中图分类号：TP3

论 文 编 号 ：10006ZY1121225

北京航空航天大學

专业硕士学位论文

基于 Hadoop/Hive 的海量数据 ETL 平台的设计与实现

作 者 姓 名　　张 永 曦

学 科 专 业　　软件工程

指 导 教 师　　康 一 梅

培 养 院 系　　软 件 学 院

The Design and Implementation of Hadoop/Hive-based platform for Massive Data ETL

A Dissertation Submitted for the Degree of Master

Candidate：Zhang Yongxi

Supervisor：Kang Yimei

School of Software

Beihang University， Beijing， China

中图分类号：TP3

论文编号：10006ZY1121225

硕 士 学 位 论 文

基于 Hadoop/Hive 的海量数据 ETL 平台的设计与实现

作者姓名 张永曦 申请学位级别 工程硕士

指导教师姓名 康一梅 职 称 副教授

学科专业 软件工程 研究方向 嵌入式软件

学习时间自 2011 年 9 月 1 日 起至 2013 年 12 月 17 日止

论文提交日期 2013 年 11 月 1 日 论文答辩日期 2013 年 12 月 17 日

学位授予单位 北京航空航天大学 学位授予日期 年 月 日

关于学位论文的独创性声明

本人郑重声明：所呈交的论文是本人在指导教师指导下独立进行研究工作所取得的成果，论文中有关资料和数据是实事求是的。尽我所知，除文中已经加以标注和致谢外，本论文不包含其它人已经发表或撰写的研究成果，也不包含本人或它人为获得北京航空航天大学或其它教育机构的学位或学历证书而使用过的材料。与我一同工作的同志对研究所做的任何贡献均已在论文中作出了明确的说明。

若有不实之处，本人愿意承担相关法律责任。

学位论文作者签名：_____ 日期：　　年　　月　　日

学位论文使用授权书

本人完全同意北京航空航天大学有权使用本学位论文（包括但不限于其印刷版和电子版），使用方式包括但不限于：保留学位论文，按规定向国家有关部门（机构）送交学位论文，以学术交流为目的赠送和交换学位论文，允许学位论文被查阅、借阅和复印，将学位论文的全部或部分内容编入有关数据库进行检索，采用影印、缩印或其它复制手段保存学位论文。

保密学位论文在解密后的使用授权同上。

学位论文作者签名：_____ 日期：　　年　　月　　日

指导教师签名：_____ 日期：　　年　　月　　日

摘　　要

随着互联网的飞速发展，互联网产生的 web 日志数据越积越多。对海量和异构的 web 日志数据进行整合，建立自己的数据仓库，从中获取更多有用的信息，是每个企业都在追求的目标。然而，在海量数据 ETL 方面，传统 ETL 工具在空间和计算效率方面都存在着明显的不足。Hadoop 分布式软件架构的出现，弥补了传统 ETL 工具的不足，Hadoop 内置的数据仓库工具 Hive 能够良好的支持对海量数据的 ETL。

本系统基于 Hadoop 以及 Hive，实现了海量数据的 ETL，通过 ETL 的三个主要步骤——数据获取、数据清洗和数据转化，将海量数据转换成星型模型并装入数据仓库，完成了企业的数据仓库的建设。同时，本系统采用工作流式任务调度器，实现了 ETL 作业的自动化运行。该工作流式任务调度器将 ETL 计算作业触发成一系列的带有偏序关系的子作业，为子作业分配执行资源，检查子作业间的依赖和约束关系并且监控子作业的执行，解决了 ETL 作业子作业之间的复杂调用关系，实现了 ETL 作业的自动化调度、运行资源的分配以及运行状态的监控。

本文首先将 Hadoop/Hive 方式与传统 ETL 工具进行对比分析，得出了其在海量数据 ETL 上的优势，然后在功能和非功能方面对系统进行了需求分析。在系统设计阶段，介绍了整个系统的框架设计，以及用户交互模块、调度模块和与 ETL 模块的详细设计。最后给出了系统的测试报告，验证系统已达到设计目标。

关键词：海量数据，Hive，Hadoop，工作流式调度器，ETL

Abstract

With the rapid development of the Internet, web log data generated by the Internet piled up. Building their own data warehouse, from which more useful information, through integration of massive and heterogeneous web log data, is every business in pursuit of the goal. However, in ETL of massive data, traditional ETL tools have obvious deficiencies in data storage and computational efficiency. Hadoop distributed software architecture appear to make up the shortage of traditional ETL tools, Hadoop built-in data warehouse Hive can be a good tool for ETL of massive data.

The system is based on Hadoop and Hive, realized ETL massive data, Through three main steps of the ETL - data acquisition, data cleansing and data conversion, converting massive data into star model and load it into data warehouses, completing the construction of the enterprise data warehouse. Meanwhile, the system uses the work-flow task scheduler to automate running of ETL jobs.Through triggering one ETL job to a series of sub-tasks with a partial order, allocating resources for sub-tasks, examining dependencies and constraints of the relationship between sub-tasks and monitoring the execution of the sub-tasks, Solves the complex relationship between sub-tasks, workflow scheduler achieves ETL job scheduling, allocation of resources and monitoring of the operational status.

Firstly, by comparing and analyzing the Hadoop/Hive with the traditional ETL tools ,this paper prove Hadoop/Hive superiority in massive data ETL calculated, then it makes needs analysis at both the functional and non-functional aspects of the system.In the system design phase, this paper describes in detail the design of the entire system design framework, and user interaction module, scheduling module and the ETL module. Finally, test reports verify that the system has reached the design target.

Key words: Mass data, Hive, Hadoop, Workflow, ETL

目　录

图清单

表清单

第一章 绪论

1.1 课题背景

伴随着互联网在全球的快速发展，互联网每天产生的数据也是越来越多，而且数据的特点很明显，海量、种类繁多，结构各异。互联网的这些海量数据大多是由用户行为所产生的，是用户行为的记录，这些数据对于商业来说，无疑是实实在在的宝藏，而且，对于社会学来说也有重大的意义[1]。这类海量数据，无疑是一座待开采的金矿，如今，能在这座金矿中提炼出实实在在的“金子”，是各个互联网公司追求的重要目标。目前，很多互联网公司都在引入各种技术，对海量互联网数据进行 ETL 并为之建立数据仓库，而最常见的数据源，就是无结构的 web 日志，这种 web 日志，就是用户访问之后所留下来的日志记录的汇总[2]，这些日志经过 ETL 过程，即数据的清洗、抽取、转换和装载之后，数据就进入了数据仓库，极大的方便了后期对数据的分析和挖掘等操作。通过这些建模后的数据的信息挖掘之后，我们可以得到很多重要的信息，比如每个用户的潜在行为模式，然后根据用户的这种行为模式，当用户访问网页时，便可以向用户推荐用户感兴趣的广告，以便提高网站的收入，同时，也实现了用户的便利化访问、访问速度的加快、以及网站服务的个性化。沃尔玛的啤酒与尿布的成功销售的案例，蕴藏在数据中的重大的商业价值被很多企业所接受，不光是互联网企业，更多传统行业的企业也希望把大量的现有的数据转换成有用的商业价值，以此来提高企业的产值和效益，提高客户的满意度和企业的竞争力。迫于企业数据的爆炸式增长，企业搜集用户数据和存储用户数据的能力越来越强，海量的堆积数据越来越多，对这些海量数据进行 ETL，即清洗、抽取、转化，最终装入数据仓库进行分析的需求越来越强烈。

Google 每天被处理的数据已经达到了 200PB，而百度每天被处理的数据也已经达到了 100PB，如此海量的数据依靠传统的单机加数据库的方式进行处理，单机的 CPU 和 I/O 已经成为了计算的瓶颈；一些利用数据仓库提供的联机分析来解决大数据的挖掘计算，例如 oracle 的 rac，对数据进行切片、上卷和下卷等分析，这种分析对数据具有强烈的依赖性，这些数据仓库中的数据都是 ETL 之后的数据，ETL 是数据分析和挖掘的首要条件，没有经过 ETL 过程产生的数据模型是无用的，用传统的方式进行数据的 ETL[3]，会有如下的问题：

（1）对于存储空间，目前的单机方式或数据库方式以无法承受。

（2）在处理速度上，传统方式已经无法满足效率的要求。

传统方式的数据处理的效率之所以低下，真正的原因在于数据处理程序在单机上是串行运行的，要提高数据处理执行效率，就要提高整个运行部分的并行度[4]。用分布式模型就可以解决这个问题，分布式模型能把一个计算任务分配给不同的机器，让这些协同完成计算任务，这样便可以实现海量数据的处理。

ETL 过程的数据清理、抽取和转换使用 Hadoop 来完成，利用数据仓储工具 Hive 以及计算框架 Map/Reduce，成倍的提高的数据处理速度。另外，利用 HDFS（分布式文件系统），也解决了存储海量数据的问题，随着数据存储量的不断增大，HDFS 可以实现永久性扩展，这种扩展又可以实现对用户的透明。利用分布式方式来完成数据的 ETL 过程，以及成为了一种不可多得的好方法。

1.2 课题研究目的与意义

ETL 过程是数据挖掘的必要前提，数据挖掘的质量的关键在很大程度上取决于 ETL 过程。根据相关统计，在数据挖掘的工作中，对数据的预处理，即把数据从原始的数据源中转载到数据仓库的过程，占整个数据挖掘工作的 60%~70%工作量[5]，所以 ETL 过程对数据挖掘的重要性不言而喻。目前随着，互联网数据的快速增长，海量数据的日益积累以及对计算机的广泛应用，数据挖掘已经俨然成为一个产业，各方对数据挖掘的需求越来越旺盛，都想从海量的数据宝藏中发掘出真金白银。2005 年的数据挖掘时常份额已经达到了 15 亿美元，而且年平均增长 30%左右。在目前的各大互联网公司，用户行为 web 日志已经成为一个价值不可估量的宝藏，整个互联网用户的多为网络行为，都可以从用户行为 web 日志中分析处理中得出，由于用户的行为具有连贯性，用户将来的行为也会可以被预测出来，这些信息不近可以为互联网企业带来丰厚的利益，为公司的高层做决策时提供参考，同时还可以支持社会学和金融学方面的决策，为科学研究寻找突破口，从以上可以看出，数据挖掘对整个社会的意义重大。传统的 ETL 的工具，已经不能满足对海量数据经行 ETL 的需求，由于数据量的爆炸式增长[6]，数据的噪音也随着增加，面对，对于这种状况，传统的 ETL 工具在空间上和时间上出现了限制，使得 Web 日志数据的 ETL 过程变得困难重重，影响了对海量数据的数据挖掘的实现。分布式计算框架 Hadoop 的数据仓库工具 Hive、MapReduce 模型的出现，解决了传统 ETL 工具的这些不足，整个 ETL 过程，即对数据的清洗、抽取、装换过程，在海量数据上变成了可能。因此，在海量数据的 ETL 计算方面[7]，我们又迈出了一步。

1.3 国内外研究现状分析

1.3.1 国内外研究现状

- ETL 的概念

ETL 是 Etract-Transform-Load 的缩写，表示数据抽取、转换和装在到数据仓库的过程。数据清洗是去掉那些不需要的、多余的噪声数据，只能获取对后期分析有用的数据的过程；数据转换是指把经过清洗过后的数据，转变成数据仓库指定的目标数据结构，目前目标数据数据结构以星型模型、雪花模型和混合模型居多[8]，很多情况下，还把数据进行汇总；数据加载是把处理过后的数据一次性的全部或增量的在加载数据仓库中。ETL 在数据仓库系统中的位置处在原始数据或数据库与数据仓库之间。ETL 同时还关系着数据的质量，它处在整个商务智能解决方案全过程的至始至终，它完成了整个系统的数据处理和计算任务的调度。

- ETL 的重要性

在对数据的分析和挖掘中，人们总希望能够随时能够访问到所需要的数据信息，这就要求一个能被用来存储一定格式的内部和外部数据的体系结构，例如用户行为数据、历史数据和现行数据以及外部关系数据。由于源数据的来源不同，它们大量、无格式、十分分散且无结构，所以它们不能被装载到数据仓库中[9]，而后期的对数据的挖掘和分析等一系列操作，都必须在一个数据清洁、数据结构清晰的数据仓库上来完成，这就必须使用 ETL 来完成，使得数据仓库能够高质量的数据。总体来说，ETL 的重要性有以下三点：

