

分类号: TP311.5

密 级: 无

单位代码: 10335

学 号: 21651098

浙江大学

硕士学位论文



中文论文题目: 云音乐产品数据分析系统的设计与实现

英文论文题目: **Design and implementation of the cloud
music product data analysis system**

申请人姓名: 吕云鹏

指导教师: 贝毅君 副研究员

合作导师: _____

专业学位类别: 工程硕士

专业学位领域: 软件工程

所在学院: 软件学院

论文提交日期 2018 年 月 日

题目

云音乐产品数据分析系统的设计与实现

作者姓名

吕云鹏

浙江大学

云音乐产品数据分析系统的设计与实现



论文作者签名:_____

指导教师签名:_____

论文评阅人 1: _____

评阅人 2: _____

评阅人 3: _____

评阅人 4: _____

评阅人 5: _____

答辩委员会主席: _____

委员 1: _____

委员 2: _____

委员 3: _____

委员 4: _____

委员 5: _____

答辩日期: _____

Design and implementation of
the cloud music product data analysis system



Author's signature: _____

Supervisor's signature: _____

Thesis reviewer 1: _____

Thesis reviewer 2: _____

Thesis reviewer 3: _____

Thesis reviewer 4: _____

Thesis reviewer 5: _____

Chair: _____

Committeeman 1: _____

Committeeman 2: _____

Committeeman 3: _____

Committeeman 4: _____

Committeeman 5: _____

Date of oral defence: _____

浙江大学研究生学位论文独创性声明

本人声明所呈交的学位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。除了文中特别加以标注和致谢的地方外，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 浙江大学 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谢意。

学位论文作者签名：

签字日期：

年 月 日

学位论文版权使用授权书

本学位论文作者完全了解 浙江大学 有权保留并向国家有关部门或机构
送交本论文的复印件和磁盘，允许论文被查阅和借阅。本人授权 浙江大学 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和传播，可以采用影印、
缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后适用本授权书）

学位论文作者签名：

导师签名：

签字日期： 年 月 日

签字日期： 年 月 日

摘要

近年来，大数据迅速发展成为互联网行业的热点，在科技界乃至世界各国引起广泛关注。人类正从 IT 时代走向 DT 时代。在 DT 时代，数据的获取更加容易及快捷。IDC 的报告显示：预计到 2020 年，数据总量将超过 40ZB。而这些正在呈现爆炸式增长的数据，存在巨大的潜在价值。但如果不能对这些数据进行有组织的存储和分析从而产生价值，那么它同时也称为一场灾难。

如何建设高效的数据模型体系，如何避免重复建设和数据不一致问题，如何提供高效易用的数据开发工具，如何设计有效的数据产品进行分析... 这些都给大数据系统的建设提出了更复杂的要求。

本文通过研究业内相关的大数据技术栈，以云音乐具体数据业务为驱动，设计和实现了云音乐产品数据分析系统。本文主要对 Spark, Hive, Hadoop, Impala 及 Azkaban 等相关技术进行简单的介绍，对大数据领域进行系统性的介绍，从而对从事大数据研究的人员提供指导意义。

本文通过归纳整理引入了大数据系统的通用框架。该处理框架主要包含数据收集、数据存储、数据计算、数据分析等四个处理阶段。并对每一个阶段进行了调研。此外，本文还结合云音乐的业务发展，提出了大数据分析系统平台的设计架构，为相关问题提供解决方案。

关键词：云音乐，数据收集，数据仓库，Spark, Hive

Abstract

During the past few years, big data has rapidly developed into a focus for the science community, business community, and even governments around the world. And thus humanity is marching from the IT era to the DT era. In DT times, People can collect more abundant data than ever before. The IDC report shows that the total volume of global data is expected to exceed 40 ZB by 2020. The potentially huge value of the explosive growth data is still to be excavated. But if these data can't be organized and stored in a structured way, if they can't be effectively exploited and excavated, and then produce value, it is also called a disaster.

How to build the efficient data models and systems to make data easy to use and avoid duplication of data. How to provide the efficient and easy to use development tools. How to design effective data products for analysis. All these questions provide more complex requirements for the construction of large data systems.

This paper designs and implements a cloud music product data analysis system based on the related big data technology stack in the industry and driven by the specific data business of cloud music. This article is a brief introduction to related technologies, such as Spark, Hive, Hadoop, Impala and Azkaban. This article gives a systematic introduction to the large data field, which provides guidance for the understanding of the large data platform, the development of large data applications, and the study of large data. This paper introduces the general framework of big data system. The framework divides the large data platform into 4 processing stages: data generation, data acquisition, data storage and data analysis. In this paper, the current research progress at each stage is investigated, and the engineering level view of architecture design is presented. Different examples of big data analysis are discussed. In addition, this paper also compares the evaluation criteria of large data systems, and sums up the scientific problems and research directions of the large data.

Key Words : Cloud Music, Data Collection, Data Warehouse, Spark, Hive

目录

| | |
|---------------------------------|-----|
| 摘要 | i |
| Abstract..... | ii |
| 图目录 | III |
| 表目录 | IV |
| 第 1 章 绪论 | 1 |
| 1.1 课题来源及类型 | 1 |
| 1.2 课题意义及国内外研究现状 | 1 |
| 1.3 课题的研究目标、研究内容和拟解决的关键问题 | 2 |
| 1.4 本文的组织结构 | 4 |
| 1.5 本章小结 | 4 |
| 第 2 章 相关技术 | 5 |
| 2.1 数据同步 | 5 |
| 2.1.1 Flume | 5 |
| 2.1.2 Sqoop | 6 |
| 2.2 数据存储 | 7 |
| 2.2.1 HDFS | 7 |
| 2.2.2 HBase | 8 |
| 2.3 数据计算 | 9 |
| 2.3.1 MapReduce | 9 |
| 2.3.2 Spark | 9 |
| 2.4 数据分析 | 10 |
| 2.4.1 Hive | 10 |
| 2.4.2 Impala | 11 |
| 2.5 任务调度 | 11 |
| 2.5.1 Azkaban | 11 |
| 2.6 本章小结 | 11 |
| 第 3 章 需求分析 | 12 |
| 3.1 资源型数据查询 | 12 |
| 3.2 市场活动数据查询 | 14 |
| 3.3 本章小结 | 17 |
| 第 4 章 架构设计 | 19 |
| 4.1 平台系统架构设计 | 19 |
| 4.2 数据收集 | 20 |
| 4.3 维度建模 | 22 |
| 4.4 数据计算 | 24 |
| 4.5 任务调度 | 25 |
| 4.6 Web 服务 | 26 |

| | |
|---------------------|----|
| 4.7 本章小结 | 27 |
| 第 5 章 系统实现 | 28 |
| 5.1 数据收集 | 28 |
| 5.1.1 DDB 同步 | 28 |
| 5.1.2 日志解析 | 33 |
| 5.2 维度建模 | 36 |
| 5.2.1 数据操作层 | 36 |
| 5.2.2 公共维度层 | 36 |
| 5.2.3 数据应用层 | 39 |
| 5.3 数据计算 | 41 |
| 5.4 Web 服务模块 | 44 |
| 5.5 本章小结 | 45 |
| 第 6 章 系统应用 | 46 |
| 6.1 资源型模块 | 46 |
| 6.2 市场活动模块 | 47 |
| 6.3 本章小结 | 50 |
| 第 7 章 总结与展望 | 51 |
| 7.1 本文的主要研究贡献 | 51 |
| 7.2 进一步研究 | 51 |
| 参考文献 | 52 |
| 作者简介 | 54 |
| 致谢 | 55 |

图目录

| | |
|-------------------------------------|----|
| 图 2.1 Flume 架构示意图 | 5 |
| 图 2.2 Sqoop 架构示意图 | 6 |
| 图 2.3 HDFS 架构示意图 | 8 |
| 图 2.4 HBase 架构示意图 | 9 |
| 图 2.5 Hive 架构示意图 | 10 |
| 图 4.1 云音乐产品数据分析系统架构示意图 | 19 |
| 图 4.2 DDB 数据库同步技术设计示意图 | 21 |
| 图 4.3 日志收集架构示意图 | 22 |
| 图 4.4 数据仓库维度模型分层示意图 | 23 |
| 图 5.1 数据收集架构示意图 | 28 |
| 图 5.2 Sqoop 导入数据类图示意图 | 30 |
| 图 5.3 Sqoop 项目结构示意图 | 30 |
| 图 5.4 MRJobBase 方法示意图 | 31 |
| 图 5.5 用户日志解析过程示意图 | 34 |
| 图 5.6 资源型及市场活动计算任务依赖示意图 | 41 |
| 图 6.1 云音乐产品数据分析系统资源型数据查询示意图 | 46 |
| 图 6.2 云音乐产品数据分析系统市场活动展示示意图 | 47 |
| 图 6.3 云音乐产品数据分析系统市场活动添加示意图 | 48 |
| 图 6.4 云音乐产品数据分析系统市场活动优先级配置示意图 | 49 |
| 图 6.5 云音乐产品数据分析系统市场活动指标查询示意图 | 49 |
| 图 6.6 云音乐产品数据分析系统市场活动指标明细示意图 | 50 |

表目录

| | |
|-------------------------------------|----|
| 表 3.1 资源型数据查询指标组合表 | 14 |
| 表 3.2 自定义埋点规则示意表 | 16 |
| 表 5.1 Sqoop 同步命令示意表 | 29 |
| 表 5.2 Sqoop 参数说明示意表 | 29 |
| 表 5.3 SqoopWrappedJob 实现逻辑示意表 | 32 |
| 表 5.4 任务配置示意表 | 32 |
| 表 5.5 common 文件配置示意表 | 33 |
| 表 5.6 评论信息表代码 | 38 |
| 表 5.7 UserCommonFactJob 实现代码 | 40 |
| 表 5.8 市场活动属性表 | 42 |
| 表 5.9 市场活动跟踪方式属性表 | 43 |
| 表 5.10 市场活动 market 参数属性表 | 43 |
| 表 5.11 市场活动渠道包属性表 | 44 |
| 表 5.12 市场活动资源型属性表 | 45 |
| 表 5.13 市场活动 API 表 | 45 |

第1章 绪论

1.1 课题来源及类型

“人类正从 IT 时代走向 DT 时代”。在 DT 时代，每天大量的数据被产生，并被快速的获取。这些巨大的数据其价值是难以估量的并等待着去被挖掘发现。然而，如果不能有效的对数据进行存储分析和利用并从中获取价值，巨大的数据也将成为一场灾难^[1]。

因此，如何建设高效的数据模型和体系，如何存储数据来使得数据易用以及避免重复建设，如何提供高效易用的数据开发工具，如何设计有效的数据产品进行分析... 等等这些问题都给大数据系统的建设提出了更复杂的要求^[2]。

在网易云音乐，随着云音乐不断的发展，用户数在 2017 年 11 月突破四亿。数据量在不断增长及数据类型不断扩展。基于这种情况，本文介绍了当前业界开源的大数据技术，并提出了云音乐产品数据分析系统的设计方案，从而建设一个基于大数据体系架构的数据查询分析决策系统。

云音乐产品数据分析系统主要对外提供数据查询服务，包括各种资源型数据如歌曲，艺人，歌单等多维度信息的快速查询，也包含市场活动等复杂业务的查询。比如某一个市场活动所带来的新增、活跃、召回人数等。这些数据为部门的决策和战略提供依据。项目从数据流上可划分成两大部分，基于 Hadoop 和 Spark 的基础数据系统和基于 J2EE(Java2 Platform Enterprise Edition, Java2 平台企业版^[3])的数据平台系统。共涉及了数据收集，ETL 处理，维度建模，数据计算，数据服务五部分。

1.2 课题意义及国内外研究现状

随着互联网行业信息化的高速发展，数据正在快速生成，而需要处理的数据种类，数量也在急剧膨胀，大数据时代已经来临。大数据通常指数据规模巨大。这些大量的数据无法通过主流软件在合适的时间范围内处理完成。因此在面对海量数据时，尽管传统数据库具有诸多优点，比如支持完整性约束、支持事物等等，但是这些软件在大规模海量数据面前处理能力还是显得力不从心。

传统关系数据库主要存在以下主要问题，一个是数据格式转化无法满足海量数据处理对性能的要求，并且存储内容有限；二是传统数据库不容易实现动态扩

展；三是传统大型关系数据库通常运行在大型设备上，成本高昂。基于以上原因，研发人员不得不思考除了传统数据库外其他能够使用的存储方法。因此如何有效地对大数据进行存储分析已经成为一个亟待解决的问题^[4]。

为了解决上述问题，国内外出现了很多优秀的开源工具及项目。如日志收集组件 Flume^[5], Kafka^[6]等，数据库同步工具 Sqoop^[7]，分布式计算 Hadoop^[8]和 Spark^[9]，以及各种任务调度工具如 Azkaban^[10]，AirFlow^[11]等等。在国内，各大公司都会开发自己的工具组件。如网易杭研所的分布式存储数据库 DDB^[12]，日志收集 DataStream 等等。这些工具的出现一定程度上很好的解决了针对这些大规模数据进行存储、查询、分析的问题。

本文主要对大数据领域相关技术进行研究以及简要介绍，从而对从事大数据研究的人员提供指导意义。本文通过学习归纳整理大数据相关技术引入了大数据系统的通用框架。该框架主要包含数据收集，数据存储，数据计算，数据分析四个处理阶段。本文并对每一个阶段进行了调研。此外，本文还结合云音乐的业务发展，提出了大数据分析系统平台的设计架构。为相关问题提供解决方案。

本文旨在以云音乐业务需求为驱动，设计并实现一套大数据数据分析系统。结合具体的业务需求，系统主要提供资源型查询及相关市场活动业务查询分析等功能。

1.3 课题的研究目标、研究内容和拟解决的关键问题

课题研究目标：

本文以网易云音乐产品数据分析系统为例，对数据分析平台系统建设过程中的数据收集，ETL 处理，数据仓库模型建设，数据计算以及数据服务等过程进行了阐述。研究目标是结合数据存储及业务查询需求，探讨如何高效的进行数据同步，及结合云音乐业务如何进行数据仓库模型设计，所需查询指标的计算任务设计和 web 服务设计，最终实现云音乐数据分析系统。对外提供资源型各指标和市场活动收益指标的查询功能。

课题研究内容：

为了更好的改进产品，云音乐出现了要求快速获取市场活动及资源型相关指标的业务需求，包括如何快速的查询歌曲的歌名，艺人，专辑等基本指标。如何查询艺人的作品指数，粉丝指数，热度等信息。如何查询一个资源带来的新增活跃的 PV（次数）和 UV（人数）。以及如何衡量一个市场活动带来的收益。这些

指标为未来的决策起到了至关重要的支撑作用。为了解决上述问题。结合实际业务，本文主要对大数据存储及计算以及对外提供服务等相关技术进行了研究。提出了一个从日志收集，数据库同步存储到计算查询的架构设计。接着本文具体介绍了日志收集，数据库同步，数据清洗，数据建模，数据指标计算及任务调度，接口设计及服务等功能的设计。

