于BitMap的Count Distinct度量计算

在视频网站用户行为分析过程中，Count Distinct度量是一个比较常用的度量方式，常用于对某些关键指标列进行去重操作，来分析不同值的个数的上升和下降来反映某些视频板块或者某些电视剧等的热度情况；虽然使用Hive等将查询语句翻译成MapReduce程序的方法可以解决这种需求，但是时效性很差，而且对于这种去重操作相当耗费系统资源；如上一节所述，对于此类整体性度量来说，缺乏有效的计算方式，并且现有的算法由于在增量数据立方计算过程中有缺陷，阻碍了算法在企业中的应用。所以本节针对Count Distinct整体性度量的特点，提出了使用BitMap数据结构存储Count Distinct度量值列表，计算此特殊整体性度量的方法。

BitMap是一种将一段区域值映射成bit位的数据结构，通过“0”或者“1”表示当前bit位表达的值是否存在，其是比较常见和重要的数据结构，经常用于位图索引、压缩数据方面，在某些关系型数据库例如Oracle等经常会用到位图数据结构。但是对于计算Count Distinct度量来说简单的位图索引数据结构并不能满足要求，即使位图本身对数据进行了压缩处理，中间产生的数据还是存在过大问题，因为简单位图必须映射所有的度量值范围即是0~n，其中n是全部不同值个数，如果度量列基数比较高则会产生很多稀疏位图；反映在数据立方计算上，即是使用传统的BitMap在计算Count Distinct时由于未经压缩会在靠近顶端的cubeid物化过程中产生过多稀疏Bitmap，造成过多空间和带宽消耗。所以在算法设计中融入RoaringBitMap作为压缩BitMap的策略，此是一种将32位索引划分到共享高16位的整数组成的数据块中的压缩算法；其中每个数据块代表个索引，每一个块最多包含个整数，否则将会使用bit位来表示，分别代表稀疏型与稠密性。对于不同的类型提供了两种不同的类型，一种是整型数组容器，另一种是bitmap容器来代表上述两种不同情况，而容器作为一级索引被存储在一个动态数组中，用于快速定位，其本身占用空间较小例如对于需要对小于100万的数进行索引，只需要最多16个容器，例如为了存储[， +100)、[2，3)、20从0到1000的倍数，相应的结构如图3.10所示。



图3.10 BitMap压缩示意图

Fig.3.10 sketch for BitMap Compression

同时数据块中保留了块中数据个数，所以对于查询总数来说，只需要将所有数据块中记录的个数简单累加即可，所以速度比较快，对于单个索引的查找问题，上述数据结构可以通过快速定位容器即第一级索引，通过快速定位到bit位。在进行位操作时，通过排序一级索引，对一级索引进行比较，而后对相同一级索引内部进行相应位操作，而对于无匹配一级索引则相当于直接插入此容器；由此在计算Count Distinct时，度量值列表中间结果所消耗的网络带宽与空间占用被降低到了一个可接受的范围，在实际应用过程中，用此BitMap所可承载的基数在百万级别，同时不太适合百万级别以上的场景，这对于一般的分析应用是可以接受的，CountDistinct度量由于是整体性度量，在上述提到的稀疏型区域等划分并不适合计算整体性度量，所以在整体性度量上将采用自顶向下计算策略，并使用上一节采用的抽样方法确定最小父cubeid减少重复计算，并且对于子cubeid<station,city,\*>与其最小父cubeid<station,city,pay>分别设为S、P来说，则对于P中的元组计算度量vid度量过程如图3.11所示。



图3.11 bitmap计算Count Distinct度量

Fig.3.11 calculate count distinct measure by bitmap

利用bitmap数据结构的特殊性，可以实现整体性度量CountDistinct如同代数式度量相似的“累加”效果，由于使用了bit位下标实现“0”、“1”代表的逻辑存在与不存在语义，并且bit位的与操作可以相当快的实现“累加”去重效果，所以计算速度很快。Bitmap结构使得算法可以利用父类cubeid元组的“累加”值直接计算子cubeid的CountDistinct操作，将整体性度量计算方法近似成代数度量，算法将会保留顶层cubeid的CountDistinct聚合结果，并与之前计算的数据立方聚合结果相加，作为增量计算的依据。

## 本章小结

本章介绍了数据预处理整个流程，并将其中数据立方聚合方面作为重点进行了分析，指出了其中存在的问题，并对存在的问题提出了解决方法，并使用多分区、多策略计算数据立方的算法，对于整体性度量Count Distinct提出使用特殊压缩结构解决中间结果空间占用较大的问题。

