密级：保密期限：

北京郵電大學

硕士学位论文

题目：分布式多层次移动感知多媒体数据管理与检索系统的设计与实现

学　　号：2012111490

姓　　名：李莹

专　　业：软件工程

导　　师：王文东

学　　院：网络技术研究院

2014 年 12 月 21 日

独创性（或创新性）声明

本人声明所呈交的论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他教育机构的学位或证书而使用过的材料。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了谢意。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名：_____ 日期：_____

关于论文使用授权的说明

学位论文作者完全了解北京邮电大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属北京邮电大学。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，允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存、汇编学位论文。

非保密论文注释：本学位论文不属于保密范围，适用本授权书。

本人签名：_____ 日期：_____

导师签名：_____ 日期：_____

分布式多层次移动感知多媒体
数据管理与检索系统的设计与实现

摘要

随着智能手机的不断普及，移动网络的迅速发展以及各种精彩应用的不断推出，互联网已经从电脑过渡到智能手机、平板等移动设备。过去的几年见证了移动互联网用户规模的大幅提升，流量规模和业务收入规模呈现爆发式增长，移动互联网已经取代了传统互联网成为行业新的领导力量。不可否认，移动设备已经成为最普及的网络接入口，是收集用户信息，下发“知识”[1]的最好载体。这迎来了新的数据采集模式-参与式感知（participatory sensing）。参与式感知过程中，参与者自由地选择收集模型和收集数据的种类。通过利用参与者个人拥有的移动设备来收集信息，参与感知创造性地将传统的以社区为基础的数据收集和科学的理念，带到移动互联网环境中。

本论文在总结现有参与式感知存储系统研究成果的基础上，设计并实现了一个分布式的移动感知多媒体数据管理和检索系统，简称 MDR 系统。该系统首先需要承担高并发的数据上传，并高效地存储移动设备收集的感应器数据（例如，经纬度、光线、温度）和多媒体数据（例如，图像、声音、录像）以及自动实现数据分布式存储和数据多备份，其中需要针对多媒体数据的存储检索做一些优化；其次为实时应用（例如，采集数据的可视化）提供快速的随机读写接口；最后为非实时应用（例如，数据分析与挖掘）提供流式读取和结果模型存储接口。最终的研究目标是构建一个快速、稳定、扩展性良好的存储检索一体式系统。

论文首先对参与式感知、现有移动感知数据存储系统的研究成果和相关理论技术背景进行了简要介绍；接着详细描述了移动感知存储系统 MDR 的功能和非功能需求分析，并对整个系统进行了详细设计和实现；然后，对整个系统进行了详尽的功能测试和性能测试，表明整个系统达到了预期的设计目标；最后，论文对全文进行了总结，对未来的工作进行了展望，并总结了作者在研究生期间的所有工作和成果。

关键词　参与式感知 分布式存储检索系统 高并发写 多媒体 实时应用 数据挖掘

DESIGN ANDIMPLEMENTATION OFASCALABLEDATA REPOSITORY SYSTEM FOR MOBILE PARTICIPATORY SENSING BASED ON HBASE

ABSTRACT

With the popularity of smart phones, the arrival of 3G and beyond 3G era and a rich mixture of apps, the Internet has gone to mobile devices from computers. The last few years have witnessed a significant increase of the proportion of Mobile Internet subscribers in all mobile users and an explosive growth of Mobile Internet business income. Undoubtedly, the Mobile Internet has become a potential alternative to replace original Internet and turn into a new industry leadership. Therefore, the mobile devices have become the most popular network access port, preferable carriers for collecting participants' and environmental information and issuing knowledge. The recent proliferation of mobile devices has ushered in a new pervasive data collection model - participatory sensing. In participatory sensing, individuals explicitly select what sensing modalities and data to be used. By enabling people to interact easily with devices they use everyday, participatory sensing brings the idea of traditional community based data collection to an online and mobile environment.

In this thesis, I propose a Scalable Data Repository System for Mobile Participatory Sensing, referred to MDR. MDR is responsible for undertaking high concurrent upload operations, and intelligent reorganization of uploaded data including numerical value such as collected by gravity sensor, gyroscope, light sensor, etc. and multimedia data such as photos, sound records and even videos in the first place. In the meantime, it provides efficient data retrieval for data mining and realtime applications. The objective of this research is to build a fast, stable, efficient and scalable storage and retrieval system.

Firstly, this thesis introduces the related theories and technical background of participatory sensing and HBase. Secondly, requirement analysis and system design of the scalable data repository system for mobile participatory sensing

based on HBase are depicted in detail. Thirdly, functional tests and performance tests are both adopted, which indicates that the entire system achieves the prospective design goals. In the end, the thesis summarizes the whole work of the system, looks forward to future work, and summarizes the work and achievements of the author during the post-graduate period.

KEY WORDS: participatory sensing; distributed storage and retrieval system; high concurrent write operations; multimedia; realtime applications; data mining

目录

第一章　绪论

1.1. 课题背景

随着智能手机的不断普及，移动网络的迅速发展以及各种精彩应用的不断推出，互联网已经从电脑过度到智能手机、平板等移动设备，从固定的办公室、书房走向人们的口袋。过去的几年见证了移动互联网用户规模大幅提升，流量规模和业务收入规模呈现爆发式增长。2013 年年底，中国移动互联网用户规模超过 5 亿。不可否认，移动互联网已经取代了传统互联网成为行业新的领导力量。移动互联网业务的特点不仅体现在移动性上，即用户可以在任意时间和地点享受互联网带来的便捷，还表现在比传统互联网拥有更丰富的业务种类、更个性化的服务和更丰富的用户数据。

如今智能手机等移动设备已经普遍配备了方向感应器、光线感应器、温度感应器、陀螺仪等多种传感器，可以帮助设备更好地识别周边环境，提高用户体验。众多传感器的出现使得手机成为一个拥有真正智能和感知能力的设备。例如重力传感器可以让我们通过摇晃手机便能控制游戏中车辆的方向，光线感应器可以让我们即使在强光下也能看清屏幕上的字体，方位传感器能够让我们进行更精准的定位。这种感知能力将会成为今后智能手机的一个重要的发展方向。微软亚洲研究院的研究项目 MoodScope[2]甚至能借助智能手机的各种传感器来侦测用户的心情，并作出恰当反馈。例如，当侦测到用户愤怒时，它就会推荐一些有趣的视频，帮助用户平和心情。

由于移动设备的快速增长以及配备了完备的传感器，参与式感知[3]在传感领域获得了越来越广泛的重视。参与式感知利用日常移动设备，例如智能手机，来形成一个具有互动性、参与性的传感器网络，使公众可以随时随地收集、分析以及分享本地知识。目前参与式感应已经被广泛地应用到城市计算、公共健康以及社会资源管理中。

通过让人们能够与他们日常使用的设备轻松互动，参与感知创造性地将传统的以社区为基础的数据收集和科学的理念，带到移动互联网环境中。这带来了很多优点：首先，由于参与传感利用现有的传感（移动设备）和通信（蜂窝或 WiFi）基础设施，部署的成本可以忽略不计；其次，使用智能手机作为传感器可以非常方便且经济可行地进行规模扩展；再者，手机平台已经广泛地建立了应用程序商店形式的分销渠道和成熟的软件开发工具，使应用的开发和部署相对容易；最后，通过设计出的应用程序来发布消息，就有可能显著地实时提高参与者的日常生活品质。

通过移动设备收集的数据具有多种形式：话筒可以收集用户的声音，摄像头收集图像、录像，GPS（Global Positioning System，全球定位系统）传感器收集用户的位置，光

线传感器收集光照信息，如何将这些多种形式的数据进行有效管理和快速检索是之后深入进行数据分析的保障和前驱。

“分布式多层次移动感知多媒体数据管理与检索系统的设计与实现”课题是为了解决在服务器端如何对大量的、多种格式的数据进行合理地管理并提供快速的检索能力。

1.2. 课题研究现状

近年来参与式感应在传感器领域获得越来越广泛的重视，并被广泛地应用到城市计算、公共健康以及社会资源管理等多种应用中。城市计算是一个通过不断获取、整合和分析城市中多种异构大数据来解决城市所面临的挑战（如环境恶化、交通拥堵、能耗增加、规划落后等）的过程[4]。参与式感知利用移动设备上配备的丰富的传感器可以为城市计算提供丰富的、多维度的数据。这些数据经过分析和处理可以用于提高人们的生活品质，帮助城市更好地发展，例如分析城市的空气质量，指导人们是否适合户外活动。另一方面，参与式感知利用人们随时携带的移动设备可以判断用户是否进行足够的户外活动，甚至利用 APP 可以测量用户血压、心率等，这些身体数据是实现公共健康的基石。例如，通过分析以上数据可以在流感蔓延之前侦测到流感的发生，并通过移动感知网络向人们传播抵抗流感的有效药物。

无论是应用到城市计算、公共健康还是社会资源管理，数据是核心也是基石。如何高效安全地存储参与式感知收集的数据是进行数据挖掘分析，指导改善人们生活的前提。在探索感应数据的存储方案上，很多研究做出了具有意义的的贡献和尝试。

MPSDataStore[5]尝试将收集数据直接存储在移动设备自身的存储器内。当收到数据请求时，MPSDataStore 根据对应关系将请求转移到拥有相应数据的移动设备当中去。通过将数据存储分布到单独的移动设备上，很大程度上地压缩了存储成本。但是，这也带来了一系列的缺点：首先，移动设备的离线会造成了短暂的数据不可用，而移动设备的格式化会造成永久的数据丢失，对于存储系统这是最严重的事故；其次，移动设备承担查询并返回数据任务，这将导致移动设备电量迅速的流失以及流量的大量上涨，影响参与者的日常使用；最后，收集数据存储太分散，以至于很难对这些数据进行数据分析和数据挖掘，用来提取更深层次的有用的知识。

另一方面，一些研究[6,7]选择了集中式存储架构来解决参与式感知数据的存储和检索问题，RDBMS（Relational Database Management System，关系型数据库）首当其冲。拥有丰富的索引技术和存储引擎架构，RDBMS 可以高效地处理 CRUD（Create, Retrieve Update, Delete，创建，读取，更新，删除）操作。但是对于感知数据存储系统，最具挑战性的问题是，如何承担高吞吐量数据上传和更新，并同时对大量的收集数据进行分析和挖掘。以 MySQL，世界上最流行的开源数据库，为例，在每秒进行数万次写数据库

的情景下，它会成为整个系统的瓶颈。由于 MySQL 等关系型数据库对海量数据的存储支持不够，当数据达到一定量之后，存储和检索的速度会变的非常慢。更坏的情况是，当并发量比较高或者请求的数量比较大时，MySQL 等关系数据库往往难以承受，响应速度变得非常缓慢甚至于造成宕机，数据丢失等恶劣的影响。虽然我们可以通过先进的设计来缓和高并发带来的压力，例如：设计合适的索引、数据库的水平和垂直分区、采取分布式缓存、甚至部署到商用计算机中，但是这样的优化操作的设计，部署和维护都是非常昂贵的。

不同于关系型数据库，key/value 数据库，比如 Bigtable[8]的或者开源的 HBase[9]，已被证明可以扩展到承担数百万的更新操作，同时依然保证高容错性和高可用性。HBase 已被应用到许多应用研究中，例如：LBS（Location Based Service，基于位置服务）[10]和 RDF（Resource Description Framework，资源描述框架）[11,12]，来承担存储系统的重任。此外，HBase 对海量的和多维度的数据的分析和挖掘提供了很好的支持[13,14]。

根据 HBase 在之前的应用研究中的成功部署，以及 RDBMS 在高并发的情况下表现的差强人意，本课题尝试基于 HBase 建立感知数据的存储系统。保证整个存储系统具有高效的写性能，并支持海量数据的存储和检索。

1.3. 课题主要研究内容

1.3.1. 数据的存储格式

主要需要存储的数据包括：传感器如重力感应器，陀螺仪，光线感应器，它们的数据格式为一系列的数值；媒体数据如图像、视频、音频，它们的数据除了自身外，还包含了地理位置，设备名，时间等附加信息。

1.3.2. 数据的分布式存储

分布式存储系统，是指将数据分散存储在多台独立的服务器节点上。传统的集中式存储系统，存储服务器容易成为系统性能的瓶颈，降低了系统的可靠性和安全性，不能满足数据挖掘等大规模数据应用的需要。分布式存储系统采用可扩展的系统结构，利用多台存储服务器组成大规模存储集群，它很好地解决了集中式存储系统面临的问题，提高了系统的存取性能、可靠性和数据安全性。

HBase 是一个具有高可靠性、高性能、高可伸缩性的分布式存储系统，利用 HBase 技术可在廉价 PC 上搭建起大规模结构化存储集群。它是 Apache 软件基金会 Hadoop[15]项目的一部分，运行在 HDFS[16]（Hadoop Distributed File System，Hadoop 分布式文件系统）之上。MapReduce[17]是一个众所周知的并行编程模型，被广泛用于处理大规模数据集。用户只需要定义一个 map 函数处理键值对并产生中间结果，以及一个 reduce 函数，

它结合了相同的中间键相关联的所有中间值，产生最终结果。图1-1展示了完整的Apache Hadoop生态系统。

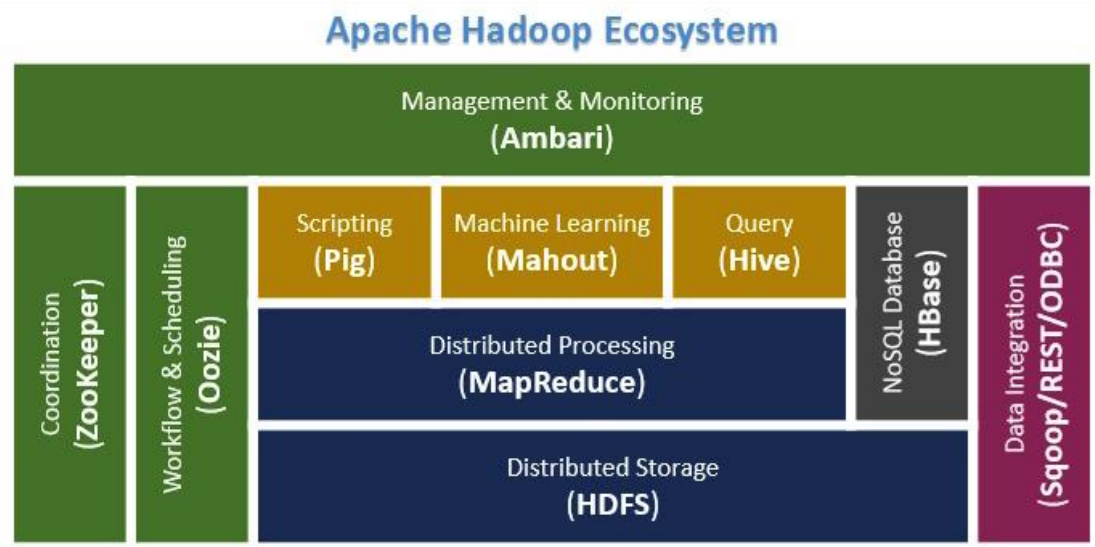

图 1-1 Apache Hadoop 生态系统[18]

HBase可自动完成请求重定向和数据库的水平分区。请求重定向指将后台数据的多个副本分布在不同的数据服务器上，一台充当Proxy角色的服务器通过特定的算法将读操作重定向到不同的服务器上，以此提高了系统的并发能力。为了让所有副本的数据都一致有效，所以写操作必须扩散到所有副本中。数据库的水平分区则指将同种数据类型中的记录通过特定的算法进行分离，从而可以部署在不同的服务器上。

1.3.3. 检索结果的格式

这里的“用户”并不是指真实用户，而是利用检索结果进行下一步数据处理的程序。如何将数据传递给下层程序是必须要考虑的问题。检索系统提供丰富且友好的数据检索接口，用户通过调用这些接口就可以取到相应的结果集。结果集的数据类型也是多样的，既可以是类似于 MySQL 的检索结果集，也可以转化为更具可视化的文本类型，例如XML（Extensible Markup Language，可扩展标记语言）格式、JSON（JavaScript Object Notation，JavaScript 对象表示法）格式等，甚至可以利用 Google 的开源技术 Protocol Buffers 或者 IDL （Interface Definition Language，接口描述语言）高效地对数据进行串行化和反串行化。

1.3.4. 多媒体文件的存储

图像、声音、视频等多媒体文件并不适合直接存储到HBase的单元格中，因为HBase的数据块大小限制为 64KB。如果通过重新配置数据块大小使其可以存储完整的多媒体

文件，HBase 的随机读取性能将会受到影响，官方指导手册推荐数据块最大不超过 1MB。可在真实场景中，即使认为大多数图片文件在 1MB 以内，但是视频文件普遍超过 1MB，从而需要对基于 HBase 多媒体文件的存储技术进行改进[19,20]。

1.3.5. 多层次 - HBase 二级索引支持

HBase 虽然在扩展性和性能方面表现卓越，但是并不能提供类似 SQL 丰富的查询方式（select * from table where col = val），只能通过单一的行键进行数据检索。也就是说，要完成这样的查询任务，HBase 只能通过遍历全表，即使通过 MapReduce 进行并行计算，仍然浪费了大量的计算资源，也使得延迟变得严重，无法应用于实时应用。

为了解决这个问题，我们试着借鉴 RDBMS 的索引技术，为 HBase 同样建立二级索引支持，使得 HBase 可以胜任实时应用的存储系统。

1.3.6. FIFO 消息队列的设计

通过 FIFO 消息队列，首先移动设备的数据上传和更新操作都以异步模式进行，提高了用户体验，这样更好地鼓励用户加入到参与式感知项目当中。其次，消息队列的平滑作用，使得即使在上传和更新请求的高峰期也不会造成数据丢失。最后，对数据库进行批量插入操作，表现的性能优于逐个单记录插入。

1.3.7. 与 RDBMS 进行性能对比

系统验证过程中，通过对比 MDR 系统和 RDBMS 在多个数据读写接口的性能表现，证明 MDR 系统不仅表现出更好的可靠性、扩展性，还拥有更高效的数据读写接口来支持实时应用，出色地承担了参与式感知项目的数据存储与检索任务。

1.4. 论文结构

论文的结构安排如下：

第一章　绪论。介绍了本论文的课题背景、研究现状和作者的主要研究内容等。

第二章　相关技术与理论。讲述参与式感知的概念和架构，Apache HBase 及其依赖的 HDFS、Zookeeper 的相关理论和技术背景，以及开发过程中用到的通信和数据格式等相关技术。

第三章　系统需求分析和概要设计。首先介绍了参与式感知项目的总体架构，阐明 MDR 系统承担的责任，然后详细说明 MDR 系统的功能性与非功能需求；接着针对需求，对比了 HBase 与 RDBMS 数据库在存储架构、扩展性、数据可靠性的表现；最后，还对系统进行概要设计，包括模块划

分、数据格式、数据结构以及接口的定义。

第四章　系统的详细设计与实现。根据系统的需求分析和概要设计，详细阐述了 MDR 系统的设计与实现，包括存储表和接口的设计与实现，最后，同时利用 MySQL 实现来进行对比试验。

第五章　系统的测试与验证。首先介绍了系统的测试环境，然后分别从功能性和性能等多方面对系统进行测试和验证。

第六章　结束语。对全文进行总结，提出了对未来工作的展望。

第二章　相关技术与理论

2.1. 参与式感知

2.1.1. 概念

参与式感知利用日常移动设备，例如智能手机，来形成一个具有互动性、参与性的传感器网络，使公众可以随时随地收集、分析以及分享本地知识[21]。目前参与式感应已经被广泛地应用到城市计算、公共健康以及社会资源管理中。

2.1.2. 架构

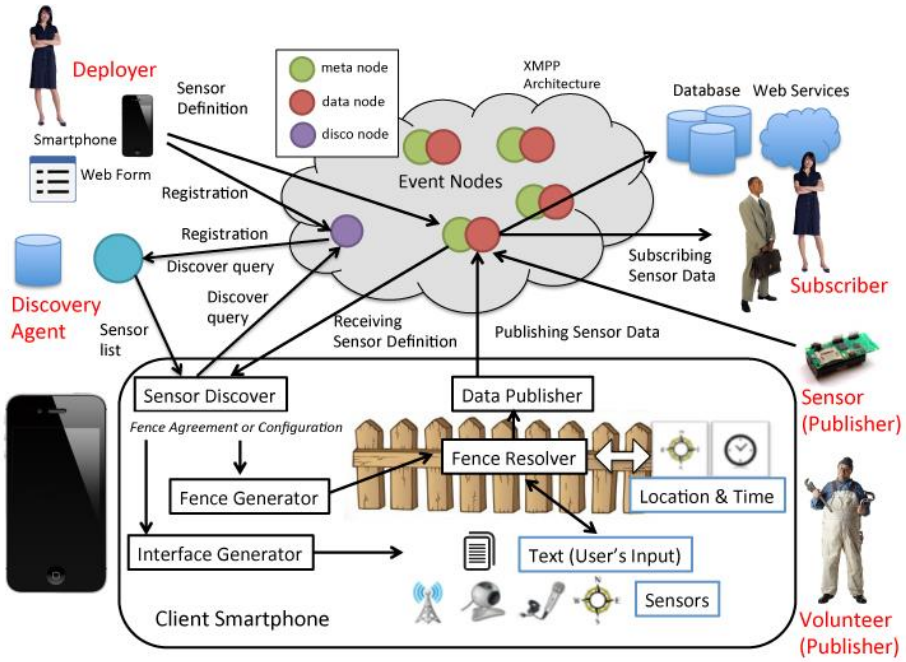

图 2-1 Participatory Sensor Andrew 参与式感知架构图[22]

图 2-1 描述了卡耐基梅隆大学提出的参与式感知的架构--Participatory Sensor Andrew。借鉴 Participatory Sensor Andrew 的架构图，介绍参与式感知项目的主要角色。首先开发者需要开发移动设备上的收集数据 APP，并确定收集数据类型，以及收集策略。然后志愿者参与进来，通过手中的移动设备不断收集数据，并最终将收集的数据上传至服务器集中存储。

服务器中运行着很多数据分析的进程，这些进程是上传数据的订阅者。每当有新的收集数据进入服务器，这些进程尝试读取收集数据中自己感兴趣的属性，并不断迭代自己的数据挖掘过程。