本文主要研究内容：

数据收集：主要业务包括用户日志收集和线上数据库的数据同步两部分。云音乐的线上数据库为 DDB，主要介绍 DDB 同步实现。

数据清洗：指对日志文件进行解析，并清洗进行 ETL 处理，生成数据仓库底层表。主要介绍 User action 日志和 Nginx 日志的解析及底层表构建过程。

数仓建设：采用维度模型抽取中间层，来屏蔽底层日志打点变化。

数据计算：采用 Spark SQL 等方式计算业务数据。并将结果存储到 Impala^[14]，DDB(分布式数据库)供查询。

数据服务：构建 web 服务，通过 restful API 提供具体查询功能。包括资源型和市场活动两大模块。

拟解决的关键问题：

本文旨在提出一个通用的大数据平台系统架构设计。主要从数据收集，数据存储，数据计算及数据服务四部分来介绍云音乐数据分析系统的设计与实现。其中每部分都结合具体的业务场景进行改进和实现。拟解决的关键问题如下：

数据收集方面，主要包含日志收集和线上数据库同步两部分，针对云音乐现有使用的线上数据库，原有开源的 Sqoop 同步方案会造成数据倾斜，因此对 Sqoop 进行二次包装，结合 Azkaban 实现每天的 DDB 同步任务。

数据仓库模型方面，数据主要存储在 Hive 及 HDFS^[13]中。针对数据来源复杂，以及适应各种类型的查询服务。对原有数据仓库的架构进行重新抽取设计，采用维度建模的思想解决数据复用及隔离底层日志打点变更带来的修改问题。

大数据查询方面，真对业务需求，需要快速的查询每个资源的信息，入每首歌某段时间内的转发，评论，有效播放量等指标。或者某个市场活动带来的新增活跃等指标。因此需要设计具体的计算逻辑。并针对快速查询的要求，系统采用 Impala, Kylin^[15]与 Hive 相结合的方式，来提高查询效率。或将具体执行完的结果存储到 DDB 以供查询。

1.4 本文的组织结构

论文主要分为以下七个部分，具体说明如下：

第一章 绪论。本章介绍了云音乐产品数据分析系统的提出背景以及目前国内外相关问题的研究现状。

第二章 相关技术。本章主要介绍了云音乐产品数据数据分析系统所用到的大数据相关技术，包括 Azkaban, Hive, Spark, Sqoop 等等。

第三章 需求分析。本章主要介绍了云音乐相关业务需求。包括资源型数据查询和市场活动数据查询两部分。

第四章 架构设计。本章主要讲述了系统的整体架构设计，涉及数据收集，数仓建设，数据计算，数据服务等模块。

第五章 系统实现。本章主要介绍了云音乐产品数据分析系统的架构实现以及各个组件模块的具体实现。

第六章 系统应用。本章主要介绍了系统的应用及效果展示。

第七章 总结与展望。本章主要对本文进行总结，并提出需要改进的地方及未来规划。

1.5 本章小结

本章主要介绍了课题的来源背景，提出了云音乐产品数据分析系统的研究内容及拟解决的问题。以及全文的组织结构。

第2章 相关技术

2.1 数据同步

2.1.1 Flume

Flume 是一个日志收集工具。它具有诸多特点，如高可用性，高可靠性，支持分布式等。Flume 支持多种数据源，其定制了多样数据发送方。同时，Flume 支持数据处理功能。并支持写到各种数据接收方。数据接收方可定制。目前已经广泛应用于业内。

以一个电子商务网站为例，系统需要获取用户的消费行为数据，来进行分析从而获取用户的消费习惯并进行相关商品的推荐。为了实现该功能，系统需要支持收集用户日志信息，并将海量的日志数据交给 Hadoop, Spark 等大数据平台上进行分析。该系统可以通过 Flume 来实现。业内也有很多其他框架实现。比如 Facebook 的 Scribe^[17]以及淘宝 Time Tunnel^[18]。

其架构图如图 2.1 所示。

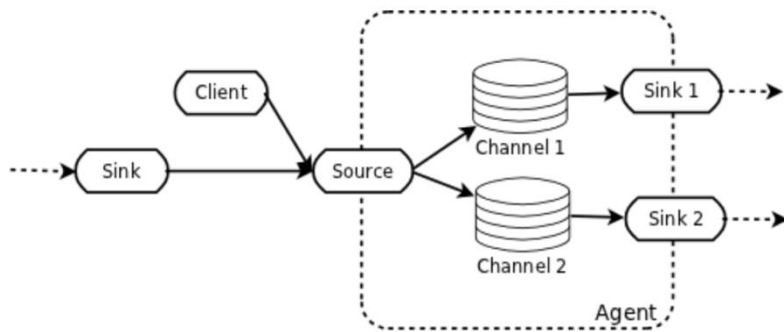


图 2.1 Flume 架构示意图

Flume 具有众多优势。Flume 可以将数据写入到任何数据接收方，比如 HDFS, HBase^[16]等等。在写入过程中，当收集数据的速度超过写入速度，Flume 会进行调整，保证速度的合理一致。保证其能够在两者之间提供一个平稳的数据通道。Flume 具有高可用性，可升级，易管理等特点。

因此 Flume 广泛应用于日志收集系统中。

2.1.2 Sqoop

Sqoop 是一种数据传输工具，其主要在 Hadoop 和关系数据库服务器之间进行数据交换等。它是由 Apache 软件基金会提供。Sqoop 主要从 MySQL, Oracle 等传统关系型数据库获取数据，并将数据复制拷贝到如 HDFS 文件系统等大数据相关存储系统中。并且也支持从 HDFS 拷贝数据到关系数据库^[20]。

产生大数据的一种重要来源便是传统的应用管理系统，该系统通常与关系型数据库进行交互。大量的数据通常是由关系型数据库产生，研发人员又需要通过大数据计算引擎和分析器如 MapReduce、Hive、HBase、Cassandra、Pig^[19]等，Hadoop 的生态系统等应运而生。而在这些组件之间，需要一个工具来进行数据的导入和导出。在这里，Sqoop 便担当着这样的角色，提供关系数据库服务器和 Hadoop HDFS 之间的可行的互动。

Sqoop 到 HDFS 的原理如图 2.2 所示。

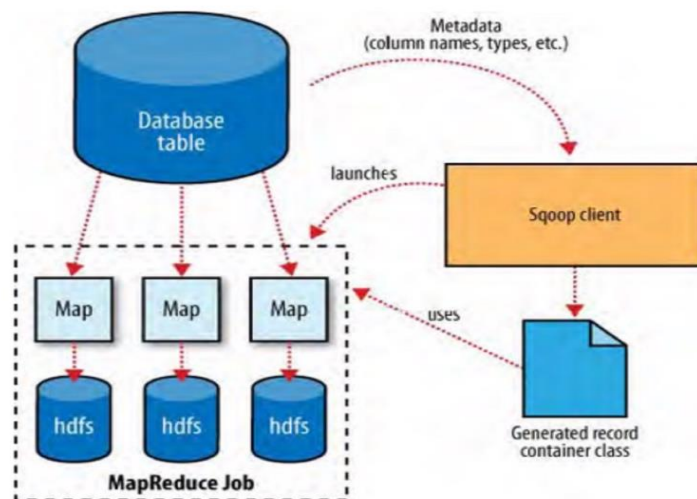


图 2.2 Sqoop 架构示意图

根据 Sqoop 的架构图，可以发现，Sqoop 是通过 MapReduce 任务从数据库中导入一个表的。该作业可以获取表中行数据，并将其写入到 HDFS。Sqoop 在 import 时，需要制定 split-by 参数。

Sqoop 可以通过配置 split-by 参数值从而进行数据的分割，每一个 split 分区会分配到不同的 map 中。在 map 中，会针对每一行进行处理。并将数据写入到 HDFS 中。Split 支持多种方式的划分，通常不同字段类别具有不同的划分器。针对 int 类型，Sqoop 会获取指定的 split-by 参数值字段中最大和最小的值，然后根

据传入的 `num-mappers` 来计算确定划分几个区域。

最后每个 `map` 会将各自的数据导入到 HDFS 中，整个导入流程结束。

一个典型的 `Sqoop` 命令如下所示。

```
$ sqoop import \  
--connect jdbc:mysql://localhost/userdb \  
--username root \  
--table emp_add \  
--m 1 \  
--where city =sec-bad \  
--target-dir /wherequery
```

其中 `import` 代表导入，`Connect` 为连接的数据库，`Where` 后面是过滤条件。

2.2 数据存储

2.2.1 HDFS

HDFS(Hadoop Distributed File System)是 Hadoop 项目的核心子项目，HDFS 是一个分布式文件系统，类似于 Google 的 GFS^[21]，其架构是典型的 Master / Slave 形式，Master 节点上启动一个进程 `NameNode`。并在 Slave 节点上启动 `DataNode` 进程。两者之间通过心跳机制进行通信。`NameNode` 进程运行的节点上会存放文件的元数据，`DataNode` 运行节点上存放具体的文件。其架构设计如图 2.3 所示。归纳起来 HDFS 具有如下特点：

1. 可以通过增加机器来横向扩展系统的存储能力。
2. 系统具有高可靠性。
3. 性价比高。采用中低端机器作为 Slave 节点。
4. 该设计适合分布式数据计算。
5. 适合存储非结构化数据。

HDFS Architecture

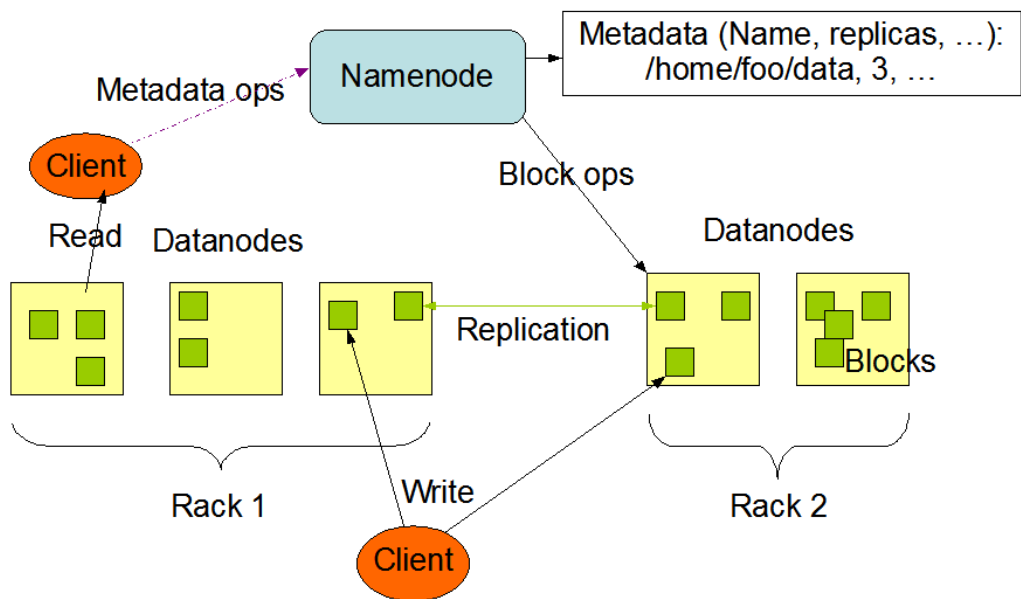


图 2.3 HDFS 架构示意图

2.2.2 HBase

HBase 是一个分布式面向列存储的数据库。HBase 是开源的, 支持横向扩展。总的来说, HBase 是一个类似于 Google 的大表设计的数据模型。它提供了快速随机访问海量结构化数据^[22]。并且 HBase 充分利用了 Hadoop 的文件系统 (HDFS) 提供的容错能力。

HBase 采用 Master/Slave 架构。作为 Hadoop 生态系统的一部分, 其主要提供对数据的随机读写功能。如图 2.4 HBase 架构图所示。在 HBase 中, 其主要包含 Client 客户端、HMaster 节点、HRegionServer 节点、ZooKeeper 集群等组件。数据实际存储在 HDFS 中。

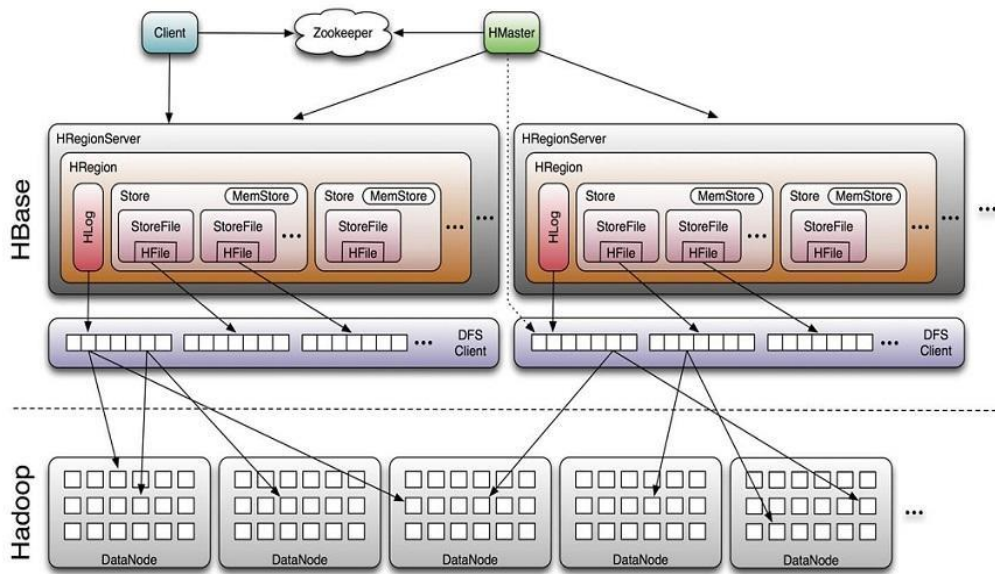


图 2.4 HBase 架构示意图

2.3 数据计算

2.3.1 MapReduce

MapReduce 是 Hadoop 中的并行计算框架，是一种编程模型。MapReduce 实际上是一个分治算法。将大问题化解为小问题，逐步解决。它的实现是 map 函数和 Reduce 函数组成^[23]。MapReduce 实际的处理过程可以理解为 Input->Map->Sort->Combine->Partition->Reduce->Output。Map 阶段由一定数量的 Map Task 组成。输入数据格式为 InputFormat，输出为 Key-Value 数据结构。Reduce 阶段由一定数量的 Reduce Task 组成。输入为 Map 的输出。

2.3.2 Spark

Spark 是加州大学伯克利分校 AMP 实验室 (Algorithms, Machines, and People Lab) 开发通用内存并行计算框架，并成为 Apache 的孵化项目。Spark 以其先进的设计理念，迅速成为社区的热门项目。社区围绕着 Spark 推出了 Spark SQL、Spark Streaming、MLLib 和 GraphX 等组件，这些组件逐渐形成大数据处理一站式解决平台^[24]。Spark 支持多种运行模式。并支持通过 yarn 进行管理调度。Spark 是由 Scala 语言编程实现。Scala 具有面向对象，函数式编程等特点。并且其运行速度快，通用型强。

2.4 数据分析

2.4.1 Hive

Hive 是一个数据分析工具。支持用户通过 SQL 对大数据进行统计和分析。它通过 SQL 来处理 Hadoop 程序，支持 100PB+量级的数据分析。Hive 的数据格式包括结构化和非结构化两种^[25]。Hive 具有如下众多特点：