1）解决数据不集中的问题

2）清洁数据的作用

3）方便的构筑了便于分析和挖掘的数据中心

- ETL 现阶段的发展现状

随着信息产业的不断发展，数据的爆炸式增长，越来越多的企业着手都建立了自己的数据仓库，它们希望通过对已有数据的分析和挖掘，能够发现用户的行为特点和习性，以便为客户提供更好的服务，产生更客观的经济效益。与此同时，一些开始提供专业的、一整套的数据仓库解决方案，包括 ETL 工具等，还有一些企业也在着手提供数据仓库的解决方案的一部分或者某个特定的功能，使用者可以结合第三方企业提供的其它产品一起使用，来进行数据的 ETL。

目前在市场上主要流行的 ETL 工具有两大类，第一类是有比较的完善的体系结构的专业的 ETL 厂商的产品，这类产品价格昂贵，但是产品的功能强大、支持复杂的业务逻辑、性能也很稳定，但是它们不能与其它的数据仓库解决方案相融合，只能独立使用。这类产品的有以下几种：Informatic、dataStageXE、Sagent Solution、Ascential。第二类是整体数据仓库供应商，它们除了提供 ETL 工具之外，还提供数据仓库展现、存储和设计工具，这类产品一般对本企业相关的厂商的产品都有很好的支持，而且能在数据处理方面发挥出最大的效率，但是这种封闭的结构对其它的厂商的产品的支持有限，它的兼容性不强，没有很好的开放性和扩展性，而且，支持的数据源的种类也非常少。这类产品的典型代表有：InfoMove、Oracle Warehouse Builder、IBM Warehouse Manager、Informix 等。目前在我国，对 ETL 工具的研究并不多，现在市面上已经存在的 ETL 工具有 BeeLoad 和 Data Integrator 等，还没有一个成熟的、完善的 ETL 工具，应用在数据仓库的系统中。

基于以上的 ETL 工具的种种不足，目前，利用 Hadoop/Hive 的方式来实现 ETL 过程的方式油然而生：

1）Hadoop 的分布式处理架构 MapReduce：

MapReduce 是 google 提出的编程模型，它专为大数据处理而生，能够很好的支持海量数据的运算。它的实现原理是将计算任务提到 Hadoop 集群中去，然后计算任务会被 Hadoop 自动分成若干个快，每个块只处理整个数据中的一部分，这样便实现数据的并行计算，提高了海量数据的计算效率。在 ETL 过程中，直接应用 MapReduce 程序来实现数据的清洗非常合适，对于无结果的 web 日志数据，它能起到很好的数据清洗效果，对提高数据的质量很有用处；

2）Hive 数据仓库架构

Hive 是在 Hadoop 基础上实现的一个数据仓储架构，它可以说专门为大数据 ETL 过程而生，它能够方便的对存储在集群上的海量数据进行分析和查询。它提供的 HQL 是一种类 SQL 语言，这种类 SQL 的语言能够方便的对海量数据进行查询等操作，Hive 的实现原理是将 HQL 转化为相应的 MapReduce 任务来执行海量数据的处理。在 ETL 过程中，用 Hive 来实现数据的清洗和抽取非常适合[10]。

3）HDFS 分布式文件系统

HDFS 是 Hadoop 实现的一个分布式文件系统，它主要用于海量数据的存储，它是一个高度容错的系统，具有高吞吐的数据访问量，非常适合大规模数据的存储，它有着

传统文件存储系统无法比拟的优势。

1.3.2 对比分析

经过上述分析，我们得到传统的 ETL 工具与 Hadoop/Hive 差别，二者在是否共享、扩展性、单位运算速度、实现难度、成本和数据量级均有不同，对比传统的 ETL 工具和 Hadoop/Hive 的方式，我们得出表 1：

表 1 并行解决方案对比表

	传统 ETL 工具	Hadoop/Hive
是否共享	否	是
扩展性	差，一般只运行在数量有限的机器上	强，基于分布式系统架构，能够扩展到几万台机器执行
单位运算速度	高，但是严重依赖于计算机的硬件配置	一般，数据的同步会对单位运算速度造成一定影响
实现难度	低	高
数据源	支持类的数据源类型少	支持多种类型的数据源，
成本	需要购买	Hadoop/Hive 为开源项目，不需要购买
数据量级	小	很大

有表 1 中可以看到，Hadoop/Hive 方式跟传统的 ETL 工具相比，有着以下几点不同，在扩展性方面，传统 ETL 工具明显存在不足，一般只支持在数量有限的机器上运行，而 Hadoop/Hive 可以扩展到很多台机器；单位运算速度上，Hadoop/Hive 的表现一般，它不依赖与单机的性能取胜；在支持的数据源上，Hadoop/Hive 能够支持各种类型的数据源，这是传统 ETL 工具所不能比的[11]；在成本方面，传统的 ETL 工具需要购买，价格昂贵，而 Hadoop/Hive 为开源项目，无需任何费用；在数量级方面，传统 ETL 工具对海量数据的支持不理想，而 Hadoop/Hive 以 HDFS 为依托，能够支持较大量级的数据。

1.4 研究目标与研究内容

1.4.1 研究目标

本文的研究目意在使用分布式系统架构 Hadoop，以及 Hadoop 的数据仓库工具 Hive 来完成海量数据的 ETL；并以平台化的方式实现整个 ETL 过程的自动化运行，对于 ETL 的 3 个步骤—数据准备，数据清理及抽取，数据转换，每个步骤在平台中都对应一种类

型的任务。平台的核心是一个基于工作流的任务调度器，该调度器可以识别用户配置的ETL 作业，并按照既定规则，将作业触发成一个具有偏序的关系的一连串子作业，其中每个子作业与 ETL 作业中的一个步骤相对应，这些子作业形成一个有向无环图 DAG，共同完成一个 ETL 作业。同时，任务调度器为子作业合理地分配执行资源，全程监控其执行，直至整个 ETL 作业执行成功。

1.4.2 研究内容

本文的研究内容主要包括：

- 对系统业务需求的分析
- 系统的总体架构设计
- 系统各个模块子系统的详细设计，这之中重点的研究内容为：
  - 用户交互模块的设计
  - 核心调度器模块的设计；
  - ETL 模块的设计；
- 系统功能的测试与验证

1.5 论文组织结构

本片论文总共分为七章。

第一章绪论部分，主要介绍了 Hadoop/Hive 应用与海量数据 ETL 的研究背景及意义，并分析了国内外类似产品的发展现状，概述了本文的研究目标和研究内容，最后给出了本文的整体组织结构。

第二章需求分析部分，首先简单介绍了系统总体的业务需求，之后对系统的业务需求进行细分，划分为若干个主要的业务功能。

第三章关键问题与解决方案，首先简单介绍了整个系统所采用的技术—Hadoop 和 Hive，然后详细介绍了利用 Hadoop/Hive 方式实现海量数据 ETL 的解决方案。

第四章系统设计，首先介绍了系统的业务模型设计，平台的架构设计，然后介绍了平台的网络拓扑结构。接着介绍了系统的几个主要功能模块，提供基础服务的接口模块和通用模块，然后给出了系统的数据库设计。

第五章系统详细设计与实现，给出了系统中各个模块的详细设计，主要分为三个模块，用户交互模块、调度模块以及 ETL 模块。

第六章系统测试，通过单元测试、系统的功能性测试与系统非功能性测试相结合的测试方式，验证了测试系统功能的正确性。

总结与展望，对系统进行了总结并展望了系统未来的目标。

第二章 需求分析

2.1 需求概述

本课题的目的是设计出一个平台，该平台可以基于 Hadoop/Hive 自动化实现 ETL 过程，包括数据获取、数据清洗以及数据转换，平台应该实现海量数据 ETL 过程的作业化执行。平台应该有完善的权限管理机制，包括用户的账户权限、数据权限以及 ETL 作业权限的管理；平台能够对 ETL 作业的编辑提供良好的支持，包括 ETL 作业的增删改查，子作业的增删改查以及作业依赖的管理；平台能够对 ETL 作业进行有效的控制，例如 ETL 作业的启动、杀死、运行状态查看等；平台应该有一个完善的调度系统，对于每一个 ETL 作业，都能够被良好的调度以及合理的分配执行资源；最重要的，平台能够基于 Hadoop/Hive，对 ETL 过程中的数据获取、数据清洗以及数据转换提供良好的支撑。

2.2 系统的功能性需求

根据以上系统需求的概述，本系统至少应该具有以下业务功能，分别是权限管理功能、作业管理功能、作业操作功能、数据 ETL 功能、作业调度功能。

■ 权限管理功能用例图如图 1 所示：

权限管理主要分为三个方面，账户权限管理、数据权限管理以及作业权限管理，图 2 是权限管理的用例图：

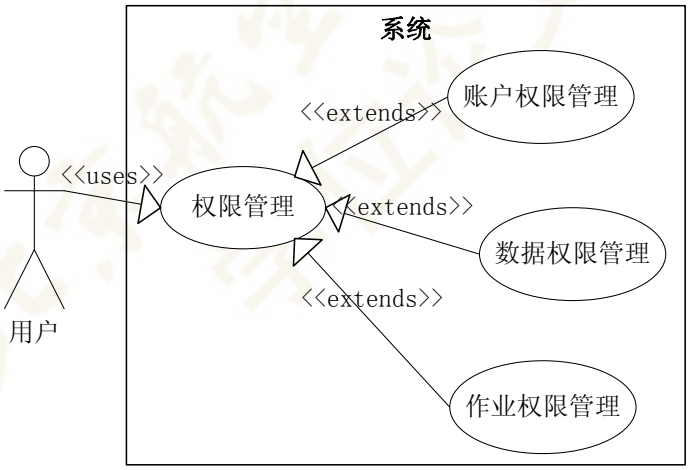

图 1 权限管理用例图

账户权限管理：用户可以修改在平台中的用户名以及密码，系统对用户的登陆权限

要进行验证；

数据权限管理：用户对自己的数据有着读写权限，并可以申请其它数据的权限，以及把自己的数据权限赋给他人；

作业权限管理：用户对自己的 ETL 作业有着增删改查的权限，同时还可以申请它人的作业权限，并把自己的作业权限赋给他人；

■ 作业管理功能用例图如图 2 所示：

作业管理包括三个部分，依赖管理、作业管理以及子作业管理，图 3 是作业管理的用例图：

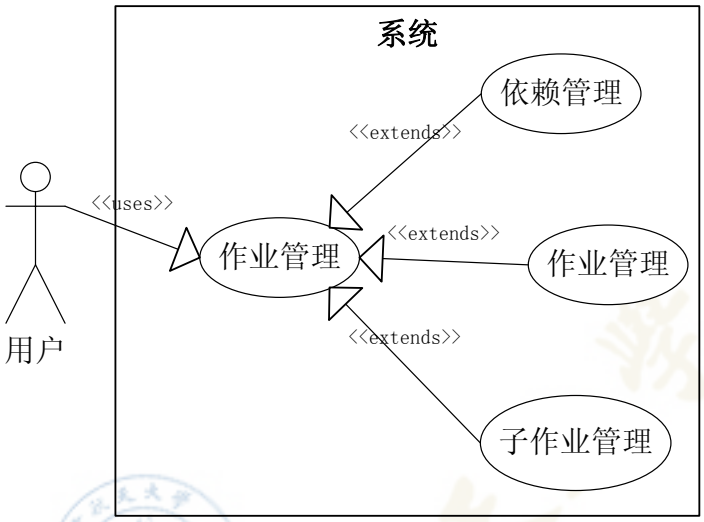

图 2 作业管理用例图

作业管理：用户可以建立自己的 ETL 作业，对 ETL 作业进行编辑，查看有权限的 ETL 的作业，并且可以对 ETL 作业进行删除；

子作业管理：对于每一个 ETL 作业，都包含着若干个子作业，这若干个子作业之间存在着偏序关系，并分别完成 ETL 过程中不同阶段的数据计算任务。用户可以子作业进行增加、删除、修改和查看，并且还可以指定子作业与 ETL 作业的从属关系；

依赖管理：作业依赖实现了子作业之间的偏序关系，它是 ETL 作业以工作流方式执行的关键。平台支持用户上传自定义的依赖脚本，对依赖进行增删改查操作，并指定依赖与子作业间的关系；

■ 作业操作功能用例图如图 3 所示：

对于已经编辑完成的 ETL 作业，可以对其对其如下几种操作，启动 ETL 作业，修改运行中 ETL 作业优先级，杀死 ETL 作业，并查看 ETL 作业运行状态：