每当数据分析进程结束迭代过程，则会将运算出的知识发布给所有的或者特定的志愿者，帮助他们更好地享受生活。

2.2. 数据格式

2.2.1. EXIF 格式

EXIF[23] (Exchangeable Image File，可交换图像文件) 是一种专门为数码相机照片设计的照片存储格式。这种格式可以用来记录数字照片的属性信息，例如相机的品牌、型号、相片的拍摄时间、相片的拍摄地点、相片拍摄时相机设置的参数，如光圈大小、快门速度等。同时它也保存了相片的拍摄数据以及照片的格式化方式。此外，还可新建一些新的标签，保存用户自定义的数据。

拿 JPEG 格式相片文件来举例，JPEG 文件以字符串“0xFFD8”开头，并以字符串“0xFFD9”结束。文件头中有一系列称为“标识”的“0xFF??”格式的字符串，这些字符串并不会被 JPEG 使用，EXIF 把信息记录在这些字符串上。如图 2-2 所示，EXIF 为 JPEG 记录了相机信息以及图片的拍摄数据。

图 2-2 EXIF 示例

2.2.2. JSON 格式

JSON [24]是一种轻量级的数据交换格式，它既容易读写，对机器来说也容易解析和生成。和 XML 相比，JSON 更容易阅读、更容易解析、占用空间更少。C、C++、Java、Python、Ruby 等语言都有相应的 JSON 库支持，JSON 采用的是与编程语言无关的文本格式，这使得 JSON 成为理想的数据交换格式。JSON 由两种原子数据结构组成，结构为 {key：value, key：value,...}的键值对以及结构为[value1, value2, …]的数组，通过这两种结构可以组合成复杂的多样的数据结构。图 2-3 展示如何利用 JSON 格式封装了三年二班的学生姓名和住址信息（城市、街道和邮政编码）。

```
{“class”:”class2 grade3”, //  班级名
 “students” : //  学生列表
      [{"name":"Michael","address":{"city":"Beijing","street":" Chaoyang
Road","postcode":100025}},
      {"name":"Tommy","address":{"city":"Shanghai","street":" Nanjing
Road","postcode":2111102}}]
}
```

图 2-3 JSON 示例

2.2.3. Protocol Buffers

Protobuf[25]是由 Google 开源的一种数据交换的格式，通过将结构化的数据序列化，广泛数据存储和数据传输。Protobuf 独立于语言和平台，Google 为其提供了 java、c++ 和 python 三种语言的实现、编译器以及库文件。由于它是一种二进制的格式，比使用 XML 进行数据交换更高效。可以把它用于分布式应用之间的数据通信或者异构环境下的数据交换。作为一种效率和兼容性都很优秀的二进制数据传输格式，可以用于诸如网络传输、配置文件、数据存储等诸多领域。

用户创建 proto 文件后，如图 2-4 所示，通过 Protobuf 内置编译器编译生成包装类，该包装类封装了属性的读写以及序列化和反序列化操作。

```
message Order
{
  required int32 time = 1; //  订单时间
  required int32 userid = 2;//  顾客 id
  required float price = 3;//  价格
  optional string desc = 4;//  说明
}
```

图 2-4 proto 文件示例

2.3. Apache HBase 及其相关技术

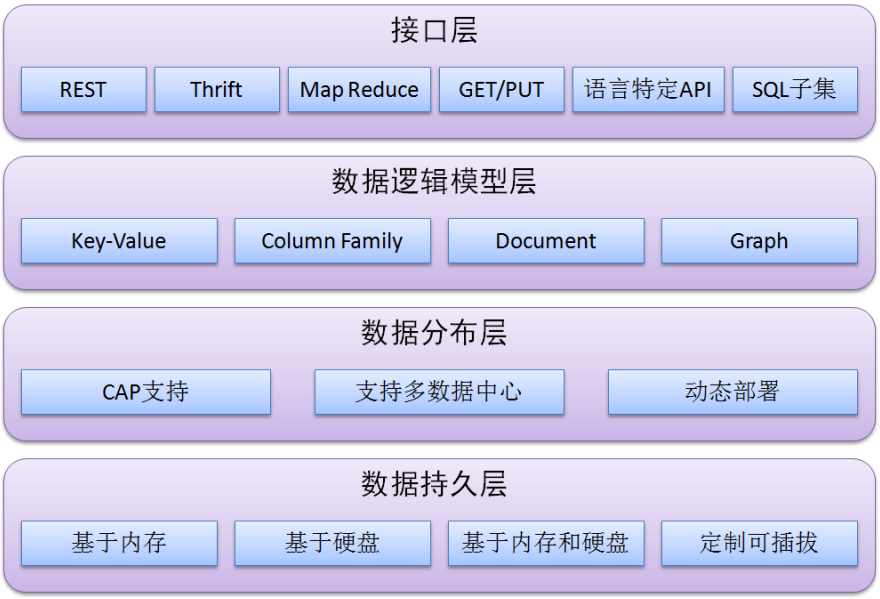

图 2-5 NOSQL 整体架构

NOSQL 型数据库的整体架构如图 2-5 所示，主要由以下几部分组成：

- 接口层：供上层应用进行数据读写的接口
- 数据逻辑模型层：描述数据的逻辑表现形式
- 数据分布层：定义数据如何进行分布
- 数据持久层：定义数据的存储形式

我们分别从以上方面介绍 HBase 相关技术和理论知识。

2.3.1. 数据逻辑模型

HBase 将数据存入一张由行键、列键和时间戳索引的表中。每一行记录都拥有一个全表唯一的行键，记录按照行键的字典序排序进行存储。与关系数据库不同的是，HBase 中每一行可以包含任意列，所以在 HBase 中可以出现某一行包含上千列而下一行只包含一列的情况。零个或多个列组成一个列族，表创建后列族的数目和名字不能再发生更改，不同的列族存储在不同的文件当中，却可以随意添加新的列。因此，列由<family>:<label>唯一确定，其中< family >代表列族名，<label>代表列族下特定的列名。在 HBase 中根据行键和列键可以唯一确定一个存贮单元，称之为单元格(cell)。单元格中的数据是没有类型的，通过二进制字节码的形式存贮。每个单元格的数据可以有多个版本，版本之间通过时间戳来区分和索引。每个单元格中，不同的数据版本按照时间倒序的排列，也就是说，最新版本的数据排在最上面。开发者可以以列族作为单位设置版本保留策略。值由{row key, column(=<family> + <label>), version} 唯一确定。HBase 的数据模型如图 2-

6 表示。

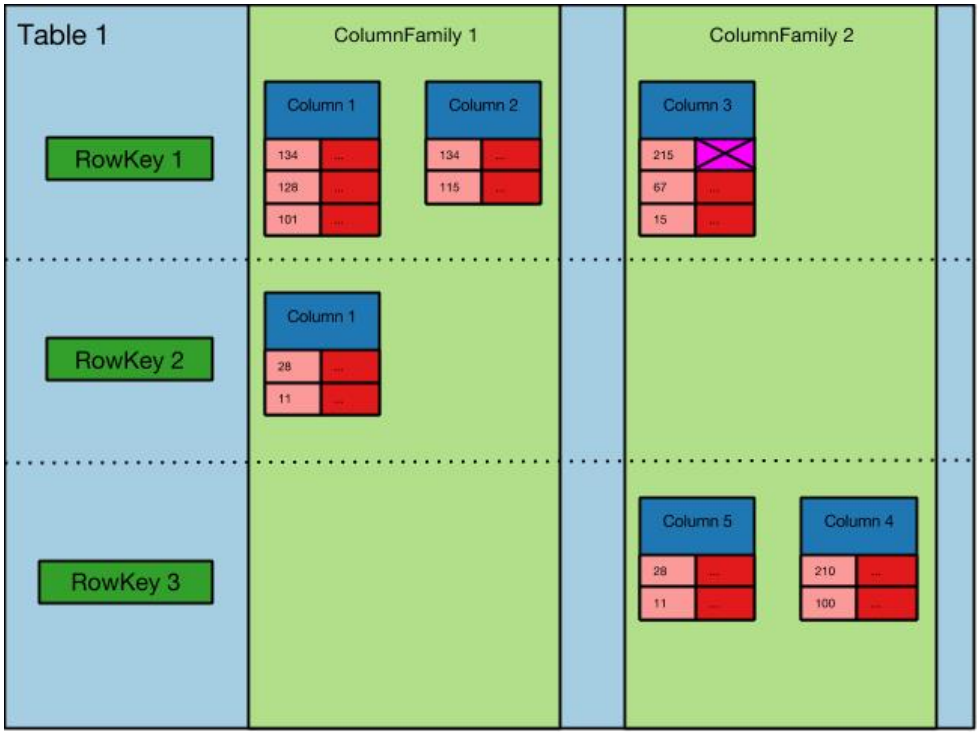

图 2-6 HBase 数据模型

RDBMS 提供多维数据丰富的查询功能的支持，但可扩展性差，而 key/value 型数据库拥有良好的扩展性，确丧失了联合处理多维数据的能力，例如 HBase 不支持绝大部分的条件查询和 Order By 排序。HBase 中只能通过过滤器在服务器筛选值，系统需要检查每一个单元格来判断是否满足用户的筛选条件。因此，我们需要精心设计表结构，甚至通过添加额外的，由特定的属性索引的表来实现高效查询。另一方面要考虑的是如何使 HBase 能够满足存储多媒体文件的需求。如我们所知，多媒体文件，例如照片、录音和视频通常太大以至于不适合直接存储在单元格当中。

HBase 将表水平分割成多个部分，每一部分称之为数据分区（Region），Region 是 HBase 中扩展和负载均衡的基本单位。每个 Region 包含了一定范围（rowkey）的数据。最初的时候，每个表只有一个 Region，随着表中的记录数的不断增加，超过一定的阈值时，Region 就会在中间位置分裂，形成两个新的，大小大致相等的 Region。每一个 Region 只能由一台 Region 服务器加载，每一台 Region 服务器可以同时加载多个 Region。

2.3.2. 架构模型

HBase 组件和架构如图 2-7 所示。

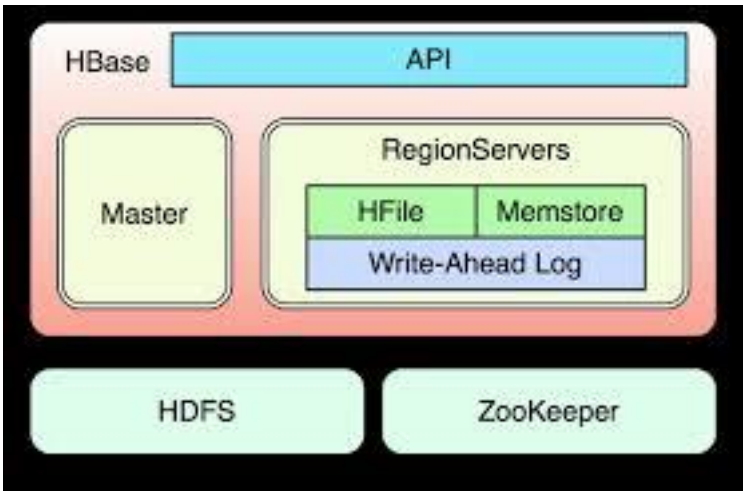

图 2-7 HBase 组件和架构

HBase 沿用了 Bigtable 的主从模式，当前版本的 HBase 只有一个主节点（HMaster）在运行。HMaster 负责将 Region 分配给各个 Region 服务器，维护整个集群的工作状态，并且均衡 Region 服务器的负载状态。HMaster 自身不负责数据存取，所有的读写请求以及操作均由 Region 服务器负责。HMaster 通过 ZooKeeper 来获取各个 Region 服务器的工作状态，当某台 Region 服务器发生故障后，HMaster 将失效的 Regions 重新分配到其它正常工作的 Region 服务器上。

Region 服务器负责为它们服务的 region 提供读写请求，HBase 所有数据存储在 HDFS 中，也提供了拆分超过配置大小的 region 的接口。所有和数据相关的操作，客户端都是通过 API 直接与 region 服务器通信来完成。

2.3.3. 数据持久层 - HDFS

HDFS 是 Apache Hadoop Core 项目的一个子项目，是 GFS（Google File System，Google 文件系统）的开源实现版本，被实现适合运行在普通硬件上的分布式文件系统。HDFS 把复杂的原理和实现隐藏在系统内部，对外看来，它就像传统的文件系统一样，通过文件路径来对文件执行 CRUD 操作。通常，HDFS 被部署在成千上万台的服务器上中，文件系统把全部数据中的一小部分分别存储在每个服务器节点上。当部分节点出现了故障，HDFS 也能迅速检测出错误并进行错误恢复，提供了良好的容错性。HDFS 的主要应用场景是流式的读取数据(而非随机读取数据)，然后进行批量处理(而非交互处理)。需要注意的是，当前版本的 HDFS 规定，文件在创建后，一旦关闭写入，就不能继续更改数据了。即 HDFS 上的文件提供的访问模式是一次性写、多次读。这样的设计是为了简化了数据的一致性问题，使得高吞吐量的数据读取变成可能。HDFS 之所以适合大数据处理，是因为它能能承担 TB 甚至 PB 级别的数据，存储数目大于百万的文件，集群节点数可以超过 10,000。

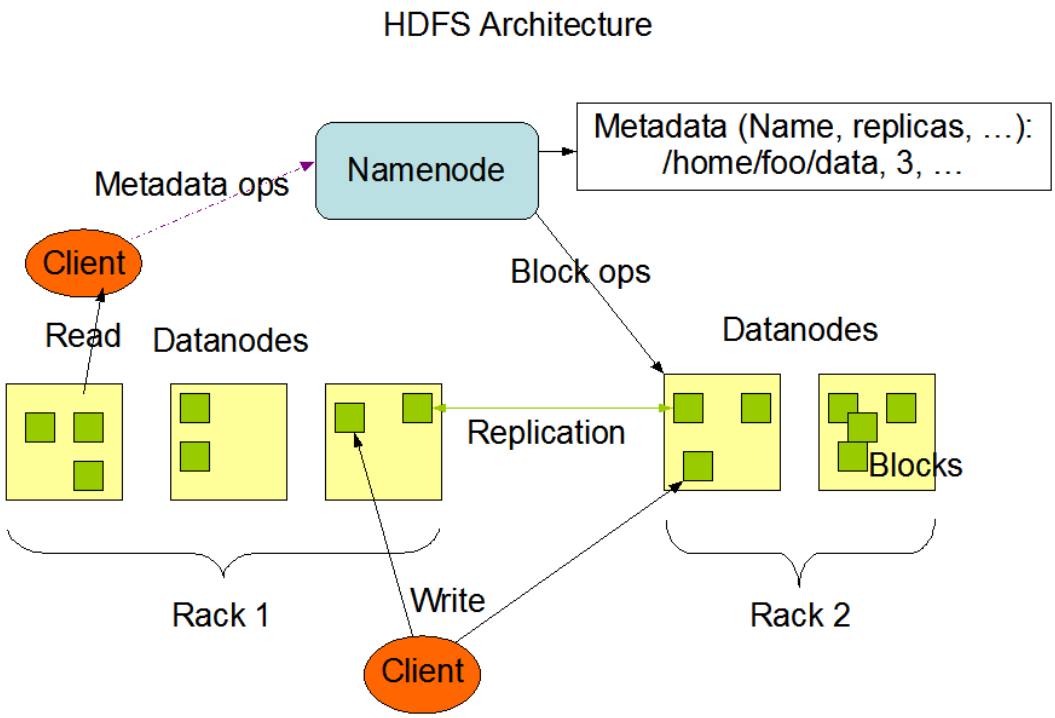

图 2-8 HDFS 体系结构[26]

HDFS 是主从体系结构的，如图 2-8 所示。一个 HDFS 集群拥有一个 Namenode 和若干个 Datanode，其中 Namenode 的职责是管理文件系统中的元数据，而 Datanode 负责实际的存储数据工作。Secondary Namenode 辅助 Namenode 并分担其工作量，紧急情况下可辅助恢复 Namenode。

文件被分割成固定大小的数据块，且每一个数据块由一个全局唯一的 64 位标识符唯一确定。Datanode 存储数据块在本地文件系统中，可以通过数据块标示符和偏移量定位并进行读写。为了提高可用性，通常每一个数据块都会在多个 Datanode 中进行多备份，默认备份版本数量是 3，用户可以更改配置文件来获得更多或者更少的备份版本数。Namenode 管理文件系统的所有元数据，包括，命名空间、访问控制信息、文件到数据块的映射关系以及数据块的访问路径等，除此之外还对数据块租约进行管理，对孤儿数据块进行垃圾回收，以及在 Datanode 之间进行数据块迁移。集群中的每一个 Datanode 都会周期性地向 Namenode 发送心跳包和块报告，Namenode 由此来验证块映射及判断每一个 Datanode 是否处于正常工作状态。一旦 Datanode 停止发送心跳包，Namenode 会标记其为宕机状态，然后客户端不会再向它作任何新的数据请求。Datanode 宕机后，存储在宕该 Datanode 上的所有数据将不再有效。Datanode 的宕机最直接的影响是，造成某些数据块的副本数量减少。Namenode 会不断监测那些副本数不足的数据块，如果副本数小于阈值，就重新复制到新的节点，这样集群的可用性才能不受影响。

2.3.4. Zookeeper

Zookeeper[27]是 Apache Hadoop 的一个子项目，应用在 Apache HBase、Apache Solr 及 LinkedIn sensei 等项目中。Zookeeper 是一个为分布式应用提供一致性服务的软件，它根据 Google 发表的<The Chubby lock service for loosely-coupled distributed systems>论文实现。在分布式应用中，由于使用者很难方便地使用锁机制，而基于消息的协调机制则不适用于某些应用，所以需要一种新的协调机制来解决这些问题。而 Zookeeper 就能提供一种可靠的、可扩展的、分布式的、可配置的协调机制来统一系统的状态。利用 Zookeeper 可以构建一个有效防止单点失效以及处理负载均衡的分布式应用系统。Zookeeper 主要是用来解决分布式应用中经常遇到的一些数据管理问题，如：统一命名服务、状态同步服务、集群管理、分布式应用配置项的管理等。

如果把 Zookeeper 看作是一个文件系统，那么文件系统中的所有文件则形成一个树状结构，Zookeeper 维护着这样的树形层次结构。树中的节点称为 Znode，Znode 具有一个唯一的路径标识，如/Member1 节点的标识就为/GroupMembers/ Member1。每个 Znode 都有一个与之相关联的 ACL（Access Control List，访问控制列表），用于决定用户可以操作的类型。Znode 分为短暂性节点和持久性节点，当创建短暂性节点的客户端与 Zookeeper 之间的会话过期后，短暂性节点会被自动删除，而持久性节点的节点即使在会话过期后仍旧存储在系统中。

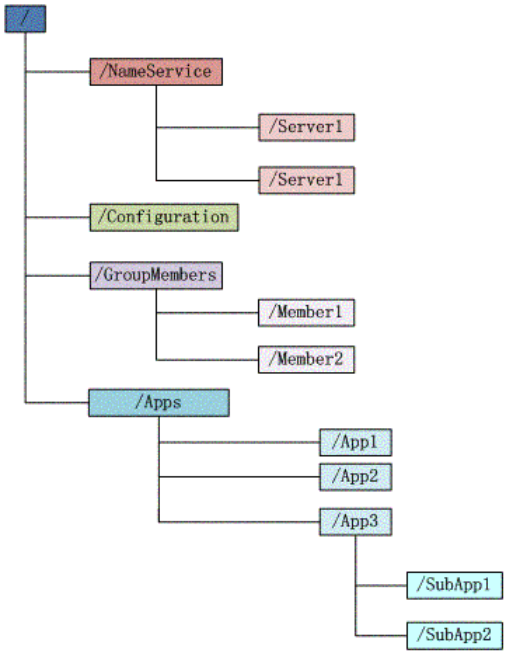

图 2-9 Zookeeper 的目录结构

Zookeeper 是由一组 Zookeeper 服务器构成的系统，客户端连接到一台 Zookeeper 服务器上，服务器会为这个客户端建立一个会话，通过这个会话客户端可以发送请求、接受响应、获取观察事件及发送心跳。Zookeeper 由一台领导者服务器和若干个跟随者服

务器组成：领导者管理写请求，领导者将收到的请求广播给所有的跟随者，当超过半数的跟随者修改数据并持久化后，领导者才会提交这个更新；读请求则由跟随者负责响应，同时跟随者把客户端提交的写请求转发给领导者。在整个 Zookeeper 系统运行的过程中，如果领导者出现问题失去了响应，那么原有的跟随者通过原子广播（Zookeeper Atomic Broadcast）协议，将重新选出一个新的领导者来完成整个系统的协调工作。Zookeeper 同步数据的流程如图 2-10 所示。

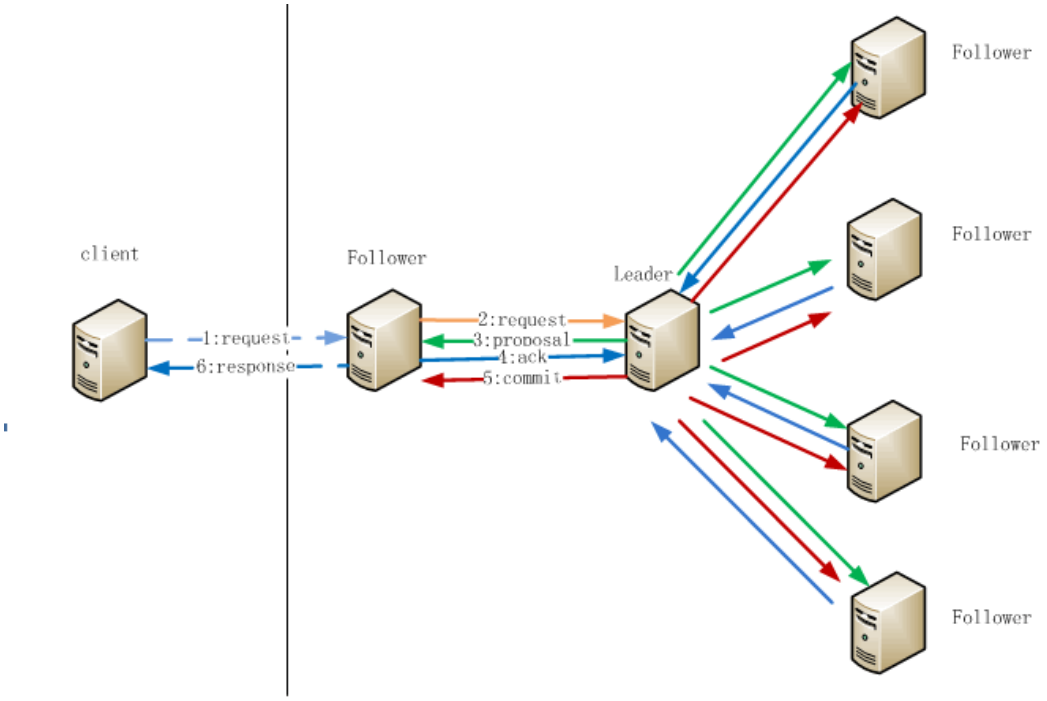

图 2-10 Zookeeper 同步数据的流程

客户端查询和维护管理命令，可以直接与跟随者交互，及时响应。而创建 Znode、更新 Znode 存储内容、更新 Znode 的 ACL，开启会话和关闭会话等操作都需要上图完整的 6 个过程。其中客户端与跟随者的通信协议采取 NIO（New Input/Output，新输入输出端），而领导者和跟随者之间采取 TCP/IP 协议。