1. 侧重于分析，而非实时在线交易。
2. 无事务机制。不像关系数据库那样可以随机进行 insert 或 update。Hive 通过 Hadoop 的 map/reduce 进行分布式处理。
3. 高扩展，Hive 支持数百台级别的扩展。

Hive 的架构设计如图 2.5 所示。主要包含了 UI、Driver、Compiler、Metastore、Execution Engine 等组件。UI 指用户提交请求到系统的接口。Driver 指相应查询请求。提供数据获取 API。Compiler 主要是解析查询，对 SQL 进行解析生出语法树，并最终翻译成操作树。Metastore 表示元数据信息，主要存储 Hive 中的元数据信息。Execution Engine 执行引擎，主要执行生成的具体执行计划。

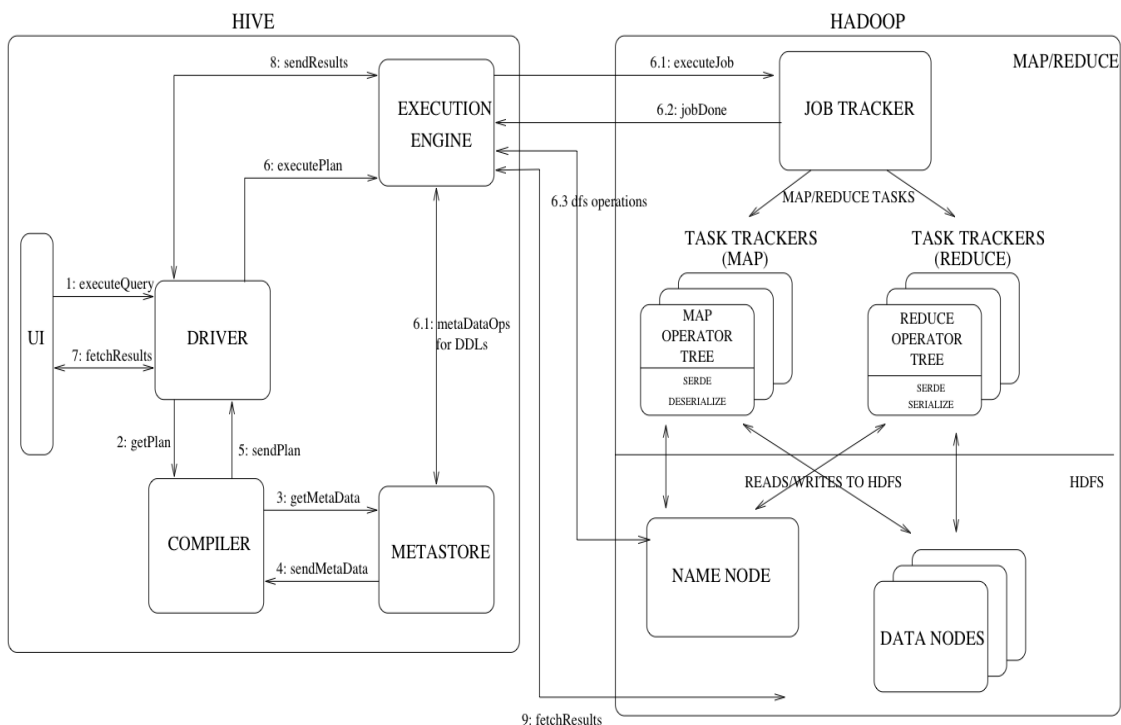


图 2.5 Hive 架构示意图

2.4.2 Impala

Impala 是 Cloudera 提供的一个支持实时交互 SQL 的大数据查询工具。它能够比 Hive 更加迅速的进行数据查询。其底层不再使用 Hive 和 MapReduce 批处理, 而是使用分布式查询引擎。其主要包含 Query Planner、Query Coordinator 和 Query Exec Engine 三部分。Impala 可以直接在 HDFS 和 HBase 中查询数据, 这样的设计大大的降低了延迟^[26]。

Impala 的设计实现主要由 Impalad、State Store 和 CLI 组成。其中 Impalad 和 DataNode 通常在一个节点上运行。它将收到的客户请求, 并翻译成查询树, 进行读写数据, 并执行查询计划。

Impala 相对于 Hive 所使用的优化技术如下所示。

1. 拒绝使用 MR 进行计算, 从而提高实时性, 提供更好的交互体验。
2. 将查询分解为执行计划树, 并非是流式的 MR 程序。

2.5 任务调度

2.5.1 Azkaban

Azkaban 是 Linkedin 推出设计的一个任务调度器。支持批量任务的定时调度, 并且支持任务之间的依赖以及任务流的划分^[27]。通常, Azkaban 任务之间的依赖通过 job 配置文件来实现。并且 Azkaban 提供一个 Web 界面进行任务的提交创建和管理跟踪。

一个 Azkaban 项目, 通常可以包含一个或多个工作流 flow。一个工作流支持多个任务 job。不同 job 之间可以构建依赖关系。并且支持多种 job 类型。一个 job 可以是一个 Linux 命令, 也可以是一 Java 程序或者 Python 脚本等等。并且可以自定义 Job 类型。创建一个 job 需要创建一个以 .job 结尾的文本文件。创建 flow 需要指定 dependencies 参数。从而构建一个图结构。

2.6 本章小结

本章主要介绍了云音乐产品数据分析系统所涉及使用到的关键技术及其原理。为平台的实现打下理论基础。

第3章 需求分析

云音乐产品数据分析系统主要对外提供数据查询服务,包括各种资源型数据如歌曲、艺人、歌单等多维度信息的快速查询。也包含市场活动等复杂业务的查询,比如某一个市场活动所带来的新增、活跃、召回人数等。这些数据为部门的决策和战略提供依据。

项目从数据流上可划分成两大部分,基于 Hadoop 和 Spark 的基础数据系统和基于 J2EE(Java2 Platform Enterprise Edition, Java2 平台企业版)的数据平台系统。共涉及了数据收集,维度建模,数据计算,数据服务四部分。

本章主要介绍了云音乐产品数据分析系统的业务需求,包含资源性数据查询,用户市场活动分析以及热点数据查询等主要业务。

3.1 资源型数据查询

系统应支持查询单项资源类型的在某个用户类别下相关指标类别下的相关指标。

所包含资源类别如下:单曲、MV、DJ、电台、视频、专辑、歌单、歌手、专栏、话题、认证用户、活动页、动态。

用户类别包含新增、日活、召回。

指标类别包含如下:播放、第一首播放、搜索、分享、分享引入、评论、订阅收藏、浏览、关注、参与话题、引入会员人数、引入会员金额。

指标包含 PV(次数)和 UV(人数)。

具体组合业务,以单曲, MV 和话题为例进行说明,详情如下所示。

单曲相关需求:

1. 查询某一首单曲某日,当日新增用户播放次数/人数;当日活跃用户播放次数/人数;当日召回用户播放次数/人数。
2. 查询某一首单曲某日,当日新增用户搜索次数/人数;当日活跃人数搜索次数/人数;当日召回人数搜索次数/人数。
3. 查询某一首单曲某日,当日新增用户分享次数/人数;当日活跃用户分享次数/人数;当日召回用户分享次数/人数。
4. 查询某一首单曲某日,当日新增用户分享引入次数/人数;当日活跃用户

分享引入次数/人数; 当日召回用户分享引入次数/人数。

5. 查询某一首单曲某日, 当日新增用户评论次数/人数; 当日活跃用户评论次数/人数; 当日召回用户评论次数/人数。
6. 查询某一首单曲某日, 当日新增用户收藏次数/人数; 当日活跃用户收藏次数/人数; 当日召回用户收藏次数/人数。

MV 相关需求:

1. 查询某一个 **MV** 某日, 当日新增用户播放次数/人数; 当日活跃用户播放次数/人数; 当日召回用户播放次数/人数。
2. 查询某一个 **MV** 某日, 当日新增用户首次播放次数/人数; 当日活跃用户首次播放次数/人数; 当日召回用户首次播放次数/人数。
3. 查询某一个 **MV** 某日, 当日新增用户搜索次数/人数; 当日活跃用户搜索次数/人数; 当日召回用户搜索次数/人数。
4. 查询某一个 **MV** 某日, 当日新增用户分享次数/人数; 当日活跃用户分享次数/人数; 当日召回用户分享次数/人数。
5. 查询某一个 **MV** 某日, 当日新增用户分享引入会员次数/人数; 当日活跃用户分享引入会员次数/人数; 当日召回用户分享引入会员次数/人数。
6. 查询某一个 **MV** 某日, 当日新增用户评论次数/人数; 当日活跃用户评论次数/人数; 当日召回用户评论次数/人数。
7. 查询某一个 **MV** 某日, 当日新增用户浏览次数/人数; 当日活跃用户浏览次数/人数; 当日召回用户浏览次数/人数。

话题相关需求:

1. 查询某一个话题某日, 当日新增用户分享次数/人数; 当日活跃用户分享次数/人数; 当日召回用户分享次数/人数。
2. 查询某一个话题某日, 当日新增用户评论次数/人数; 当日活跃用户评论次数/人数; 当日召回用户评论次数/人数。
3. 查询某一个话题某日, 当日新增用户收藏次数/人数; 当日活跃用户收藏次数/人数; 当日召回用户收藏次数/人数。
4. 查询某一个话题某日, 当日新增用户浏览次数/人数; 当日活跃用户浏览次数/人数; 当日召回用户浏览次数/人数。
5. 查询某一个话题某日, 当日新增用户参与次数/人数; 当日活跃用户参与次数/人数; 当日召回用户参与次数/人数。

整个资源型组合查询指标如表 3.1 所示。在表 3.1 中，其中‘1’表示所需查询指标，未填写表示不需要计算该指标。

表 3.1 资源型数据查询指标组合表

| | 单曲 | MV | DJ | 电台 | 视频 | 专辑 | 歌单 | 歌手 | 专栏 | 话题 | 活动页 | 动态 |
|----|----|----|----|----|----|----|----|----|----|----|-----|----|
| 播放 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | | | |
| 首播 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | | | |
| 搜索 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | | | |
| 分享 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 分引 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | | | | |
| 评论 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | 1 | 1 | 1 | 1 |
| 收藏 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | |
| 浏览 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 关注 | | | | | | | | | 1 | | | |
| 参与 | | | | | | | | | | 1 | | |
| 会员 | 1 | | | | | 1 | | | | | | |

3.2 市场活动数据查询

为了推广云音乐，提高增长。市场部会推出一系列市场活动，包括校园歌手大赛，地铁网易云音乐乐评，农夫山泉乐评等。那么如何衡量一个市场活动带来的收益？经过试验及验证，通过计算一个市场活动带来的新增，活跃，召回，次日留存，7日留存，30日留存等指标来指标数字化一个活动的收益。一个市场活动可以是各种形式及内容的。将其内容进行分类，共计五种跟踪计算方式，分别如下所示：

1. Market 点击 IP+OS+OSVER 匹配
2. 渠道包匹配
3. IDFA/IMEI 匹配
4. 埋点计算
5. 资源型计算

方式一：Market 点击 IP+OS+OSVER 匹配

Market 参数：市场活动可以有一个市场活动主页，例如某个活动的 URL 为 <https://music.163.com/m/banner/special/aixiyibuliang?market=aixiyibuliang>，那么该活动对应的 market 参数为 aixiyibuliang。

一个市场活动，可以绑定一个 Market 参数。当一个用户或者设备访问 URL 时，系统通过日志可以获取到该设备的 IP、OS、OSVER（版本号）等信息。同时当用户浏览及进行操作时，系统可以获取到该用户的设备信息及操作信息。可以通过 IP、OS、OSVER 进行关联匹配来计算市场活动关于 Market 参数类别下的新增、活跃、召回等指标。具体计算规则如下所示：

1. 当用户多次点击某个链接，以最后一次点击时间做归属。
2. 计算带来的新增用户，当用户点击完该活动的 URL 后，在 24h 内注册为云音乐用户，将其视为该市场活动带来的新增。
3. 计算带来的召回用户，当老用户点击完该活动的 URL 后，在 2min 内有登录行为，并且前 7 日未有登录行为，将其视为该市场活动带来的召回。
4. 计算带来的活跃用户，该市场活动带来的活跃用户包括带来的新增用户，以及在老用户点击完该活动的 URL 后，2min 内登录的用户，将其视为市场活动带来的活跃用户。
5. 计算带来的留存用户，今日带来的新增用户，次日有登录行为，将其作为该活动的次日留存。7 日后有登录行为，将其作为活动的 7 日留存。30 日后有登录行为，将其视为 30 日留存。

方式二：渠道包匹配

渠道包：渠道包指的是该 APP 在各大应用市场发布的 apk 包中配置的 value 值。如果该 APP 是通过百度推广发布的，value 可能是 baidu，360 可能是 360。主要是为了标记不同的渠道信息。

每个活动由各个渠道分发。需要计算该活动是和哪个渠道有关的。比如是通过云音乐官网 还是通过百度推广带来的新增。用户日志中会记录渠道包信息。具体的计算规则如下。

1. 计算带来的活跃用户：统计当日活跃用户中，渠道包信息为该活动绑定的渠道包。将其作为该市场活动带来的活跃。
2. 计算带来的新增用户：统计当日活跃用户中，为新增用户。并且新增信息中渠道包信息为该活动绑定的渠道包，将其作为该市场活动带来的新

增。

方式三：IDFA/IMEI 匹配

设备号：每一个设备都有唯一的设备号，比如安卓有 IMEI，Iphone 有 IDFA。设备号唯一标识了一台设备。

当市场活动与第三方合作，第三方提供在其平台所推广的用户设备号，可以通过第三方提供的设备号和用户日志中所记录的设备号进行匹配，从而来进行计算。具体匹配规则如下所示。

1. 计算带来的新增用户，所提供的设备号当日有注册行为，成为新用户。将其视为该活动带来的新增用户。
2. 计算带来的活跃用户，所提供的设备号当日有激活行为或者成为新增用户，将其视为该活动带来的活跃用户。

方式四：埋点计算

系统支持用户自定义匹配规则，然后根据自定义规则进行计算相关指标。即用户可以输入相应的埋点规则，根据规则去查询数据。再利用查询到的数据进行新增，活跃，召回等指标的计算。

埋点格式如表 3.2 所示。

表 3.2 自定义埋点规则示意图

| |
|---|
| <p>action=page json 内容: type=webvity, 代表是活动页面 url: 活动链接, eg: https://music.163.com/independ/kingcross2/index?id=16007 target: 活动名称, eg:crossking targetid: 活动 id, eg:16007 page: 活动的页面, eg:mainpage previous vip—活动主页 往期节目 购买会员页 is_webview: 0-代表 APP 内</p> |
|---|

方式五：资源型计算

资源：指云音乐中各种内容。包括：单曲，MV，DJ，电台，视频，专辑，歌单，歌手，专栏，话题，认证用户，活动页，动态等等。

市场活动，可以绑定关联某一个具体的资源。比如推广一个 MV，一个歌单

等等。最细可以关联到某一个资源的某种行为上。通过为活动设定相应关联的资源及行为，来计算该活动带来的新增，活跃，召回，留存等指标。具体的资源及行为组合详见资源型数据查询需求中所列内容。该部分也作为市场活动的一种计算规则。