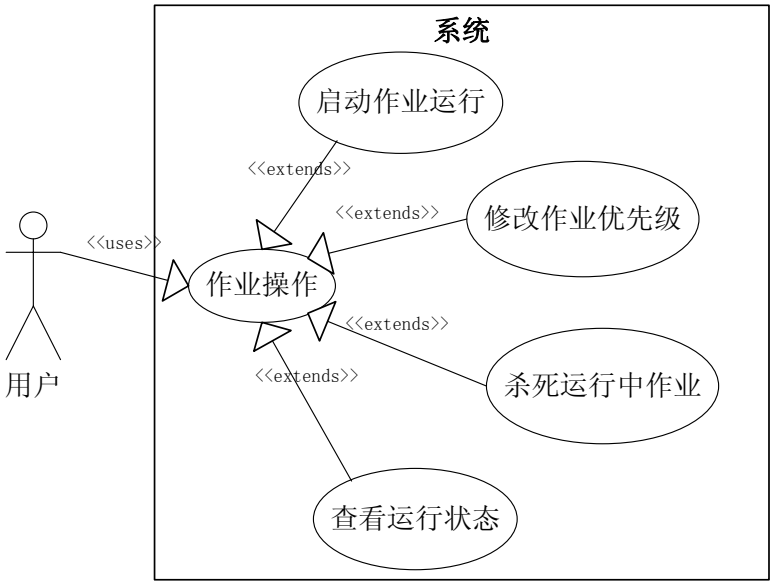

图 3 作业操作用例图

启动作业运行：用户启动一个 ETL 作业，系统会对该作业实施调度并执行；

修改作业优先级：用户修改一个正在运行中的作业的优先级，会改变度器为该 ETL 作业分配执行资源的策略；

杀死运行中作业：用户杀死一个正在运行的 ETL 作业，则调度器停止对改作业的调度执行，数据的 ETL 计算终止；

查看运行状态：用户可以查看一个 ETL 作业的状态，若是该作业正在运行，则按照偏序关系显示所有的子作业的运行状况；

■ 数据 ETL 功能用例图如图 4 所示：

平台最核心的功能便是将数据进行 ETL 计算，其中 ETL 主要包含三个步骤——数据获取，数据清理以及数据转换，每一个 ETL 作业中可包含以上三种类型的子作业：

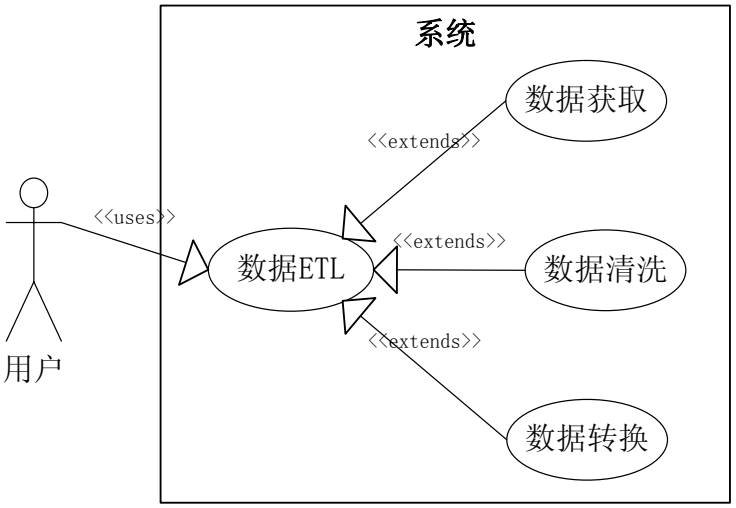

图 4 数据 ETL 用例图

数据获取：用户可以获取想要分析的数据；

数据清洗：用户可以对已有的数据进行清洗以及抽取，以便利用 Hive 进行数据转换；

数据转换：用户可以将已经被清洗过的数据转换成星型模型，载入数据仓库，方便后期进行联机多为分析等操作；

■ 作业调度功能用例图如图 5 所示：

ETL 作业的自动化执行所依托的是平台对 ETL 作业的强大的调度功能，目前调度功能应分包含 4 个部分，作业触发、子作业资源分配、子作业状态转换以及子作业的执行：

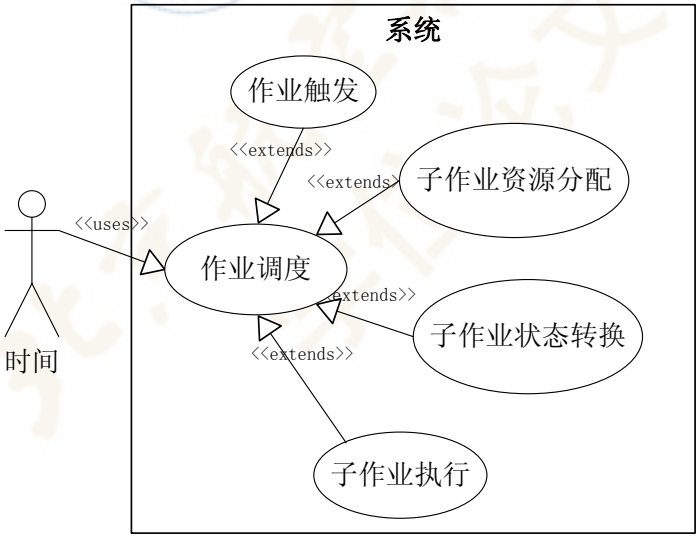

图 5 作业调度用例图

作业触发：平台需要根据用户的配置，将 ETL 作业触发成为一系列的带有偏序关

系的子作业，每个子作业的代表着数据 ETL 计算的不同阶段，这些子作业共同完成了数据的 ETL 计算；

子作业资源分配：平台需要为每个即将执行的子作业分配执行资源，分配资源要根据合理的算法，为了应对资源不足的情况，应该有合理的子作业排队机制；

子作业状态转换：ETL 作业触发出来的一系列子作业是有偏序关系的，即先后执行顺序，上一段阶段的子作业执行完毕，下一阶段的子作业才能由未调度执行状态转换为调度执行状态，系统应该能够实现这种子作业状态的转换；

子作业执行：系统能够实现子作业的执行，能够根据不同的子作业类型为子作业选择合适的执行资源，对子作业的执行状态进行监控，并对执行资源的情况作出反馈；

2.3 非功能性系统需求

■ 异常恢复

异常恢复要求系统能够在网络异常、机器异常的情况下，系统能够适应这种异常，并作出相应应对措施[12]。

1）调度机器应该有主机和备机，若是主机出现宕机的情况下，调度服务能够自动切换到备机上。

2）执行机出现异常的情况下，调度系统应能够发现执行机的异常，并把这台执行机置为不可用。

■ 可靠性需求

系统在高并发、高负荷的状况下仍然能够保证正常运行。

■ 可扩展性需求

系统应具有可扩展性，对于执行机资源，能够适量的添加，而对系统的稳定性不会造成什么影响。

2.4 本章小结

本章对平台的需求做了详细的分析。平台的主要功能是基于 Hadoop/Hive 实现数据获取、数据清洗和数据转换，并利用强大的调度系统使其自动化执行。本章结合用例图，对系统的需求进行了详尽的描述与归纳，将系统的需求分为 5 大部分，权限管理、作业管理、作业操作、数据 ETL、作业调度。最后提出了系统的非功能性要求，需要系统在异常恢复、可靠性需求、可扩展性需求上达到一定的要求。

第三章 关键问题及解决方案

由第二章需求分析得知，本系统的主要目的在于实现海量数据的 ETL，那么，如何实现海量数据的 ETL 便成了整个系统的关键所在。前文已经提到，ETL 过程的主要分为数据获取、数据清洗以及数据转换这三个阶段，所以海量数据的 ETL 问题便转换成了如何实现海量数据的获取、海量数据清洗以及海量数据转换这三个问题。

3.1 关键技术

3.1.1 Hadoop

Hadoop 是一个由 Apache 开发的一个分布式系统架构。它能够在用户不了解底层细节的情况下开发分布式程序。它能够抽分布式集群的进行大规模的数据计算和存储。分布式文件系统（Hadoop Distributed File System），是由 Hadoop 实现的[13]，简称 HDFS。HDFS 能够部署在价格低廉的硬件上，而且还具有高容错的特点。对于具有海量数据的应用程序，它能够实现高效率传输。同时，Hadoop 也实现了 MapReduce 编程模型，实现了大规模数据的运算。

- HDFS

HDFS 是基于块存储的文件系统，换而言之，对于 HDFS 上的每一个文件，都是被分割成多个大小固定的块进行存储的，这些快被存储在集群中不同的数据节点上(DataNode），文件的每一个块都是被随机存储到某一个数据节点上的[14]。

对用户来说 HDFS 就是一个分级的文件系统，用户可以实现数据的增加、删除、修改、移动等常规的文件操作。HDFS 是一个基于特定的节点构建的，这是由它自身的特点所决定。节点包括两大类，第一类是 NameNode，它存储着数据的元数据；还有一类是 DataNode，它提供存储块；由于 NameNode 只存在一个，所以 HDFS 有一个很大的缺点，就是单点失败。

文件的每个块都存储在不同的 NameNode 上，与传统的 RAID 架构不同，块的大小默认值为 64MB，以及文件的分块数都可由用户来决定。NameNode 控制所有文件操作。所有在 HDFS 内部的通信都基于 TCP/IP 协议[15]。

- MapRedue

将一个任务进行分割，让绝大部分工作交给 MapReduce，是 MapReduce 框架的基本思想。用户需要做的就是写两个函数：map()和 reduce()。

键值对分别是 map 函数的输出和 reduce 函数的输入。首先，系统将 HDFS 上的文件作为输入，Hadoop 将输入分成不同的块，分别交给不同的 map 函数进行处理。同时，map 函数将处理的结果分合成键相同的逻辑块，交给 reduce 函数进行处理。由于 Hadoop 将任务分配到了集群上不同的节点，使得 map 和 reduce 能够并行执行。

MapReduce 通过将一个大的任务分解成成几个独立的任务，是海量数据能够得到并行处理。

MapReduce 要并行的处理海量数据，就需要将一个任务划分到大量的不同机器中去。在这种模型中，不允许执行同一任务的不同组件共享数据，否则就会影响到模型的可扩展性。每当 map 函数产生新的键值对的时候，组件之间才会通信，此时，这些输出键值对将会交到下一个执行的阶段。

3.1.2 Hive

Hive 是一个数据仓库架构工具，是由 facebook 发展起来的，主要用于处理海量用户日志数据。目前，Hive 已经成为了 Hadoop 下的一个子项目。Hive 的主要目的是让用户可以用一种类 SQL 的语言 HQL 对存储在 HdFS 上的数据进行查询和分析，同时，Hive 还增加一些 MapReduce 的优化功能。Hadoop 和 Hive 的组合带来了我们使用分布式文件系统、MapReduce 和 SQL 的便利。在 Hadoop 上进行大数据处理，我们必须编写自己的 MapReduce 程序，对于一个比较简单的数据操作逻辑，MapReduce 程序的编写还比较简单。但是对于一些用户来说，它们的数据处理需求经常发生变化，它们得重新编写 MapReduce 程序，这是 Hadoop 处理非结构化数据的唯一方式。

而对于结构化的数据，正是 Hive 所擅长的。Hive 运行在 Hadoop 上，通过 HQL，可以实现对存储在 HDFS 上的海量数据的操作[16]。Hive 引入了许多数据库中类似的概念：如表、行、列和模式等。对于用户来说，用户不必去学习 MapReduce 编程，类 SQL 语言 HQL 就可以实现对数据的处理。此外，为了支持这个特性，数据必须以表的形式存储在 HDFS 上，表的元信息存储在一个关系型的数据库中，如 MySQL 和 Derby 等。

- Hive 的优势

Hive 的优势主要体现在如下几方面：

1)支持较多的数据类型，除了基本类型还有 List、Map、Struct。

2)Hive 中的 Thrift 服务器支持任务程序和 Hive 交互，并将其当做后端服务器。Thrift 支持语言装换，任何程序都可以和 Hadoop 通信。

3)对于结构复杂的数据，我们可以自己写 Derser 程序来解析。将数据存入 HDFS，表结构存入到元数据库。

4)Hive 支持 SQL 中的 Where、Join、Groupby 和 Orderby 等操作，同时我们还以把查询结果重定向一个新表中，同时，我们还可以添加 MapReduce 程序和 Hive 作为 HQL 的查询的一部分。

5)为了提高 Hive 的性能，Apaehe 还开发了优化器(Optimizers)。可以通过调整按照不同应用程序的不同需求，我们可以调整少量配置参数来提高 Hadoop 和 Hive 的性能。