2.4. MySQL

MySQL 是一个主流的开源的关系型数据库管理系统，被大家广泛使用。MySQL 数据库系统使用最常用的数据库管理语言-SQL 来完成进行数据库的操作。MySQL 的主要优点是安装体积小、运行速度快、总体拥有成本低廉，且开源和可以免费使用，所以成为了众多中小型网站的首选数据库。MyISAM 和 InnoDB 是 MySQL 常用的数据存储引擎。MyISAM 是旧版本 MySQL 的默认存储引擎，优点是访问速度快，但是不支持行锁和事务安全。从 MySQL-5.5.5 开始，InnoDB 存储引擎成为了 MySQL 的默认存储引擎。

它提供了具有提交、回滚和崩溃恢复能力的事务安全。此外，InnoDB 除了全表锁，还支持行锁，适合承担高并发的写操作。

2.5. RMI

RMI[28]（Java Remote Method Invocation, 远程方法调用）是一种针对 Java 编程语言中用于实现远程过程调用的 API。它可以让程序调用运行在另一台机器上程序的方法。RMI 的远程方法调用可以使 Java 编程人员能够在网络环境中分布式执行操作。这种机制的出现，给分布式系统的设计和实现都带来了便利。只要按照 RMI 的规定和要求设计程序，使用不必关心方法如何被远程调用等细节。

RMI 会把参数和返回对象都进行序列化，这样就可以在网络上进行传输，传输到另一端后，还需要进行反序列化构造出同序列化前相同的对象实例。RMI 目前使用 JRMP（Java Remote Messaging Protocol，Java 远程消息交换协议）进行通信，JRMP 是专为 Java 的远程对象制定的协议。用 Java RMI 开发的程序可以运行在任何支持 JRE（Java Run Environment，Java 运行环境）的平台上。同时 RMI 使用 Java 内置的安全机制来保证执行程序对用户系统的安全性。

2.6. 本章小结

本章首先了介绍参与式感知的基本概念和架构，开发中使用的数据格式技术，然后详细阐述 Apache HBase 的数据逻辑模型和系统架构模型，并简要介绍了 HBase 依赖的 HDFS 和 Zookeeper 的核心原理和技术，最后介绍了关系型数据 MySQL 以及开发中使用通信技术。本章介绍了课题需要涉及的一些基本知识，为课题之后的需求分析、设计和实现打下了基础。

第三章　系统需求分析与概要设计

在之前的章节中，对开发 MDR 系统的背景知识和相关技术进行了详细介绍，余下章节将主要阐述开发 MDR 系统的过程，包括需求分析、概要设计、详细设计、系统实现以及功能性和非功能性测试。

3.1. 参与式感知项目总体架构

为了更清晰地阐述 MDR 系统的功能性需求和非功能性需求，我们首先先简略地介绍整个参与式感知项目的总体架构，以及 MDR 系统所处的位置和需要承担的责任。参与式感知项目的总体架构图如图 3-1 所示。

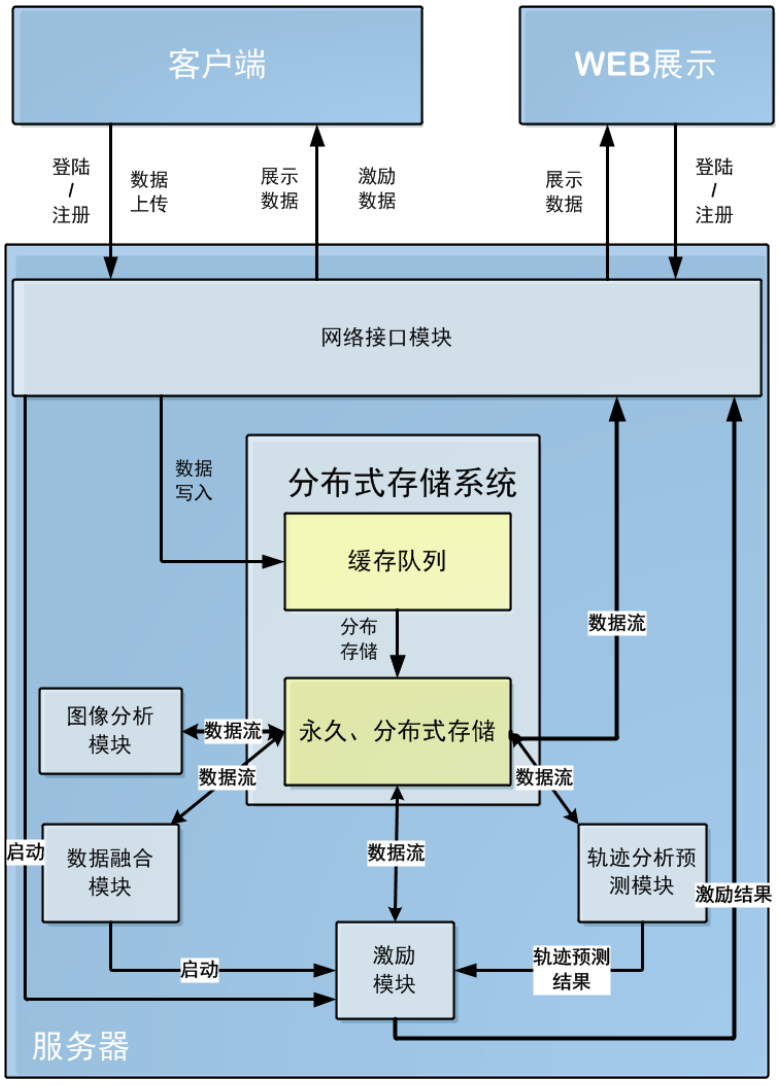

图 3-1 参与式感知项目总体架构

参与者首先通过客户端 APP 或者网页进行用户注册，之后通过客户端 APP 进行数

据采集并自动选择合适的场景（考虑网络状况、电池状况等）将数据上传至服务器。采集分为主动采集和被动采集，被动采集按照采集策略自动开启并采集需要的感应器数据，主动采集由参与者主动进行拍照、录音、录像等操作并将数据保存本地等待合适的场景进行上传或者直接上传。

网络接口模块功能负责接受客户端上传的数据，并将数据写入 MDR 系统；同时负责接受和处理客户端或者 web 展示系统发送的 HTTP 请求(包括注册/登陆和展示请求)；以及主动下发激励结果给参与者。

MDR 系统首先负责与网络接口模块进行交互，包括用户管理、实时存储用户上传的数据以及客户端和 web 端的数据展示。除此之外，MDR 系统分别为图像分析模块、数据融合模块、激励模块、轨迹分析预测模块设计存储表结构，并提供可靠的高效的读写接口。

启动激励模块运行有两种情况，分别是：数据融合模块发现数据缺失或 HTTP 展示请求发现数据缺失。激励模块启动后，调用轨迹预测模块获得轨迹预测结果，并从 MDR 系统获得其他所需数据，运算得出激励结果，最后调用网络接口模块主动下发给客户端。web 展示系统根据地理位置展示近期内地图上各 POI（Point of Interest，兴趣点）点的 PM2.5（Fine Particulate Matter，细颗粒物）值，以及在该地点采集的照片、光照等信息。

3.2. 功能性需求分析

MDR 系统首先负责实时存储来自参与者收集并上传的数据，其次对上传的数据进行再次的整理和组织（例如合并冗余信息、筛选无效数据、建立二级索引等），最后根据图像分析模块、数据融合模块、激励模块、轨迹分析预测模块等应用模块的具体需求和使用场景设计存储表的结构，并为以上应用模块提供可靠高效的数据读写接口。

读写接口分为流式读取、随机读取以及存储应用模块计算的中间结果和最终结果。流式读取指一次读取大量的数据来进行数据挖掘，不要求实时性，例如轨迹分析预测模块读取特定用户的所有地理位置信息；随机读取指读取少量的数据，例如读取用户的密码权限等，对实时性要求比较高；最后，MDR 系统需要为挖掘模块计算的中间结果和最终模型提供数据库写支持，例如图像分析模块需要存储用来预测 PM2.5 的模型。具体的系统用例如图 3-2 所示。

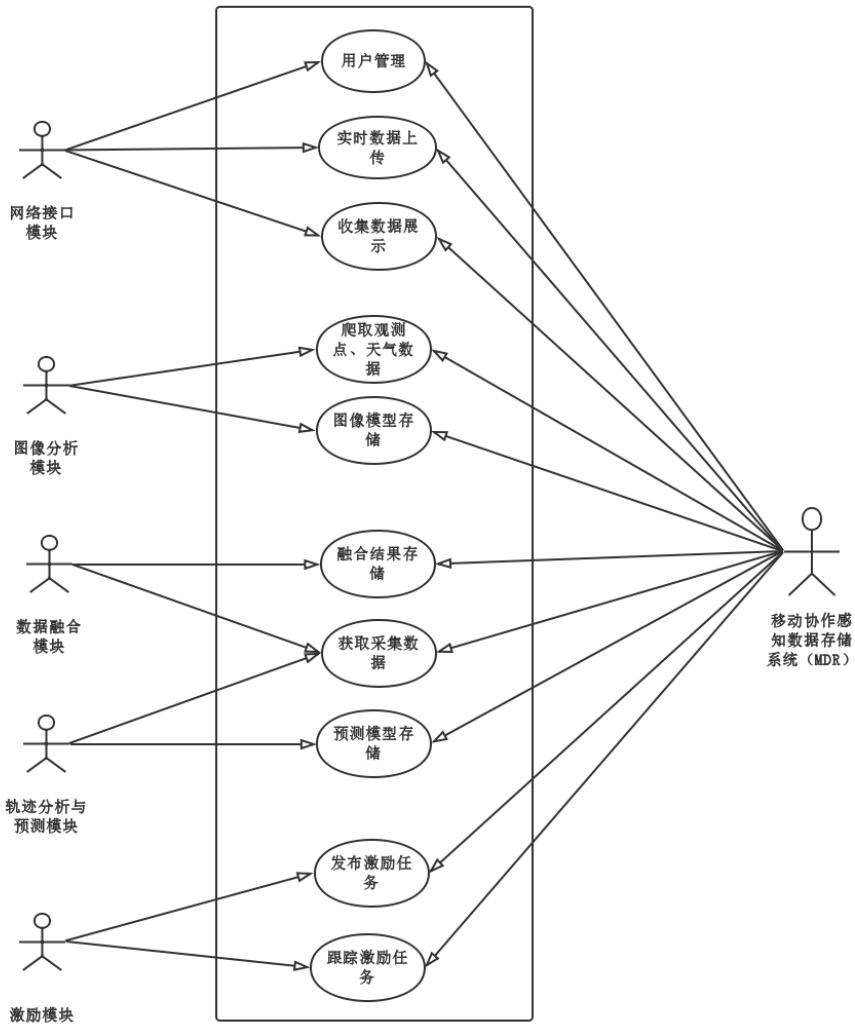

图 3-2 MDR 系统用例图

MDR 系统主要为应用模块提供以下主要功能：

- 用户管理

用户需要首先在系统中进行注册，之后用户上传的所有数据都会和用户 id 相关联，这样可以根据用户的上传数据的价值对用户进行奖赏，针对用户有针对性地推送激励，以及根据用户移动的轨迹点序列进行用户轨迹分析和预测。用户管理功能包括用户注册和用户登录功能，除此之外还管理用户的权限、优先级以及收益等。

- 实时上传数据

上传数据分为感应器数据（经度、纬度、光照、温度、声音以及方位等）和多媒体数据（图像、录音以及视频等），所有的感应器数据和多媒体数据都标有相应的用户 id 和数据的收集时间。感应器数据通过 JSON 格式进行编码，图像或者其他文件数据通过 EXIF 技术添加属性。

据我们所知，HBase 可以自动完成数据的多副本和分布式存储，但是图像或其它多媒体文件并不适合直接存入 HBase 的单元格中。业界有多种方式来存储和管理多媒体数据。以图像为例，最显而易见的方法，每个图像保存为单独一个 HDFS 文件，文件路径存储在 HBase 中。不幸的是，HDFS 被设计成适用巨大的文件（通常是 64MB），设计大文件是为了减少元数据的数量，从而使得 Namenode 能够加载所有元数据到内存中。但单个图像为约 1~2MB 的大小，这比 HDFS 块大小小得多，从而导致巨大的元数据以及检索变慢。为了解决这个问题，HDFS 允许打包多个小文件到 HAR（Hadoop Archive[29] ，Hadoop 文档文件）文件来降低 Namenode 的内存使用情况，同时还允许透明地访问原来的小文件。然而，HAR 文件在创建之后无法进行任何改变。要添加或删除里面的文件，则必须重新创建 HAR 文件，不适合频繁的数据改动。所以我们需要探索如何高效透明地完成多媒体文件的分布式存储。

- 收集数据展示

由参与式感知项目总体架构所知，收集数据展示分为 web 展示和客户端展示。首先用户采集并上传的数据可能包含很多冗余的信息，例如用户同一时间点连续采集了多条数据，而这些数据都提供相同的信息，通过将冗余的信息进行合并可以获得更精简的数据，得到的模型具有更好的鲁棒性。

web 展示模块通过聚合采集数据，在地图中形成多个 POI 点，并展示 POI 点下收集到数据，包括照片、光照信息、噪声数据以及 PM2.5 值，所以要求用户上传的照片包含经纬度信息，不包含经纬度信息的照片是无效的数据，需要被筛选出。当地图被放大或者缩小时，聚合算法将粗粒度的 POI 点分裂成多个子 POI 点或者将细粒度的多个 POI 点聚合成一个父 POI 点。例如，最普遍的读取数据接口，获取一周内某 POI 点下收集的所有采集照片。不幸的时，HBase 并不提供快速的针对非主键的特定属性的检索能力，只能遍历上传的所有图像，判断是否属于该 POI 点，性能非常低效。对于 web 展示来说，低效的查询会严重地损害用户体验。要求 MDR 系统提供高效的针对属性的查询能力。

同样的，客户端会展示最近一段时间所有用户上传的图像，形成照片墙，鲜明地对比不同城市、不同地点的 PM2.5 情况。要求 MDR 系统同时提供快速的、针对时间的检索。

- 爬取空气质量、天气数据

图像分析模块需要爬取并保存城市里所有观测点的在每个整点时刻观测到的 PM2.5 浓度、PM10（Inhalable Particles，可吸入颗粒物）浓度以及 AQI（Air Quality Index，空气质量指数），城市在每个整点的温度、湿度、降雨量、风速以及压强等。这些数据和上传的图像一起被用来建模。要求 MDR 系统提供对爬取数据的快速存储以及快速检索能力。

- 图像模型存储

图像分析模块建模结束后，图像分析模块需要为不同的手机型号存储并时常更新模型，至此分析阶段结束。预测阶段，当新图像到来，会从数据库中读出新图像对应的手机型号在拍照时刻之前的最新模型，处理图像并获得图像代表的 PM2.5 值。要求 MDR 系统存储模型的所有版本，并提供针对版本时间的快速检索能力。

- 融合结果存储

数据融合模块对收集的光照、声音或者温度数据进行融合计算，将地图分为排列整齐的连续的小方格，并为每一个小方格通过融合算法计算得到一个代表该方格较为准确的值。这样参与者可以快速地知道所处的地理位置当前的光照、声音或者声音值，帮助用户更好地选取阳光更充足或者噪声更小的地点。需要 MDR 系统提供方格到收集数据的快速映射，以及融合结果的存储接口。

- 预测模型存储

轨迹分析与预测模块通过分析参与者的运动轨迹，建立预测模型，预测参与者到达指定地点的概率。轨迹分析与预测模块的预测结果是激励算法的重要决策属性，通过轨迹分析与预测模块的预测结果可以更精准地推送激励任务，丰富数据的同时也鼓励参与者不断贡献。要求 MDR 系统提供针对用户快速检索轨迹的能力，以及预测结果的存储能力。

- 发布、跟踪激励任务

激励任务指参与者到达指定地理位置采集指定种类数据，选择稍后上传或者立刻上传数据。当数据融合模块发现数据缺失或者展示 HTTP 请求发现数据缺失，激励模块会发布合适的激励任务，调用轨迹预测模块获得最有可能到达指定位置的参与者列表，同用户信息一起，通过激励算法得出合适的参与者列表，调用网络接口模块将激励任务推送给以上的参与者列表。随后激励模块跟踪激励任务的接受情况和激励任务的完成情况，并最后更新参与者的诚信度、优先级等数据。要求 MDR 系统提供对用户信息、激励任务信息的快速读写接口。

3.3. 非功能性需求分析

- 性能

首先，系统应该能承受住可能的大量并发的数据上传，必须确保数据库的高效写性能，否则影响参与者持有设备的正常使用，甚至可能造成数据丢失；其次为了让收集数据展示功能提供友好的用户交互过程，要求系统保证提供高效的读性能；最后，多个应用模块要求系统提供可接受的读写性能。性能测试的主要指标有响应时间、并发数、吞吐量等。

- 可靠性

参与式感知项目中，数据是其最宝贵的物质资产，硬件可以购买，软件可以重写，但是长时间积淀下来的数据一旦失去，不能再重来，所以要求数据保持高可靠性。高可靠的数据包含以下含义：

数据持久性，保证数据可持久存储，在任何情况下都不会出现数据丢失，为了实现数据的持久性，不但在写入数据时需要写入持久性存储，还需要将数据备份一个或多个副本，存放在不同的物理存储设备上，在某个存储故障或灾害发生时，数据不会丢失；

数据可访问性，在多份数据副本分别存放在不同存储设备的情况下，如果一个数据存储设备损坏，就需要将数据访问切换到另一个数据存储设备上，如果这个过程不能很快完成，或者在完成过程中需要停止终端用户访问数据，那么这段时间数据是不可访问的；

数据一致性，在数据有多份副本的情况下，如果网络、服务器或者软件故障，会导致部分副本写入成功，另外部分副本写入失败，这就造成了副本之间的数据不一致，数据内容冲突。

- 伸缩性/扩展性

伸缩性是指在不改变系统的软硬件设计的前提下，只需要增加或者减少服务器的数量就可以扩大或者缩小系统的服务处理能力。可伸缩性更侧重系统的水平扩展，通过廉价的服务器集群实现分布式来线性地提升性能，同时降低资源成本和系统维护成本。

我们希望以后参与式感知项目可以做成一个云平台，越来越多的参与者加入，存储了海量的数据，数据类型也越来越丰富，基于这些数据衍生了大量的、多维度的数据分析与挖掘应用。

CAP 定理认为任何分布式系统只可同时其中满足二点，没法三者兼顾，如图 3-3 所示。

图 3-3 CAP 原理

Consistency(一致性), 数据一致更新，所有数据变动都是同步的；Availability(可用性), 好的响应性能；Partition tolerance(分区容错性)，相当于对通信的时限要求。系统如果不能在时限内达成数据一致性，就意味着发生了分区的情况，必须就当前操作在 C 和 A 之间做出选择。

3.4. NOSQL 与 RDBMS 对比

为了更好地满足以上列举的功能性和非功能性需求，分别对 NOSQL 数据库（HBase 为例）和 RDBMS（以开源的 MySQL 为例）进行技术调研，仔细对比两种方案的利弊。

首先是存储机制方面，MySQL 采取面向行的存储机制，连续地存储整行，而 HBase 是列式存储(所有列值是连续存储保存的)，列式存储基于这样的一个假设：对于特定的查询，不是所有的列都是必须的，尤其在分析型应用里，这种场景很常见，且列式存储允许表在实际存储时不存储 NULL 值，NULL 值不占用存储空间。列存储拥有以下优点：减少 I/O 操作；列的数据类型是天生相似的，即便逻辑上每一行之间有轻微的不同，但仍旧比按行存储的结构聚集在一起的数据更利于压缩，好的压缩比有利于在返回结果时降低带宽的消耗。图 3-4 展现了行式存储结构和列式存储结构。

行式存储结构：

Row-id	Name	Birthdate	Salary	Dept
1	Dave	5-Jul	100	MKT
2	Bob	4-Apr	75	ENG
3	Ron	12-Dec	115	MKT
4	Joe	2-Mar	120	ENG

列式存储结构：

Name 列：

Row-id	Value
1	Dave
2	Bob
3	Ron
4	Joe

Birthdate 列：

Row-id	Value
1	5-Jul
2	4-Apr
3	12-Dec
4	2-Mar

图 3-4 行式存储结构与列式存储结构

对比写性能，RDBMS 提供所谓的 ACID(Atomicity, Consistency, Isolation, Durability，原子性，一致性，隔离性，持久性)特性，这意味着数据是强一致性的（数据的变化是原子的，一经改变即时生效）。此外，参照完整性约束不同表结构之间的关系。导致 RDBMS 的写性能很低效。而 HBase 的存储基于 LSM 树（Log-Structured Merge Tree，LSM 树）。LSM 树和 B+树相比，它牺牲了部分读性能，但能够大幅地提高写性能。它的基本工作原理是把一颗大的树分拆成多棵小树，LSM 树会先写入到内存中（内存中随机写的性能远高于磁盘），在内存中构建一颗有序小树，随着小树变大，内存中的小树会刷新到磁盘上。当读时，因为无法提前知道数据存在哪棵小树上，所以必须遍历全部小树。但在每颗小树的内部数据是有序的。