除了上述物种计算方式，系统还应支持以下功能：

1. 市场活动支持多种跟踪计算方式并存。因为当一个市场活动指定多个计算方式时，需要对其进行 User ID 粒度上去重计算，从而避免数据偏高。
2. 同一时期，多个市场活动可以并行存在，同一个用户可能参与多个活动，需要对活动进行优先级设置，并按照优先级进行归因计算，防止整体新增指标数据偏高。

上面所描述的为指标计算规则，除此之外，该系统还应支持相关活动管理功能。具体包含以下内容。

1. 系统应提供活动新增模块，支持用户配置活动。支持用户定义活动名称，跟踪计算方式，支持多选。支持用户修改活动优先级。
2. 系统跟踪方式应包含 market 参数，渠道包，设备号，站内外打点，资源型物种类别。并支持类别多选。
3. 系统应提供市场活动的查询功能，包括模糊匹配。
4. 系统应提供市场活动的删除功能。
5. 系统应提供市场活动的下线功能。
6. 系统应提供市场活动的修改功能。如更正计算规则等。
7. 系统应提供去重计算，按照优先级进行归因计算。
8. 系统应提供相关指标查询，支持用户查询某一段日期范围内，特定活动或全部活动带来的新增，活跃，召回，留存等指标。
9. 系统应提供所查数据的离线下载功能，供分析人员使用。
10. 系统应支持某一个活动的数据下载以及全部活动全部时间的数据下载。

3.3 本章小结

本章节主要分析了云音乐产品数据分析系统的业务需求。包括资源型数据查

询以及市场活动数据查询两大部分。其中市场活动数据查询又包括市场活动管理需求和市场活动查询需求。为后续系统设计与实现提供需求指导。

第4章 架构设计

4.1 平台系统架构设计

为了满足云音乐的业务需求,系统需要获取用户数据进行分析。并最终对外提供服务。项目从数据流上可划分成两大部分,基于 Hadoop 和 Spark 的基础数据系统和基于 J2EE(Java2Platform Enterprise Edition, Java2 平台企业版)的数据平台系统。共涉及了数据收集,维度建模,数据计算,数据服务四部分。

云音乐产品数据分析系统的整体架构设计如图 4.1 所示。

需要通过 DS 来收集用户日志,将其存在 HDFS 中,并将线上数据库相关数据也同步到 HDFS 上。再然后对日志进行解析,建立数据仓库。以维度模型来设计中间表。并将中间表作为数据输入,依据业务计算规则通过 Spark 进行计算。最终将计算结果存储到 Impala 以及 DDB 等数据库中。整个过程都是通过 Azkaban 进行每日定时调度。最终通过 web 服务访问结果数据库,对用户提供服务。查询服务包括资源型相关数据查询,也包括市场活动相关数据查询。

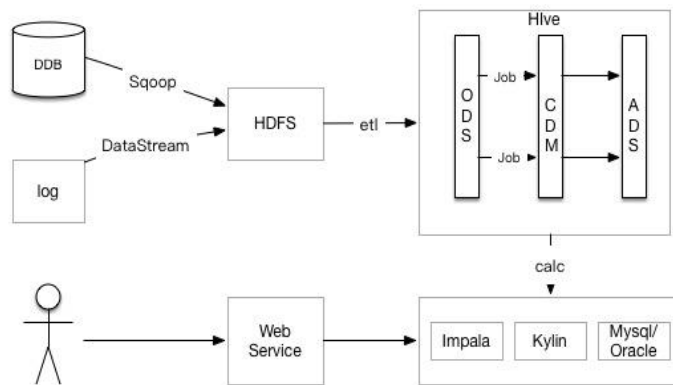


图 4.1 云音乐产品数据分析系统架构示意图

云音乐整体架构设计主要分为数据收集、数仓建设、任务计算、任务调度、Web 服务共五部分。其中数据计算任务为为天定是通过 Azkaban 进行调度。具体

设计如下所示。

1. 数据收集
 - a) DDB 数据库同步
 - b) 日志收集
 - c) 日志解析
 2. 维度建模
 - a) ODS 中 Hive 底层表构建
 - b) CDM 中数据明细层建设
 - c) CDM 中数据汇总层建设
 - d) ADS 中应用层建设
 3. 任务计算
 - a) 数据源主要为 hive，按照业务需求，进行计算任务，输出结果存储到 Impala, Kylin, MySQL 等
 - b) 业务计算包括资源型计算和市场活动计算
 4. 任务调度
 - a) 通过 Azkaban 进行相关任务调度
 5. Web 服务
 - a) 提供查询分析服务，主要采用 Spring MVC 进行实现
 - b) 前端实现
- 将在后续章节进行详细的介绍

4.2 数据收集

DDB 数据库同步：

主要目的是将线上数据库的数据，每天定时 dump 到 HDFS 上，并在 Hive 中创建相应的业务表，主要存放到 music_db_front， music_db_backend 两个库下面。经过调研，系统拟采用的是 Sqoop 进行实现。

Sqoop 主要在关系型数据库和大数据分析工具及存储之间担当一个数据导入导出的工具角色。设计如图 4.2 所示。

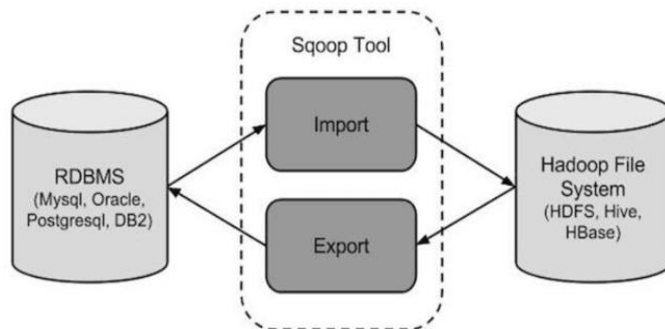


图 4.2 DDB 数据库同步技术设计示意图

利用 Sqoop 的特性，遍历 DDB 中的业务表，然后全量更新到 HDFS 中。

日志收集：

主要对用户日志和 Nginx 日志进行收集，采用杭研自研实现的 DataStream 进行收集，也可以采用业内开源的 Flume 进行收集。其中用户产生的日志主要分为两种。主要架构设计如图 4.3 所示。云音乐日志从产生到被收集处理主要包含了以下几个组成部分。

1. 用户客户端：用户操作 PC / Web / Android / Mac / iPhone 等各种客户端产生的日志，产生的日志主要分两种。
2. 客户端日志：此类日志为用户行为产生的日志，会暂时缓存在客户端，然后批量上传到音乐服务器，由于网络，流量以及音乐有离线播放等种种原因，此类日志上传会有一定的延迟。
3. 服务端日志：此类日志是用户直接调用后台接口产生的日志，日志由服务端直接产生收集，无延迟。
4. 音乐服务器：产生服务端日志以及给客户端提供日志上传接口收集客户端产生的客户端日志。这类日志会被存储在服务端本地，然后由 DS Agent 负责上传到 DS 服务器。
5. DS Agent(TailFileAgent)程序：安装在每台音乐服务器上，收集音乐服务器存储在本地的日志，然后实时上传到 DS 服务器。
6. DS 服务器端：接收 Agent 上传上来的的日志，分发到 HDFS、HBase、Kafka 等各个端。
7. DA 统计程序：从 HDFS 读取 DS 上传的文件，统计日活等各项指标；或从 Kafka 读取实时数据，进行实时统计。

其中 DS 会根据数据到达 DataRoute 的时间来决定 HDFS 归档的目录，每天一

个目录，拿用户日志 `useraction` 来说，DSAgent 在 2016-12-04 上传的日志会被归档在目录 `/datastream/useraction/2016-12-04` 里。这样子通过文件目录结构的形式，系统将每天的日志都进行落盘存储。

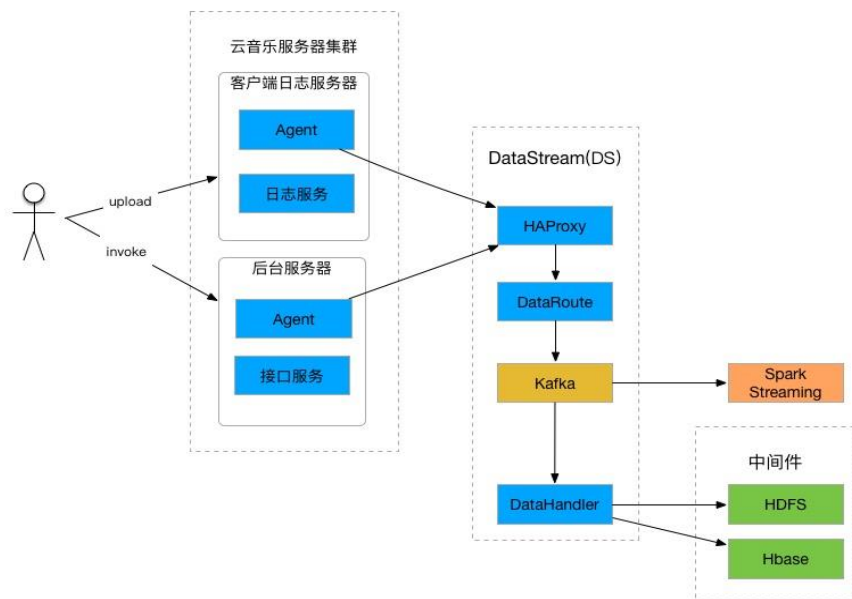


图 4.3 日志收集架构示意图

4.3 维度建模

数据仓库是一个面向主题的数据集合，具有面向主题的，集成的，随时间变化的等特点。通常用于对管理决策过程的支持。数据仓库具有多种用途。例如可以通过数据仓库整合现有业务建立统一中心。支持数据报表产出，进行决策。支持为业务提供数据，形成数据互相反馈闭环。可以整合公司现有的所有业务数据，建立统一的数据中心。数据仓库包含众多内容，如架构、建模和方法论等。具体如下所示：

1. 以 Hadoop、Hive、Spark 等组件为中心的数据架构体系。
2. 数据建模思想。
3. 调度系统，元数据管理等辅助系统。

在此讨论如何对数据仓库进行建模，来满足云音乐的业务需求。通常数据仓库模型包括四种模型，分别是 ER 模型、维度模型、DataVault、Anchor 模型。

ER 模型：用实体和实体关系来描述业务架构。是数据仓库之父 Immon 所提的一个 3NF 模型。其在范式理论上遵循 3NF 原则。但是在数据仓库中的 3NF 是面向主题的抽象，而不是针对具体业务流程进行的实体关系抽象。它支持数据的整合和整体性。正如 Immon 所希望达到的：“single version of the truth”。然而这种模型并不适合云音乐这种飞速扩展的业务。

维度模型：维度建模是以分析决策的需求作为出发点来构建模型的。它能够解决大规模数据的复杂查询以及快速响应。维度模型包含多种设计。比如常用的星型模型。该模型很好的满足了云音乐产品数据分析系统的要求。

DataVault 模型：该模型是在 ER 模型的基础上诞生的。是 Dan Linstedt 设计的一种模型方法论，设计的出发点是为了实现数据整合，该模型并非为数据分析进行设计的。因此不适合云音乐产品数据分析系统的设计需求。

综上，该系统拟采用维度模型来建模。其中维度模型主要包含两个重要的概念。即事实表和维度表。因此在维度建模的思想下，结合云音乐的自身业务。数据仓库三层模型架构设计如图 4.4 所示。



图 4.4 数据仓库维度模型分层示意图

该结构为了解决数据的复用性，及屏蔽后台日志打点变化造成上层业务逻辑变化等一些列问题，抽取中间层即 CDM 层。该层主要采用维度建模的思想，采用星型模型进行设计。主要包含两层，一层是明细层，以维度表和无事实的事实表为主，主要是为了屏蔽底层变化。另一层是汇总表，主要是进行数据复用，避免不必要的重复计算问题。

数据操作层主要涉及底层数据。包括线上数据库，用户日志等一切原始数据或者经过简单处理的原始数据。其主要数据设计如图 4.5 所示。该模块主要是将收集到的用户日志和 Nginx 日志进行解析，生成数据仓库 Hive 中的底层表。

数据维度层主要包括数据明细层和汇总层。其中明细层主要以资源型，及行为事实为建表依据。汇总层以 **user** 和资源型之间的关系为主。表之间的计算任务通过 **Spark SQL** 任务进行实现。明细层设计如图 4.5 所示。

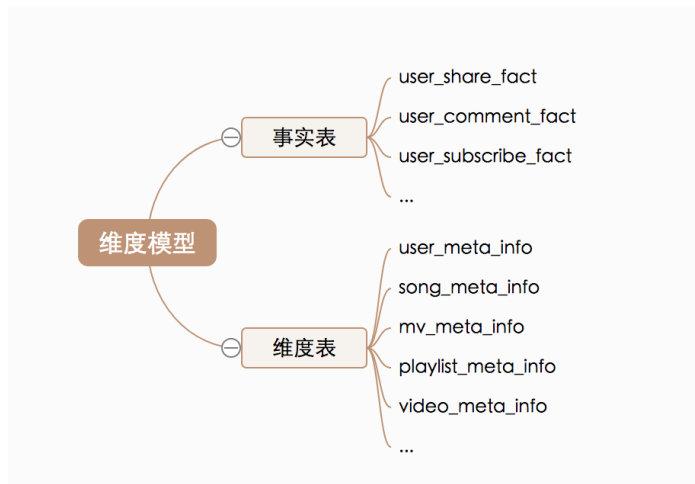


图 4.5 数据维度层明细层设计示意图

汇总层设计如图 4.6 所示。



图 4.6 数据维度层汇总层示意图

数据应用层，包含对外系统所设计的数据库。针对第三章提到的业务需求。主要包括市场活动信息表，跟踪表，指标表等。

4.4 数据计算

在云音乐产品数据分析系统中，系统主要采用 **Spark** 来作为主要的计算方式。其中不同的业务需求，有不同的计算逻辑。以市场活动跟踪模块为例，主

要计算不同活动带来的新增、活跃、召回等指标。对于每一个活动，支持多种可配置的跟踪方式。计算依赖图如图 4.7 所示。

4.5 任务调度

本系统所设计的计算任务，都是通过 Azkaban 进行调度。Azkaban 的介绍详见第二章任务调度的介绍。Azkaban 通过将项目打成 zip 压缩包上传到平台上。然后解析构建调度依赖图。并按照具体的时间规则进行周期调度。在这里，具体的 azkaban 环境的配置与搭建不做介绍。需要关注的是，如何将 Spark 任务包装成 Azkaban 任务进行调度。为此定义了一个 AzkabanSparkJob 基类。所有 Spark Azkaban 任务都继承该类，并实现其 run 方法。