- HiveQL

Criteria 查询对查询条件实现了面向对象的封装，更加符合编程人员的使用习惯，HQL 在在查询的基础上提供了进一步提供了更加灵活和丰富的查询，HQL 在覆盖了 Criteria 的所有查询功能，还提供了类似于 SQL 语句的查询方式：

Select/update/delete……from……where……groupby……having……orderby……asc/desc，由此可见，HQL 查询语句与 SQL 语句非常的相似，这一节我们专门对 HQL 操作进行讲解。

- 创建表

```
CREATE TABLE page_view(viewTime INT, userid BIGINT,
                       page_url STRING, referrer_url STRING)
PARTITION BY(dt STRING, country STRING)
ROW FORMAT DELIMITED
        FIELDS TERMINATED BY “\t”
STORED AS SEQUENCEFILE;
```

CREATE TABLE page_view(…)语句的作用是创建一个表，指明表明，列名以及列的类型。列的类型支持 INT、BIGINT、STRING 等常规类型。

PARTITION BY(…)语句指定该表的 Partition，Partition 的一般隐式的存在于表数据的 HDFS 目录上，而不是显示的存在于表的列中，但是，在查询语句中，Partition 与表的常规列没有任何区别。

ROW FORMAT…语句是十分重要的，它规定了表数据的存储格式，比如该例子中用的是”\t”，则在真实的数据每一个的不同列是以”\t”作为间隔的，若是不设置，则以 Ctrl-A 为间隔符。

- 导入数据

INSERT OVERWRITE TABLE page_view PARTITION(dt='2008-06-08', country='US')

SELECT pvs.viewTime, pvs.userid, pvs.page_url, pvs.referrer_url, null, null, pvs.ip

WHERE pvs.country = 'US';

这一操作的目的是想把 SELECT 查询出来的数据写入到表 page_view 当中，另外，也可以用 LOAD DATA LOCAL INPATH 语句进行数据导入。

- 函数、操作符

同 SQL 一样，HQL 支持丰富的函数和操作符。HQL 支持的函数有字符串函数、条件函数、日期函数、类型转换函数、聚集函数，而 HQL 支持的操作符有逻辑运算符、算术运算符、关系运算符。

3.2 海量数据存储以及获取

3.2.1 海量数据存储

根据目前的现状，对于每个大型互联网公司来说，每天积累的日志量是巨大的。对于一些重要的日志，每天单份日志的积累就能达到 1TB，这个数量级是十分庞大的，而这些重要的日志，其存储期限可能达到几个月之久，同时，为了保障数据的安全性，还要对每一份重要的日志进行备份，若是勉强备份一份，则该份日志的大小又将翻倍，试想，用普通的单机或者小型的集群来存储这些日志是不可行的。

为了解决海量数据存储的这一问题，我们引入了 Hadoop 分布式架构的 HDFS 分布式文件系统，HDFS 解决了海量数据的存储问题，并能作为存储介质以供 MapReduce 和 Hive 进行数据计算，HDFS 分布式文件系统有如下几点优势：

1）能够存储并处理超大文件

超大文件指的是数百 MB 乃至数百 PB 的数据，而对于 HDFS 来说，目前已经储存 PB 级别的数据了，而且 HDFS 还能够支持扩展；

2）海量数据高效率的访问效率

HDFS 的设计建立在更多地响应"一次写入、多次读写"任务的基础上。这意味着一个数据集一旦由数据源生成，就会被复制分发到不同的存储节点中，然后响应各种各样的数据分析任务请求。这种并行的数据访问方式能够获得较高的数据访问效率。

3）储存成本低廉

Hadoop 对硬件的要求比较低，所以 Hadoop 能够运行在价格低廉的设备上，而且

Hadoop 还有可靠的数据安全机制，保证了数据的完整性、健壮性；

3.2.2 海量数据获取

数据获取是 ETL 过程中的重要步骤，如何实现海量数据的获取，是实现海量数据 ETL 的前提。在本系统中，海量数据的获取是以批量获取的方式来实现的，而这种批量获取的实现，依靠的是 Hadoop 提供的 HDFS 操作命令，命令如表 2 所示：

表 2 HDFS 操作接口表

HDFS 操作命令	功能描述
fs -mv	将指定的文件移动到相应的位置
fs -rm	删除指定的文件
fs -put	从本地拷贝文件到 HDFS 上
fs –mkdir	创建文件目录
fs –cp	同一集群中，拷贝文件到目标位置
fs -discp	不同集群中文件的拷贝

本系统实现海量数据的获取是通过表 2 提供的命令，这里命令可以实现海量数据的批量操作，数据的获取主要分为以下三种情况：

1）若是获取线上服务器的数据，则先用 wget 命令把数据下载到本地，然后使用 fs –put 上传至集群指定位置；

2）若是同一集群之间的数据，则用 fs –cp 实现数据的获取；

3）若是不同集群之间的数据，则使用 fs –discp 实现数据的获取，同时，为了实现集群间的负载均衡，通过参数 mapred.job.map.capacity，distcp.map.speed.kb，mapred.job.priority 合理的调节数据拷贝作业的 map 个数、数据传输速度以及作业的优先级[17]；

3.3 海量数据的清洗

海量数据的清洗主要是指海量数据的清洗和抽取两个步骤，无结构的原始日志数据经过清洗之后，变成能被 Hive 元数据表所描述的数据；清洗过程是对清洗过后的数据进行多表聚合、多表连接、数据比较等操作，得到最终的 Hive 表数据，该表数据被用来进行数据转换；

3.3.1 MapReduce 完成海量数据的清洗

MapReduce 非常适合处理无结构海量数据，它支持几乎任何编程语言，能够实现复

杂的数据处理逻辑。在数据的清洗阶段，MapReduce 实现了对原始数据的清洗，原始数据的清洗过程如图 6 所示：

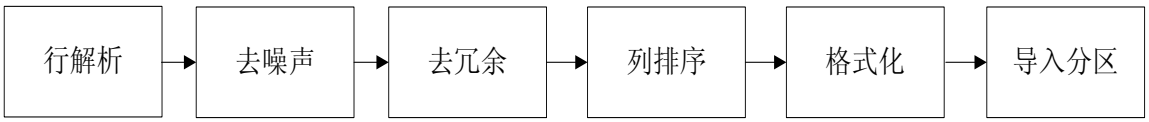

图 6 原始日志数据清洗流程图

1）行解析，数据是按照行来处理的，对于每一行，用户按照固有的逻辑，利用正则表达式将行进行解析；

2）去噪声，对于无法被解析的行，或者与预期偏差较大的行，要把这一行丢掉；

3）去冗余，已经解析完之后的行，根据需求，要去掉不需要的字段；

4）列排序，根据 Hive 表的列顺序，要对所有字段进行排序，使其符合 Hive 元数据表结构的定义；

5）格式化，把处理过后的每一行的所有列，用 Hive 约定的分隔符进行分隔，以便 Hive 对数据的识别；

6）导入分区，对于已经处理完后的每一列，导入 Hive 存储分区，Hive 只能对其分区内的数据进行操作；

以上过程是是由 MapReduce 程序来实现的，而程序的运行方式，是通过 Hadoop Streaming 的方式，该方式如下：

```
$HADOOP_HOME/bin/hadoop    jar $HADOOP_HOME/hadoop-streaming.jar \
    -input myInputDirs \
    -output myOutputDir \
    -mapper mapper.php \
    -reducer reducer.php
```

上段代码是一个简单的 Hadoop Streaming 提交命令，其中 Map 和 Reduce 程序分别由参数-mapper 与参数-reducer 指定，参数-input 与参数-output 分别指定整个 MapReduce 程序的输入数据和数据输出路径[18]；

3.3.2 Hive 完成数据的抽取

对于已经被清洗之后的数据，需要进行对数据的抽取工作，数据的抽取是 ETL 过程中非常重要的一个步骤，它把经过清洗之后的数据源，进行数据表聚合、多表连接、

数据比较等操作，最后把需要转换的数据由多张 Hive 数据表抽取到一张 Hive 数据表中。

ETL 过程中数据的抽取是由 Hive 实现的，对于结构化的数据，正是 Hive 所擅长的。Hive 运行在 Hadoop 上，通过 HQL，可以实现对存储在 HDFS 上的海量数据的操作。Hive 引入了许多数据库中类似的概念：如表、行、列和模式等，对于这些数据，Hive 处理起来是非常擅长的。

ETL 数据抽取的操作，主要有以下几种类型：

1）数据的关系操作包括选择、投影、连接、交和并，以及差等；

2）数据的聚集计算，包括 Sum、Count、Min、Max 等；

3）对单行数据的值域检查以及判断非空操作，以及对值域检查和函数转换等；

而对应 Hive 的 HQL 中，对 ETL 数据抽取的若干种操作类型，都能找到良好的支持：

1）HQL 支持对数据表的 Select…from 操作，支持对若干张数据表的 Join、Union all 等操作；

2）HQL 支持对结果集的 Sum、Count、Min、Max 等操作，并支持 Where 条件之后的 Group by、Order by 等操作；

3）HQL 如 SQL 一样，支持 Where 操作对数据的值域判断、值域检查等操作；

3.4 海量数据的转换

数据转换是 ETL 过程的重要步骤，海量数据经过获取、清洗之后，得到一份要加载到数据仓库的最终数据，这份最终数据是在 Hive 中以一张表的方式存在，利用 Hive 操作表数据，将数据转换成星型模型，并导入数据仓库。

3.4.1 星型模型

星型模型是一种由一点向外的呈辐射状的建模范例，它的中间有一个单一对象，沿半径向外延伸连接到多个对象。其中中间的对象叫做事实表，外部链接的多个对象叫做维度表。其中在事实表中，每一列存储的都是数字类型的键，而键对应的具体信息，则存储在外侧的维度表当中。对这个星型模型的查询从事实表开始，首先，对事实表进行查询，获取相关的维度表的指针，也就是事实表中存储的键的值，而事实表中的键对应的具体信息，则到相应的维度表表中去获取。对事实表和维度表的查联合在一起时，就可以检索到大量的信息。图 7 是一个星型模型的具体实例。

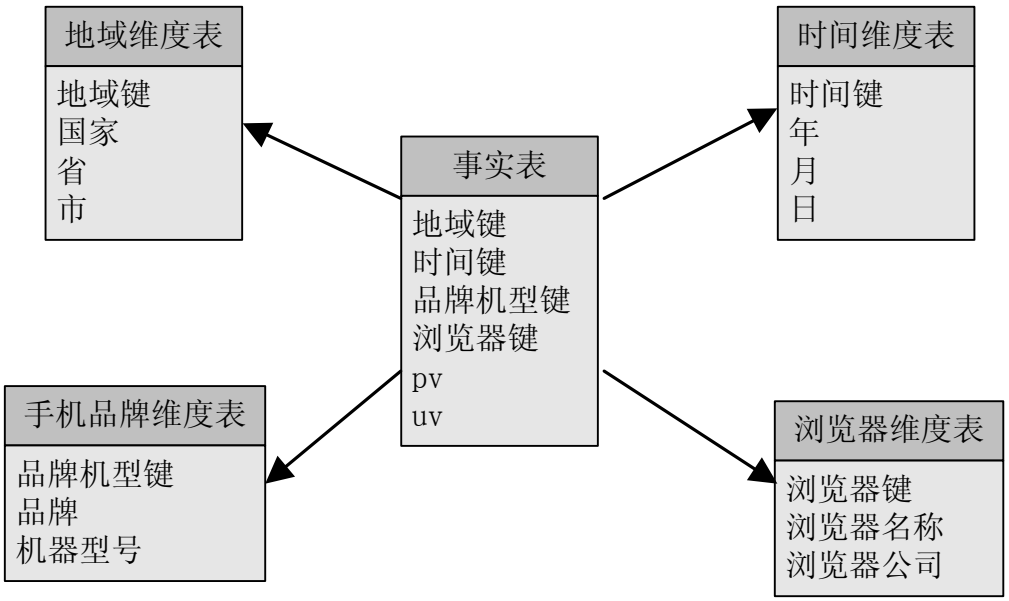

图 7 星型模型图

在图 7 我们可以看到一个星型模型的具体实例，这个星型表是用来记录不同时间、不同地域、不同品牌以及不同浏览器的手机访问某网站的 pv 和 uv。在图中，事实表有四个主要的字段，地域键、时间键、品牌机型键和浏览器键，这四个字段分别对应着地域维度表、时间维度表、手机品牌维度表以及浏览器维度表。对星形表的查询，是通过对事实表和维度表的联合查询进行的，例如，从事实表的获取的一条数据为“1,1,1,1,1000,100”，联合维度表，获取到的信息为“某省某市，在某年某月某日、某个手机品牌的某个机型，用某公司的某浏览器访问本公司的首页的 pv 和 uv”。