其次分析扩展性，当访问量越来越大，数据越来越多，RDBMS 一般进行如下扩展：首先进行读写分离；当写压力持续增加的时候，对主服务器进行垂直扩展（移植到拥有更快速 CPU，更大内存的商业机器上），同时在数据库前添加缓存；但是当数据量持续增加，后台的 JOIN 语句执行变慢，索引量也持续增大，数据库性能下降；此时必须对数据库进行水平扩展，但是对 RDBMS 进行水平扩展的代价是十分昂贵的。RDBMS 非

常适合事务性操作，但不长于大规模数据分析。除此之外，RDBMS 的等待和死锁的概率，与并发数据的平方以及事务规模的 3 次方甚至 5 次方相关。相比之下，HBase 的扩展性主要依赖其可分裂的 Region 及可扩展的分布式文件系统 HDFS 实现。当一个 Region 中写入的数据太多，达到配置的阈值时，Region 会分裂成两个 Region,并将 Region 在整个集群中进行迁移，以使 Region 服务器的负载均衡。对于表中的每行数据只由一台 Region 服务器所服务，因此 HBase 具有强一致性，HBase 在设计上完全避免了显式的锁，提供了行原子行操作，这使得系统不会因为读写操作性能而影响系统扩展能力。

应该永远保证数据是可访问的，HBase 底层使用了 HDFS 进行存储，HDFS 已经自动完成了多副本和副本间同步，保证了系统的容错性，而 RDBMS 并没有这方面的功能。虽然 RDBMS 本身可以根据 LOG 进行备份和恢复，但是如果整个硬盘坏掉，数据不可恢复。为了保证可用性，应该对数据进行多副本保存到服务器集群，但是如何保证副本同步也是个难题。

最后，HBase 主要应用是数据挖掘和分析，对实时性要求较低。而 RDBMS 则是对实时性响应较高，作为数据挖掘而言，即使当前可以胜任，但是一旦数据量庞大，直接导致数据检索速度急剧下降，即使是 Oracle 数据库执行的挖掘任务都有可能跑不出结果。并且数据挖掘查询返回的数据量通常非常巨大，HBase 的查询返回结果拥有更大的压缩比，节约带宽资源。

3.5. 系统概要设计

3.5.1. 系统模块划分

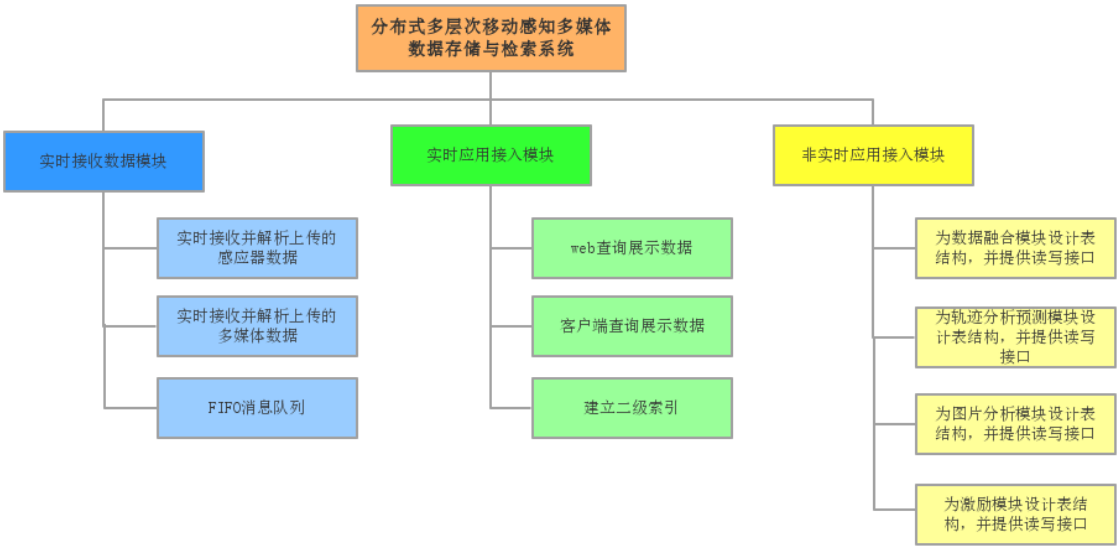

图 3-5 系统模块划分

如图 3-5 所示，根据读写数据的不同方式将 MDR 系统划分为 3 个子模块，分别为实时接收数据模块、实时应用接入模块和非实时应用接入模块。其中实时接收数据模块包括实时接收并解析上传的感应器数据和多媒体数据以及实现 FIFO 消息队列；实时应用接入模块负责为 web 展示和客户端展示数据提供实时的数据查询接口，此外为了使 HBase 适应实时读操作，需要实现二级索引，实现针对二级属性的快速查询；非实时应用接入模块为数据挖掘模块例如数据融合模块、轨迹分析预测模块以及图片分析模块设计存储表结构，并提供相应的读写数据的接口。

3.5.2. 实时接收数据模块

根据参与式感知项目的总体架构，参与者可以选择主动收集或者被动收集感应器数据和多媒体数据，并最终通过客户端将收集数据进行上传操作。网络接口模块负责接收数据，随后将数据写入 MDR 系统中。网络接口模块并不关心数据的格式和内容，数据的解析工作留给 MDR 系统完成。考虑到可能的高并发的数据上传操作，需要针对高并发的数据上传进行优化设计。

解决方案是，在内存中维护一个 FIFO 消息队列，网络接口模块通过 RMI 远程过程调用将数据写入到消息队列中便立即返回，MDR 系统中的消费者线程不断从 FIFO 队列中读取数据元素，根据不同的数据类型进行解析操作后插入到对应的数据表格。通过 FIFO 消息队列，首先数据上传和更新操作都以异步模式进行，提高了用户体验，这样更好地鼓励用户加入到参与式感知项目当中。其次，消息队列的平滑作用，使得即使在上传和更新请求的高峰期也不会造成数据丢失。最后，对数据库进行批量插入操作，表现的性能优于逐个单记录插入。开放给网络接口模块写数据的 RMI 接口定义如图 3-6 所示。RMI 具有以下优点：首先网络接口模块和 MDR 系统可以部署在不同的机器上，扩展性强；其次 RMI 使用简单，类似函数调用；最后 RMI 是面向对象的，可以方便地传递序列后的对象。

```
/*  远程接口必须扩展接口 java.rmi.Remote */
public interface QueueInterface extends Remote
{
        /*  将数据写入到消息队列中
        * dt  指定是感应器数据、多媒体数据还是用户数据
        * content  存放数据内容
        *  返回 True 代表数据写入成功，False 代表写入失败/
        public Boolean insert(DataType dt, byte[] content) throws RemoteException;
}
```

图 3-6 网络接口模块写数据的 RMI 接口定义

- 感应器数据格式

感应器数据通过 JSON 格式进行编码和解析，格式如图 3-7 所示。客户端的一次上传操作往往包含多条采集数据，每条采集数据包括采集时间、光线、噪声、经纬度、方位等。

```
{
    "userid": “leeying”,// 用户
    "sensoryData": [
    {"time": "2014-10-12 14:28:03", // 收集时间
    "light": 73,// 光照
    "noise ": 28,// 噪音
    "lon": 39.964,// 经度
    "lat": 116.358,// 纬度
    "orienX ": 43.0, "orienY ": 135.6, "orienZ ": 90.7},// 方向传感器数据
   /*剩余收集数据 */]
}
```

图 3-7 感应器数据 JSON 格式

- 多媒体数据格式

多媒体数据通过 EXIF 技术填入和提取所需属性，开源类 com.drew.imaging.jpeg.JpegMetadataReader 提供了快速简便的读 JPEG 图像中 EXIF metadata 的接口。多媒体数据采集后，客户端通过 EXIF 技术为多媒体文件设置表 3-1 列举的标签的值，同样地，MDR 系统通过 EXIF 技术将以上属性提取出来。

表 3-1 多媒体文件 EXIF 附加信息

Tag	Description
Data/Time	拍摄时间
GPS Longitude	拍摄地点经度
GPS Latitude	拍摄地点维度
Username	贡献者用户名
PM2.5	PM2.5 浓度值

3.5.3. 实时应用接入模块

- 二级索引的实现

MDR 系统需要为 web 展示模块和客户端展示提供实时查询数据的能力，HBase 虽然在扩展性和性能方面表现卓越，但是并不能提供类似 SQL 丰富的查询方式（select * from table where col = val），只能通过单一的行键进行数据检索。也就是说，要完成这样

的查询任务，HBase 只能通过遍历全表，即使通过 MapReduce 进行并行计算，仍然浪费了大量的计算资源，也使得延迟变得严重，无法应用于实时应用。

为了解决这个问题，我试着借鉴 RDBMS 的索引技术，为 HBase 同样建立二级索引支持，使得 HBase 可以胜任实时应用的存储系统。

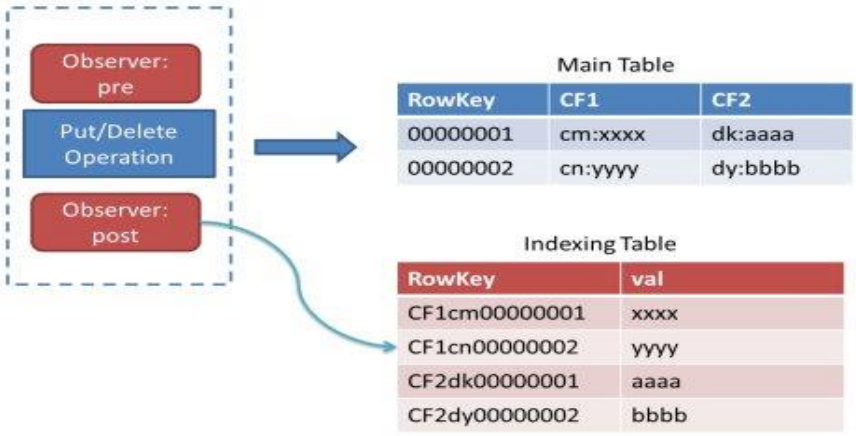

图 3-8 HBase 实现二层索引

为了提高二级属性查询操作的性能，我们可以为数据表建立索引表，如图 3-8 所示，索引表与数据表性质完全相同，只是存储的不是数据而是索引。通常索引表的行键选择为被查询的属性的值，行键按照字母序排列，所以可以非常高效地定位到该属性值在索引表的位置。索引表的内容可以选择存储数据结果，或者存储数据表的行键值，前者不需要二次读取但是造成大量数据冗余浪费存储资源，而后者需要进行 2 次读取操作才能获得数据。

- 展示数据的数据结构设计

```
Public class MultiMedia {// 基类，多媒体数据结构
    private Date timestamp;  // 采集时间
    private String userid；// 采集用户
    private double lon; // 采集经度
    private double lat; // 采集纬度
    private Byte[] content; // 数据内容
     private String poi; //POI 值
}
Public class Pic extends MultiMedia {// 扩展类，图片数据结构
     Private int fpm; //PM2.5 浓度
}
```

图 3-9 展示数据的数据结构

3.5.4. 非实时应用接入模块

3.5.4.1. 针对图片分析模块的数据结构和接口设计

- 数据结构设计

图片分析模块需要实时爬取并存储天气和空气质量数据，这些数据是之后训练模型的重要属性。天气和空气质量的数据结构如图 3-10 和 3-11 所示。

```
// 天气数据结构
Public class WeatherDataUnit{
        private int temp; // 温度
        private int humi; // 湿度
        private int wsped; // 风速
        private int precip; // 降雨量
        private int pressure; // 压强
        private int weather; // 天气类型
}
```

图 3-10 天气的数据结构

```
// 空气质量数据结构
Public class AirQualityDataUnit{
        private int fpm; // PM2.5 浓度
        private int cpm; // PM10 浓度
        private int aqi; // AQI 指数
}
```

图 3-11 空气质量的数据结构

- 读写接口设计

图片分析模块需要实时爬取并存储天气和空气质量数据，并且需要读取最接近时间戳的整点空气质量或者天气信息，这些数据和上传的图像一起被用来建模。读写接口如图 3-12 所示。

```
// 存储空气质量信息接口
public Boolean set(String city, int siteId, Date timestamp, int fpm,int cpm, int aqi)
// 读取最接近时间戳的整点的空气质量信息,没有则返回NULL
public AirQualityDataUnit getNearst(String city, int siteId, Date timestamp)

// 存储天气信息接口
public Boolean set(String city, Date timestamp, WeatherDataUnit weatherDataUnit)
```

```
// 读取最接近时间戳的整点的天气信息,没有则返回NULL
public WeatherDataUnit getNearst(String city,Date timestamp)
```

图 3-12 空气质量和天气信息的读写接口

图像分析模块建模结束后，图像分析模块需要为不同的手机型号存储并时常更新模型，至此分析阶段结束。预测阶段，当新图像到来，会从数据库中读出新图像对应的手机型号在拍照时刻之前的最新模型。模型的读写接口如图 3-13 所示。

```
// 保存phoneID在timestamp生成的crf内容
public Boolean set(int phoneID, byte[] crf, Date timestamp)

// 返回手机型号phoneID在timestamp之前的最新CRF模型
Public byte[] getLastestCRF(int phoneID, Date timestamp)
```

图 3-13 模型数据的读写接口

3.5.4.2. 针对数据融合模块的数据结构接口设计

- 数据结构设计

数据融合模块可以针对光照、噪声、温度进行融合操作，数据融合模块首先将收集数据根据收集地点的经纬度划分到不同的网格中。然后分别对网格中的数据进行融合，得到代表该网格数据的光照、噪声或者温度值。融合数据和网格的数据结构如图 3-14 所示。

```
// 融合数据的数据结构
Public class FusionData{
        Private int light; // 光照
        Private int noise; // 噪声
        Private int temp; // 温度
        Private Date timestamp; // 收集时间
}

// 网格的数据结构
Public class Grid{
        Int x;// 水平横坐标
        Int y;// 垂直纵坐标
        Int side; // 边长
}
```

图 3-14 融合数据和网格的数据结构

- 读写接口设计

```
// 计算感应器数据表中的记录的网格
Public Grid computeGrid(double lon, double lat);

// 读取网格中时间区域的采集数据
Public List<FusionData> getDataInGrid(Grid grid, Date beginTime, Date endTime);

// 记录网格的融合结果
Public Boolean setFusionData(Grid grid, FusionData)
```

图 3-15 针对数据融合模块的读写接口

3.5.4.3. 针对轨迹预测模块的数据结构和接口设计

- 数据结构设计

轨迹预测模块首先从感应器数据存储表中读取用户的运动轨迹信息，然后将轨迹信息中的轨迹点映射到网格中，得到用户的运动网格序列。最后分别计算用户从任意网格移动到任意网格的转移概率。轨迹信息的数据结构如图 3-16 所示。

```
// 轨迹点信息
Public class TrackPoint{
    Private double lon; // 经度
    Private double lat; // 纬度
    Private Date timestamp; // 时间戳
}
//  轨迹信息
Public class Track{
    Private int tid;// 轨迹标识
    Private String username;// 用户
    Private List<TrackPoint> track; // 轨迹点序列
}
```

图 3-16 轨迹信息的数据结构

- 读写接口设计

```
// 读取轨迹信息
Public Track getTrack (int tid);

// 将运动轨迹映射到网格中
Public List<Grid> computeGrid(Track track);

// 记录转移概率
Public Boolean setTransferProbability(Grid start_grid, Grid end_grid, Stirng username,
                                      double prob);
```

图 3-17 轨迹信息的数据结构

3.5.4.4. 针对激励模块的数据结构和接口设计

- 数据结构设计

激励模块主要需要针对两种数据结构进行操作，分别为激励任务和用户响应的数据结构，数据结构的定义如图 3-18 和 3-19 所示。

```
// 激励任务的数据结构
Public class IncentiveJobDataUnit{
        Private int dataType; // 收集数据类型
        Private double lon; // 经度
        Private double lat; // 纬度
        Private double payment; // 报酬
        Private Boolean flag; // 是否已经发布
        Private Date publishTime; //发布时间
}
```

图 3-18 激励任务的数据结构

```
// 用户响应的数据结构
Public class UserResponseDataUnit{
         Private String username;   // 响应用户
         Private Date responseTime; // 响应时间
         Private double payment; // 要求报酬
}
```

图 3-19 激励任务的数据结构

- 读写接口设计

激励模块需要针对激励任务的发布状态，用户响应激励任务情况，用户完成激励任务情况进行读写操作，接口设计如图 3-20 所示。

```
// 新增一个激励任务
Public Boolean setIncentiveJob(IncentiveJobDataUnit job, Boolean flag)
// 更新激励任务发布状态
Public Boolean setPushed(IncentiveJobDataUnit job, Boolean flag);

// 记录用户响应激励任务
Public Boolean setUserResponseIncentiveJob(IncentiveJobDataUnit job,
                                           List<UserResponseDataUnit> users)
// 查询激励任务的所有响应用户信息
Public List<UserResponseDataUnit> getAllResponseUsers(IncentiveJobDataUnit job)

// 激励用户完成激励任务
Public Boolean setUserFinishedIncentiveJob(IncentiveJobDataUnit job, String username,
                                           Data finishedTime)
```

图 3-20 针对激励模块的读写接口

3.6. 本章小结

本章首先对MDR系统进行详细的需求分析，其中包括参与式感知项目的总体架构、功能性需求分析、以及非功能性需求。然后我们分别从数据存储架构、写性能、扩展性、数据可靠性等方面比较了 HBase 和 RDBMS。读完本章我们应该了解到 MDR 系统应该具有哪些功能性和非功能性要求，了解我们需要完成哪些任务，为之后系统的设计与实现奠定了基础。最后，对进行了系统概要设计，包括模块划分，以及各模块之间数据格式定义、数据结构定义和接口定义。

第四章　系统的详细设计与实现

上一章首先简要介绍了参与式感知项目的总体架构和 MDR 系统在参与式感知项目的位置，然后详细分析了 MDR 系统的功能性和非功能性需求，最后对系统进行了概要设计。以此为基础，本章将介绍 MDR 系统的详细设计与实现。

4.1. 实时接收数据模块

4.1.1. 上传数据实时存储流程设计

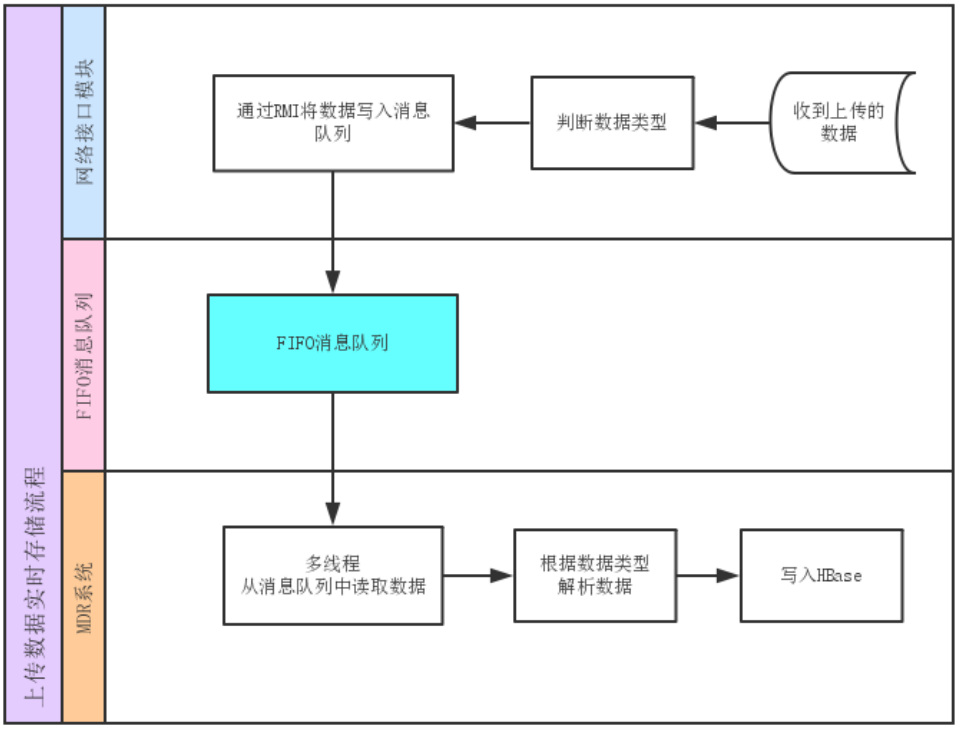

图 4-1 上传数据实时存储的流程图

图 4-1 描绘了上传数据实时存储的流程。考虑到 I/O 操作的慢速性，MDR 系统在内存中维护的一个消息队列，上传数据先写入到消息队列中，随后再异步写入 HBase 数据库。引入消息队列有以下优点：首先如果在不使用消息队列的情况下，用户的上传数据直接写入数据库，在高并发的情况下，会对数据库造成巨大的压力，同时也使得响应延迟加剧。在使用消息队列之后，用户的上传数据发给消息队列后立即返回，再由消息队列的消费者线程从消息队列中获取数据，异步写入数据库。由于消息队列处理速度远远快于数据库，因此用户的响应延迟可得到有效改善；其次消息队列具有很好地削峰作用，通过异步处理，将短时间内高并发产生的上传数据存储在消息队列中，从而削平了高峰期的巨大流量，防止数据丢失；最后，消费者线程可以一次解析并同时插入多条记录到

数据库，批插入会比逐个插入数据表现出更好地性能，从而提高了写性能。

网络接口模块接收到上传数据后，为数据打上类别标签（感应器数据或者多媒体数据），然后通过 RMI 写入消息队列。

当 MDR 系统启动之前，事先根据配置文件里的消费者线程数量，启动同样数量的消费者线程。消费者线程负责从消息队列中取出头元素，进行解析后写入数据库，具体流程图如图 4-2 所示。