# 预处理系统的设计与实现

本章主要介绍预处理系统，首先从预处理系统解决的问题、产生背景对视频网站用户行为分析的数据预处理系统做简单的定位与介绍，随后本文采用先总体后细化的原则，首先对系统的总体框架进行了简单的阐述，随后对组成系统的各个组件的功能做了详细的介绍。

## 系统背景

随着互联网硬件与软件的快速发展，人们在视频网站所花费的时间日益增多，而对于视频网站本身来说，企业内部数据也呈急速增长的态势；而对于视频网站来说，用户访问量，各地区访问分布，访问量走势这些问题都需要对数据进行多维分析汇总，而利用现有技术例如hive很容易在一次分析人物过程中产生巨大的资源消耗，影响其他任务的正常运行，同时使用此工具时间成本也是很可观的投入；而在视频网站内部经常需要对时间跨度大的数据进行分析，得出总体趋势，时间跨度大意味着数据量巨大，这样的分析任务单靠临时任务汇总显然是不可行的，这种问题即使是半年内某个地区的访问量这样的问题都会扫描海量的数据，而且这种问题随着时间的推移结果也一直在变；而且对于分析主题数据，经常会重复使用，简单的手工抽取、提数、汇总既繁琐又容易出错不适合长期使用；并且对某些主题的分析，还有时间上的限制，需要快速得到结果方便决策。

所以本系统采用空间换时间的策略，预先聚合数据立方，代替单步分析任务对大跨度时间数据进行聚合计算；同时数据立方本身支持数据多维分析任务的需求，为上层应用提供针对某种主题的快速分析与聚合操作。本文数据预处理系统是针对数据预聚合处理而设计，作为企业内部从数据仓库层到数据集市层的数据立方聚合中间系统。

## 系统开发环境

### 系统开发语言

对于系统开发而言，语言选择有多个，但是目前Java语言使用广泛，

### 系统开发工具

## 系统总体设计

基于上述系统背景，将数据预处理系统总体分三层为数据抽取层（ETL）、数据立方构建层、数据集市，严格来说数据立方构建层的存储层即是数据立方计算的结果存放层，这里为了表达清晰，将结果层归并为数据集市，作为结果存放的数据区域；总体系统架构如图4.1所示。



图4.1 architecture of preprocessing system

Fig.4.1 预处理系统架构

输入层作为ETL层，ETL层负责数据清洗、加载和转换，主要是去掉与数据立方计算不相关的维度和度量，只保留与用户建立的数据立方模型相关的维度和度量列，并对数据进行空缺值等的填充，统一度量等级，统一称谓和IP与所属地址转换等工作；数据抽取对象是数据仓库中的数据仓库明细层(DWD)与数据仓库轻微汇总层(DWS)，细节数据层数据按主题进行组织，保留历史数据，数据粒度与生产系统一致，除了满足当前数据服务外还可满足一定未来不可预见的应用需求，数据汇总层是细节数据到数据集市过度的数据层，是对细节数据轻微汇总后的数据，所以在多数情况下，数据的抽取发生在DWS层，只有DWS层不能提供分析支撑时，才会发生在DWD层；处理完毕的数据将会由ETL层加载到指定数据存储区域，作为系统的输入数据进行进一步的加工，即数据立方聚集计算。

数据立方构建层是本数据预处理系统的核心内容，是数据立方聚合计算的核心部分，其中数据立方构建层由任务管理层、聚合引擎层、存储层、元数据管理层组成；其中任务管理层主要负责数据立方任务的构建管理任务，包括任务执行状态监控、任务类型管理、数据立方聚集计算任务的新建、提交、修改、删除，数据立方模型编辑等功能；聚合引擎层是数据立方聚合的关键层，是对上层任务管理层的支撑层，负责生成执行链，按顺序执行MapReduce聚合任务，聚合引擎层的具体算法与流程在第三章中有所描述；存储层是系统存储结果数据立方数据的地方，所使用的是Hbase数据库，存储层将数据立方结果格式化成Hbase格式文件例如HFile，导入Hbase数据库中，并管理数据立方Hbase表空间。元数据管理层主要负责管理系统中元数据信息，元数据信息主要包括数据立方模型描述，分析工程信息，数据立方存储位置与表信息，数据立方源数据信息等，元数据是系统关键部分。

## 系统核心模块设计与实现

本节介绍本数据预处理系统核心子模块的设计，

### 任务管理层

### 聚合引擎层

### 元数据管理层

### 数据存储层

## 本章小结

# 实验

# 结论与展望

## 工作总结

## 未来展望

# 参考文献

# 致 谢