3.4.2 数据转换过程

数据转换的使命就是完成对已经被抽取和清洗的数据进行数据格式的转换，使其能够被装载到数据仓库中，以便后期对数据进行多维分析或者其它操作。

数据转换过程的读取的数据源表是前一阶段被抽取以及清洗过的数据，该数据的是以 Hive 表的存在的，它在 HDFS 上以能够被 Hive 识别的方式存储，即能够被 Hive 元数据所描述，并在能够使用 HQL 对数据进行处理。

数据原本存储在数据源表中，数据源表中包含着星型模型中所有维度表的列，经过数据转换后，原有的数据以星型模型的形式存储，简单分析得知，数据存储在星型中，要更节省存储空间，源数据表中的若干个字段在星型模型中的事实表中用一个键替代即可[19]。

3.4.3 Hive 实现海量数据的转换

在数据转换过程，有以下几个待解决的问题：

1）维度表的生成；

2）事实表与维度表的键值映射、事实表的生成；

3）载入数据仓库的实现；

以下是上述问题的解决方案，Hive 提供的 HQL 具有强大的数据操作功能，同时 Hive 对数据的存储和查询效率的都做到了良好的优化。

- 维度表的生成

维度表记载着维度信息，它是由数据源表的若干列组合在一起生成的，每个维度表都代表着一定维度信息，比如地域维度信息、时间维度信息信息等。

维度表的生成过程如下，利用 Hive 提供的 UDF 函数 Concat_ws (string SEP, string A, string B…)，将维度列从数据表中 Select 出来，然后利用 Concat_ws 函数将若干列合成一列，列之间用 Hive 事先约定好的列分隔符隔开（默认"\t"），形成一个单列临时表，然后对改单列进行 Distinct 操作，生成维度表数据。

- 事实表与维度表的键值映射、事实表的生成

当维度表生成之后，接下来生成事实表。根据星型模型的定义，事实表存储的都是维度表的主键键值，以及 PV、UV 值等信息，如何实现键值映射变成了关键问题。

这里利用的 Hive 提供的一个强大的功能，Map…Using…，其作用是可以将 Select 出来的数据作为标准输入，调用键值映射脚本（Hive 支持任何语言的脚本），该脚本读取已经生成的维度表，将每一行与维度表进行键值映射，映射后的行作为标准输出，写回到 Hive。

- 载入数据仓库的实现

把数据转换后生成的星型模型装载到数据仓库当中，是 ETL 的最后一个阶段。目前，本系统实现的数据仓库以 HDFS 为存储介质，以 Hive 为查询工具的。本系统中，转换后的数据载入数据仓库主要分为两个步骤，即维度表的载入与事实表的载入。因此，载入数据仓库的问题转化为事实表与维度表的存储问题。数据载入数据仓库是通过 Hive 提供的动态分区机制来实现的。

Hive 的存储是按照分区来进行的，每一个分区对应在 HDFS 上是一个独一无二的路径，Hive 的动态分区机制，实现了数据在 Hive 分区上的动态存储。实现数据的动态存储，需设置 Hive 的两个重要参数：

hive.exec.dynamic.partition.mode = nonstrict;

hive.exec.dynamic.partition = true;

数据转换后生成的星型模型，即维度表与事实表，被存储在一张 Hive 中。而维度表与事实表，被分别存储在该 Hive 表的不同分区中。首先，建立 Hive 时，利用 HQL 语句 Create Table…PARTITION BY (table_type STRING, table_name STRING)，为 Hive 表指定两个不同的分区列，这两个分区列分别是表类型以及表名称，表类型区分事实表还是维度表，而表名称表示事实表或维度表的名称，这两个分区列和在一起，唯一标识一张维度表或事实表。然后，在星型模型数据生成的过程中，通过指定分区方式将事实表数据和维度表数据动态插入到对应的分区中，在数据转换的同时，实现了数据仓库的载入。

3.5 本章小结

本章对首先对平台所用关键技术 Hadoop 和 Hive 做了一些简单的介绍。分别介绍了 Hadoop 与 Hive 的定义，以及 Hadoop 的 HDFS 和 Mapreduce，Hive 的优势以及 HQL 的几种简单操作。然后，对系统的关键问题，海量数据的存储以及获取，海量数据的清洗，海量数据的转换，给出了基于 Hadoop 以及 Hive 的解决方案。

第四章 系统设计

4.1 概述

系统设计是软件开发过程中的核心部分，系统设计的好坏直接关系到系统开发的成败。本章以第二章的需求分析为基础，给出了整个系统的设计方案。

4.2 平台的业务模型设计

由第二章需求分析可知，平台主要的功能是对海量数据进行 ETL，最终将海量的 web 日志数据转换成星型模型，装在到数据仓库，以便后期进行联机分析等操作。平台的定位为一个数据平台，图 8 是平台的业务模型图：

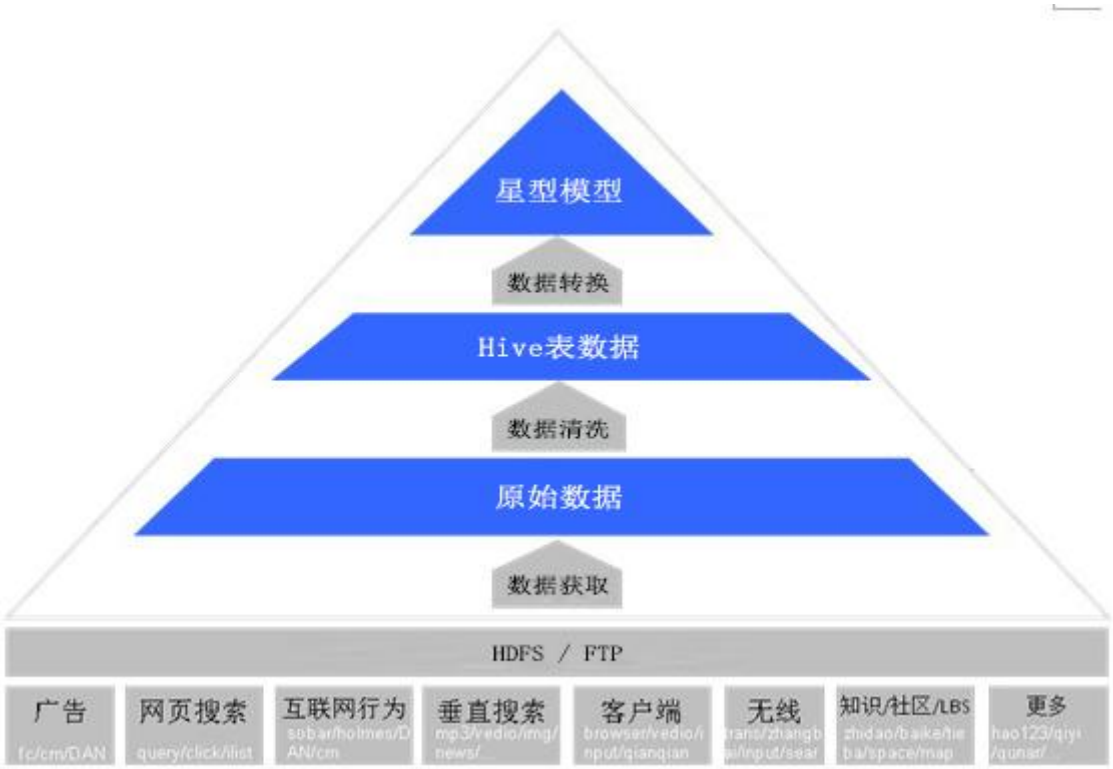

图 8 平台业务模型图

图 8 展示了平台的数据处理过程。首先，平台的数据源是存储在 HDFS 或者线上机器上的 web 日志，通过数据获取步骤，平台得到了待处理的原始数据；然后，对原始数据进行清洗以及抽取，把要转换的数据集中到一张 Hive 表当中；最后，对该 Hive 表数据进行转换，得到数据的星型模型，载入数据仓库。

4.3 系统架构设计

系统的功能主要有以下几大块：

1）一个采用工作流式是任务调度器，主要负责将用户编辑的 ETL 作业触发成一系列相互之间具有偏序关系的的子作业务，然后根据特定机制为这些子作业提供计算资源，以及相应类型的运行插件，并监控器执行，对异常情况进行通报；

2）在一个 ETL 作业中，根据 ETL 的阶段的不同，对应着不同的子作业类型，每种类型的子作业完成 ETL 过程某个阶段的数据处理任务，所以需要有不同的插件来负责不同类型的 ETL 子作业的执行；

3）需要一个交互系统来与用户交互，可以实现 ETL 作业以及子作业管理、作业运行时操作、子作业依赖管理、权限管理以及 Hive 的元数据管理；

在分层架构基本设计思想的指导下，系统按照层次进行划分，使得系统结构更为清晰，每个层次各施其职，降低系统模块间的耦合性，图 9 是系统的总体架构图：

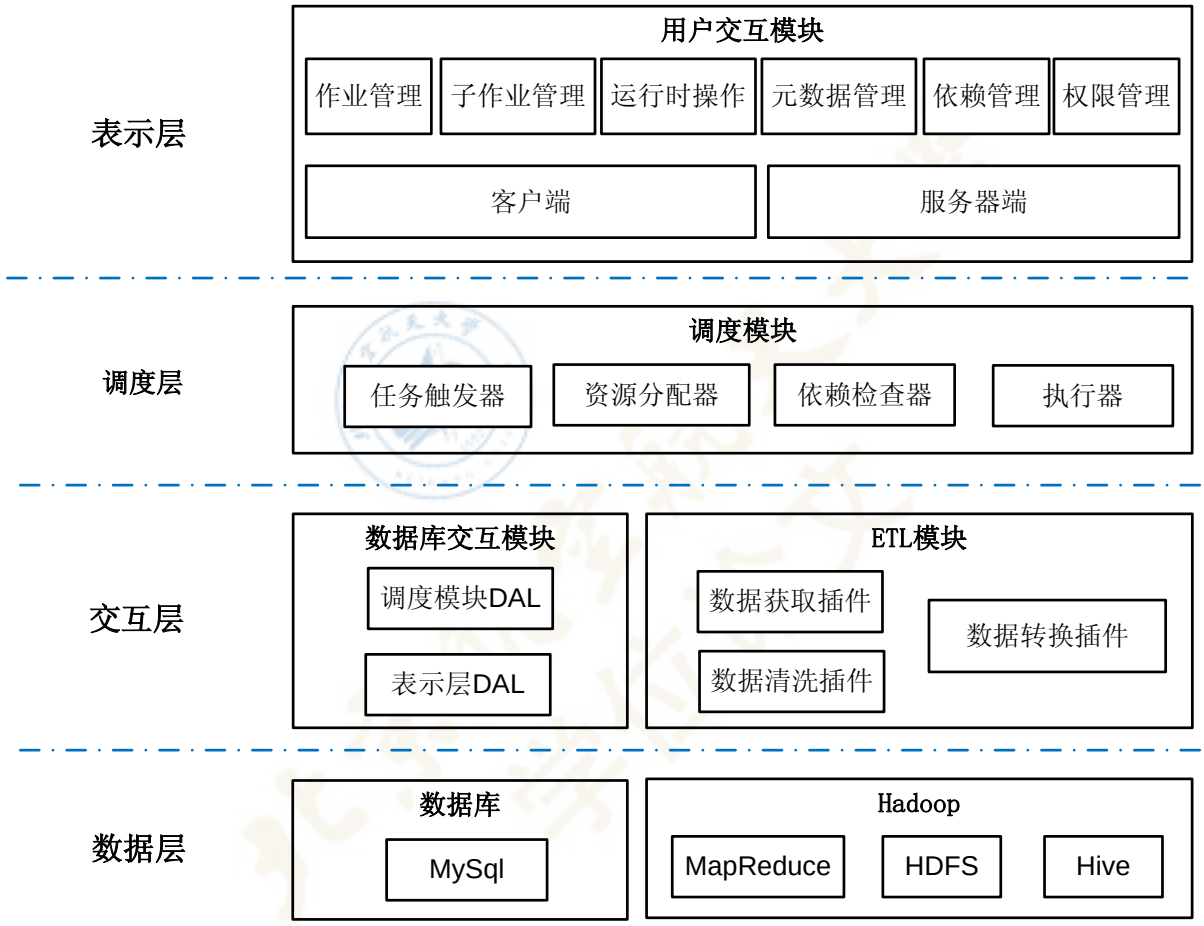

图 9 系统总体架构图

- 数据层

在数据层中，主要包括两大模块，Hadoop 以及关系型数据库 MySQL，二者的分工是这样的，平台相关的数据都存在 MySQL 中，比如与调度相关的作业描述信息，作业运行时信息等；而海量数据都存储在 Hadoop 的分布式文件系统 HDFS 中；