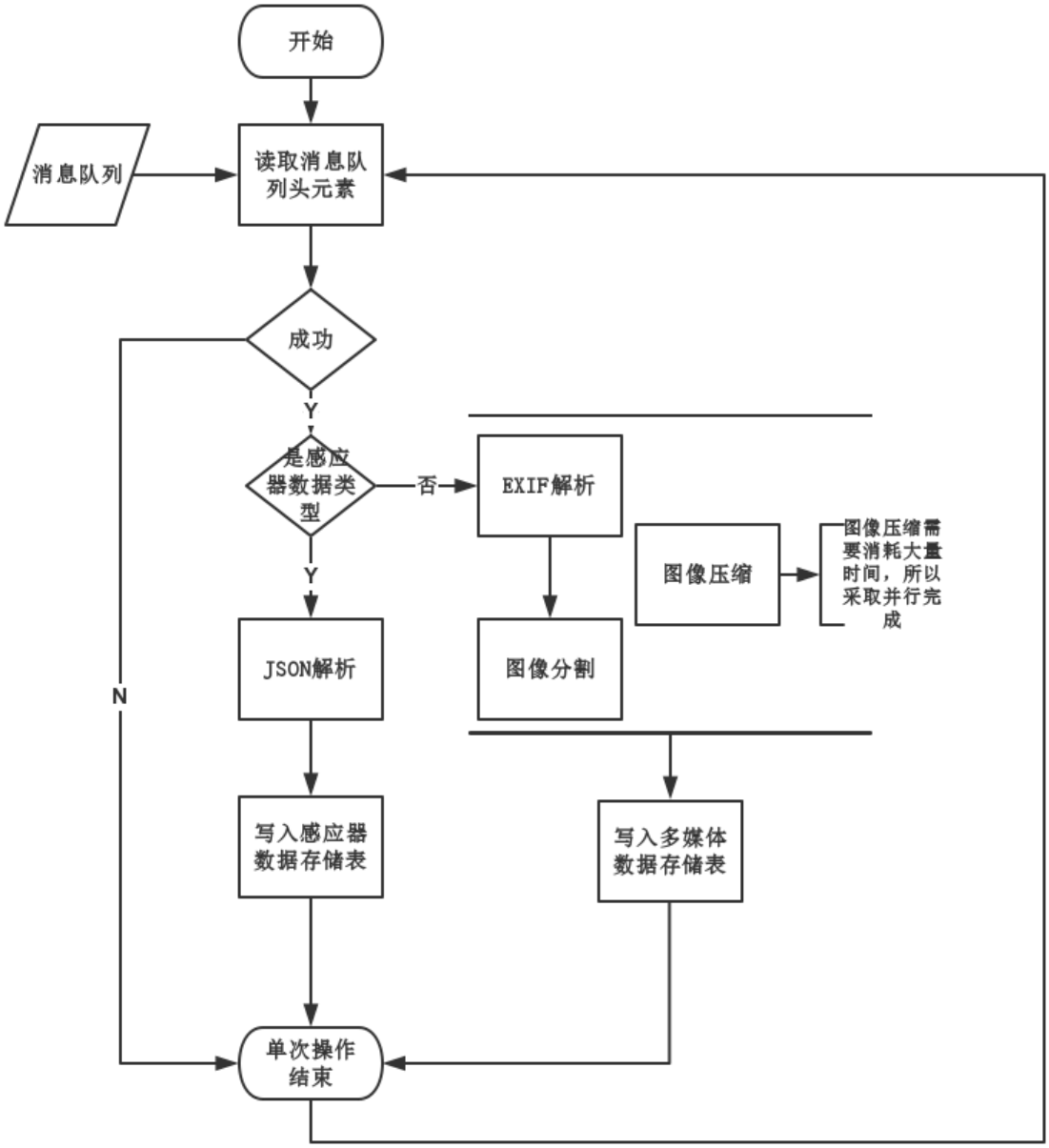

图 4-2 消息队列的消费者线程的工作流程图

消费者线程不断尝试从消息队列读取并移除头元素，如果读取失败，则单次操作结束。如果成功，则根据类别标签进行判定，如果是感应器数据，则进行 JSON 解析，并把解析结果 Put 写入到感应器数据存储表；如果是图像数据，则对图像进行

EXIF 解析提取相关属性，并按照预定的分片大小对图像进行分割，与此同时对图像进行压缩，因为压缩图像需要消耗大量的时间，通过并行化可以提高系统性能，最后将原图像和压缩后的图像都存储到多媒体数据存储表中。

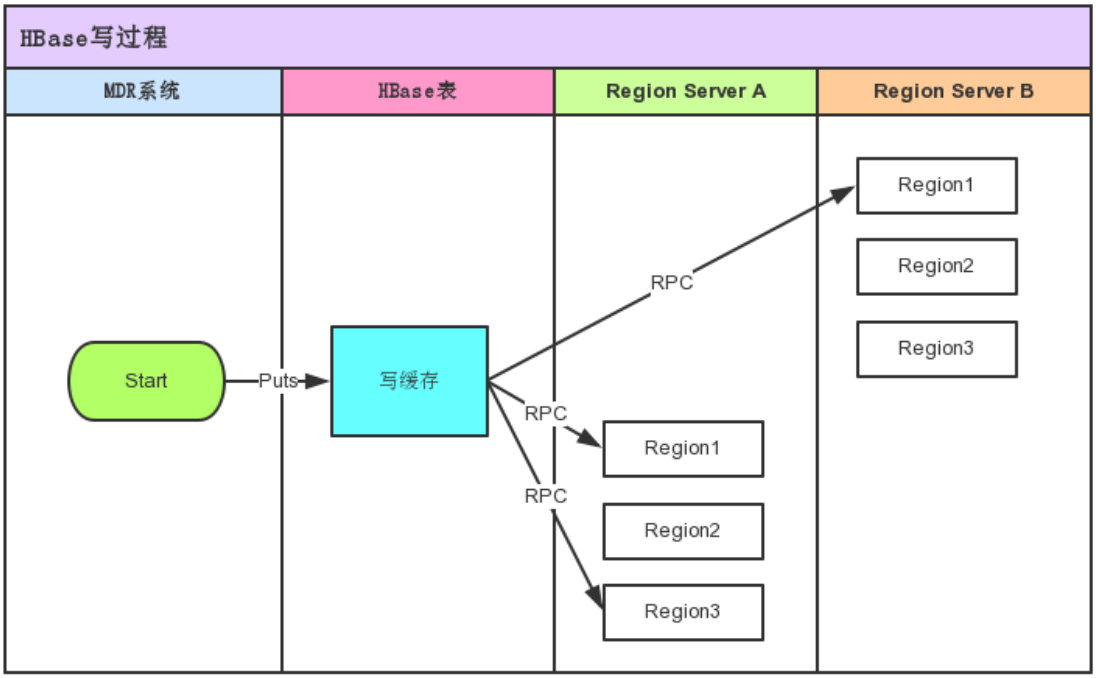

图 4-3 HBase 写流程图

另一方面，图 4-3 描述了 HBase 写流程。根据 HBase 的官方手册每个存储表都包含一个写缓存（WriteBuffer），当且仅当写缓存打开且包含的写请求超过配置的写缓存大小时才真正地提交数据写请求。提交写请求时，根据“写请求位于不同 Region”将写缓存的所有请求进行归类，多线程写，线程数与这批数据涉及的 Region 个数相同，写请求通过 RPC（Remote Procedure Call，远程过程调用）异步执行的。如果写缓存未开启，任何一个写请求就会引发一次 RPC，造成性能低下，为了避免以上情况，分别打开了感应器数据存储表的写缓存，并将写缓存大小设置为 2MB，和多媒体数据存储表的写缓存，并设置缓存大小为 20MB。写缓存之后的执行性能很高，并不需要过多地进行优化。

数据上传的性能测试会在系统测试与验证章节中进行详细描述，确保 MDR 系统可以承担高并发的数据上传任务。

4.1.2. 收集数据表设计

由需求分析可知，收集的感应器数据与多媒体数据的使用场景不同，且两种数据类型之间的属性差别非常明显，为了提供更好的并发性和扩展性，选择将感应器数据与多媒体数据分开存储到不同的数据库表中。表 4-1 与表 4-2 分别展示了感应器数据存储表以及多媒体存储表的结构。

表 4-1 感知器数据存储表的结构

	行键	SensoryData 列族					
		Tid	Lon	Lat	Light	Temp	Noise
说明	格式： 收集时间/用户 ID	轨迹 id	经度	纬度	光照	温度	声音

如表 4-1 所示，感应器数据包括用户 id、收集时间、经度、纬度、光照、温度、声音以及方位等。为了满足全局唯一，行键由用户 id 与收集时间共同编码，此外为了加快比较速度收集时间采取了万年历开始以来的毫秒数。SensoryData 列族保存了所有收集的感应器数据，SensoryData 列族将每种感应器数据都保存到单独的列中，这样可以使得应用模块自由地选择读取需要的列数据，节约了带宽。参与者在加入参与式感知项目中，可以手动选择贡献什么数据，所以感应器数据存储表可能是一张非常稀疏的表格，非常幸运的是，HBase 的存储机制允许不存储 NULL 值，即 NULL 值不占用任何存储空间，很大程度上节约了存储资源。除此之外，HBase 允许在列族创建之后仍旧可以添加新的列，所以可以随时添加收集的数据种类，而不必修改现在的存储结构，提供了非常良好的可扩展性。

表 4-2 多媒体数据存储表的结构

	行键	Multimedia 列族			
		Lon	Lat	Content	（Fpm）
说明	格式： 收集时间/用户 ID	经度	纬度	多媒体文件内容	（图像的 PM2.5 值）

如表 4-2 所示，多媒体数据包括用户 id、收集时间、经纬度、多媒体文件内容等。同感应器数据存储表相同，多媒体数据库存储表的行键也是由用户 id 与收集时间进行编码，Multimedia 列族存储收集的多媒体文件以及附加属性（例如采集地点的经纬度，如果是图像文件，则还包括通过图像分析得到的 PM2.5 值）。但是多媒体文件的大小往往非常大，已经超过了适合 HBase 随机读取的范围，无法简单地像存储经纬度等数值一样地直接存储多媒体文件到单元格中。具体的存储方案在下一节“HBase 多媒体文件存储方案”中进行详细描述。

4.1.3. HBase 多媒体文件存储方案

由之前的系统需求分析可知，我们希望可以将多媒体文件进行分布式存储，虽然可以通过多种方案将多媒体文件存入 HDFS 中，HBase 中存储文件路径和偏移量。但是此类方法需要根据 HDFS 提供的接口进行编程，非常繁琐且可维护性低。

另一方面，HBase 的存储结构是一种面向列的存储结构，即按列族来保存及处理数据，同一列族的数据是连续存储的。HBase 以 KV 的形式来存储每个列族的每行单元格

中的数据，形成一定数量的数据块，一定量的数据块又形成一个 HFile[30]文件。HBase 的数据最终以 HFile 的形式保存在 HDFS 上。如果用单元格存储多媒体文件，上述存储数据的过程实际上隐含了把众多的多媒体文件打包成 HDFS 文件的过程。

图像、声音、视频等多媒体文件并不适合直接存储到 HBase 的单元格中，因为 HBase 的数据块大小限制为 64KB。如果通过重新配置数据块大小使其可以存储完整的多媒体文件，HBase 的随机读取性能将会受到影响，官方指导手册推荐数据块最大不超过 1MB。可在真实场景中，即使认为大多数图片文件在 1MB 以内，但是视频文件普遍超过 1MB，从而需要做一定的改进来实现基于 HBase 多媒体文件的存储。本文的想法是，如果文件超过了数据块限制的最大值，则将文件分割成多个切片后再保存，使所有切片的大小小于等于数据块大小限制的上限[31]。为了还原完整的原始数据，需要再设计一种机制来保存同一多媒体文件的所有切片的顺序。

幸运地是，HBase 的时间戳可以方便地记录各个切片的顺序。以图像文件为例，当这个图像文件超过了 1M，我们将图像文件切成了 N 片，然后我们将这 N 个切片按切片顺序依次保存到同一个的单元格中，HBase 会在保存时为每个切片打上保存时的时间戳。当需要取出数据时，我们根据行键和列键找到这个单元格，取出所有切片，然后把切片按时间戳进行排序并合并切片，即取出了完整的原始文件。多媒体数据存储表存储图像文件示例如表 4-3 所示，其中，列 Multimedia:Content 同时存储图像的多个分片，每个分片被打上了不同的时间戳（t5、t6、t7）。读数据的时候，按照 t5,t6,t7 的顺序合并文件分片便恢复完整的原始文件。

表 4-3 多媒体数据存储表存储图像文件示例

行键	Multimedia: Lon	Multimedia: Lat	Multimedia: Content	Multimedia: Fpm
…….				
645447832150013004 568833000682007444 20	Byte[]	Byte[]	“<JPG….>”←t5 “<JPG….>”←t6 “<JPG….>”←t7	Byte[]

任何多媒体文件内容都可以看做是一串连续的 byte 数组，Java 提供了非常高效的对 byte 数组的拷贝函数 System.arraycopy，可以快速完成切片和分片合并工作，代码如图 4-4 所示。

```
/**
 * 将content切片成sectionSize大小的分片列表
 * @param content 多媒体数据
 * @param sectionSize分片大小
 * @return
 */
publicstatic List<byte[]> splitPic(byte[] content, int sectionSize) {
    List<byte[]> subPics = new ArrayList<byte[]>();
    int index = 0;
    while(index < content.length){
        int len = Math.min(content.length - index, sectionSize);
        byte[] subPic = new byte[len];
        // 通过指定拷贝byte数组的范围来得到分片结果
        System.arraycopy(content, index, subPic, 0, len);

        subPics.add(subPic);
        index += len;
    }
    return subPics;
}
```

图 4-4 分割多媒体数据的代码示例

4.2. 采集数据展示

4.2.1. web 展示

由参与式感知项目总体架构所知，web 展示模块通过聚合采集数据，在地图中形成多个 POI 点，并展示 POI 点下收集到数据，包括照片、光照信息、噪声数据以及 PM2.5 值。当地图被放大或者缩小时，聚合算法将粗粒度的 POI 点分裂成多个子 POI 点或者将细粒度的多个 POI 点聚合成一个父 POI 点。例如，最普遍的读取数据接口，获取一周内某 POI 点下收集的所有照片。不幸的是，HBase 并不提供类似 RDBMS 的快速的针对非主键的特定属性的检索能力，只能遍历所有图像，判断是否属于该 POI 点，性能非常低效。对于 web 展示这种实时应用来说，低效的查询会严重地损害用户体验。

为了使得 web 展示模块与用户之间交互操作更加友好，我尝试为 POI 创建索引，建

立 POI 到收集数据的快速映射。POI 索引表的结构如表 4-4 所示。

表 4-4 POI 索引表的结构

	行键	MultimediaIndex 列族	SensoryData Index 列族	Info 列族
		Mrowkey	Srowkey	Info
说明	POI 标识	多媒体数据存储表记录的行键	感应器数据存储表记录的行键	POI 点的信息，包括经纬度和展示级别等

如表 4-4 所示，MultimediaIndex 和 SensoryDataIndex 列族分别存储 POI 下的数据在多媒体数据存储表和感应器数据存储表的索引-即为行键值，同时 MultimediaIndex 和 SensoryDataIndex 列族将 POI 点不同日期的数据保存在单元格的不同版本中。另外，据我们所知，任意 POI 点下可能会有多条收集数据，幸运地是，HBase 对列族下面的列没有任何限制，我们可以使用单独的一列存储一条行键，当读取所有收集的多媒体数据时候，只需要选择 MultimediaIndex 列族即可。Info 列族存储了 POI 点自身的信息，包括经纬度、展示级别等。

通过 POI 索引表，一次 HBase Get 操作可以找到任意 POI 点下所有的收集数据在存储表中的行键，再通过一次 HBase Get 操作便可快速定位到存储表中的记录。通过多层次的索引，使得针对非主键的属性的检索性能非常高效，适合实时应用。

4.2.2. 索引表的建立

根据 HBase 官方指导手册，可以利用 HBase 提供的协处理器（coprocessor）自动完成索引表内容的填写。协处理器允许用户在 region 服务器上运行自己的代码，即允许用户执行 region 级别的操作，并且可以实现与 RDBMS 中触发器类似的功能。用户不关心操作具体在哪执行，HBase 的分布式框架会帮助用户把这些工作变得透明。

协处理器可以在客户端 API 调用执行前或刚刚执行后拦截他们。为了自动完成 POI 索引表内容的填写，需要在多媒体数据存储表中添加 POI 列族，如表 4-5 所示，POI 列族仅包含一列即为 POI 标识，值由经度纬度计算得来。

表 4-5 多媒体数据存储表的结构

	行键	Multimedia 列族				POI 列族
		Lon	Lat	Content	Fpm	Poi
说明	格式： 收集时间/用户 ID	经度	纬度	多媒体文件内容	（图像的 PM2.5 值）	POI 标识

另外，我们编写类 POIIndexObserver, 继承 BaseRegionObserver 并重载 postPut 方法，将 POIIndexObserver 类编译后加载到多媒体数据存储表中，如表 4-5 所示。之后每

当对多媒体数据存储表中的记录计算 POI 值并写入 POI 列中，会自动将此<POI, 行键>对存储在 POI 索引表中。

```
’ MultimediaTable', {METHOD => 'table_att', coprocessor$1 => true
  'file:///usr/local/hbase/hbase-
data/POIIndexObserver.jar|Web.POIIndexObserver|1001'},
```

图 4-5 多媒体数据存储表加载协处理器

4.2.3. 客户端展示

客户端展示过去一段时间内参与者采集后上传的所有照片形成照片墙，通过照片墙可以非常鲜明地对比不同地点的 PM2.5 值。为了节约带宽资源，照片墙展示的照片均为缩略图，当用户手动点击任意的缩略图后再展示该缩略图的无压缩版本。

图 4-6 客户端展示收集数据顺序图

MDR 系统为客户端展示提供的接口如下：

- public List<Pic> getPicList(Date beginTime，Date endTime)接口，获取 beginTime 至 endTime 区间内参与者采集的所有照片。如表 4-2 可知多媒体存储表的行键格式是收集时间/用户 id，另外知道可以限制 HBase 的扫描操作在特定的范围内进行，同时 HBase 的行键按照字母顺序排序。所以我们通过将收集时间编码在前以及 HBase 提供的 setStartRow(beginTime.getTimes())和 setStopRow(endTime.getTimes() + 1)函数来为扫描操作指定特定的范围。将扫描操作限定在特定范围内，加快了检索的速度，提高了用户的用户体验。

- public Pic getOrignalPic(Pic compressPic)接口，通过压缩图片查询原图结果。图片 Pic 的数据结构如图 3-8 所示，包括拍照时间、拍照用户、地理位置以及内容等。getOrignalPic 接口参数 compressPic 只需要包含拍照时间和拍照用户，MDR 生成对应的行键，后查询到原图数据并返回。

客户端展示的性能测试会在系统测试与验证章节中进行详细描述，确保本系统可以为客户端的实时展示提供高效的接口。

4.3. 图像分析模块

4.3.1. 实时存储爬取的空气质量和天气数据

图像分析模块建模的过程中除了图像以及相关信息外，还需要空气质量信息，包括观测点观测到的 PM2.5、PM10、AQI 等以及天气信息，包括温度、湿度、压强等。其中，空气质量信息和天气信息需要每小时爬取一次。存储空气质量和天气信息的表结构分别如表 4-6 和表 4-7 所示，两张表的 Info 列族都分别用来存储爬取的信息，不同的是，一个城市拥有多个观测点，每个观测点观测到不同的空气质量，所以空气质量信息存储表的行键是城市、观测点和观测时间的编码，而天气信息存储表的行键是城市和观测时间的编码。通过以上的编码方式，可以非常高效地获得城市在一段时间区间内的所有天气信息以及非常高效地获得城市中任意观测点在一段时间区间内的所有空气质量信息。

表 4-6 空气质量信息存储表的结构

	行键	Info 列族		
		Fpm	Cpm	Aqi
说明	格式：城市/观测点/观测时间	PM2.5 指数	PM10 指数	空气质量指数

表 4-7 天气信息存储表的结构

	行键	Info 列族					
		Temp	Humi	Wspeed	precip	Pressure	Weather
说明	格式：城市/观测时间	温度	湿度	风速	降雨量	压强	天气状况

由概要设计可知，对空气质量信息存储表和天气信息存储表的读取操作都是读取在收集时间之前或之后的最接近的整点记录。因为 HBase 不支持 RDBMS 的 order by 操作，寻找距时间戳最近的有效记录的流程如图 4-7 所示。

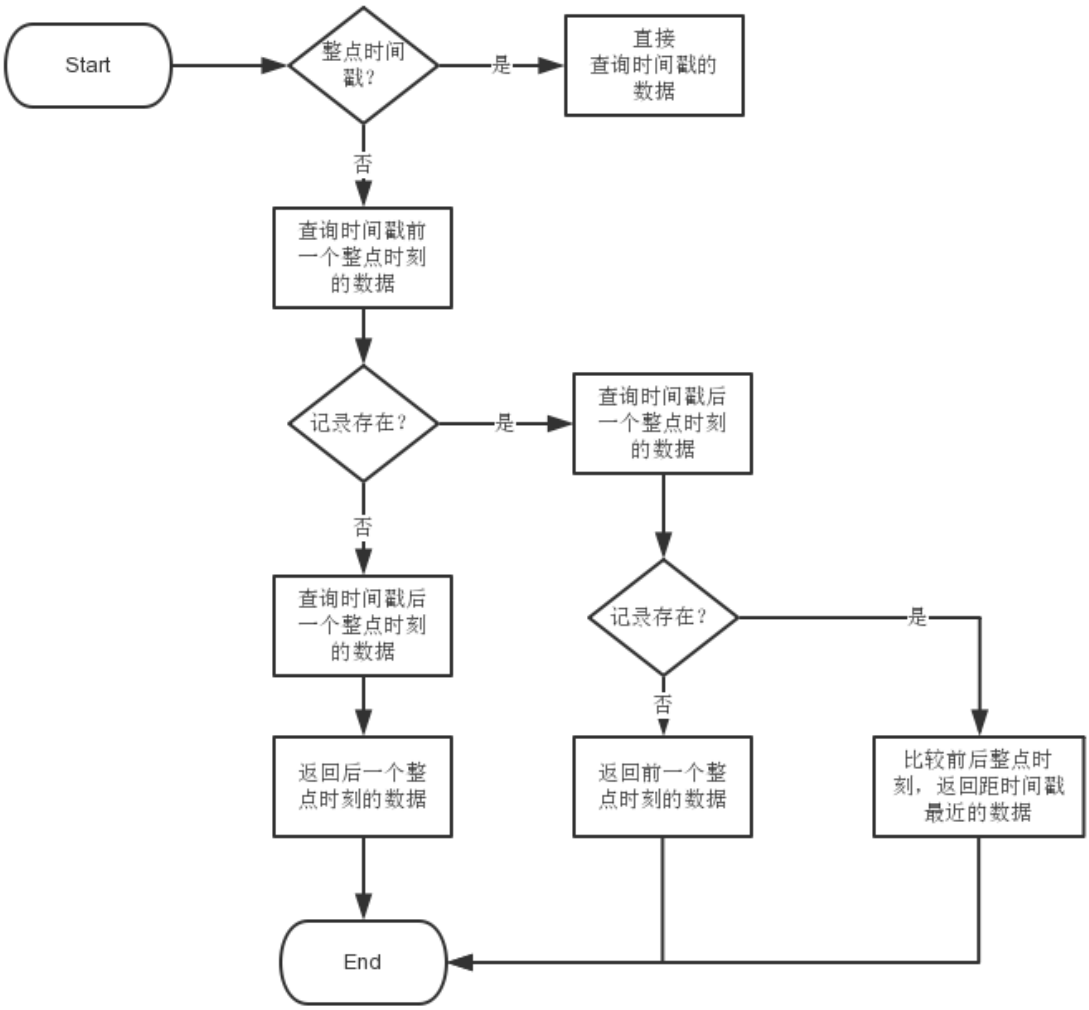

图 4-7 查询距时间戳最近的有效记录的流程

首先判断时间戳是否为整点，如果是整点时刻则直接查询数据并返回。否则尝试查询时间戳前一个整点时刻的数据，如果记录不存在，则查询时间戳后一个整点时刻的数据并返回；如果记录存在则尝试查询时间戳后一个整点时刻的数据，如果记录不存在则返回前一个整点时刻的数据；如果前后整点时刻的记录均存在，则比较前后整点时刻，返回距时间戳最近的数据。