AzkabanSparkJob 主要解决了问题如下所示。

1. Spark 程序 Jar 包依赖，从 hdfs 加载文件。
2. 任务执行环境监测，包括输入目录，输出目录。
3. 执行 execute 核心方法。
4. 临时文件及目录清除。

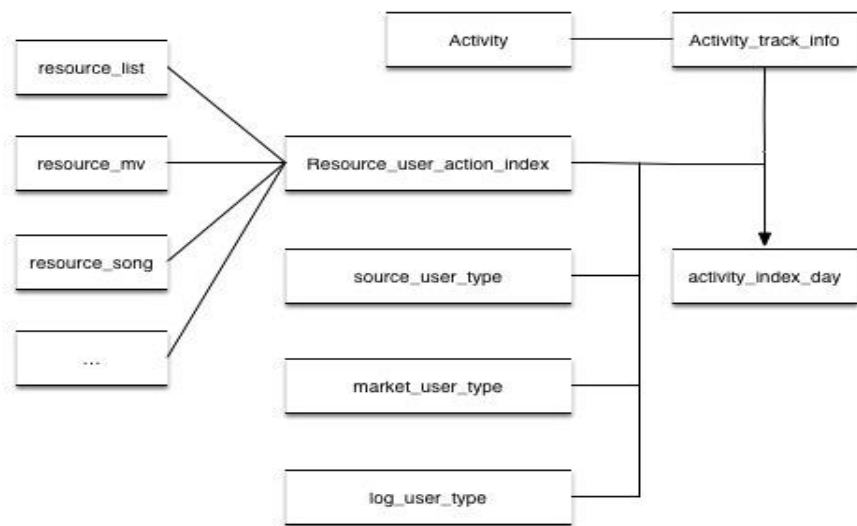


图 4.7 市场活动数据计算逻辑设计示意图

再然后，定义了一个 SparkJobDriver，作为程序的入口。主要通过反射的机

制, 构建一个 Spark 任务对象。并执行其 execute 方法。AzkabanSparkJob 的 execute 方法会检查执行时间, 加载依赖配置并执行其 run 方法。

每一个具体的 Spark 任务, 都需要继承 AzkabanSparkJob 基类, 并实现其 run 方法。在 run 方法中实现该任务的具体业务逻辑。通过这样子的架构设计, 避免了代码冗余, 使得项目易于管理及维护。

其实现逻辑如下所示。

1. 声明 sparkConf 对象。
2. 如果 sparkConf 中设置为 spark 任务, 打印任务开始时间。
3. 设置 sparkConf 属性信息。
4. 声明 sparkContext 对象。
5. 通过参数获取任务类型。
6. 实例化任务对象。
7. 调用该 spark 任务的 execute 方法。执行具体的逻辑。
8. 如果 sparkConf 中未设置为 spark 任务, 打印 Spark 任务未开启。

4.6 Web 服务

主要对外提供数据服务功能。整个系统采用 Spring、MVC 架构实现。架构设计如图 4.8 所示。

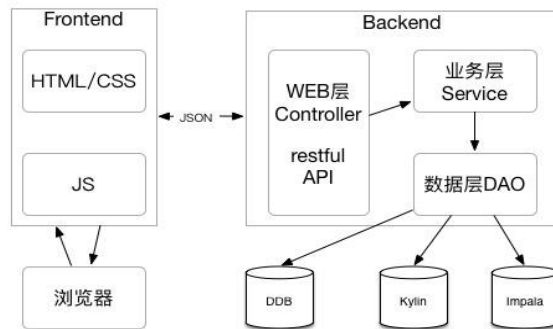


图 4.8 Web 服务架构设计示意图

整个系统分为前后端和数据存储三部分。前后端交互数据格式为 JSON。后台对外提供 rest 接口 API。并且架构划分为 Controller, Service, Dao 三层。系统所需数据支持多种数据来源。包括 DDB、Kylin、Impala 等。

4.7 本章小结

本章主要介绍了云音乐产品数据分析系统的整体架构以及各个模块的架构设计。为后续实现打下基础。

第5章 系统实现

第四章节根据云音乐的业务需求及大数据相关技术，设计了云音乐产品数据分析系统的架构设计及个格模块的架构设计。本章主要根据前文的设计来介绍数据收集，数仓建设，数据计算，任务调度，Web 服务各个模块的具体实现。

5.1 数据收集

该模块主要涉及两部分，一部分是线上数据库 DDB 同步到 Hive 上，进行数据同步。另一部分是用户日志收集及解析。功能示意图如图 5.1 所示。将在下面进行详细介绍。

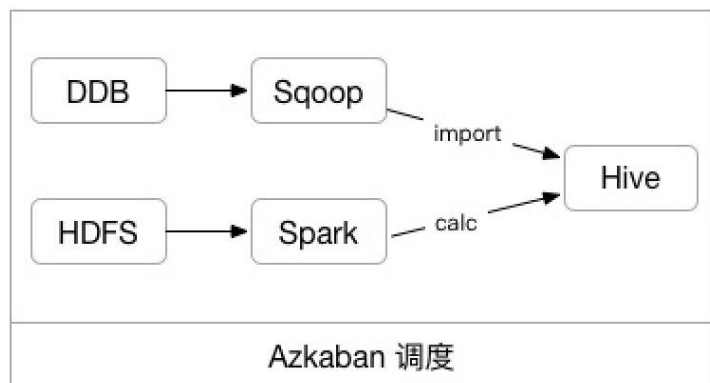


图 5.1 数据收集架构示意图

5.1.1 DDB 同步

出于业务需求，每天需要将线上数据库 DDB 同步到数据仓库中。模式为全量同步。主要采用 Sqoop 来实现。Sqoop 的介绍详见第二章。Sqoop 的同步原理是在 dump 表时，会首先查询数据库 key 的最大值和最小值，然后根据配置好 Map 的数量，以及最大的 ID 数以及最小的 ID 数来计算出均匀的 ID 区间，然后分配到每个 Map 上查询导出数据。

常用的 Sqoop 同步方式如表 5.1 所示。通过指定表及分割列来进行拷贝。并在命令行中指定文件格式等信息。

表 5.1 Sqoop 同步命令示意表

```
import
--connection-manager XXXXX
--driver XXXXX
--connect "${db.uri}"
--table ${table}
--split-by ${table.key}
--delete-target-dir -m ${job.connections} --as-jsonfile
--compression-codec com.hadoop.compression.lzo.LzopCodec
```

其中上述命令的详细参数说明见表 5.2.

表 5.2 Sqoop 参数说明示意表

| 参数名 | 参数说明 |
|--------------------------|-------------------------------|
| --connection-manager | 指定要使用的连接管理器类 |
| --driver <class-name> | 指定要使用的 JDBC 驱动类 |
| --connect <jdbc-uri> | 指定 JDBC 连接字符串 |
| --table <table-name> | 导入的源表表名 |
| --split-by <column-name> | 指定按照哪个列去分割数据 |
| --delete-target-dir | 如果指定目录存在，则先删除掉 |
| --as-jsonfile | 将数据导入到 json 文件中 |
| --compression-codec <c> | 指定 Hadoop 的 codec 方式(默认 gzip) |

Sqoop 本身在 dump 数据库表示本身的机制会存在这样的问题。如果数据库表本身的 ID 很不均匀，有很大的空洞区间时，会导致某些 Map 上有需要负责很大的数据量的导出，而某些 Map 基本没有数据导出，会导致整体的导出速度很慢，没有充分利用机器资源。解决方案是对 Sqoop 进行二次封装。手动添加空白的 id 区间保证数据 Sqoop 绕过这些空白的 ID 区间进行数据导出。

同时同步任务应该是每天定时同步线上数据库到 Hive 或者 HDFS 中。因此，

系统采用 Azkaban 进行任务调度。可以将一个 Sqoop 同步任务包装成一个 Azkaban 任务进行调度。具体实现过程如图 5.2 所示。



图 5.2 Sqoop 导入数据类图示意图

项目结构如图 5.3 所示。其中 com.xx.mapred 包下面定义了 Job 类型。Jobs 目录下定义了具体的 dump 任务。

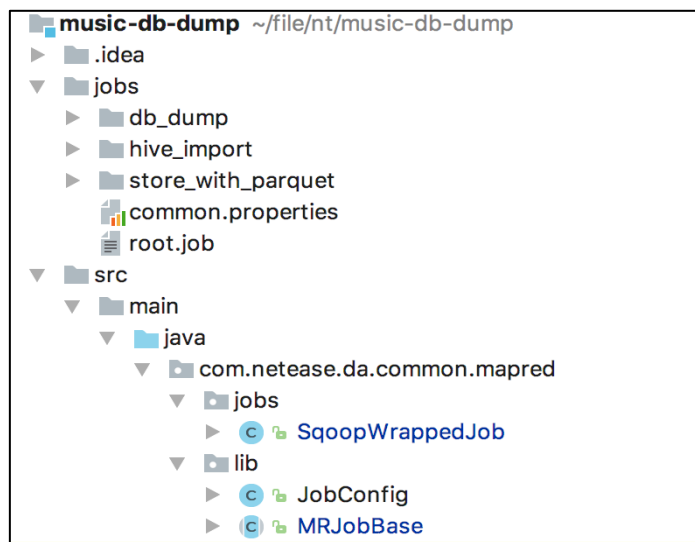


图 5.3 Sqoop 项目结构示意图

为了简化运行方式，Hadoop 自带了一些辅助类。Configured 实现了 Configuration 获取和设置。Dump 任务调度通过 Azkaban 进行调度。JobConfig 主要定义了 Job 的属性信息。包括 job 输入路径，输出路径，延迟时间，调度日期，输出分区大小等。MRJobBase 是定义的一个抽象类，主要包含创建一个 Job，以及加载设置属性，加载 Jars 到本地等公共方法的实现。定义实现的方法如图 5.4 所示。

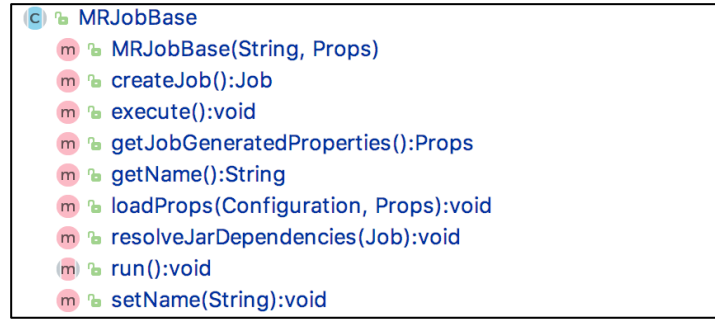


图 5.4 MRJobBase 方法示意图

其中，createJob 方法根据 configure 信息，定义了 job 对象。SqoopWrappedJob 继承了 MRJobBase 抽象类。主要实现了 run 方法。run 方法为程序运行的核心。其主要过程如下：

1. 删除 job 的输出目录 JOB_OUTPUT
2. 获取 table.stages 表分区策略
3. 如果 table.stages 设置为 null，直接按照 job 的配置信息执行 runWrappedJob 方法。
4. 如果 table.stages 设置有 id 分片。针对每一个 stage 执行一个 runWrappedJob 任务。
5. 移动结果文件到目标目录。
6. 删除中间结果
7. 给输出目录建立 LZO 索引。
8. 清除历史数据
9. 关闭连接，任务结束。

SqoopWrappedJob 类中 run 方法的具体实现如表 5.3 所示。另外系统会为每一个表的 dump 都定义一个 job 文件。其中每一个 job 文件内容如表 5.4 所示，会配置该 job 的一些信息。如同步的表名，同步策略，连接数等信息。

为了避免重复以及易于管理。针对整个公共 job 的配置信息在 common.properties 中。Job 的配置信息主要包括所运行任务的名字，任务的输出目录，任务所涉及的表名，任务涉及的表的 key，以及历史任务保留天数等等这些信息。

表 5.3 SqoopWrappedJob 实现逻辑示意表

```

public void run() throws Exception {
    //1 删除已有输出目录
    FileSystem.get(this.getConf()).delete(new
    Path(this.getConf().get(JOB_OUTPUT)), true);
    //2 获取 TABLE_STAGES
    String tableStages = this.getConf().get(TABLE_STAGES);
    //3 判断是否需要优化，并执行任务
    if (StringUtils.isBlank(tableStages)) {
        runWrappedJob(this.getConf().get(JOB_OUTPUT),
        this.getConf().get(WHERE));
    } else {
        int index = 1;
        for (String stage : tableStages.split(",")) {
            String stageOut = String.format("%s/.stage%d",
            this.getConf().get(JOB_OUTPUT), index);
            String stageWhere = String.format("%s and %s",
            this.getConf().get(WHERE), stage);
            runWrappedJob(stageOut, stageWhere);
            moveFiles(index, stageOut);
            FileSystem.get(this.getConf()).delete(new Path(stageOut),
            true);
            index++;
        }
    }
    //4 为输出目录添加索引
    addLzoIndex();
    //5 根据配置信息，清空历史数据。只保留最近的数据
    clearHistoryData();
}

```

表 5.4 任务配置示意表

```

table=Music_SongAppBIStatus
table.stages=SongId<42462601, SongId>=42462601
job.connections=6
table.key=SongId

```

其中一个配置文件示例如表 5.5 所示。

表 5.5 common 文件配置示意表

| 参数名 | 参数说明 |
|--------------------|--------------|
| classPath | 所需加载的类路径 |
| job.class | 所运行任务的类名 |
| wrapped.job.class | Azkaban 任务类型 |
| job.connections | 任务连接数 |
| main.args | 主要命令参数 |
| job.output | 任务输出目录 |
| table.key | 表主键设置 |
| job.history.remain | 任务历史记录保留天数 |

5.1.2 日志解析

本章主要介绍用户日志收集的设计与实现。

通过日志收集，系统能够拿到每一天的用户日志文件。文件以 JSON 的格式存储在 HDFS 中。目前要做的就是日志解析，将系统所需要的信息存储到 hive 中，供后续分析使用。