- 交互层

交互层在整个系统中起到非常重要的桥梁作用。在数据库交互模块中，包括两个模块，分别是调度器模块 DAL 和表示层 DAL，DAL 即 Data Access Layer——数据访问层，这两个模块提供了调度层和表示层访问数据库的接口方法；ETL 模块中，主要包括三个部分，数据获取插件，数据清洗插件，数据转换插件，对应着 ETL 中的三个步骤，它们通过调用 Hive 以及 MapReduce 的方式实现了海量数据的 ETL。

- 调度层

调度层是整个平台的核心部分，它对平台上运行的所有 ETL 作业起着调度作用，平台的 ETL 作业由它根据时间点或用户命令触发成一系列的 ETL 子作业，这一些列子作业是带有偏序关系的，每一个子作业都依赖于另一个子作业的产出，或者它的产出是触发下一个子作业运行的前提。若是子作业的运行条件被触发，接下来的就是进入待分配资源的状态，调度器会根据一定的原则，比如子作业的类型，优先级等属性，给它挂载到某个执行机资源的队列上，然后然后根据相应的算法，该子作业被取出，分配到执行机上去执行，执行机会根据子作业的类型，给子作业分配一个对应的运行插件，最终完成子作业的执行，子作业完成后，调度系统会检查是否有另外的子作业依赖这个子作业的产出，若是有的话，则启动那个子作业，若是没有，则整个 ETL 作业运行完成，把这个作业的状态置为成功。

- 表示层

表示层的用户交互模块，起着与用户的交互的关键作用。用户通过用户交互模块，实现了与系统的交互，用户可以通过客户端进行 ETL 作业以及子作业管理、作业运行时操作、子作业依赖管理、权限管理以及 Hive 的元数据管理。用户交互模块主要分为两部分，客户端与服务器端，整个用户交互模块式典型的 C/S 架构，客户端和服务器端通过 RCP 协议进行通信。

4.4 拓扑结构设计

拓扑结构设计主要描述了为应用软件提供所需运行环境的所有计算机硬件资源，其中主要包括硬件服务器、网络线路、网络设备等，我们知道硬件是基础，应用是关键。

一个合理有效的基础设施布局是支持相关应用高效运行的基础， 根据项目的建设要求，在构建系统基础设施层时，我们主要是以高性能 PC 服务器为主，再结合现有存储、安全、网络环境进行统一集成。其具体拓扑如图 10 所示。

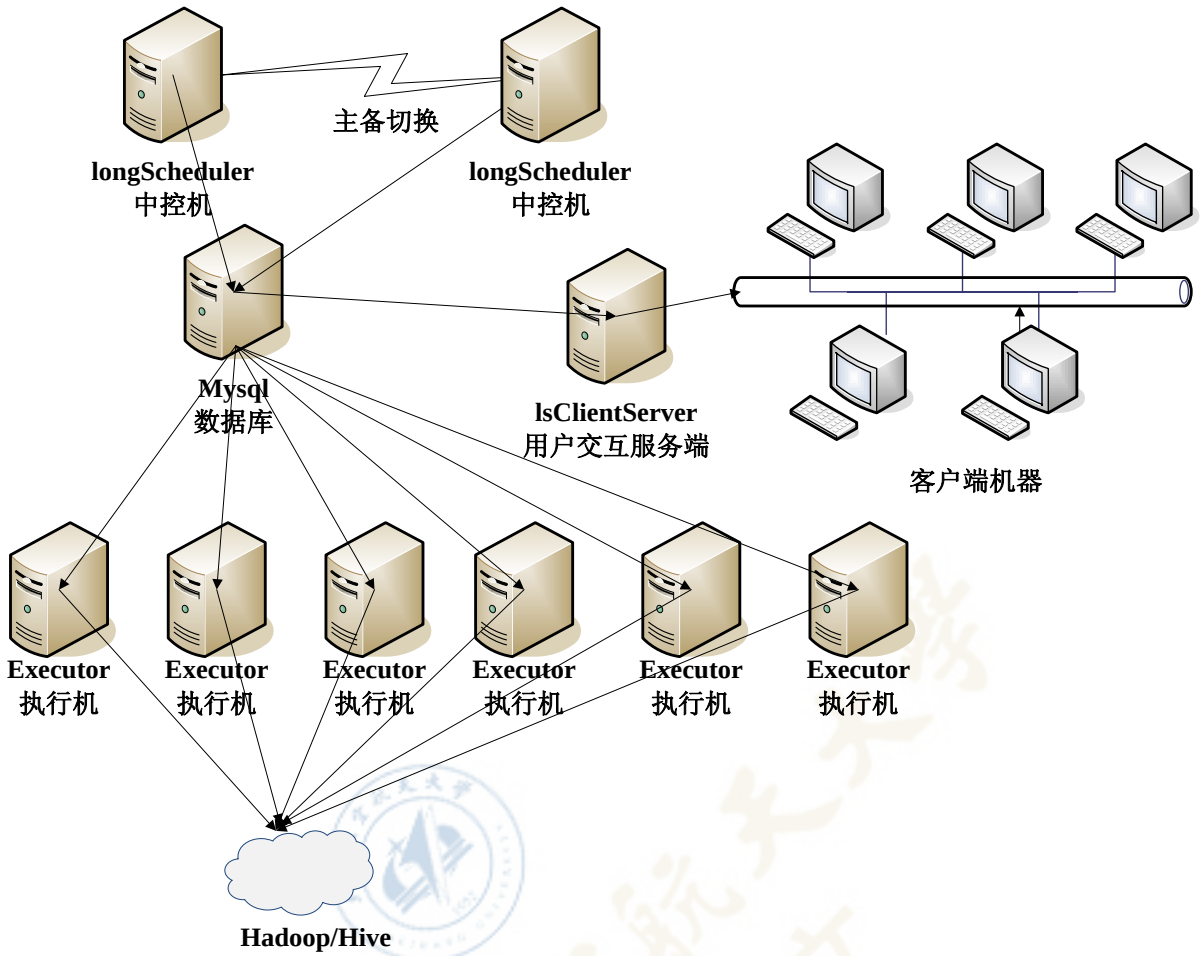

图 10 系统网络拓扑图

由上一节系统架构，设计出系统的网络拓扑图，在图 10 中主要包括以下几种机器：中控机、数据库机、执行机、分布式集群 Hadoop/Hive（由于集群机器众多，在图中用云朵形状表示），用户端交互模块的服务端，以及各种用户机器。为了方便描述，跟系统总体架构设计对应，拓扑结构图仍按照系统总体架构的分层结构叙述：

- 表示层

表示层的用户交互模块，主要包含用户端和服务端，在网络拓扑图中分别对应客户端机器和用户交互服务端机器。其中客户端模块程序分别装载在不同的客户端机器上，每一个客户都可以通过自己的机器来访问用户交互服务端系统，同时，用户交互服务端

机器与数据库机器相连，从而实现了用户对整个系统的访问。

- 调度层

调度层有两个模块，分别是调度模块和执行模块。

调度模块对应图中的两台中控机，任务触发器、资源分配器和依赖检查器都运行在这两台中控机的某台机器上，从图中可以看出，两台机器之间实现了主备切换，即具备了容灾措施，调度层是系统的核心，若是中控机出问题，则整个系统面临崩溃，主备切换则保证了若是其中一台宕机，则调度模块的所有相关进程都会在另一台中控机上实现重启。

执行器模块对应图中的数台执行机，每台执行机上都运行着一个执行器进程以及很多 Worker 进程。执行机对于 ETL 作业是重要的运行资源，只有被分配到执行机上，作业才会得到执行。

- 交互层

交互层中有两个模块，数据库交互模块以及 ETL 模块。

其中数据库交互模块中有两部分，调度模块 DAL 以及表示层 DAL，其中调度层 DAL 运行在中控机以及执行机上，而表示层 DAL 运行在用户交互服务端机器上，这些机器都与数据库机器相连。

ETL 模块有三个部分，数据获取插件、数据清洗插件和数据转换插件，这三个插件都运行执行机上，它们通过执行机的 Hadoop 客户端或 Hive 客户端与分布式集群相连，以实现对海量数据的 ETL 过程的控制。

- 数据层

数据层有两个模块，Hadoop 以及数据库。其中数据库用的是 MySQL，单独部署在数据库机服务器上，从图中可以看出数据库服务器是整个系统的核心，整个系统各个模块之间的交互都靠数据库服务器来完成。图中云朵形状表示的就是 Hadoop 集群，Hadoop 集群中除了 Hive 之外，还包括分布式计算框架 MapReduce 以及分布式文件系统 HDFS，海量数据的运算和存储都靠它来完成。

4.5 系统功能模块划分

由以上分析可知，可将系统分为以下几大主要模块，用户交互模块，调度模块和 ETL 模块，以下是几个模块的简单介绍。

4.5.1 用户交互模块

用户交互模块起着与用户交互的作用，用户通过此模块与系统进行交互，该模块分为以下两个部分组成：

- 客户端：用户通过命令行的方式，可以执进行 ETL 作业以及子作业管理、作业运行时操作、子作业依赖管理、权限管理以及 Hive 的元数据管理；
- 服务器端：服务器端运行在用户交互服务器上，不断的监听着来自用户端发来的请求，根据用户不同的请求，调用不同的业务模块，完成用户的请求；同时，服务端还对用户的账号权限和操作权限进行验证；

4.5.2 调度模块

调度模块包括任务触发器，资源分配器，依赖检查器和执行器，以下是调度模块各个组成部分的介绍：

- 任务触发器的作用主要是根据 ETL 作业的周期，把 ETL 作业按时触发成具有系列偏序关系的多个 ETL 子作业，这些偏序关系即用户来配置的子作业依赖，被触发出来的子作业，若是其依赖条件达到，便被分配到资源分配器中。
- 资源分配器的作用是接收来自任务触发器发送来的子作业，根据子作业的特性，比如子作业类型，或者子作业优先级，用一定的资源分配算法，把这个子作业挂到某个执行机资源的队列上去，若是某执行机出现资源空缺，则资源分配器根据相应的规则把其队列中的一个子作业推送到该执行机中。
- 依赖检查器的作用是促使一个 ETL 作业中各个子作业依照偏序关系执行，各个子作业依照偏序关系可形成一个有向无环图（DAG），其中两个相邻的子作业，当上游的子作业运行完成并产生结果之后，下游的子作业就会进入运行状态。依赖检查器会循环检查系统中每个没有运行的子作业，若是其上游的所有的子作业都已运行成功，则该子作业进入运行状态。
- 执行器运行在执行机上，它接受来自资源分配器的子作业，根据这个子作业的类型，分配相应类型的 Worker 以及执行插件；同时，执行器监还控着执行机的资源状况，以心跳的形式每隔 5 秒钟向资源分配器报告执行机的资源情况，使资源分配器能够合理的为子作业分配运行资源。

4.5.3 ETL 模块

ETL 模块是整个系统的重点，它通过与 Hadoop/Hive 的交互，完成了分布式集群对

海量数据的获取、清洗以及转换直至星型模型的建立，数据仓库的载入。ETL 模块主要分为三部分，数据获取插件，数据清洗插件，数据转换插件，以下是对几个插件的介绍：

- 数据获取插件：数据获取插件，顾名思义，它的作用就是用来获取原始数据。由于数据源被存储在不同的地方，或许是线上机器，或许是本集群的不同路径，或许是其它集群，而后续对数据进行清洗或者转换的时候，必须要求数据存储在指定集群的指定路径下，数据获取插件实现了这的作用就是把数据下载到指定的集群路径下；
- 数据清洗插件：数据清洗插件完成对了数据的清洗以及抽取工作，它利用 MapReduce 把原始的日志数据变成带有格式，即能够被 Hive 元数据所描述的数据；然后在此基础上利用 Hive，对数据进一步抽取，得到符合预期的数据以供后期数据转换之用；
- 数据转换插件：数据转化插件完成了对数据的转换工作，以及数据仓库的载入，它负责把已经被清洗和抽取的数据，按照一定的业务逻辑，转化为相应的目标数据结构，加载到数据仓库中，以供后期的商业多为分析等。目前系统支持的目标数据结构类型是星型模型；