4.3.2. 图像模型存储

图像分析模块成功建模结束之后，需要存储模型结果。图像分析模块认为，对于同种手机型号，其硬件设备特别是相机相同，结果模型对于同种手机型号是通用的，所以只需要为每种手机型号存储结果模型。但是，随着时间以及数据的不断更新，结果模型的版本会不断变化。当需要利用结果模型对图像进行处理的时候，图像分析模块需要查询图像收集时间之前的最新版本的结果模型，所以 MDR 系统需要记录下模型的所有版

本和提供快速的针对时间戳的模型的查询接口。

我们知道，HBase 可以针对列族设定版本保留策略，可以设置最多可以保留的版本数量或者版本的 TTL（Time To Live，生存时间）。图像模型存储表的结构如表 4-8 所示，Info 列族存储图像模型的所有版本，通过时间戳区分不同的版本。

执行查询操作的时候，通过利用函数 get.setTimeRange(0, timestamp,getTime() - 1)设置读取的时间范围，利用函数 get.setMaxVersions(1)设置读取的版本数目，可以查询到某手机型号在 timestamp 之前的最新模型。

表 4-8 图像模型存储表的结构

行键	Info 列族
手机型号	T1 - CRF 模型 T2 - CRF 模型

4.4. 数据融合模块

4.4.1. 存储表结构设计

由之前的需求分析可知，数据融合模块需要将收集的数据根据收集的地理位置组织到地图上的连续排列的网格中。然后，数据融合模块读取网格内的所有光线数据或温度、声音数据，通过特定的融合算法例如克里金算法，来预测代表网格的最终光线或温度、声音值，如果网格中没有任何数据，可以通过周边网格的数据来进行预测。

MDR 系统应为数据融合模块提供任意网格到感应器数据列表的快速映射的接口，如果直接在感应器数据存储表中进行全表扫描操作，判断数据是否落在网格中，当表中数据量持续增加，扫描操作会需要消耗越来越多的时间，最终变为不可用。

为了提高网格到感应器数据列表映射的性能，通过额外的网格索引表来快速完成网格到收集数据的快速映射，网格索引表的结构如表 4-9 所示。

表 4-9 网格索引表的结构

	行键	Index 列族	Fusion 列族		
		Srowkey	Light	Temp	Noise
说明	格式：网格横坐标/网格纵坐标	多媒体数据存储表记录的行键	光照	温度	声音

如表所示， Index 列族存储任意的网格在任意的时间片段内的数据在感应器数据存储表的索引-即为行键值，同时 Index 列族将网格不同时间片段的数据保存在单元格的不同版本中。据我们所知，任意网格下可能会有多条收集数据，使用单独的一列存储一条行键，当读取所有收集的感应器数据时候，只需要选择 Index 列族即可。数据融合模块最终将融合结果写入 Fusion 列族对应的单元格和版本中。

通过网格索引表，一次 HBase Get 操作可以找到任意网格下所有的收集数据在数据存储表中的行键，再通过一次 HBase Get 操作可以快速定位到数据存储表中的记录。通过多层次的索引，使得查询性能非常高效。

另外，我们在感应器数据存储表中添加 Grid 列族，如表 4-10 所示，Grid 列族包含两列分别是网格的横纵坐标，值由经纬度计算得来。同样为感应器数据存储表加载协处理器拦截客户端的 Put 操作，每当为感应器数据存储表的记录计算所属网格的时候，<网格，收集数据的行键>映射对自动写入网格索引表。

表 4-10 感应器数据存储表的结构

	行键	Info 列族						Grid 列族	
		Tid	Lon	Lat	Light	Temp	Noise	Grid_x	Grid_y
说明	格式： 收集时间/用户 id	轨迹 id	经度	纬度	光照	温度	声音	网格横坐标	网格纵坐标

4.4.2. 数据融合模块与 MDR 系统交互流程设计

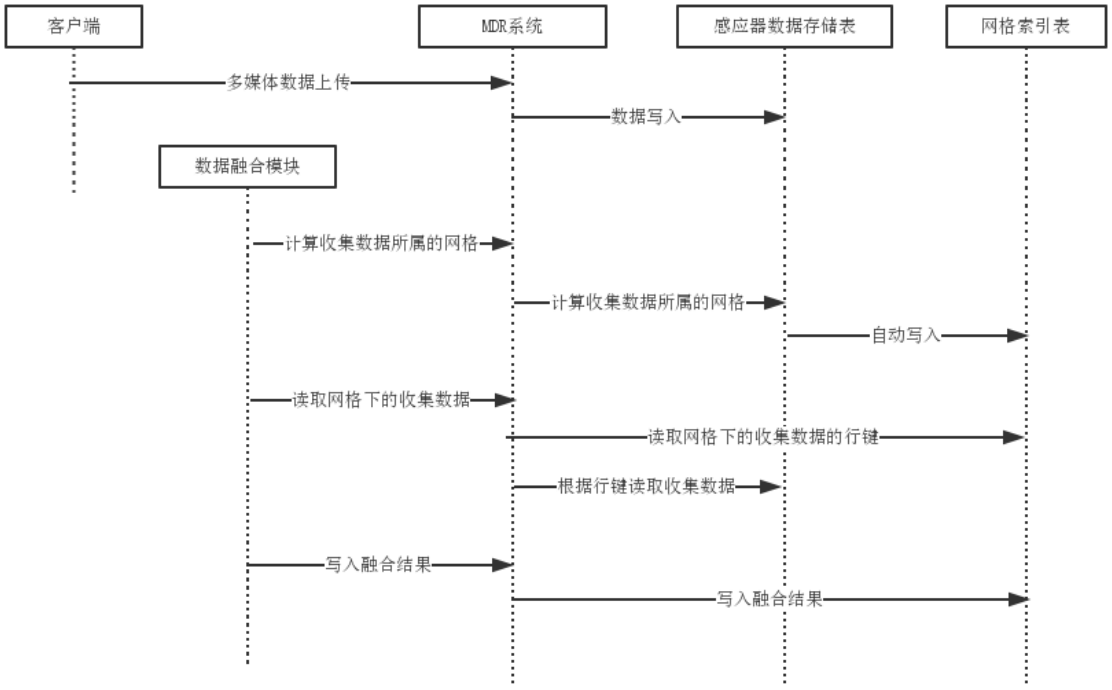

图 4-8 数据融合模块与 MDR 系统的交互的顺序图

图 4-8 展示了数据融合模块与 MDR 系统的交互。首先客户端实时上传的数据由 MDR 系统写入到感应器数据存储表中。执行数据融合之前，数据融合模块需要将感应器数据存储表中的记录根据收集的地理位置组织到地图上的连续排列的正方网格中，数

据融合模块需要实现 MDR 系统提供的 GridComputeInterface 接口，接口定义如图 4-9 所示。

```
public interface GridComputeInterface{
        // 根据经纬度计算得到 Grid
        Public Grid compute(double lon, double lat);
}
```

图 4-9 数据融合模块需要重载的 GridComputeInterface 接口定义

随后数据融合模块执行计算收集数据所属的网格操作的时候，MDR 系统查询感应器数据存储表中 Grid 列族为空的记录，并执行数据融合模块实现的 Grid compute(double lon, double lat)方法，将结果写入到 Grid 列族中。通过上文提到的索引，网格索引表会自动记录网格到收集数据的行键的映射。此时完成划分操作。

融合阶段，数据融合模块需要读取网格内的所有光线数据或温度、声音数据，MDR 系统首先从网格索引表中读取网格内收集数据的行键结果，接着利用行键从感应器数据存储表中取出收集数据。数据融合模块读到收集数据后，执行融合算法，并将融合结果写入到网格索引表中。此时完成写入操作。

4.5. 轨迹预测模块

4.5.1. 存储表结构设计

轨迹预测模块通过马尔科夫链，计算参与者从任意地理位置转移到另一个位置的转移概率，来预测参与者到达指定地理位置的概率。这些数据是激励模块精确推送激励任务的重要属性。

轨迹预测模块与数据融合模块类似，同样首先需要将用户的轨迹路线上的轨迹点组织到地图上的连续排列的正方网格中，称为区域。

轨迹预测模块读取感应器数据存储表的轨迹信息，轨迹信息包括用户 id 以及该用户的一连串轨迹点序列（按时间排序的经纬度对）。通过将经纬度映射到区域中，可以得到用户的运动的区域序列。最后将以上处理结果存入轨迹表中，轨迹表的结构如表 4-11 所示，轨迹表由轨迹 id 索引，Info 列族存储了用户 id 和用户运动的区域序列，其中区域序列是是连续的字符串。

表 4-11 轨迹表的结构

	行键	Info 列族	
	tid	Userid	Region Sequence
说明	轨迹标示	用户 id	区域序列

表 4-12 区域转移概率表的结构

	行键		Probability 列族	
	起始区域	终点区域	一阶转移概率	二阶转移概率
说明	From_region	To_region	P1	P2

随后，轨迹预测模块通过马尔科夫链，利用任意用户的所有轨迹，即所有的运动的区域序列，计算任意用户从任意区域运行到另一区域的转移概率。并将计算结果存入区域转移概率表，如表 4-12 所示，列 Probability:P1 记录了所有的用户从起始区域到达终点区域的一阶转移概率，类似的，列 Probability:P2 记录了所有的用户从起始区域到达终点区域的二阶转移概率。其中列 Probability:P1 和列 Probability:P2 中的任意单元格都存储了所有用户的转移概率，通过 protobuf 将所有用户及其对应的转移概率序列化到字节数组中然后存入 HBase。读取的时候通过反串行化获得用户及其转移概率。

4.5.2. 轨迹预测模块与 MDR 系统交互流程设计

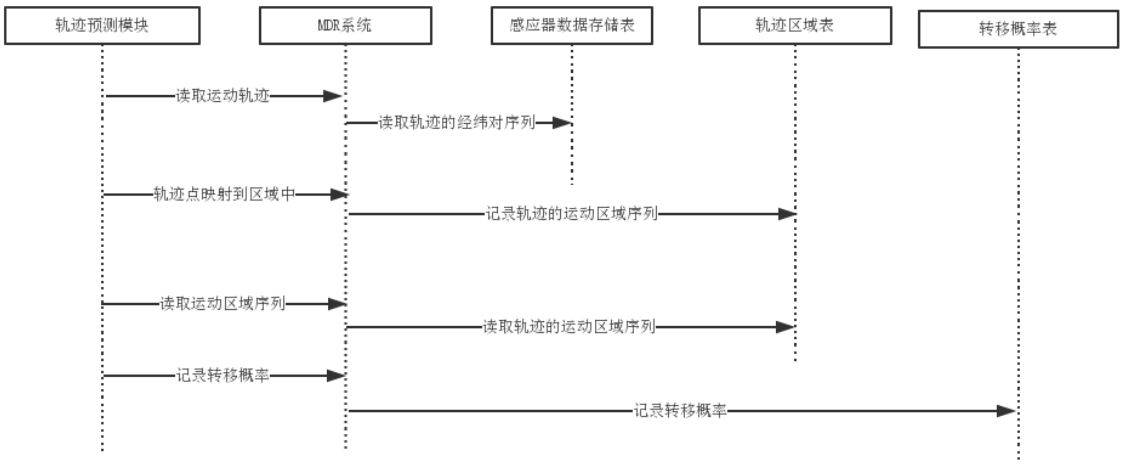

图 4-10 轨迹模块与 MDR 系统的交互顺序图

图 4-10 阐述了轨迹模块与 MDR 系统的交互流程。轨迹预测模块读取感应器数据存储表的所有轨迹信息，轨迹信息包括用户 id 以及该用户的一连串轨迹点序列（按时间排序的经纬度对）。通过将经纬度映射到区域中，可以得到用户的运动的区域序列。最后将以上处理结果存入轨迹区域表中。

随后，轨迹预测模块通过马尔科夫链，利用任意用户的所有轨迹，即所有的运动的区域序列，计算任意用户从任意区域运行到另一区域的转移概率。并将计算结果存入区域转移概率表。

4.6. 发布、跟踪激励任务

4.6.1. 存储表结构设计

激励模块涉及两张存储表的操作，分别为用户表和激励表。用户表存储参与式感知项目的所有参与者的信息，如表 4-12 所示，除了用户名、密码外，还包括权限、收益和参与次数，这些信息是进行激励任务精确发送的重要属性。激励任务的属性包括发布时间、收集数据的类型、收集地点、报酬以及是否已经推送等，并且可以方便地添加新的属性。激励任务表的结构如表 4-13 所示，行键是对收集地点和发布时间的编码以便达到全局唯一，Job 列族存储激励任务的属性，Response 列族存储激励任务的响应和完成情况。

表 4-12 用户表的结构

	行键	INFO 列族			
		Password	Priority	Earn	times
说明	格式：用户名	密码	权限	总收益	参与次数

表 4-13 激励任务表的结构

	行键	Job 列族			Response 列族			
		dataType	Payment	Flag	Accept Users	Finish User	Finish Time	Finish Payment
说明	格式：经度/纬度/任务发布时间	收集数据类型	报酬	是否推送	接受任务的用户列表	完成任务的用户	完成时间	完成报酬

4.6.2. 发布跟踪任务的流程设计

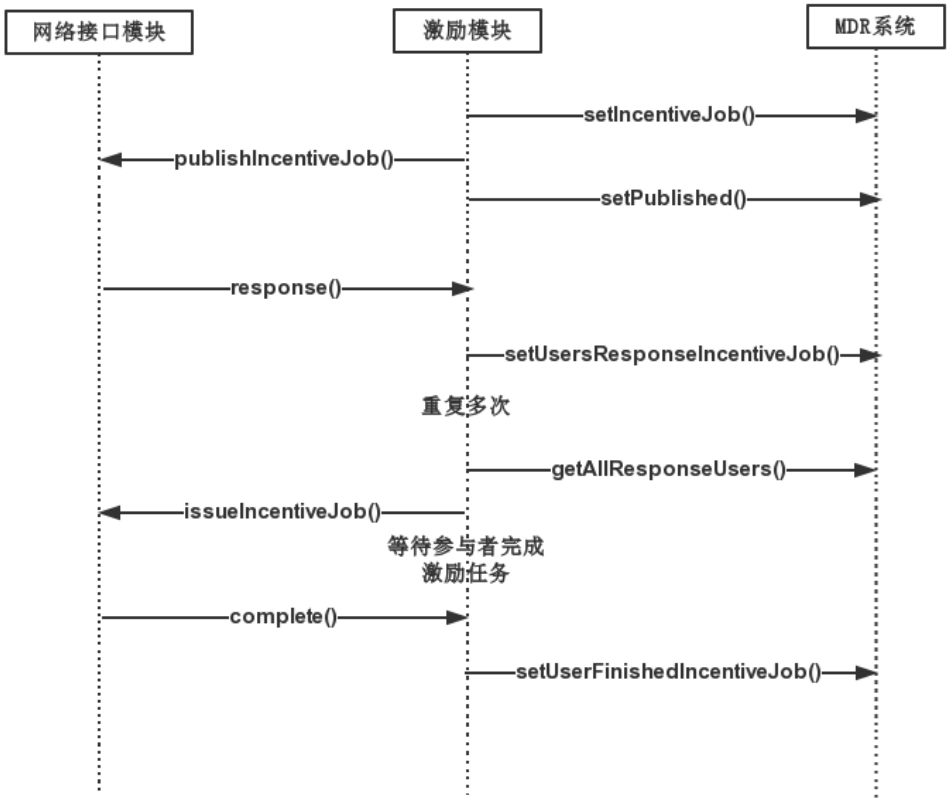

图 4-11 发布、跟踪激励任务表的顺序图

发布、跟踪激励任务的流程如图 4-11 所示：

- 首先激励模块发现需要采集特定地理位置的数据，它通过 setIncentiveJob 函数向 MDR 系统提交一个新的激励任务
- 提交激励任务成功后，激励模块通过特定算法选取一批合适的参与者，调用网络接口模块的 publishIncentiveJob 函数向被选中的参与者推送激励任务，并修改激励任务的状态为已推送。
- 随后一段时间内如果任何被选中的参与者选择接受任务，则激励模块通过 setUsersResponseIncentiveJob 函数向数据库添加接受用户
- 一段时间后，激励模块通过 getAllResponseUsers 接口读取所有的接受任务的参与者，并选择最终授予激励任务的参与者，然后开始等待跟踪激励任务的完成
- 当参与者回应激励任务并上传指定数据之后，激励模块通过 setUserFinishedIncentiveJob 函数记录激励任务的完成用户以及完成时间，并为用户发放激励报酬
- 最后激励模块为用户发放报酬，并更新用户的总收益以及参与次数

4.7. 对比试验：MySQL

为了在性能测试阶段做对比测试，我们尝试使用传统关系型数据库来承担数据存储的任务，MySQL 是当今世界最流行的开源关系型数据库，所以我们尝试利用 MySQL 来实现 MDR 系统所需的功能。

根据前面章节可知，MySQL 拥有多种数据存储引擎，其中 InnoDB 提供了行级锁(locking on row level)，保证了即使在高并发写的情景下也能保证性能。所以我们最终选择 InnoDB 作为存储引擎。其次，我们知道每次执行 SQL 命令都需要与数据库建立连接，建立连接需要消耗很多时间，为了加快 SQL 命令的执行，我们建立一个连接数据库的连接池。之后，任何 SQL 命令的执行不需要重新建立数据库连接，而是从连接池里取一个可用的连接，节约了时间，获得更好的性能。数据库连接信息通过配置文件进行配置，配置文件如图 4-12 所示：

```
<?xmlversion="1.0"encoding="UTF-8"?>
<!-- 数据库连接信息 -->
<databaseConnection>
    <driver>com.mysql.jdbc.Driver</driver>
    <url>jdbc:mysql://localhost:3306/</url>
    <dbs>
        <db>PS</db>
        <user>root</user>
        <pass>root</pass>
        <minConn>10</minConn>
        <maxConn>10</maxConn>
        <timeout>2000</timeout>
</dbs>
</databaseConnection>
```

图 4-12 SQL 语句配置

通过配置文件可以配置数据库的地址、数据库名、登入数据库的用户名和密码、数据库连接池的最少数量和最大数量以及 SQL 命令执行的超时时间。执行 SQL 命令的流程图如图 4-13 所示：

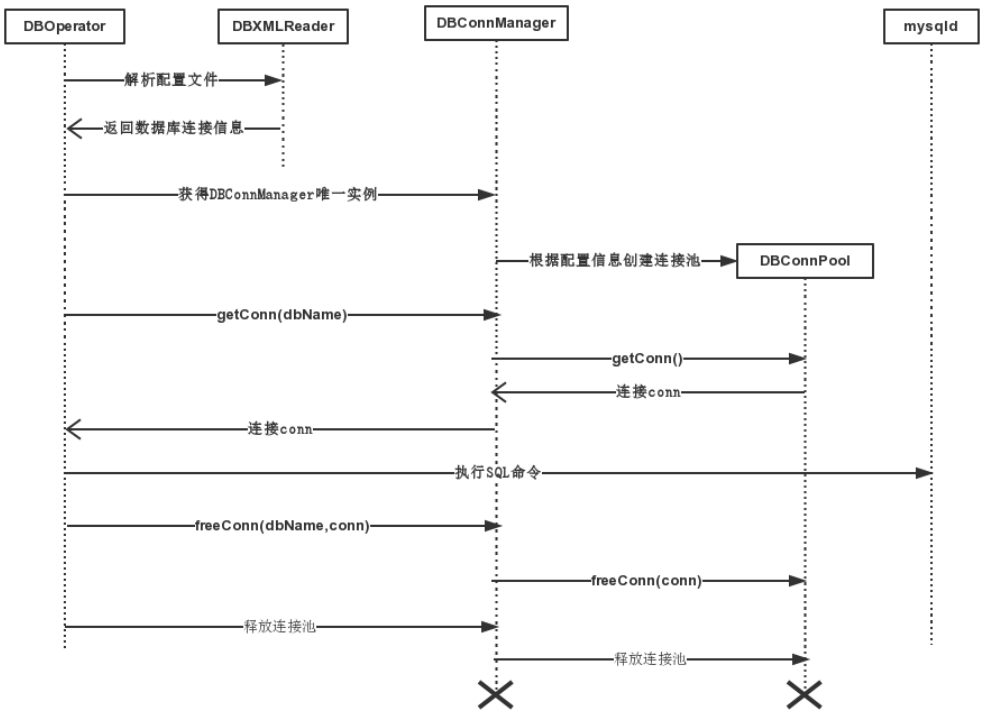

图 4-13 通过连接池 SQL 语句执行顺序图

其中，DBOperator 充当使用者的角色，DBXMLReader 负责解析图 4-12 的配置文件，DBConnManager 管理所有数据库的连接池，但其并不负责维护连接池，只是将连接请求转发给特定的 DBConnPool，而 DBConnPool 则维护特定数据库的所有连接。DBConnManager 实现为单例模式，只创建连接池一次。