以 useraction 为例，说明日志解析过程和存入 hive 表。图 5.5 所示是整个 UA 日志的解析过程。

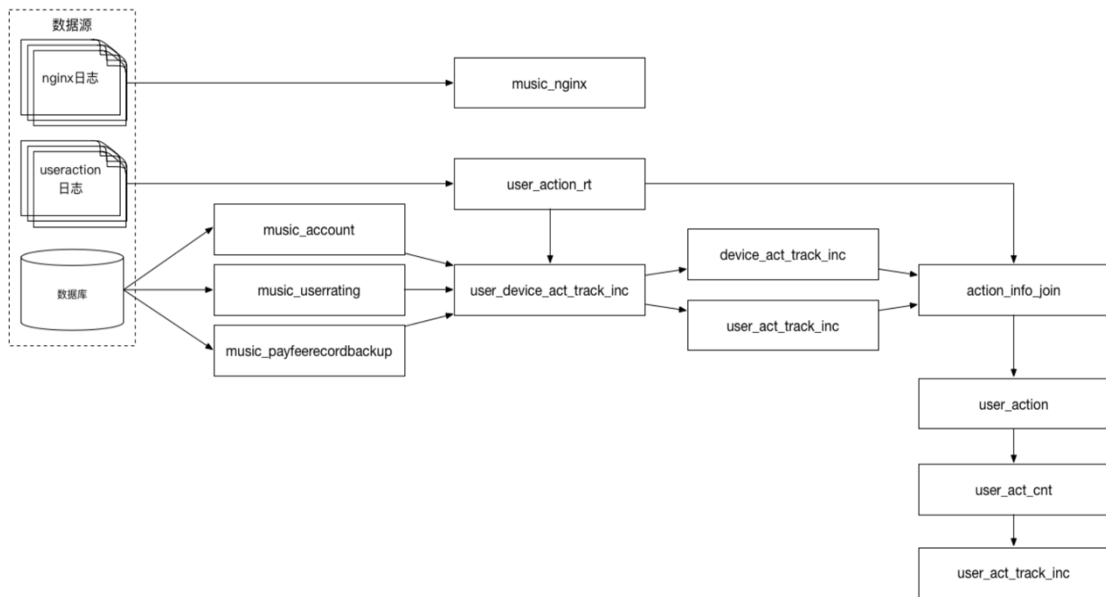


图 5.5 用户日志解析过程示意图

图中每一个节点都是一个 Spark Job，每个 job 的输出是一张 hive 表。输入是文件路径。

各节点表说明如下所示。

1. **music_nginx**: 读取 Nginx 日志生成相应的 hive 表。
2. **music_account**: 从应用数据库中 **music_account** 表 dump 出来的快照表，主要读取这张表用户的状态信息，是否为过渡账号。
3. **music_userrating**: 从应用数据库中 **music_userrating** 表 dump 出来的快照表，主要读取用户等级信息
4. **music_payfeerecordbackup**: 从应用数据库中 **Music_PayFeeRecordBackup** 表 dump 出来的快照表，读取用户 vip 信息。
5. **user_action_rt**: 定时读取 /datastream/music/useraction 的归档日志，预处理日志，生成准实时的日志表 **user_action_rt**。输出文件补充说明：这个任务每小时增量处理一次，生成的文件会议 action 的名称为前缀，如 play 的日志会存在 /user/da_music/hive/warehouse/music_dw.db /user_action_rt/dt=2016-11-21/type=valid/play_*.lzo 这样的文件中，其中 type=valid 表示格式正确的日志，type=error 表示格式有问题的日志。
6. **user_device_act_track_inc**: 用户设备登陆记录表，每天一份的快照表记录了所有用户最近 30 天的登陆记录。

7. `device_act_track_inc`: 从设备维度统计的登陆记录表,按设备记录所有设备最近 30 天的活跃情况, 每天一份所有设备的快照。
8. `user_act_track_inc`: 从用户维度统计的登陆记录表,按设备记录所有设备最近 30 天的活跃情况, 每天一份所有用户的快照。
9. `action_info_join`: 从 `device_act_track_inc` 读取设备信息、从 `user_act_track_inc` 读取用户信息吗, 将用户信息和设备信息 `merge` 到 `user_action_rt` 处理后的行为日志中。输出目录 `/user/da_music/out/music_dw/action_info_join/${etl_day}/`, 输出文件会按照日志 `logtime` 的时间输出近三天的日志。
10. `user_action`: 读取当天以及后两天的 `action_info_join` 的结果, 补充修复日志, 去除重复日志, 并日志进行去重处理, 将日志按照 `action` 进行分区 `load` 到 `hive` 中, 生成 `user_action` 表以及 `sys_action` 表。
11. `user_act_cnt`: 分解行为日志中 `json` 信息, 将功能点相关信息和操作对象信息剥离成 `context` 和 `target_info` 两个字端, 去除 `ip`、`device`、`seq` 等信息, 将用户行为进行一次计数一次抽象操作。
12. `user_act_cnt_inc`: 读取 `user_act_cnt` 的数据, 统计用户对这个功能点第一次行为、最后一次行为、以及最近 30 天的行为信息, 每天一份快照, 累计数据。

以 `Nginx` 日志解析为例进行说明。`MusicNginxJob` 实现了解析 `Nginx` 日志。该任务继承于 `AzkabanSparkJob` 类。`AzkabanSparkJob` 主要实现 `Azkaban` 任务相关的接口。`MusicNginxJob` 主要解析过程如下:

1. 设置正则表达式
2. 匹配文件中每一行, 解析具体的字段
3. 将获取的结果写入到表 `music_nginx` 中。

其中解析每行的伪代码如下所示。

1. 抓取每行数据 `body`。
2. 按照正则匹配 `ip`, `time`, `requestStr`, `code`, `size`, `refer` 等字段。
3. 解析 `musicNginx` 相关参数。
4. 进行计数统计。
5. 返回解析结果, 保存到变量中。最后一起进行存储。

除去上述所提到的数据库同步以及用户日志收集两部分内容, 一个完善的数

据还应涉及到其他系统数据的同步迁移等问题。在此不做涉及。

5.2 维度建模

数据是一切的根本，如何组织存储大量的数据来进行高效的分析，成为一项重要的工作。为了数据高效的使用以及尽可能简便易用的复用，将整个数据仓库设计成三层体系结构。分别是数据操作层，公共维度模型层，应用数据层。其中公共维度模型层分为数据明细层和数据汇总层。

5.2.1 数据操作层

数据操作层包含了所有的原始以及底层数据。这些数据构成了整个数仓的最底层。在该系统中，数据操作层数据主要包含用户日志相关表，Nginx 日志表，曲库表，线上数据库表等数据。大部分表都采用按天进行分区，从而方便管理及查询。该层所包含的具体数据划分如下所示。

1. 业务数据库数据: 通过 Sqoop 每天凌晨 dump 数据库表快照到 HDFS 上。
2. 日志数据: 包括服务器 useraction 和 Nginx 日志。
3. 其他系统同步过来的数据: 包含移动 DA 系统，推广系统等。

其中用户日志表及 Nginx 表的生成及解析可以参考用户日志收集模块的设计与实现。

5.2.2 公共维度层

为了解决数据的复用性，及屏蔽后台日志打点变化造成上层业务逻辑变化等一些列问题，抽取中间层数据维度层，即 CDM 层。该层主要采用维度建模的思想，基于星型模型进行设计。而为了减少重复及数据服用，将该层又分为两层，一层是明细层，以维度表和无事实的事实表为主，主要是为了屏蔽底层变化。另一层是汇总表，主要是进行数据复用，避免不必要的重复计算问题。

数据明细层:

按照维度建模的思想，云音乐产品数据分析系统采用星型模型。事实表包含无事实的事实表。以用户行为为例，来进行主题划分。每一个主题，相应的建立一个维度表。维度表包含如下主题:

1. mv 表 mv_meta_info
2. 用户信息表 user_meta_info
3. 视频信息表 video_meta_info

4. 歌曲信息表 `song_meta_info`
5. 歌单信息表 `list_meta_info`
6. 专辑信息表 `album_meta_info`
7. 艺人信息表 `artist_meta_info`
8. 动态信息表 `event_meta_info`
9. 电台信息表 `dj_meta_info`
10. 评论信息表 `comment_meta_info`

每一个主题都维护一个底层的维度表，包含了该 `object` 的各种基本属性。以 `comment_meta_info` 为例来说明维度表是如何构建的，另外维度表的数据来源基本上来自于数据操作层。上述所有的维度表按照日期天进行分区，存放在 `dw` 库下面。以上所有的 `Azkaban Job` 每天会进行定时调度，相应的维度表每天增加 `dt` 分区。

并且在上述表中，所有的数据每天都是全量的。这么做的好处，就是可以保持数据的完整性以及可回溯。另外这些表都是以 `Parquet` 的格式进行存储。可以充分的减少存储空间。表 5.6 展示的是评论信息表的建设过程。

事实表主要按照行为进行划分。基本上是每一个主题对应于各种行为所组成的事实表。事实表包含如下信息。

1. 用户播放事实表 `user_play_fact`
2. 用户分享事实表 `user_share_fact`
3. 用户点赞事实表 `user_zan_fact`
4. 用户下载事实表 `user_download_fact`
5. 用户评论事实表 `user_comment_fact`
6. 用户购买事实表 `user_buy_fact`

以用户评论事实表为例，进行说明。详细解析过程见表 5.7 所示。

维度明细层按照星型模型建模的思想构建。

汇总数据层，主要包含一些汇总表。公共指标汇总数据一般根据维度表数据和明细表数据加工生成。这里面，主要采用宽表化手段构建公共指标数据层，提升公共指标的复用性，减少重复加工。

表 5.6 评论信息表代码

```

class CommentMetaInfo(@@(transient@param) sc: SparkContext) extends
AzkabanSparkJob(sc) {
  override def run(): Unit = {
    val hive = sqlContext.newSession()
    val output = sparkConf.get("spark.job.output") + s"/${etlDay}"
    hive.sql(
      """
        |select id, resourcetype resource_type,
        |get_json_object(ResourceJson, "$.id") resource_id,
        |get_json_object(ResourceJson, "$.userId") resource_user_id
        |from music_db_front.music_commentthread
      """.stripMargin).createOrReplaceTempView("comment_thread")
    hive.sql(
      """
        |select id, thread_id, comment_ip,
        |reply_info['content'] content,
        |reply_info['reply_comment_id'] reply_comment_id,
        |reply_info['reply_user_id'] reply_user_id,
        |comment_time, status, user_id, liked_count
        |from
        |(
        |  select id, threadid thread_id, userip comment_ip,
        |  parse_contents(contents) reply_info,
        |  cast(substr(time, 0, 10) as bigint) comment_time,
        |  cast(status as int) status, userid user_id,
        |  cast(likedcount as bigint) liked_count
        |  from music_db_front.Music_Comment
        |)
      """.stripMargin).createOrReplaceTempView("comment_info")
    hive.sql(
      s"""
        |select cast(comment_info.id as bigint) id,
        |cast(user_id as bigint) user_id,
        |content, comment_time, comment_ip,...
        |from comment_info join comment_thread on
comment_info.thread_id = comment_thread.id
      """.stripMargin).repartition(100).write.mode(SaveMode.Overwrite).
parquet(output)
    hive.newSession().sql(s"load data inpath '$output' " +
      s"overwrite into table music_dimension.comment_meta_info")
    HadoopUtil.getHadoop.fsRm(output)
  }
}

```

维度汇总层数据主要包含以下一下信息表:

1. 用户歌曲日行为表 `user_song_action_day`
2. 用户专辑日行为表 `user_album_action_day`
3. 用户 mv 日行为表 `user_mv_action_day`
4. 用户视频日行为表 `user_video_action_day`
5. 用户歌单日行为表 `user_playlist_action_day`
6. 用户动态日行为表 `user_event_action_day`
7. 用户艺人日行为表 `user_artist_action_day`

以 `user_song_action_day` 为例进行说明。

主要描述了在某一天某个用户和某首歌之间的行为，比如播放了多少次，分享了多少次，下载了多少次等等。该表也包含了部分用户信息，比如用户 ID、用户真实 ID、设备号、时间戳、设备系统等等。该表会按照日进行分区。这么做的好处是方便数据管理与复用。

5.2.3 数据应用层

主要是针对具体业务所创建的中间表。这部分数据为对外服务提供的数据。针对云音乐数据产品系统，该部分的数据设计，将在数据计算模块详细展开。

表 5.7 UserCommonFactJob 实现代码

```

class UserCommentFactJob(@transient@param sc: SparkContext) extends
AzkabanSparkJob(sc) {
  override def run(): Unit = {
    val hive = sqlContext.newSession()
    val output = sparkConf.get("spark.job.output") + s"/${etlDay}"
    val typeMap = Map(
      "album" -> 3, "song" -> 4, "topic" -> 6, "event" -> 2, "mv" -> 5,
      "dj" -> 1, "shortvideo" -> 62, "list" -> 0)
    hive.udf.register("get_type_id", (resourceType: String, threadId: String) => {
      var typeId: Int = -1
      if (resourceType != null) typeId = typeMap.getOrElse(resourceType, -1)
      if (typeId == -1 && threadId != null) {
        try {typeId = threadId.split("_")(2).toInt
        } catch {case _: Throwable => Unit} }typeId))
    hive.sql(
      s"""
      |select userid user_id, anonymous, rid, deviceid device_id, os,
osver os_ver, ip,
      |appver app_ver,
      |cast(logtime/1000 as bigint) log_time,
      |props['netstatus'] net_status,
      |fix,
      |props['abtest'] ab_test,
      |cast(props['cid'] as bigint) comment_id,
      |props['tid'] thread_id,
      |props['type'] resource_type,
      |get_type_id(props['type'], props['tid']) resource_type_id,
      |cast(props['id'] as bigint) resource_id,
      |if(action = 'commentreply', 1, 0) is_reply,
      |cast(props['rid'] as bigint) reply_id,
      |cast(props['resourceOwnerId'] as bigint) resource_user_id
      |from music_dw.user_action
      |where dt = '${etlDay}' and action in ('comment',
'commentreply')""").stripMargin
      ).withColumn("resource_type_id", expr("if(resource_type_id < 0, null,
resource_type_id)")
      ).repartition(1).write.mode(SaveMode.Overwrite).parquet(output)
      sqlContext.newSession().sql(s"load data inpath '${output}' overwrite
into table music_fact.user_comment_fact partition (dt='${etlDay}')"
    )
  }
}

```

5.3 数据计算

根据第三章所提到的业务需求，需要计算各个资源各种行为所带来的新增，活跃，召回等指标。以及需要计算各个市场活动相应的指标。根据以上业务需求描述。计算过程如图 5.6 所示。

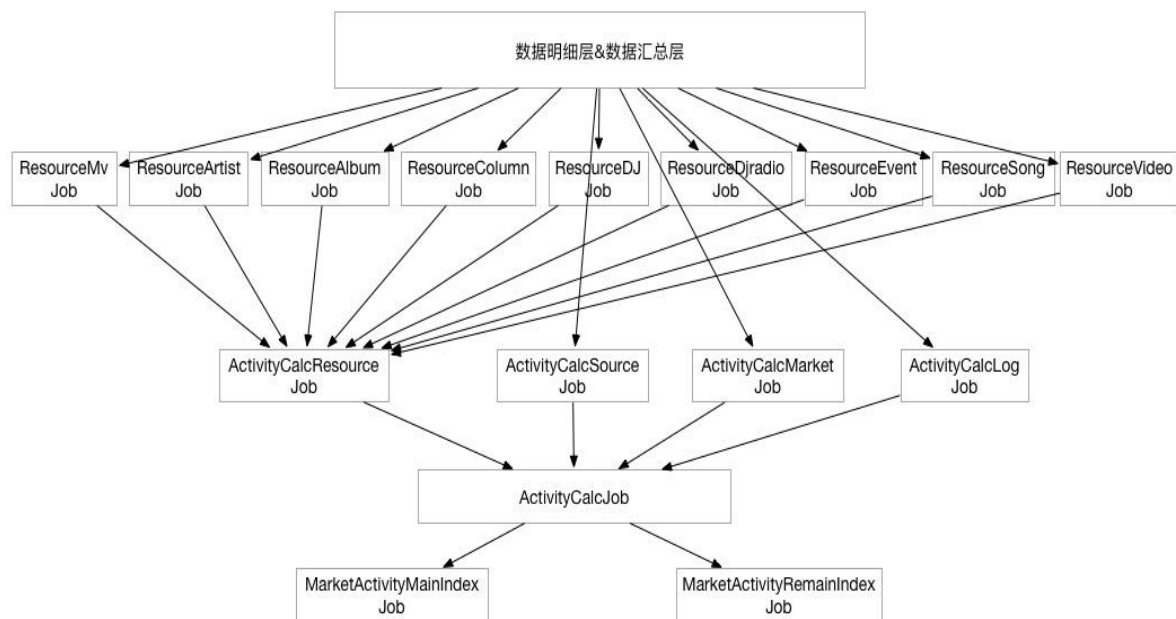


图 5.6 资源型及市场活动计算任务依赖示意图

根据任务依赖图，粗略的将计算任务划分为 5 个层次。每一个节点代表一个 Spark Job。其中所有任务的数据输入都是数据仓库中数据明细层和汇总层中所涉及的表。以下是各个任务节点的解释及详细含义。