4.5.4 系统的接口模块设计

除了以上系统主要的模块之外，系统还用到了其它辅助模块或者与外部接口模块，实现了系统内部模块之间的交互，以及对外部系统的调用。

Hadoop 客户端模块：Hadoop 客户端模块实现了系统对 Hadoop 分布式集群功能的调用。用过 Hadoop 客户端模块，系统实现了对 Hadoop 的操作，包括对 HDFS 分布式文件系统的操作，以及对 MapReduce 作业的操作。

数据访问模块：数据访问模块实现了对数据库的访问，它提供了对各种对数据库访问的接口函数，封装了调度器层和表示层对数据库的访问逻辑。

日志模块：系统运行中各种重要的信息需要以日志的形式来记录，以便系统出现故障时对系统问题的排查，以及维护人员对系统的运行状态的查看。根据日志的重要程度，日志级别可分为记录日志，调试日志，警告日志和错误日志。对于系统不同的事件，可采用不同级别的日志进行记录。

操作系统命令模块：在一些情况下，例如对操作系统文件的操作，以及对进程的操作等，需要调用操作系统的原生命令。本平台基于 linux 系统，操作系统命令模块对 linux

shell 做了一层封装，加上了操作失败多次重试机制，保证了平台调用的操作系统命令能够正确执行，从而平台的健壮性得到了提高。

4.6 数据库设计

数据库的设计需要制定相应的规则，这样才能很好的保证系统的正常运行和稳定。

1．表明和字段的命名规则：在系统中，表名和字段名都将依据业务的英文单词进行命名。长的英文单词采用单词的缩写，表名和字段名由多个单词组成的，每个单词的首字母要用大写。

2．字段的类型和长度：为保证本系统数据源的正确和可靠，依据字段的实际应用和类型适当的选用 MySQL 的数据类型。一般来讲，整数采用整型（int）类型，占用 1-2 个字节；小的文本数据采用 Varchar 类型，并设置相应的长度；日期类型的数据采用 Date 类型等等。

3．数据库在设计的时候要尽量减少数据冗余，减少重复的数据字段。数据冗余的字段数据占用过多的物理内存空间，而且会对具有多个表的一致性的操作带来问题。

4.6.1 数据库概念结构设计

基于需求分析对数据库的概念结构进行设计，可以从用户的需求中抽象出能充分并且真实的反映现实世界关系的概念模型。用实体关系模型——ER 模型来描述概念模型，它是描述现实世界概念结构的有效方法，是表示概念模型的一种直观方式，图中实体用矩形来时表示，实体的属性用椭圆形来表示，它们之间用没有箭头的直线相连，菱形表示实体之间的关系，并通过两条直线把两个实体相连接。

根据之前的需求分析和设计可知，用户可编辑一个 ETL 作业，ETL 作业中包含一系列的子作业，这些子作业的关系都是偏序关系，每一个子作业的都依赖于其上游子作业的产出，或者其本身的产出提供给下游使用。所以，一个 ETL 作业应该包括多个子作业，子作业中包含该子作业的各种信息，例如子作业的执行逻辑，子作业被被触发的条件，以及子作业的产出。子作业的产出对于本平台来说，可以关联到一个 Hive 的元数据，日志数据等。对于子作业以及子作业的产出，又分别包含了依赖与触发两种关系。同时，一个子作业生成出一个子任务由调度系统来进行调度并执行。由以上的分析，可得出调度系统的 E/R 图，如图 11 所示：

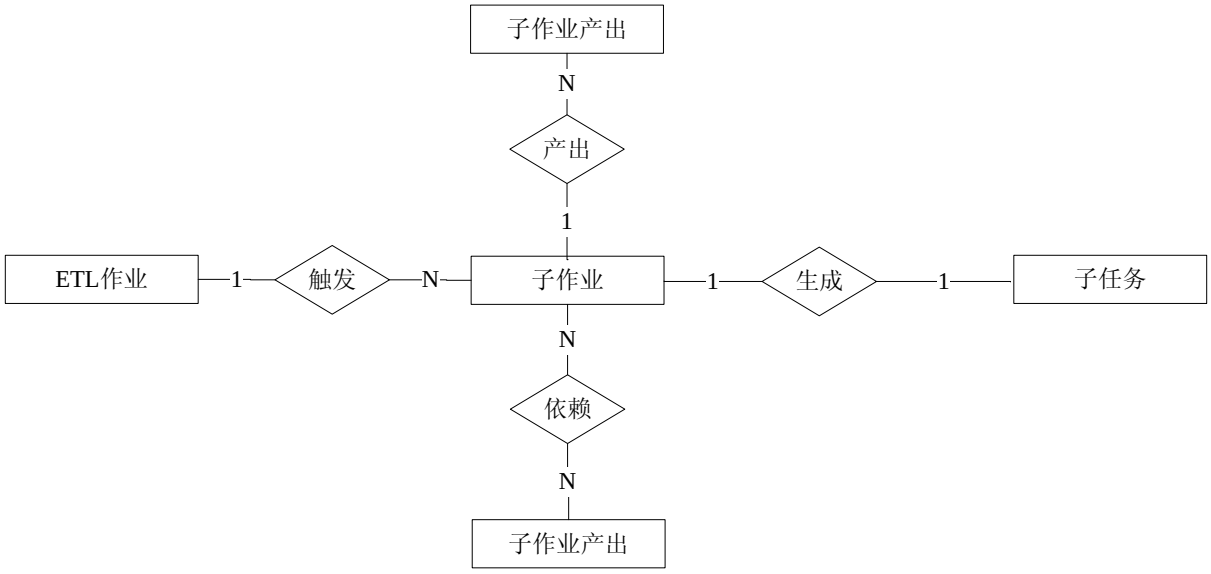

图 11 调度模块 E/R 图

ETL 作业和子作业的关系为触发关系，它们之间的比例关系是 1:N，一个 ETL 作业可以被触发成多个子作业，一个子作业只属于一个 ETL 作业，而一个 ETL 作业可以包含多个子作业；

子作业和子任务的关系为生成关系，它们之间的比例关系为 1:1，一个子作业可以实例化一个子任务，可以认为子作业是是子任务的描述，子任务是子作业的实例，是子作业的运行时描述信息，就像是类与对象之间的关系一样，一个子任务只属于一个子作业，而一个子作业只可产生一个子任务。

子作业和和子作业产出之间的关系有两种，依赖关系以及产出关系。在依赖关系中，它们之间的比例关系为 N:N，一个子作业可以依赖多个子作业产出，而一个子作业产出可以被多个子作业所依赖；在产出关系中，子作业与子作业产出的关系为 1:N，一个子作业可以有多个产出，而一个子作业产出只属于一个子作业。

4.6.2 数据库的表结构设计

根据上一节数据库的逻辑设计，可得出系统主要有以下 6 张表：ETL 作业表、子作业表、子作业表、子作业依赖表、子作业输出表、子作业产出表。

其中 ETL 作业表的结构如下：

表 3 ETL 作业表（TblETLJob）

字段名	类型	描述
id	int	ETL 作业 id，主键
name	string	ETL 作业名称
freq	int	周期，单位为分钟
priority	int	优先级
creator	string	创建者
modifier	string	修改者
createTime	datetime	创建时间
modifyTime	datetime	修改时间
lastSchuleTime	datetime	上一次被调度时间
warning_sms	string	短信报警人
warning_mail	string	邮件报警人

ETL 作业表记录每一个 ETL 作业的基本信息：

- freq 字段表示 ETL 作业的运行周期，是一个调度相关的字段，主要有 15、60、1440 四个值，表示 ETL 作业分别为刻钟级、小时级以及天级，例如刻钟级，表示每隔一刻钟，改作业被调度一次，以此类推；
- lastSchuleTime 指的是上一次被调度的时间，记载着 ETL 作业的调度相关信息；
- warning_sms、warning_mail 分别表示短信报警人和邮件报警人，若是 ETL 作业运行时出现异常情况时，则发送报警给相关责任人；

表 4 子作业表（TblJob）

字段名	类型	描述
id	int	子作业 id
etlId	int	ETL 作业 id，外键
name	string	子作业名称
category	int	子作业的类型
freq	int	周期，单位为分钟
status	int	状态
priority	int	优先级
jobVersion	int	子作业版本信息
creator	string	创建者
modifier	string	修改者
createTime	datetime	创建时间
modifyTime	datetime	修改时间
delayTime	string	延迟启动时间
deadline	string	完成时间阈值

子作业表记录着子作业的基本信息，以下介绍几个重要字段的作用：

- etlId 记录着 ETL 作业的 id，它属于外键，这个字段关联了子作业和 ETL 作业，建立了它们之间的从属关系；
- category 表示子作业的执行类型，比如数据获取作业，数据清洗作业，以及数据转换作业等，对应着交互层 ETL 模块的应该调用的执行插件，来完成 ETL 的不同阶段的任务；
- jobVersion，表示子作业的版本号
- deadline 表示子作业完成的时间截止点，是个绝对时间，一般从子作业的开始被调度算起；

表 5 子作业输出表（TblJobOutput）

字段名	类型	描述
jobid	int	子作业的 id，外键
version	int	对应的 version
resType	int	输出的类型
resourceId	int	输出的资源 id

子作业输出表记录着子作业的输出：

- jobid 字段为外键，它表示该输出归属的子作业；
- version 表示版本号，与子作业的版本号相对应；
- resType 表示子作业输出的资源类型，比如 Hive 元数据类型，表示该子作业的产出是 Hive 相关的数据；
- resourceId 指的是子作业输出的资源编号，每个 Hive 表在数据库中都会有一个编号；

表 6 子作业依赖表（TblJobDep）

字段名	类型	描述
jobid	int	基准子作业
version	int	子作业的 version
depType	int	依赖类型，依赖其它子作业/依赖 Hive 表
depRes	int	依赖的资源
depTime	string	依赖的时间范围，保持为(-1 day, 1day)的形式

子作业依赖表记录着子作业的依赖信息，也是两个子作业间的偏序关系，子作业除了可以其它子作业的产出，也是直接依赖其它子作业的状态信息，若是其它子作业完成，则该子作业直接被触发；

- jobid 字段表示被描述的子作业 id；
- version 表示该子作业的版本号；
- depType 表示该子作业的依赖类型，可是是其它子作业是否完成的状态，也可以是其它子作业的产出；
- depTime 表示被依赖的子作业或者产出的 partition 信息，例如依赖的是某个子作业的产出，（0day，1day）表示这个作业今天所有的产出；

表 7 子作业表（TblTask）

字段名	类型	描述
taskid	int	task 的 id
jobid	int	子作业的 id
jobVersion	int	子作业的版本
exeVersion	int	executor 的版本
productid	int	产品线 id
baseTime	datetime	关联时间
freq	int	子作业的运行周期
type	int	子作业的类型
state	int	任务状态
lmachineid	int	选定的执行机 id
priority	int	优先级
checkTime	datetime	最后一次刷新状态时间
startType	int	启动方式(0 for auto, 1 for manual)
retry	int	重试次数，首次执行为 0
depInfo	string	依赖信息，记录所有依赖的 job 或日志(json)
startTime	datetime	任务开始时间
dueTime	datetime	任务超时时间
endTime	datetime	任务结束时间

子作业表示记录着子作业的相关信息，它是子作业的运行时信息：

- jobid 字段表示被描述的子作业 id；
- baseTime 表示该子作业的时间 partition，它与 jobid 字段联合表示一个子作业在某个时间点产生的子作业；
- freq 表示对应的子作业的周期；
- state 表示子作业的运行状态机的当前状态，例如，子作业是否进入运行状态，看这个字段的值是否为 202；
- lmachineid 表示子作业被分配到到执行机 id，字段若是为空的话，则表示该子

作业目前还未被分配到执行机；

- startType 表示该子作业被触发的方式，是用户手动触发，还是有系统自动触发；
- depInfo 依赖信息，若是该子作业的还未完成的上游子作业信息；
- startTime、endTime 和表示该子作业的开始时间和完成时间；

表 8 子作业产出表（TblDataResouce）

字段名	类型	描述
resourceid	int	资源 id
englishName	string	资源英文名称
chineseName	string	资源中文名称
productid	int	产品线 id
genJobid	int	生成此数据的 jobid
description	string	备用的描述信息
resType	int	资源类型，依赖其它子作业/依赖 Hive 表
dataId	int	资源的 id
baseTime	dataTime	资源的基准时间