DBOperator 首次运行的时候，调用 DBXMLReader 执行配置文件的解析工作，获得数据库的连接信息，并将数据库的连接信息作为参数获得 DBConnManager 的唯一实例。DBOperator 通过 DBConnManager 的唯一实例来请求连接以及释放连接，DBConnManager 将用户的真正请求转发给特定数据库的 DBConnPool 执行。最后 DBOperator 调用函数释放所有连接，清空连接池。

连接池里的连接可能会出现都被占用的情况，请求空闲连接的过程图如图 4-14 所示。

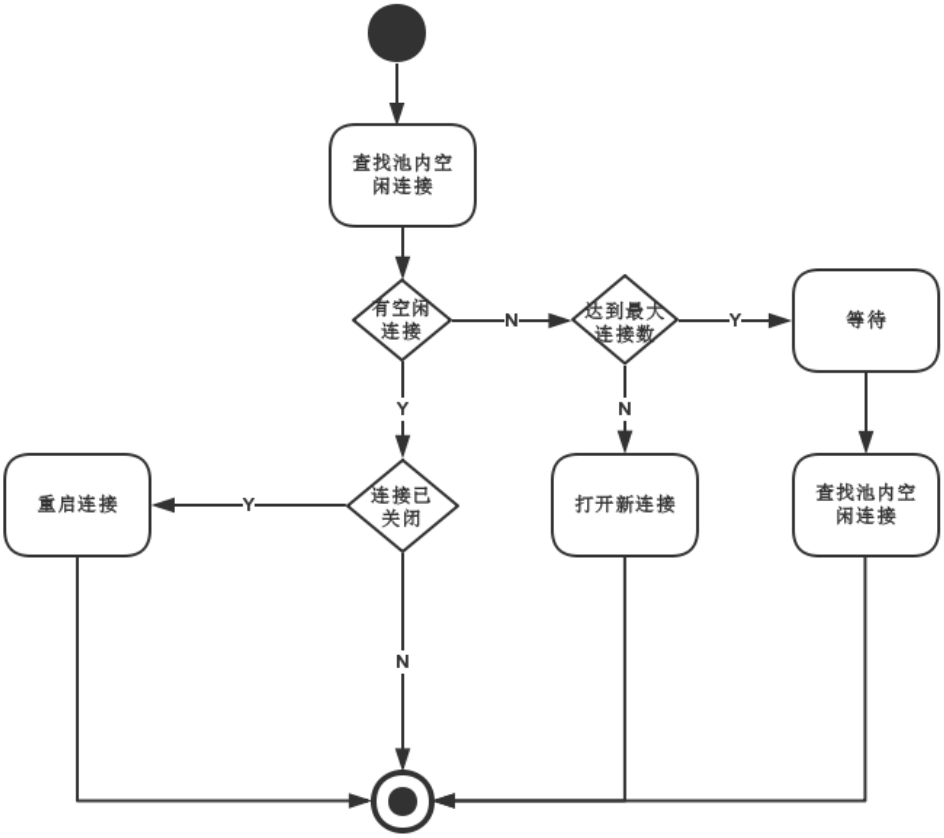

图 4-14 查找空闲连接的流程图

首先尝试在连接池中查找空闲的连接，如果查找成功则判断连接是否是关闭状态，如果关闭则需要重新打开连接再返回；当连接池中没有空闲连接的时候，先判断池内的连接数是否达到了配置文件中的最大连接数，如果没有则新建连接，并返回；当池内的连接数已经达到了最大值，则等待配置文件中 SQL 执行超时时间后，再次尝试在连接池中查找空闲连接，不管是否成功找到都返回结果。

4.8. 开发与运行环境

系统的开发环境如下：

- 后台编程语言：Java（JDK 1.7.0）
- 集成开发环境：Eclipse
- 打包与构建工具：Ant 1.9.4
- 编辑器： vim
- 版本控制：svn

系统的运行环境如下：

- 操作系统：Ubuntu 12.04
- 内核版本：Linux 3.2.0-20-generic

- 非关系型数据库：Apache HBase 0.94.19
- 关系型数据库：MySQL 5.5.38
- 底层文件系统：Hadoop Distributed File System 0.23.10

4.9. 本章小结

本章首先在需求分析以及 HBase 的技术原理的基础上，详细描述了 MDR 系统如何针对不同模块的功能需求和性能需求，来设计存储表的结构和接口；然后，同时也构建了一个基于 MySQL 的数据存储检索系统，为了之后与 MDR 系统做对比试验；最后介绍了系统的开发和运行环境。

第五章　系统测试

本章详细描述 MDR 系统的测试工作，首先介绍系统的测试环境，然后针对各个子系统进行单元测试，接着对整个系统进行集成测试，演示并检验运行效果，最后对测试结果进行分析。

5.1. 测试环境

因为参与式感知项目目前仍处于开发阶段，MDR 系统目前以单机的模式进行部署运行。幸运地是，后期可以非常简单和方便地演变成分布式模式。服务器的配置如下：CPU 是 Intel(R) Xeon(R) 2.00 GHz，16GB 内存和 750GB 硬盘。服务器的操作系统版本为 Ubuntu 12.04，内核版本为 Linux 3.2.0-20-generic，数据库为 Apache HBase 0.94.19 和 MySQL 5.5.38，系统的打包发布工具是 Ant 1.9.4。

5.2. 功能性测试

功能性测试，也称为黑盒测试，指不了解程序内部运行逻辑的情况下，通过精心设计的测试用例，验证程序的每个功能在任何场景都能正确工作。对系统的功能测试我主要采用了黑盒测试，同时设计测试用例的时候考虑输入不合法、边界值等异常情况来验证系统的鲁棒性。

5.2.1. 数据上传

通过表 5-1 所示的功能性测试用例，可以验证数据上传功能的正确性。例如用例 1-1 中，尝试使用多个线程同时向消息队列中插入数据，同时多个线程从消息队列中读取数据，实验结果表明即使在多线程读写的场景下，依然可以保证每个线程正确地向消息队列写入或读取数据，同时消息队列空时读失败，消息队列满时，写失败。

如图 5-1 所示，模拟数据写入操作之前，感应器数据存储表中的记录为空。

```
hbase(main):008:0> scan 'sensoryData'
ROW                                           COLUMN+CELL
0 row(s) in 0.0790 seconds
```

图 5-1 模拟数据写入操作之前，感应器数据存储表为空

接着模拟数据并发写入操作，开启 10 个生产者线程随机生成如图 4-1 格式的 JSON 字符串来模拟用户收集的数据，通过 RMI 写入到消息队列中，同时开启 10 个消费者线程不断从消息队列中读取数据并写入到感应器数据存储表中。测试中，任意生产者线程

进行 10,000 次写入操作，每次写入 4 条数据。此时感应器数据存储表中的记录如图 5-2 所示，第一列为行键值，格式为时间戳/用户名，其中用户名由随机挑选的英文字符生成。第二列分别列出了不同 column（包括经度、纬度、声音等）的最新版本的值，因为 HBase 保存的都是连续的字节数组，例如经度的值是 double 类型，转换成字节数组展示成?\xEL+\Xe3.\xFE\xED。

```
1389716006000/yDpFem                column=info:Longitude, timestamp=1419080211348, value=?\xE1+\xE3.\xFE\xED)
1389716006000/yDpFem                column=info:Noise, timestamp=1419080211348, value=@;\x00\x00\x00\x00\x00\x00
1389716063000/UfPgNR                column=info:Latitude, timestamp=1419080214788, value=?\xD0\xB2\xCAY\x0C\x81\x1C
1389716063000/UfPgNR                column=info:Longitude, timestamp=1419080214788, value=?\x93\xEB\x99\xE0x\x91\x80
1389716063000/UfPgNR                column=info:Noise, timestamp=1419080214788, value=@M\x00\x00\x00\x00\x00\x00
1389716064000/srmGNv                column=info:Latitude, timestamp=1419080209766, value=?\xEE\xB5b\xCC\xD7I\x91
1389716064000/srmGNv                column=info:Longitude, timestamp=1419080209766, value=?\xEE\x98\x14v:\xA3V
1389716064000/srmGNv                column=info:Noise, timestamp=1419080209766, value=@D\x80\x00\x00\x00\x00\x00
1389716078000/FGoNhG                column=info:Latitude, timestamp=1419080214905, value=?\xE9- \xD4\xBAF0
1389716078000/FGoNhG                column=info:Longitude, timestamp=1419080214905, value=?\xECW\x18\xBB\xD1\xDF,
1389716078000/FGoNhG                column=info:Noise, timestamp=1419080214905, value=@<\x00\x00\x00\x00\x00\x00
1389716078000/QUsIGg                column=info:Latitude, timestamp=1419080216553, value=?\xC30\xF3\x9B\xFD\x1F\x00
1389716078000/QUsIGg                column=info:Longitude, timestamp=1419080216553, value=?\x94bw\xFAm\x5C\xC0
1389716078000/QUsIGg                column=info:Noise, timestamp=1419080216553, value=@S\x00\x00\x00\x00\x00\x00
1389716084000/Neagou                column=info:Latitude, timestamp=1419075988125, value=?\xE6\xED\xDF\xC0\xAB\xF9\xEC
```

图 5-2 感应器数据存储表中记录截图

当所有生产者线程完成数据写入操作，并且消息队列中的数据都已经被写入到 HBase 中时，再次计算感应器数据存储表中记录数目如图 5-3 所示，与写入的数据条数相同。通过实验可以证明即使在多线程的环境中，依然可以保证数据的完整。

```
Current count: 290000, row: 1416777850000/pStFar
Current count: 300000, row: 1416813017000/gXBBbe
Current count: 310000, row: 1416845688000/WMTPJH
Current count: 320000, row: 1416874804000/uGOgKh
Current count: 330000, row: 1416907632000/IkrMQF
Current count: 340000, row: 1416936571000/trpCuy
Current count: 350000, row: 1416966589000/JrvAJp
Current count: 360000, row: 1417001288000/yXydzp
Current count: 370000, row: 1417032973000/pNcDBK
Current count: 380000, row: 1417066916000/mmmkNU
Current count: 390000, row: 1417097757000/xBGGHP
400000 row(s) in 15.0300 seconds
```

图 5-3 模拟数据写入操作之后，感应器数据存储表的记录数

表 5-1 数据上传功能性测试

用例编号	测试用例	预测结果	测试结果
1-1	线程安全的有容量限制的消息队列	即使在多线程的环境下，读写消息队列依然工作正确，当消息队列空时，读失败，当消息队列满时，写失败。	符合
1-2	RMI 测试	验证网络接口模块可以通过 RMI 对消息队列进行写操作	符合
1-3	上传感应器数据类型	网络接口模块通过 RMI 写入 JSON 格式数据到消息队列之后，消费者线程从消息队列中读取数据并解析后插入到数据库中	符合

（续上表）

表 5-1 数据上传功能性测试

用例编号	测试用例	预测结果	测试结果
1-4	JSON 解析	将感应器数据解析为 Put 操作	符合
1-5	上传图像数据类型	网络接口模块通过 RMI 写入 JPEG 格式数据到消息队列之后，消费者线程从消息队列中读取数据并解析后插入到数据库中	符合
1-6	EXIF 解析	将图像数据解析为 Put 操作	符合
1-7	图像分割	如果图像的大小超过设置的最大值，则将图像分割成多个分片，每个分片均小于或等于最大值，并依次存入数据库	符合

5.2.2. 采集数据展示

通过表 5-2 所示的功能性测试用例，可以验证采集数据展示功能的正确性。通过在数据库中写入精心准备的伪数据，然后验证各接口读取的数据是否符合预期来验证正确性。其中用例 1-7 和用例 2-5 通过对图像文件进行分割再合并操作，通过比较输入文件与输出文件来验证分割和合并操作的正确性。

表 5-2 采集数据展示功能性测试

用例编号	测试用例	预测结果	测试结果
2-1	客户端展示照片墙	展示由客户端指定时间内所有用户上传的照片（压缩图）。	符合
2-2	客户端查看原图	客户端通过点击任意的压缩图来展示该压缩图的原图	符合
2-3	Web 获取 POI 点信息	Web 展示模块根据地图的缩放情况，读取视图中包含的 POI 点信息	符合
2-4	web 展示收集的照片和 PM2.5	web 展示模块通过用户选择 POI 点，读取该 POI 点下最近采集的照片以及照片的 PM2.5 值	符合
2-5	图像合并	如果从数据库读出的是图像分片列表，需要根据分片的时间戳按照顺序合并，恢复原图	符合

集成测试中，web 端展示截图和客户端展示截图分别如图 5-4 和图 5-5 所示，并且展示数据与数据库中一致，多媒体数据例如图片展示正常，证明支持采集数据展示中各接口功能正常。

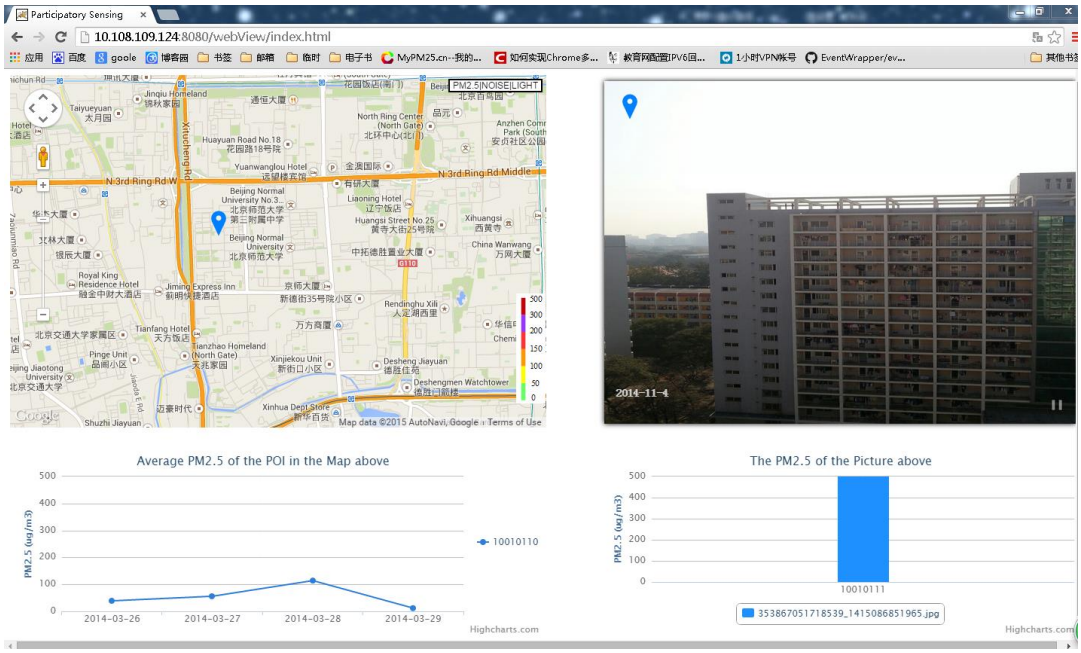

图 5-4 web 展示截图

图 5-5 客户端展示截图

5.2.3. 图像分析模块

通过表 5-3 所示的功能性测试用例，可以验证对图像分析模块提供接口的正确性。通过模拟爬取程序利用写接口向空气质量存储表，天气存储表、图像模型存储表写入数据，接着通过读接口分别从空气质量存储表、天气存储表、图像模型存储表中读取数据，比对输入数据和输出数据来验证读取接口工作正确。

表 5-3 图像分析模块接口测试

用例编号	测试用例	预测结果	测试结果
3-1	存储空气质量信息	将空气质量信息（城市、观测站 id、时间、PM2.5 浓度、PM10 浓度、AQI）成功写入数据库	符合
3-2	存储天气信息	将天气信息（城市、时间、温度、湿度、风速、降雨量、气压、天气类型）成功写入数据库	符合
3-3	获取时间最近的观测站信息	从数据库读取城市相同、观测站相同且存储时间与给定的时间参数最接近的信息（PM2.5 浓度、PM10 浓度、AQI）	符合
3-4	获取时间最近的观测站信息	从数据库读取城市相同且存储时间与给定的时间参数最接近的信息（温度、湿度、风速、降雨量、气压、天气类型）	符合
3-5	存储图像模型	将手机型号对应的图像模型成功写入数据库	符合
3-6	图像模型查询	从数据库中读取手机型号相同且模型形的版本是给定的时间参数之前的最新版本	符合

5.2.4. 数据融合模块

通过表 5-4 所示的功能性测试用例，可以验证对数据融合模块提供接口的正确性。通过从感应器数据存储表中抽样出一份测试数据，验证对数据融合模块提供的接口在测试数据上运行正确。以此判断接口的正确性。

表 5-4 数据融合模块接口测试

用例编号	测试用例	预测结果	测试结果
4-1	计算网格 Grid	感应器数据存储表中，每一条记录由经纬度计算数据所属的网格，并将计算结果存入感应器数据表	符合
4-2	获取指定的网格内收集的数据	感应器数据存储表中，读取属于指定网格的感应器数据列表	符合
4-3	获取一段时间内的收集的数据	感应器数据存储表中，读取一段时间内的数据列表	符合
4-4	获取所有的网格信息	感应器数据存储表中，读取所有的网格	符合
4-5	存储融合结果	融合结果表中存储，每个网格最终的融合结果	符合

5.2.5. 轨迹预测模块

通过表 5-5 所示的功能性测试用例，可以验证对轨迹预测模块提供接口的正确性。通过从感应器数据存储表中抽样出一份测试数据，验证对轨迹预测提供的接口在测试数据上运行正确。以此判断接口的正确性。

表 5-5 轨迹预测模块接口测试

用例编号	测试用例	预测结果	测试结果
5-1	获取一条轨迹路线上用户的运动轨迹点列表	感应器数据存储表中，获取一条轨迹路线上用户的运动轨迹点	符合
5-2	记录轨迹路线用户的运动区域列表	轨迹表中，记录轨迹路线用户的运动区域列表	符合
5-3	读取轨迹路线用户的运动区域列表	轨迹表中，读取轨迹路线用户的运动区域列表	符合

（续上表）

表 5-5 轨迹预测模块接口测试

用例编号	测试用例	预测结果	测试结果
5-4	记录一阶转移概率	转移概率存储表中，记录用户从起始区域到终点区域的一阶转移概率	符合
5-5	记录二阶转移概率	转移概率存储表中，记录用户从起始区域到终点区域的二阶转移概率	符合

5.2.6. 激励模块

通过表 5-5 所示的功能性测试用例，可以验证对激励模块提供接口的正确性。通过模拟激励模块发布新的激励任务、模拟用户响应、完成激励任务，验证对激励提供的接口正确性。

表 5-6 激励模块接口测试

用例编号	测试用例	预测结果	测试结果
6-1	新增激励任务	激励任务存储表中，新增一条描述了收集数据类型、收集地点、报酬、推送状态的激励任务	符合
6-2	修改激励任务状态	激励任务被推送后，修改该激励任务在激励任务存储表中的状态为已推送	符合
6-3	添加接受激励任务的用户	激励任务存储表中，将新接受的用户记录在接受用户列表中	符合
6-4	读取接受激励任务的用户列表	激励任务存储表中，读取所有的接受激励任务的用户列表	符合
6-5	确定激励任务的最终完成者	激励任务存储表中，记录激励任务最终完成者的完成时间和最终激励	符合
6-6	更新用户信息	用户存储表中，更新用户的总收益以及参与次数	符合

5.2 非功能性测试

非功能测试主要集中测试实时上传数据、实时数据展示的性能，以及多媒体数据读写的性能。通过比对 MDR 系统和 MySQL 在以上测试环境中的表现来验证 MDR 系统

可以胜任参与式感知项目的数据存储与检索工作。

5.2.1. 数据上传性能测试

参与式项目的数据全部来自参与者上传，而数据又是所有数据分析和挖掘的基石，所以高效地实时上传数据是系统最重要也是最基本的功能，我们希望即使在高并发的压力下，MDR 系统仍旧可以正常工作。

我们通过随机生成如图 4-1 格式的 JSON 字符串来模拟用户收集的数据，然后通过 RMI 写入到消息队列中，模拟用户上传的操作。再者，可以通过增加模拟用户上传操作的线程数目来进行压力测试。测试中，任意模拟线程进行 10,000 次上传操作，每次上传 4 条数据。我们分别将模拟线程设为 1、10、20、100 进行多组性能测试，每组性能测试中消息队列的容量设为 1,000，HBase 中感应器数据存储表的写缓存设为 2M。基于 HBase 性能测试的结果如表 5-7 所示。

表 5-7 基于 HBase 数据上传性能测试结果

模拟线程数目	消费者线程数目	运行时间（秒）	上传请求/秒	数据丢失率
1	20	16.4	609.8	0%
10	20	66.7	1499.3	0%
20	20	72.5	2758.6	0%
100	20	646.4	1547.0	0%
100	50	577.1	1732.8	0%
100	100	427.9	2337.0	0%

由表 5-7 可知，当模拟线程小于 100 的时候，处理请求的吞吐量快速增长，当模拟线程达到 100 的时候，吞吐量因为请求不能被快速处理而有所降低，但当我们尝试增加消费者线程数目的时候，吞吐量又有显著的增长。但是当线程数目过多会造成资源竞争频繁，影响性能，造成吞吐量无法保持直线增长。在所有进行的性能测试中，没有发生数据丢失的情况。

另一方面，我们将底层 HBase 数据库换成 MySQL,进行了同组试验。感应器数据存储表的结构如图 5-6 所示。

```
+------------+--------------+------+-----+-------------------+-----------------------------+
| Field      | Type         | Null | Key | Default           | Extra                       |
+------------+--------------+------+-----+-------------------+-----------------------------+
| dataId     | int(11)      | NO   | PRI | NULL              | auto_increment              |
| username   | varchar(50)  | YES  |     | NULL              |                             |
| uploadtime | timestamp    | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
| lon        | double(11,7) | YES  |     | NULL              |                             |
| lat        | double(11,7) | YES  |     | NULL              |                             |
| light      | int(11)      | YES  |     | NULL              |                             |
| noise      | int(11)      | YES  |     | NULL              |                             |
+------------+--------------+------+-----+-------------------+-----------------------------+
```