ResourceXXJob: 主要计算各个资源型各个行为相关数据。输出结果都存储在 music_dw.resource_user_action_index 表中。该表按照资源类型和 dt 进行分区。

ActivityCalcSourceJob: 主要计算渠道包相关数据。结果存储在表 music_dw.source_user_type_day。该表主要按照 dt 和用户类别进行分区。

ActivityCalcMarketJob: 主要计算 market 参数相关数据。结果存储在表 music_dw.market_user_type_day。该表主要按照 dt 和用户类别进行分区。

ActivityCalcJob: 在市场活动维度上，合并资源型，渠道包，market 参数等数据。结果存储在 music_dw.activity_user_index_type。该表按照 dt 和跟踪类别进行分区。该任务会进行内部去重。

MarketActivityMainIndexJob: 主要计算市场活动带来的新增, 活跃, 召回等指标。计算过程中, 会进行活动之间的优先级归因去重, 防止指标偏高。输出结果存储在线上数据库 DDB 中。

MarketActivityRemainIndexJob: 主要计算市场活动带来的次日留存及周留存, 月留存等指标。计算过程中, 会进行活动之间的优先级归因去重, 防止指标偏高。输出结果存储在线上数据库 DDB 中。

在上述计算过程中, 会涉及到很多中间表。市场活动基础表为 `bi_music_market_activity`。存在 DDB 中, 每天 `dump` 到 `hive` 里面。主要包含了活动 `id`, 名字, 活动开始时间, 结束时间, 状态, 创建者等信息。具体属性如表 5.8 所示。

表 5.8 市场活动属性表

| 列名 | 类型 | 说明 |
|-------------|--------|--------------------------|
| id | String | 市场活动 ID |
| name | String | 所运行任务的类名 |
| level | String | 市场活动优先级 |
| start_date | String | 市场活动开始时间 |
| end_date | String | 结束时间 |
| status | String | -1 表示删除, 0 表示进行中, 1 表示结束 |
| create_user | String | 创建人 |
| day | String | 日期 |

每一个活动支持多种计算方式, 包括 `market` 匹配, 埋点匹配, 渠道包, `idfa/imei`, 资源型五大类。这些匹配规则由用户填写, 存在 DDB 中, 每天 `dump` 到 `hive` 里面。表属性如表 5.9 所示。

`market` 参数计算逻辑: 计算当天所有的 `market` 参数带来的新增、活跃、召回三个指标, 存到 `music_dw.market_user_type_day` 中。表结构如表 5.10 所示。其中

按照 dt 和 index_type 属性进行分区。

表 5.9 市场活动跟踪方式属性表

| 列名 | 类型 | 说明 |
|------------|--------|---|
| id | String | 市场活动跟踪方式 ID |
| activityId | String | 市场活动 ID |
| type | String | logsearch (埋点匹配) market (market 匹配) resource (资源型) source (渠道包) idfaimei |
| info | String | 计算方式, Json 格式存储 |
| day | String | 日期 |

表 5.10 市场活动 market 参数属性表

| 列名 | 类型 | 说明 |
|-----------------|--------|------------------------------|
| source | String | source 类别 |
| url_click | bigint | url 点击时间 |
| user_first_time | bigint | 用户当天第一次登陆时间 |
| deviceid | String | 设备 id |
| rid | String | 用户真实 id |
| ip | String | IP 地址 |
| osver | String | 设备版本号 |
| index_type | String | 用户类型, 包括 new, recall, active |

渠道包计算: 计算当天所有渠道带来的新增、活跃、召回 三个指标, 存到 music_dw.source_user_type_day 中。计算细节见 SourceJob。

表属性如表 5.11 所示。按照 `dt` 和 `index_type` 进行分区。

表 5.11 市场活动渠道包属性表

| 列名 | 类型 | 说明 |
|------------|--------|-------------------|
| source | String | 市场活动渠道包 |
| device_id | String | 设备 ID |
| userid | bigint | 用户 ID |
| rid | bigint | 用户真实 ID |
| ban | int | 是否封禁，0 表示没有，1 表示有 |
| os | String | 设备系统 |
| Index_type | String | 指标类型，包括新增，活跃，召回 |

资源型计算：计算各个资源当天所有行为的明细。将具体数据存到 `music_dw.resource_user_action_index`。具体任务见 `ResourceXXXJob.scala`。具体属性表设计如表 5.12 所示。

最后，按照活动优先级和不区分优先级分别聚合，来计算新增、活跃、召回、留存等指标。数据存储在 DDB 中。以上所描述的所有计算任务，最终都会通过 Azkaban 进行调度。每天晚上凌晨 1 点定时计算。

5.4 Web 服务模块

目前所有的数据都已经存储在 DDB 和 Kylin 中，需要做的是将这些数据通过 web 平台对外提供服务。Web 平台主要采用 Spring, MVC 的架构设计。系统主要包含 Controller 层，Service 层，Dao 层三层结构。针对第三章提出的业务需求。大致包含三个功能模块，分别是资源型，市场活动管理，市场活动数据查询。

以市场活动为例。其主要包含市场活动的添加，修改，删除，查询等接口，以及包含数据的查询和导出等功能。其所实现的功能如表 5.13 所示。平台前端和后台数据传输格式为 JSON。从而方便数据的解析与获取。

表 5.12 市场活动资源型属性表

| 列名 | 类型 | 说明 |
|---------------|--------|------------------------|
| resource_id | String | 市场活动渠道包 |
| device_id | String | 设备 ID |
| userid | bigint | 用户 ID |
| rid | bigint | 用户真实 ID |
| action | int | 行为, 包含 play, comment 等 |
| pv | bigint | 次数 |
| resource_type | String | 资源类型, 包括 mv,song 等 |

表 5.13 市场活动 API 表

| 方法 | API | 说明 |
|-----------------|------------------------------------|----------|
| getActivityList | /api/tracking/activity/list | 获取市场活动列表 |
| deleteActivity | /api/tracking/activity/delete | 删除某个市场活动 |
| saveActivity | /api/tracking/activity/save | 保存某个市场活动 |
| changeLevel | /api/tracking/activity/changelevel | 修改活动优先级 |
| getIndex | /api/tracking/activity/indexlist | 查询活动数据 |
| export | /api/tracking/activity/export | 活动数据导出 |

5.5 本章小结

本章节主要介绍了云音乐产品数据分析系统的实现过程。主要包括数据收集, 数据仓库建设, 数据计算, 任务调度, web 服务五个模块。整个实现流程包含了数据的收集到处理到服务一系列操作。

第6章 系统应用

本文第三章提出了云音乐的业务需求，即最终该系统要提供资源型查询和市场活动数据指标查询两部分功能。第五章主要介绍了该系统的实现过程。本章节主要介绍了这两部分功能的效果展示。

6.1 资源型模块

页面提供如下选择控件。

资源类别，以下拉列表形式提供。包括单曲，MV，DJ，电台，视频，专辑，歌单，歌手，专栏，话题，认证用户，活动页，动态等类别。

用户类别，以下拉列表形式提供。包括新增，活跃，召回三种类别。

指标类别，以下拉列表形式提供。包括 PV（人次），UV（人数）两种。

资源名称，支持自定义输入，可输入歌曲 ID，艺人 ID 等自定义查询条件。

日期控件，支持输入所查询日期范围。

查询结果提供日期，ID，名称，播放，第一手播放，搜索点击，站外分享，分享引入，评论，收藏/订阅，引入的会员人数，引入的会员金额等信息。

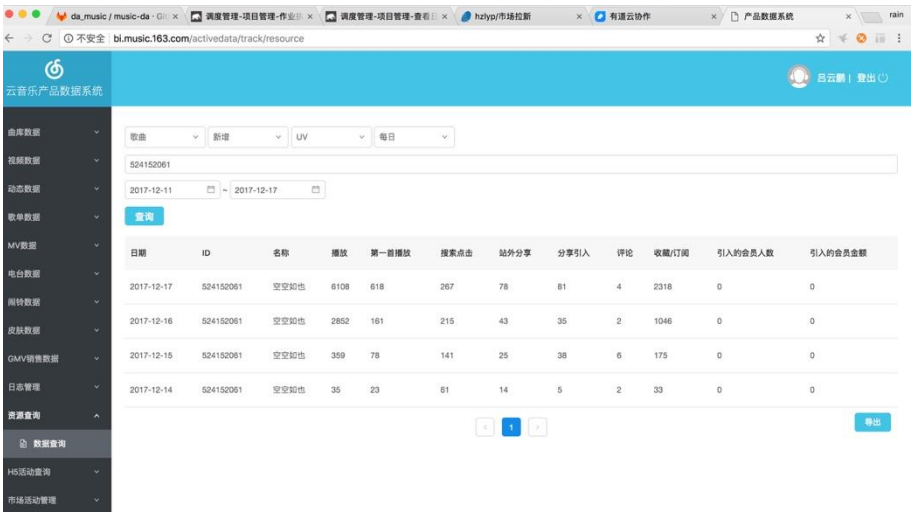


图 6.1 云音乐产品数据分析系统资源型数据查询示意图

以图 6.1 为例。查询歌曲 ID 是 524152061 这首歌在 2017-12-11 到 2017-12-17 这几日每天的各个指标的 UV 数。特此声明，不同指标之间未进行去重。

该模块支持导出功能，导出格式为 CSV 格式。

6.2 市场活动模块

该模块旨在通过计算每个市场活动带来的新增，活跃，召回等指标来评价某个活动的好坏。

同一时期，可以同时展开多个活动，在要求计算时，要求按照市场活动的优先级进行用户归因去重。从而避免数据偏高。同一活动，支持多种计算方式进行跟踪。针对统一活动，多种计算方式，支持用户粒度的去重计算。同一计算方式，支持不同层次行为的统计，如资源型的播放，第一次播放可同时勾选。因此，系统支持不同行为之间的去重求并集运算。

基于以上逻辑，该模块包含市场活动管理，即配置活动计算规则和优先级。和市场活动数据查询两个字模块。具体应用效果如下所示。

市场活动展示：

市场活动氛围更好进行中和已结束两种状态。市场活动所展示指标包含优先级，活动名称，开始时间，结束时间，创建者，状态，计算方式等内容。

| | | | | | | |
|---|-------------------|----------|-------|---------------------------|-----|----------|
| 云音乐产品数据系统 | | | | | | |
| bi.music.163.com/activedata/market/activity/manager | | | | | | |
| 吕云鹏 退出 | | | | | | |
| 动态数据 | | | | | | |
| 歌单数据 | | | | | | |
| MV数据 | | | | | | |
| 电台数据 | | | | | | |
| 南铃数据 | | | | | | |
| 皮肤数据 | | | | | | |
| GMV销售数据 | | | | | | |
| 日志管理 | | | | | | |
| 资源查询 | | | | | | |
| H5活动查询 | | | | | | |
| 市场活动管理 | | | | | | |
| 活动管理 | | | | | | |
| 数据查询 | | | | | | |
| 市场推广管理 | | | | | | |
| 进行中 已结束 活动优先级 | | | | | | |
| 新增活动 | | | | | | |
| 共983条数据 | | | | | | |
| 优先级 | 活动名字 | 开始时间 | 结束时间 | 创建者 | 状态 | 操作 |
| 994 | 移动官网 | 2017/9/4 | ----- | hzbao@corp.netease.com | 进行中 | 关闭 编辑 删除 |
| 993 | 百度阿拉丁 | 2017/9/5 | ----- | xiaochao@corp.netease.com | 进行中 | 关闭 编辑 删除 |
| 992 | 百度SEM投放 | 2017/9/5 | ----- | hzyoujie@corp.netease.com | 进行中 | 关闭 编辑 删除 |
| 991 | 校园音乐人推荐 (7月11日开始) | 2017/9/6 | ----- | hxieqid@corp.netease.com | 进行中 | 关闭 编辑 删除 |
| 990 | 张大仙你是妖怪推广 | 2017/9/6 | ----- | hxieqid@corp.netease.com | 进行中 | 关闭 编辑 删除 |
| 989 | 农夫山泉-歌单 | 2017/9/7 | ----- | hzcay@corp.netease.com | 进行中 | 关闭 编辑 删除 |

图 6.2 云音乐产品数据分析系统市场活动展示示意图

市场活动支持关闭，编辑，删除等操作。具体效果图如图 6.2.1 所示。

市场活动编辑：

支持用户新增或者变异一个市场活动，主要编辑内容包括添加或修改活动名

称，添加跟踪计算方式两类。其中活动名称要求全局唯一不重复。

支持多种跟踪方式同时设置。跟踪方式包括资源型，埋点匹配，IDFA/IMEI，Market 参数等。

支持多种跟踪方式多次配置。

资源型支持多资源类别及 ID 配置。并支持多种行为进行并集勾选。

如图 6.3 所示，展示了新增一个叫做移动官网的市场活动。

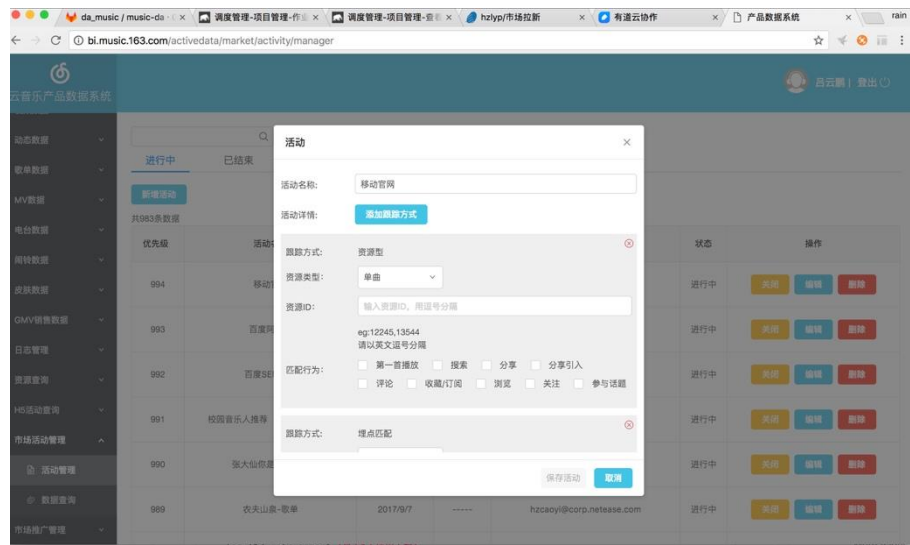


图 6.3 云音乐产品数据分析系统市场活动添加示意图

当用户创建完一个活动后，默认优先级是 1，此刻，用户可以通过优先级管理进行调整。整个设计可以通过拖拽来实现更改活动的优先级。排位越靠前，优先级越高。优先级高的活动，会优先进行归因计算。