图 5-6　MySQL 数据库中的感应器数据存储表的结构

如图 5-6 所示，感应器数据存储表的主键为 dataId,当新数据插入，dataId 自增一，其他属性除了 uploadtime 都可为 NULL，username 是变长字符串，lon 和 lat 为 double 类型，light 和 noise 为 int 类型。由系统详细设计的章节可知，我们通过连接池来节约向 MySQL 建立连接的时间，连接池的容量与消费者线程保持一致。基于 MySQL 性能测试的结果如表 5-8 所示。

表 5-8 基于 MySQL 数据上传性能测试结果

模拟线程数目	消费者线程数目	连接池容量	运行时间（秒）	上传请求/秒	数据丢失率
1	20	20	23.64	423.61	0%
10	20	20	260.14	384.41	0%
20	20	20	525.63	380.50	0%
100	100	100	6697.21	149.32	13.2%

通过比较表 5-7 和表 5-8 的性能测试结果，HBase 表现出更好的写性能，展现出更高的吞吐量。当模拟线程数目达到 100 时，MySQL 发生了数据丢失，影响了系统的可用性。

与此同时，我们还记录比较了消费者线程解析 JSON 并进行单次数据库写操作的运行时间，当使用 HBase 作为底层存储系统的时候，平均运行时间是 0.63ms，99.8%的操作在 10ms 之内完成，只有 0.0429%的操作超过了 100ms。长时间的操作是因为写缓存满了，一次 RPC 被启动。另一方面，MySQL 处理一次操作的平均时间为 5.21ms，约为 HBase 的 8 倍，其中 83.7%的操作在平均时间内完成，当线程数增长到 100 时候，一次写入操作消耗的最长时间长达好几分钟。

5.2.2. 图像读写性能测试

图像读写性能测试包括对图像导入和图像读取两种操作的性能测试，其中图像导入包括两个步骤：图像分割和图像块写入；图像读取也分为两个步骤：读取图像块和图像块拼接。我们进行了 4 组性能测试，图像数据集的大小分别为 100MB、515MB、1.1GB 和 5.1GB。4 组性能测试的运行时间如表 5-9 所示。图像导入的平均吞吐率为 60MB/s+，图像读取的平均吞吐率为 120MB/s。

表 5-9 基于 HBase 的图像读写性能测试结果

	5.1GB	1.1GB	515MB	100MB
图像顺序导入	96000ms	18597ms	8958ms	1825ms
图像顺序读取	40478ms	8932ms	4092ms	756ms

另一方面，我们将底层 HBase 数据库换成 MySQL,进行了同组试验。多媒体数据存储表的结构如图 5-7 所示。

```
mysql> describe multimedia;
+-----------+--------------+------+-----+-------------------+-----------------------------+
| Field     | Type         | Null | Key | Default           | Extra                       |
+-----------+--------------+------+-----+-------------------+-----------------------------+
| dataId    | int(11)      | NO   | PRI | NULL              | auto_increment              |
| username  | varchar(50)  | YES  |     | NULL              |                             |
| Time      | timestamp    | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
| Longitude | double(11,7) | YES  |     | NULL              |                             |
| Latitude  | double(11,7) | YES  |     | NULL              |                             |
| fpm       | int(11)      | YES  |     | NULL              |                             |
| path      | varchar(100) | YES  |     | NULL              |                             |
| content   | mediumblob   | YES  |     | NULL              |                             |
| poi       | char(8)      | YES  | MUL | NULL              |                             |
+-----------+--------------+------+-----+-------------------+-----------------------------+
```

图 5-7　MySQL 数据库中的多媒体数据存储表的结构

如图 5-7 所示，多媒体数据存储表的主键为 dataId,当新数据插入，dataId 自增一，其他属性除了 uploadtime 都可为 NULL，username 是变长字符串，lon 和 lat 为 double 类型，content 类型为 mediumblob，存储多媒体文件的内容，mediumblob 可以存储 16M 以下的文件。基于 MySQL 的性能测试的运行时间如表 5-10 所示。图像导入的平均吞吐率平均为 40MB/s，图像读取的平均吞吐率为 140MB/s。

表 5-10 基于 MySQL 的图像读写性能测试结果

	5.1GB	1.1GB	515MB	100MB
图像顺序导入	152337ms	28536ms	12146ms	2290ms
图像顺序读取	46768ms	8310ms	3702ms	720ms

通过比较表 5-9 和表 5-10 的测试结果，在图像导入操作中，HBase 表现出更优异的吞吐率，如图 5-3 所示（柱状图左边为 MySQL 右边为 HBase，图 5-4 相同）。与此同时，从图 5-8 我们可以看出，MySQL 的写速度随着数据量的增大而有所降低。

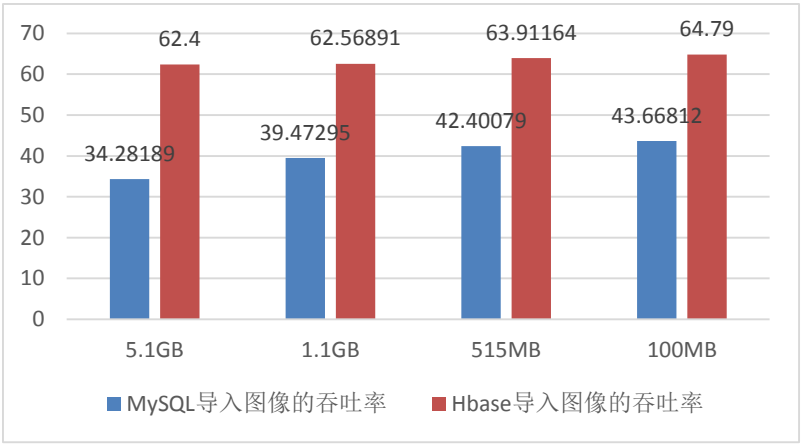

图 5-8 图像导入吞吐率的柱状对比图

如图 5-4 所示，在图像读取操作中，当数据量比较小的时候，MySQL 的读取吞吐率略高于 HBase，但当数据量略大的时候（51GB，4184 张照片）MySQL 的吞吐率有

所降低，而 HBase 保持吞吐率不变。

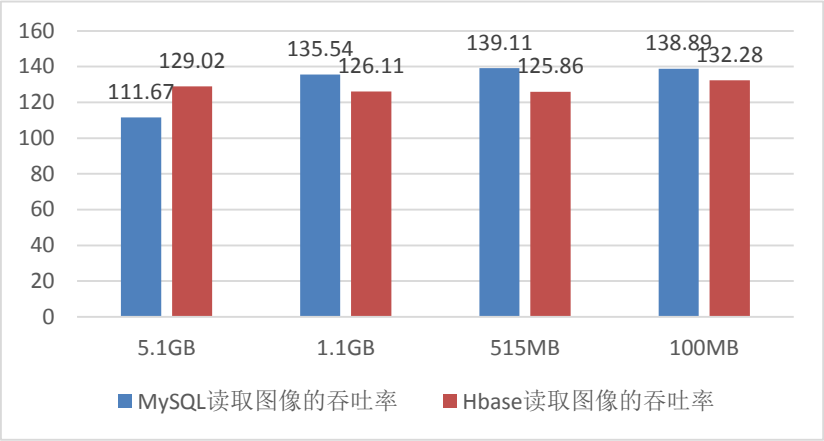

图 5-9 图像读取吞吐率的柱状对比图

5.2.3. web 展示实时接口性能测试

- 获取 POI 信息接口：public Set<JSONObject>getPOIsInfo(), web 在展示收集数据之前，需要根据地图的缩放情况，将在视图中展示 POI 信息载入内存。我们进行了 4 组性能测试，POI 点的个数分别为 100，1000，10,000 和 100,000，每组测试都比较了 HBase 环境和 MySQL 环境的性能表现。其中 MySQL 数据库 POI 表的结构如图 5-5 所示。针对 HBase 环境和 MySQL 环境的获取 POI 信息接口的性能测试的吞吐率对比结果如图 5-6 所示（下折线为 Mysql 上折线为 HBase）。

```
mysql> describe POI;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| poi       | char(8)      | NO   | PRI | NULL    |       |
| min_level | int(11)      | YES  |     | NULL    |       |
| max_level | int(11)      | YES  |     | NULL    |       |
| Longitude | double(11,7) | YES  |     | NULL    |       |
| Latitude  | double(11,7) | YES  |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

图 5-10 MySQL 数据库中 POI 表的结构

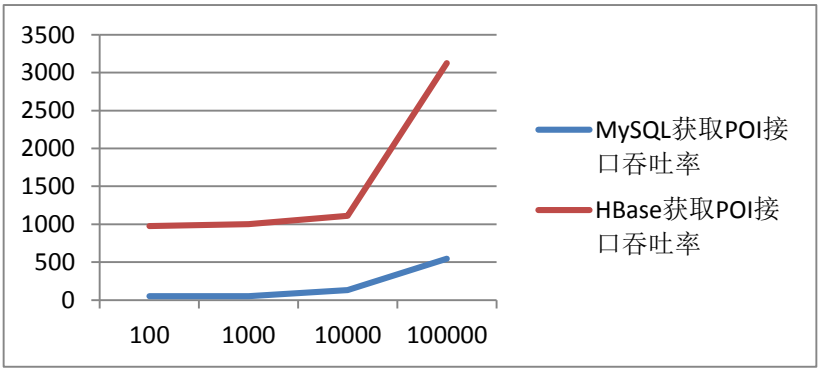

图 5-11 获取 POI 信息接口的性能测试的吞吐率结构

如图 5-11 所示，当 POI 数据分别为 100，1000，10,000 的时候，HBase 环境的吞吐率约为 1000 req/ms,远远高于 MySQL 环境。另当 POI 数据增长到 100,000 个的时候，HBase 的吞吐率呈指数增长，是 MySQL 环境平均吞吐率的 6 倍。

- 获取 POI 点下收集的图片接口： public List<Pic> getPic(String poiPrefix, Date timestamp)，获取符合 poiPrefix 的所有 POI 点在 timestamp 日期收集的图片，若 timestamp 为 null 则获取所有收集图片。我们进行了 4 组性能测试，图像数据集的大小分别为 100MB、515MB、1.1GB 和 5.1GB，每组性能测试分别将数据集中的图片随机放在实际的 31 个 POI 点中，并随机请求 POI 点下收集的图片。针对 HBase 环境和 MySQL 环境的获取 POI 下收集的图片接口的平均吞吐率如表 5-12 所示（柱状图左边为 MySQL 右边为 HBase），柱状图如图 5-7 所示。

表 5-12 获取 POI 点下收集的图片接口的性能测试结果

	5.1GB	1.1GB	515MB	100MB
HBase 环境下接口的吞吐率（MB/s）	198.37	278.95	245.47	256.62
MySQL 环境下接口的吞吐率（MB/s）	104.75	208.14	179.99	188.68

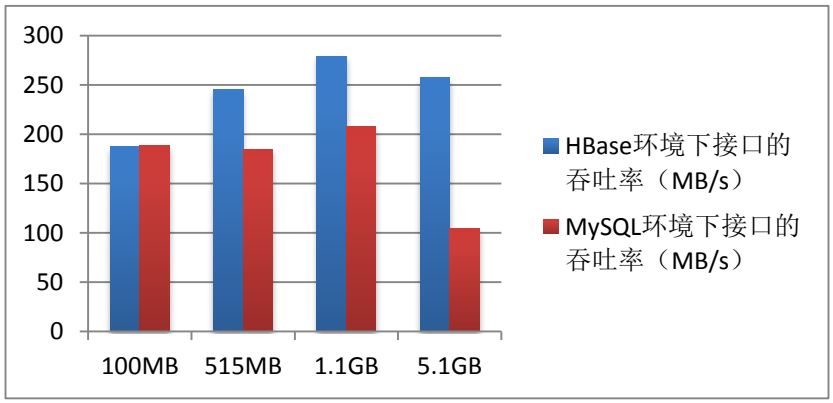

图 5-12 获取 POI 下收集的图片接口的性能测试的吞吐率结果

如图 5-12 所示，4 组性能测试中，HBase 环境下的接口表现出更良好的吞吐率，证明通过二级索引，HBase 也可以为实时应用提供存储和存取服务。

5.3 本章小结

本章节详细介绍了 MDR 系统的测试结果，验证系统已经达到了需求分析章节的期望。功能测试部分详细列举并验证 MDR 系统对外提供的接口，非功能性测试部分着重

对实时应用和接口进行性能测试，将 HBase 与 MySQL 的性能结构进行详细对比。

第六章　结束语

6.1. 全文总结

本论文构建了一个稳定的、高效的、扩展性良好的分布式数据存储检索系统，存储的数据包括感应器数据和多媒体数据。分布式数据存储检索系统可以承担高并发的数据上传，并为实时应用提供了高效的读写接口，为非实时应用的数据挖掘工作提供了流式读写的接口。

本论文创新之处有两点：

首先引入了开源的分布式 key/value 数据库 HBase，抛弃了被 ACID 约束的传统关系型数据库，大大提高了写性能，使得承担高并发的数据上传成为可能。与此同时，HBase 提供了更好的扩展性，使得参与式感知系统发展成一个云平台成为可能；最后运行在 HBase 之上的 MapReduce 非常适合海量数据的挖掘和分析。但是 HBase 无法像 RDBMS 一样提供对多维数据丰富的查询能力，所有的 key/value 数据库都只有在确定 key 的情况下搜索 value 才能达到最高的性能，否则只能做全表扫描操作。项目中主要利用 3 种方案解决二级属性查询问题：方案一将属性加到行键中，因为对行键的检索非常高效，可以提高属性的检索速度；方案二通过限定扫描的区间提高检索效率；方案三通过建立属性的二级索引达到提高检索效率的目的。

在 HBase 上实现了将大型多媒体文件存储自动分片化。多媒体文件的大小已经超过 HBase 适合的数据块的大小，所以无法直接将多媒体文件存入 HBase 的单元格。解决这个问题方法是，将大于数据块大小限制的多媒体文件进行切片，将每个分片的大小限制在数据块大小限制之内，然后将每个分片保存起来。我们利用时间戳来记录分片存储顺序，将多媒体文件的各个切片按顺序逐一保存在同一个单元格，HBase 会自动打上时间戳。如此以来，只需要根据行键和列键就可以找到同一多媒体文件的所有切片。然后按照每个切片时间戳的时间顺序把各个切片进行合并，即可恢复出原始的多媒体文件。

本论文首先介绍了参与式感知的背景和意义，以及目前关于如何存储感知数据和多媒体数据的研究成果，并尝试摒弃传统的关系型数据库，利用非关系型数据库来承担数据存储检索的任务。HBase 是一个开源的、高可靠性、高性能、具有良好伸缩性的分布式结构化的存储系统。然后，简略地介绍了完整的参与式感知项目，详细阐明了 MDR 系统的需求，并在需求分析和 HBase 自身属性的基础上进行数据库表结构和接口的详细设计，最后通过完备的功能性和非功能性的测试来验证系统。

6.2. 未来工作展望

在未来的工作中，还需要从以下几个方面来改进本文中实现的数据存储系统：

- 支持录音、视频类型的数据存储与数据挖掘
- 将本论文中的数据存储系统部署到服务器集群中去，将单机的消息队列替换成分布式消息队列
- 通过协处理器将计算任务放在“数据服务器”端执行，而不是将数据全部提取出来由“HBase 客户端”执行，减少传输数据，节约资源

6.3. 研究生期间工作

在研究生期间，论文作者参与的工作主要有：

一、参加国家重点 863 项目“Triple Networks Integration based on Openflow”

- 参与网络状况的实时 web 展示系统的技术调研和可行性分析
- 参与系统的需求分析
- 参与系统详细设计
- 负责 web 服务器端开发
- 负责 MySQL 数据库开发
- 负责 SNMP 协议以及 MIB 数据库开发

二、参加华为合作项目“Autonomous Network based on TCP/IP”

- 负责 4 人小组的协调工作
- 参与系统的需求分析
- 参与系统详细设计
- 负责底层协商协议的开发
- 负责协商协议客户端与服务器端开发

三、参加国家自然基金项目“Participatory Sensing System”

- 参与系统的需求分析
- 参与系统详细设计
- 负责数据存储的前期技术调研和可行性分析
- 负责数据存储系统的开发
- 参与数据存储系统与其他模块的接口设计

6.4. 学术输出

- 以第一作者身份向科技论文在线发表学术论文一篇，题目是《Realization, Simulation and Analysis of DASH Service Based on QualNet》，检索号：201412284

参考文献

[1] Wikipedia. http://en.wikipedia.org/wiki/DIKW_Pyramid.

[2] LiKamWa R, Liu Y, Lane N D, et al. Moodscope: building a mood sensor from smartphone usage patterns[A].//Proceeding of the 11th annual international conference on Mobile systems, applications, and services[C]. ACM, 2013: 389-402.

[3] Wikipedia. http://en.wikipedia.org/wiki/Participatory_sensing.

[4] http://research.microsoft.com/en-US/projects/urbancomputing/cn.aspx

[5] Junya Niwa, Kazuya Okada, Takeshi Okuda, and Suguru Yamaguchi. Mpsdatastore: a sensor data repository system for mobile participatory sensing[A]. //In Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing[C], pages 3–8. ACM, 2013.

[6] Diego Mendez, Alfredo J Perez, Miguel A Labrador, and Juan Jose Marron. P-sense: A participatory sensing system for air pollution monitoring and control[A]. //In Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on[C], pages 344–347. IEEE, 2011.

[7] Salil S Kanhere. Participatory sensing: Crowdsourcing data from mobile smartphones in urban spaces[A]. //In Mobile Data Management (MDM), 2011 12th IEEE International Conference on[C], volume 2, pages 3–6. IEEE, 2011.

[8] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data[J]. ACM Transactions on Computer Systems (TOCS), 2008, 26(2):4.

[9] Wiki H. HBase: bigtable-like structured storage for Hadoop HDFS[J]. 2012-02-23)[2012-04-17]. http://wiki. apache, org/hadoop/Hbase.

[10] Shoji Nishimura, Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. Md-hbase: a scalable multi-dimensional data infrastructure for location aware services[A]. //In Mobile Data Management (MDM), 2011 12th IEEE Interna- tional Conference on[C], volume 1, pages 7–16. IEEE, 2011.

[11] Jianling Sun and Qiang Jin. Scalable rdf store based on hbase and mapreduce[A]. //In Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on[C], volume 1, pages V1–633. IEEE, 2010.

[12] Vaibhav Khadilkar, Murat Kantarcioglu, Bhavani Thu- raisingham, and Paolo Castagna. Jena-hbase: A distributed, scalable and efficient rdf triple store[A]. //In Proceedings of the 11th International Semantic web Conference Posters & Demonstrations Track[C], ISWC-PD, volume 12, pages 85–88. Citeseer, 2012.

[13] Ankur Khetrapal and Vinay Ganesh. HBase and hypertable for large scale distributed storage systems[J]. Dept. of Computer Science, Purdue University, 2006.

[14] Murali Krishna, Balaji Kannan, Anand Ramani, and Sriram J Sathish. Implementation and performance evaluation of a hybrid distributed system for storing and processing images from the web[A]. //In Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on[C].New York: IEEE, 2010:pages 762–767.

[15] Apache Hadoop. http://hadoop.apache.org/.

[16] Shvachko K, Kuang H, Radia S, et al. The hadoop distributed file system[A]. //Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on. IEEE[C], 2010: 1-10.

[17] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

[18] Hadoop ecosystem, http://www.mssqltips.com/sqlservertip/3262/big-data-basics--part-6--related- -projects-in-hadoop-ecosystem/

[19] Liu Y, Chen B, He W, et al. Massive imag apache e data management using HBase and MapReduce[A]. //Geoinformatics (GEOINFORMATICS), 2013 21st International Conference on. IEEE[C], New York: IEEE,2013: 1-5.

[20] Wu C, Quan J, Yuan Y, et al. Research on Quickly Search in Massive Remote Sensing Images Based on Hbase[A].//3rd International Conference on Computer Science and Service System[C]. Atlantis Press, 2014.

[21] Burke J A, Estrin D, Hansen M, et al. Participatory sensing[J]. Center for Embedded Network Sensing, 2006.

[22] Participatory Sensor Andrew. http://wise.ece.cmu.edu/redmine/projects/psandrew/wiki

[23] EXIF. http://zh.wikipedia.org/zh/EXIF

[24] JSON. http://www.json.org/json-zh.html

[25] Protobuf. http://en.wikipedia.org/wiki/Protocol_Buffers

[26] HDFS architecture. http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[27] Zookeeper. http://zookeeper.apache.org/

[28] RMI. http://en.wikipedia.org/wiki/RMI

[29] Apache Hadoop. http://hadoop.apache.org/docs/stable1/hadoop_archives.html.

[30] Apache Hadoop. https://hbase.apache.org/apidocs/org/apache/hadoop/hbase/io/hfile/HFile.htm

[31] 朱晓丽，赵志刚. 一种基于 HBase 的海量图片存储技术[J]. 信息系统工程, 2013 (8): 22-24.

致谢

时光荏苒，岁月如歌。在北邮的研究生生活将要结束，我要向在此时期向我提供过帮助和指导的所有人，表示最真挚的感谢。

首先，我要感谢我的导师王文东教授，非常感谢他对我的教育和学术上的指导，让我在研究生期间过得非常充实，受益匪浅。不仅如此，王老师还关心每位同学的生活情况。我还要感谢龚向阳教授和阙喜戎副教授。每当我对学术上有疑问的时候，龚老师和阙老师总是耐心热情地和我探讨，给了我很多宝贵的建议和意见。

我要感谢实验室的全体同学，沈跃辉、张咏悦、王璐、徐登佳、袁龙运等等，与你们一起奋斗过的日日夜夜，都是我研究生期间宝贵的经历和回忆。从大家的身上我学到了很多，也学会了如何团结一致，互相协作。

感谢陈建树同学的耐心引导，对我的鼓励和宽容。让我找到了生活目标并愿意陪伴着我为之奋斗下去。

此外，我还要感谢我的家人，特别是我的父母，谢谢你们对我无条件的支持，以及给予的爱与关怀，让我拥有战胜一切的力量。

最后，非常感谢诸位专家教授们在百忙之中评审我的学位论文，并给予指导。

作者攻读学位期间发表的学术论文目录

[1] 李莹. 基于 QualNet 的 DASH 业务的实现、仿真与分析[EB/OL].北京：中国科技论文在线 http://www.paper.edu.cn/releasepaper/content/201412-284,2014-12-10.