优先级调整效果具体设计如图 6.4 所示。

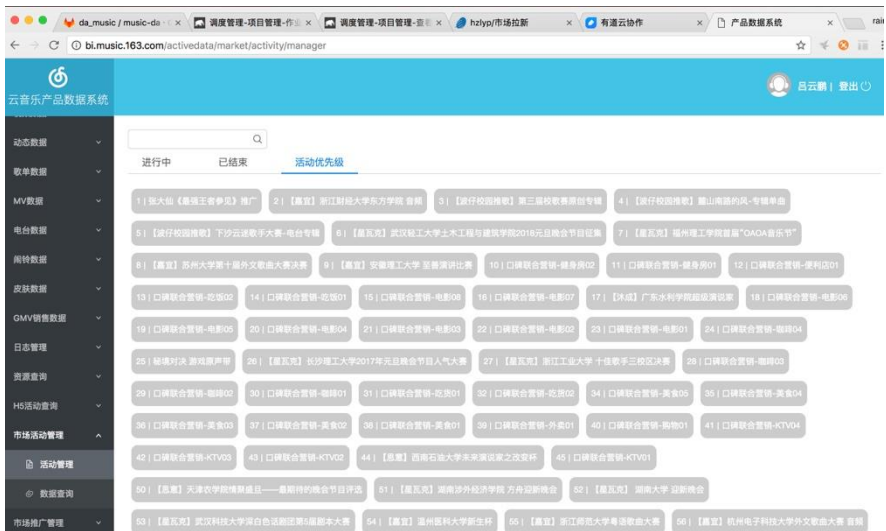


图 6.4 云音乐产品数据分析系统市场活动优先级配置示意图

市场活动查询：

配置好的市场活动，次日会进行任务调度，依据规则，进行计算。并将结果存储在 DDB 中供系统进行查询。系统支持按照设备端进行查询，包括 IOS 端，Android 端,PC 端和不区分。系统支持查询所选日期范围内进行中活动相关指标。包括日期，优先级，活动名称，开始时间，结束时间，创建者，状态，新增不区分优先级，新增区分优先级，日活不区分优先级，日活区分优先级，召回不区分优先级，召回不区分优先级，1 日留存，7 日留存，30 日留存，播放，播放率。

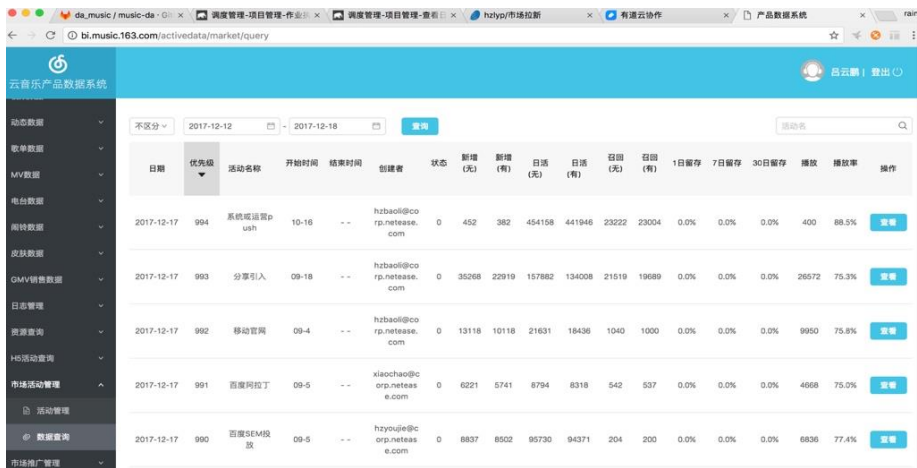
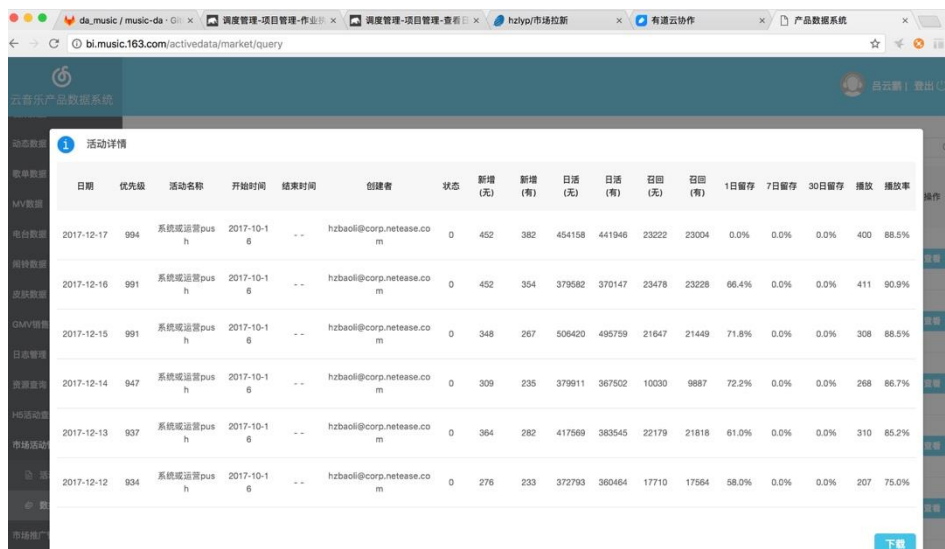


图 6.5 云音乐产品数据分析系统市场活动指标查询示意图

用户可以通过输入所查询的起始日期，来查看所选范围内，每个活动所带来

的新增，召回，活跃等指标。默认显示所选日期的最后一天。如图 6.5 所示，查询 2017-12-12 至 2017-12-18 日期范围内，各个活动的相关指标。

系统支持用户点击所选中列，查看该活动每日详情。并支持下载 excel 文件，供分析使用。如图 6.6 所示，展示了所选活动名为系统或运营 push 这六天的详细指标。



| 日期 | 优先级 | 活动名称 | 开始时间 | 结束时间 | 创建者 | 状态 | 新增(无) | 新增(有) | 日活(无) | 日活(有) | 召回(无) | 召回(有) | 1日留存 | 7日留存 | 30日留存 | 播放 | 播放率 |
|------------|-----|-----------|------------|------|--------------------------|----|-------|-------|--------|--------|-------|-------|-------|------|-------|-----|-------|
| 2017-12-17 | 994 | 系统或运营push | 2017-10-16 | -- | hzbaoil@corp.netease.com | 0 | 452 | 382 | 454158 | 441946 | 23222 | 23004 | 0.0% | 0.0% | 0.0% | 400 | 88.5% |
| 2017-12-16 | 991 | 系统或运营push | 2017-10-16 | -- | hzbaoil@corp.netease.com | 0 | 452 | 354 | 379582 | 370147 | 23478 | 23228 | 66.4% | 0.0% | 0.0% | 411 | 90.9% |
| 2017-12-15 | 991 | 系统或运营push | 2017-10-16 | -- | hzbaoil@corp.netease.com | 0 | 348 | 267 | 506420 | 495759 | 21647 | 21449 | 71.8% | 0.0% | 0.0% | 308 | 88.5% |
| 2017-12-14 | 947 | 系统或运营push | 2017-10-16 | -- | hzbaoil@corp.netease.com | 0 | 309 | 235 | 379811 | 367502 | 10030 | 9887 | 72.2% | 0.0% | 0.0% | 268 | 86.7% |
| 2017-12-13 | 937 | 系统或运营push | 2017-10-16 | -- | hzbaoil@corp.netease.com | 0 | 364 | 282 | 417569 | 383545 | 22179 | 21818 | 61.0% | 0.0% | 0.0% | 310 | 85.2% |
| 2017-12-12 | 934 | 系统或运营push | 2017-10-16 | -- | hzbaoil@corp.netease.com | 0 | 276 | 233 | 372793 | 360464 | 17710 | 17564 | 58.0% | 0.0% | 0.0% | 207 | 75.0% |

图 6.6 云音乐产品数据分析系统市场活动指标明细示意图

6.3 本章小结

本章节主要介绍了云音乐产品数据分析系统的具体应用效果。包括资源型查询，市场活动管理及数据分析等功能。

第7章 总结与展望

7.1 本文的主要研究贡献

本论文主要介绍了云音乐产品数据分析系统的设计与实现。旨在以云音乐数据分析为背景，提出了一个大数据系统的通用架构，包括数据收集，数据存储，数据仓库建设，数据分析及任务调度和服务提供等一系列处理阶段。并对每个阶段的相关技术进行了简单阐述以及每个阶段进行了设计与实现。

本论文中，第二章主要介绍了大数据系统所涉及的相关前沿技术。第三章提出了云音乐产品数据分析系统的业务需求。系统需要通过分析用户数据，来查询各种资源的热度及新增，活跃，召回等相关指标。第四章主要根据业务需求，提出了整体的系统框架，包括数据采集，数仓建设，数据分析，对外服务四个部分。并提出了每一个模块的设计。第五章根据第四章的设计，介绍了该系统如何实现，以及实际实现中所遇到的困难以及如何解决。其中包括二次封装 **Sqoop** 进行数据库同步，以及维度模型建模来使数据高效组织及易使用等等。第六章主要展示了云音乐产品数据分析系统的应用，包括系统界面展示及相关使用说明。主要包含了资源型查询及市场活动查询两大部分。

本文旨在以云音乐业务需求为驱动，设计并实现一套大数据数据分析系统。并通过该系统的数据分析对云音乐的相关市场决策提出指导意义。

7.2 进一步研究

系统业务需求方面，未来计划增加更多的数据服务，包括热点资源数据分析，日志管理等，用以丰富平台所对外提供的功能。

整体系统架构方面，在数据库同步方面，目前采用包装 **Sqoop** 程序，手动添加空白的 **id** 区间保证数据 **Sqoop** 绕过这些空白的 **ID** 区间进行数据导出，但是这也就是只是一个临时方案，手动维护 **ID** 区间也是一个很麻烦的事情，后续平台组这边可以根据 **DDB** 的特性进行数据分片。拟采用通过 **Spark** 自己实现一个 **dump** 功能。在维度模型建模这块，应当丰富数据汇总层相关表建设，尽量避免上层使用最底层数据。现有系统数据结果是通过离线定时计算，存储到 **DDB** 中，未来可以考虑支持更加实时的计算。

以上的这些问题都值得在今后的工作中继续研究。

参考文献

- [1] GANTZ J, REINSEL D. The Digital Universe in 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East[J]. IDC Analyze the Future, 2012, 2007: 1-16.
- [2] Mayer-Schönberger V, Cukier K. Big data: A revolution that will transform how we live, work, and think[J]. Houghton Mifflin Harcourt, 2013.
- [3] J2EE: <http://www.oracle.com/technetwork/java/javaee/overview/index.html>.
- [4] 陈亚楠, 李涛. 基于 Hadoop 模式的数据查询系统设计[J], 经营管理者, 2014: 78.
- [5] Apache Flume[EB/OL]. <http://flume.apache.org/>, 2018-04-01.
- [6] Kafka[EB/OL]. <http://kafka.apache.org/>, 2018-04-01.
- [7] Sqoop[EB/OL]. <http://sqoop.apache.org/>, 2018-04-01.
- [8] Hadoop[EB/OL]. <http://hadoop.apache.org/>, 2018-04-02.
- [9] Spark[EB/OL]. <http://spark.apache.org/>, 2018-04-02.
- [10] Azkaban[EB/OL]. <https://azkaban.github.io/>, 2018-04-02.
- [11] Airflow[EB/OL]. <http://airflow.incubator.apache.org/>, 2018-04-02.
- [12] DDB[EB/OL]. <http://www.iteye.com/news/32029>, 2018-04-02.
- [13] HDFS[EB/OL]. http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html, 2018-04-02.
- [14] Impala[EB/OL]. <http://impala.apache.org/>, 2018-04-02.
- [15] Kylin[EB/OL]. <http://kylin.apache.org/>, 2018-04-02.
- [16] HBase[EB/OL]. <http://hbase.apache.org/>, 2018-04-02.
- [17] Scribe[EB/OL]. <https://github.com/facebookarchive/scribe>, 2018-04-02.
- [18] TimeTunnel [EB/OL]. <http://code.taobao.org/p/TimeTunnel/wiki/index/>, 2018-04-02.
- [19] Pig[EB/OL]. <http://pig.apache.org/>, 2018-04-02.
- [20] Hidalgo C A. How to Transform big data into knowledge[J]. 2013-04-24.
- [21] Ghemawat S, Gobioff H. The Google File System[J]. Proc of the 19th ACM Symposium on Operating System Principles, 2003: 29-43.
- [22] George L. HBase: The Definitive guide[M]. USA: O'REILLY, 2011.

- [23] White T. Hadoop 权威指南,南京[M], 东南大学出版社, 2011.
- [24] Zaharia M, Chowdhury M, Franklin M, Shenker S, Stoica I. Spark: Cluster computing with working sets[J]. HotCloud 2010.
- [25] Yu H, Wang D. Research and implementation of massive health care data management and analysis based on Hadoop[J]. IEEE, 2012.
- [26] T liu, M Mrtonosl , Impala:a middleware system for managing autonomic, parallel sensor systems[J]. Acn Sigplan Notices.2003:14.
- [27] Li GJ, Cheng XQ. Research Status and Scientific Thinking of Big Data[J]. Bulletin of the Chinese Academy of Sciences, 2012, 27(6): 647.

作者简历

教育经历:

2016 年 9 月至 2018 年 6 月 硕士就读于浙江大学软件学院软件工程专业

2012 年 9 月至 2016 年 6 月 本科就读于东北大学软件学院软件工程专业

工作经历:

2017 年 2 月至 2017 年 12 月 网易 云音乐事业部 数据研发实习

2015 年 9 月至 2016 年 5 月 IBM 中国研究院 CRL 后台研发实习

攻读学位期间发表的论文和完成的工作简历:

2017 年 12 月至 2018 年 5 月 云音乐数据产品分析系统设计与实现及论文编写。

致谢

再次致谢，又到致谢，已从东北大学换成了浙江大学，恍若昨日。

二十载苦读，将是如何？心有戚戚。

首先谨以最真诚的敬意感谢我的导师。自入学，助我教我知我。

他以大家的气度，深邃的思想，广阔的视野而又豁达的人生观让我能时刻保持初心，不畏艰难，一步一步的走在这条道路上。

再次，感谢实验室 S310 的小伙伴，是你们陪伴我走完了难忘的浙大学习生活。一起学习，一起找实习，一起游戏，感谢祝兄，大卫兄，良师益友。

也要感谢我的室友，杨伟杰，缪为正，陈仪同学。生活中容忍帮助我，学业上指正我，得意时，与我尽欢。失意时，陪我共度。感谢你们在浙里的陪伴，不敢忘此情。

同时感谢云音乐中的同事，每当我遇到问题和迷惑，总会耐心指导帮助我。感谢飞哥，在我人生迷惑之时，帮我指明方向，教导我如何思考和成长。感谢军正，志毅，有显给我讲解项目，帮我完成一个又一个任务。谢谢你们。

最后，我要感谢我的父母与女朋友，在我学习和生活中，给我安慰和鼓励，是你们给了我最大的感情支持。

路漫漫其修远兮，吾将上下而求索。

诚惶诚恐之际，惟寄此情于心，谨拜文以闻。

吕云鹏

于浙江大学软件学院

2018 年 3 月