子作业产出表记录着子作业的产出，以下为重要字段的详细解释：

- resourceid 为资源 id，主键；
- genJobid 为该产出的所属的子作业 id，与具体的子作业相关；
- resType 表示该产出的资源类型，例如 3 代表 Hive 数据；
- dataId 是资源的标识，比如某个 Hive 元数据的编号；
- baseTime 表示该资源的 partition，例如和子作业的 baseTime 保持一致；

4.7 系统的接口模块设计

系统分为了几个重要的模块，包括调度模块、用户交互模块、执行模块和 ETL 模块等，这些模块之间存在着相互调用的关系，模块与模块之间相互独立，它们都调用接口与其它模块进行交互。同时，系统与外部系统的交互也需要通过调用接口才能完成，以下是各个接口的设计方案。

- Hadoop 客户端模块

Hadoop 客户端模块通过对 Hadoop 客户端的命令行调用实现了执行机与 Hadoop 的集群的交互，这种交互包括了对 HDFS 的交互和对 MapReduce 作业的操作，Hadoop 客户端模块实现了以下接口：

- fs_mkdir()接口函数实现了在 HDFS 上建立目录的操作；
- fs_chmod()接口函数实现了在 HDFS 上更改目录权限的操作；
- fs_mv()接口函数实现在 HDFS 上移动文件或目录的操作；
- fs_cp()接口函数实现在 HDFS 上拷贝文件的到指定目录的操作；
- fs_copyFromLoca()接口函数实现了把本地目录下的文件拷贝到 HDFS 上的操作；
- fs_ls()接口函数实现了查看 HDFS 文件的操作；
- fs_rm()接口函数实现了删除 HDFS 目录或者文件的操作；
- jar_streaming()接口函数实现了 Hadoop Streaming 程序的提交；

● 数据库访问模块

数据库访问模块是整个系统的桥梁，整个平台是以数据库为中心的，各个模块之间的信息交互都通过数据库来完成，数据库访问模块显得显得尤为重要。

数据访问模块主要分为以下几个重要的部分：

- 作业数据库访问；
- 子作业数据库访问；
- 子任务数据库访问；
- Hive 元数据数据库访问；

其中每个数据访问模块都需要有增删改查的接口，比如子作业数据库访问模块。增加一个新的子作业，设计接口函数为 addJob()；删除一个子作业，设计接口函数为 unsetJob()，把此子作业置为无效；修改一个子作业，设计接口为 updateJob()，根据新的配置修改子作业的属性；查询一个子作业，设计接口 getJobInfoByIdAndVersion()，根据子作业的 id 和 version 来获得一个子作业，接口 getJobBasicInfoById()，表示根据 id 来或者子作业的基本信息。

● 日志模块

日志服务是一个系统的基础服务，它提供了系统日志的分级记录，并且为日志提供了统一的格式、时间戳。系统运行中各种重要的信息需要以日志的形式来记录，以便系统出现故障时对系统问题的排查，维护人员对查看系统的运行状态。根据日志的重要程度，日志级别可分为记录日志，调试日志，警告日志和错误日志。

- 系统模块需要记录正常的日志信息，调用日志模块接口函数 info()；
- 系统模块需要记录调试的相关信息，调用日志模块接口函数 debug()；
- 系统模块需要记录警告日志信息，调用日志模块接口函数 warning()；

- 系统模块需要记录错误日志信息，调用日志模块接口函数 error()；

- 操作系统命令模块

平台运行在 linux 操作系统上，一些场景下会调用 linux 来实现某些必要的操作，比如对文件的操作，对进程的操作等，此时需要调用操作系统的原生命令。但是直接调用 linux 的原生命令对于系统来说是不安全的，因此需要对这些命令做一层封装。以下是操作系统命令模块最重要的几个接口函数：

- Execute()接口函数是这个模块最重要的函数，健壮性执行函数，若是这个函数执行的操作系统命令失败，该接口函数会隔一段时间再次重试，直到重试三次返回命令执行失败。
- Wget()接口函数实现了对 Wget 命令的调用，通过该命令，系统可以从远端机器上获取数据文件，这个函数通过调用 Execute()接口实现了 Wget 命令的健壮性执行。

4.8 本章小结

本章给出了系统的总体设计方案。首先是系统的业务模型设计；其次是系统总体架构设计，主要分为四个层次，表示层、调度层、交互层和数据层；然后介绍了系统的网络拓扑结构设计，以及系统的功能模块设计，系统的数据库设计，最后给出系统的接口模块设计。

第五章 系统详细设计与实现

本章具体讲系统主要模块的详细设计，包括用户交互模块的详细设计、调度模块详细设计以及 ETL 模块的详细设计。最后给出本章小结。

5.1 用户交互模块的设计

用户交互模块完成了用户与平台的交互，通过该模块，用户可以实现实现 ETL 作业以及子作业管理、作业运行时操作、子作业依赖管理、权限管理以及 Hive 的元数据管理等。用户交互模块被设计成经典的 C/S 架构，即客户端和服务器结构，客户端和通信是使用 RPC 协议实现的。用户交互模块被分为两个部分，客户端部分和服务端部分，客户端负责接收用户输入的命令，服务端负责接收客户端发送来的请求，处理接收到的请求并返回处理结果给客户端。

5.1.1 用户交互模块总体功能设计

用户交互模块实现了用户与整个平台的交互，用户可以通过用户交互模块完成以下几大功能：用户登录功能、ETL 作业管理功能、ETL 子作业管理功能、ETL 作业运行时操作功能、子作业依赖管理功能、Hive 元数据管理功能以及 ETL 作业权限管理功能。图 12 是用户交互模块的功能图：

图 12 用户交互模块功能图

用户登录：用户登录主要功能包括了验证用户账号权限、以及验证用户操作权限两部分功能，实现了平台用户权限的验证；

ETL 作业管理：ETL 作业管理以及 ETL 子作业管理分别实现对 ETL 作业和 ETL 子作业的增删改查，ETL 作业的数据处理逻辑在这里编辑；

ETL 作业运行时管理：用户可以启动 ETL 作业，并在 ETL 作业运行的过程中对 ETL 作业进行操作，例如改变 ETL 优先级、杀死 ETL 作业等；

子作业依赖管理：子作业的之间偏序关系是子作业依赖产生的，用户可以对子作业依赖进行编辑，进而实现 ETL 作业个子作业间偏序关系的管理；

Hive 元数据管理：用户可以对 Hive 的元数据进行增删改查操作，以此来实现对 Hive 对 HDFS 上数据的识别以及处理；

ETL 作业权限管理：用户可以申请别人的 ETL 作业权限，还可以把自己的 ETL 作业权限付给他人；

5.1.2 用户交互模块架构设计

出于对用户操作的响应速度、交互安全性、以及用户交互功能的丰富性上的考虑，用户交互模块采用传统的 C/S 架构，这正是利用了 C/S 架构响应速度快、安全性能容易保证以及客户端的功能强大的优势；同时，客户端和服务端的通信采用 RPC 远程调用协议实现，通过 RPC 协议，客户端程序可以直接调用运行在网络中另一端的服务器上的服务程序，进而达到了客户端与服务端的交互目的。一个完整的 RCP 模块主要包含以下三层：

（1）服务层(service)：服务层定义了具体的调用接口并实现了这些接口，系统的逻辑处理主要在这一层实现；

（2）协议层(protocol)：这一层主要定义了传输数据的协议，包括传输数据的格式以及编码方式；

（3）传输层(transport)：这一层主要实现了客户端与服务端的通信（例如 Socket）以及服务进程的类型（多线程、非阻塞等）；

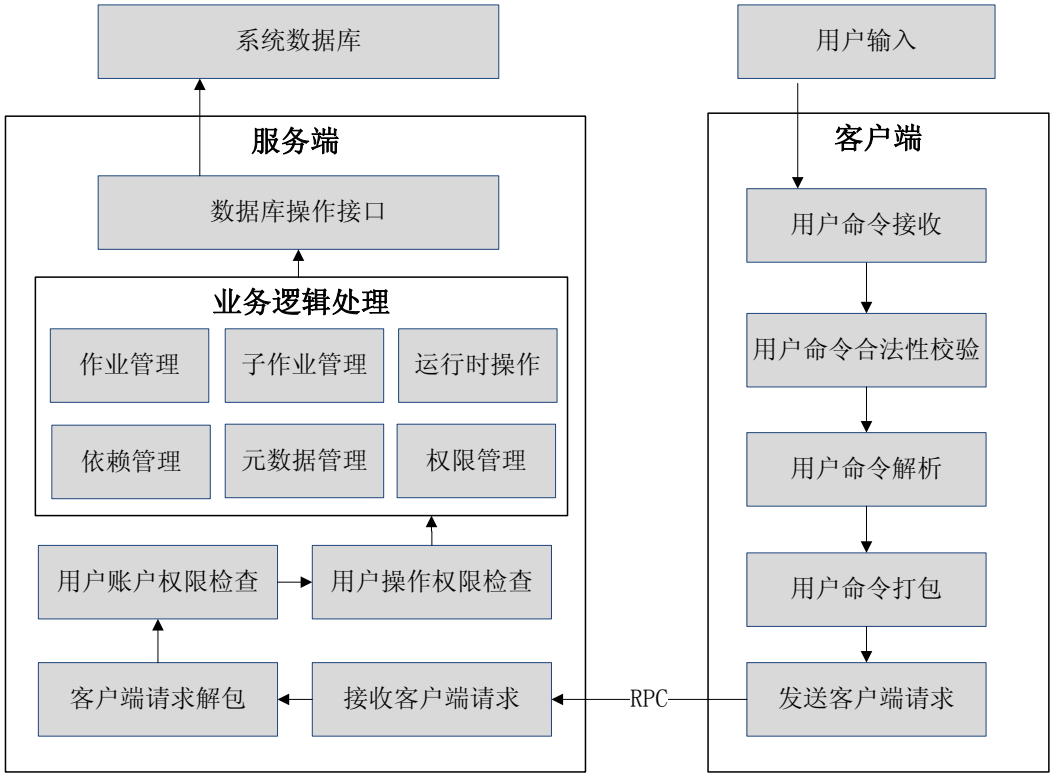

图 13 用户交互模块架构图

图 13 是用户交互模块的架构图，该图展示了用户交互模块的结构以及一个用户请求的过程。首先，用户在客户端输入命令，客户端接收用户命令，并对用户的命令进行合法性校验与解析，然后把解析后的用户命令进行打包，并发送到服务端；服务端接收客户端发来的请求之后，对请求解包，并进行用户的账户权限检查以及操作权限检查，检查通过后，对请求进行业务逻辑的处理，例如作业管理以及、依赖管理等，通过数据库操作接口把相应的信息写入数据库，系统便接收到了用户的请求。

整个基于 RPC 通信的模块是用开源的 Thrift 来实现的，Thrift 是一个基于静态代码生成的跨语言的 RCP 协议栈，它可以生成几乎任何编程语言的传输层和协议层的代码，用户便可以专心去实现业务逻辑部分的代码：服务器端的开发人员专心实现服务接口的定义以及实现，客户端开发人员只需调用客户端的接口为其服务。

在本模块中，传输层模块使用的是常见的 TSocket——阻塞式 I/O 进行传输；而在协议层上，为节约带宽，提高传输效率，模块的使用 TBinaryProtocol——二进制类型的传输协议；由于服务端使用 PHP 语言来实现的，PHP 是个仅支持单线程的语言，服务端类型使用的是 TSimpleServer——标准单线程服务端的阻塞式 I/O。

5.2 调度模块的设计

调度模块是整个平台中举足轻重的模块，它承担着平台数以万计的 ETL 作业的调度，调度包括 ETL 作业的分析，并把分析出的一系列子作业触发成子任务，根据相应算法为子任务分配执行机资源，监控作业的执行，直至最后整个 ETL 作业的成功完成。调度模块的重要性不可小觑，它的设计质量的好坏直接决定着整个平台的运行效率。

5.2.1 子作业状态机的设计

子任务是子作业的的执行实体，平台中调度模块的调度基本的对象就是子任务。子任务的产生过程是这样的，首先，平台解析一个 ETL 作业，找到这个 ETL 包含的所有子作业，然后是对子作业的实例化，把子作业都触发成一个子任务，该子任务继承了子作业的所有特性，包括子作业间的偏序关系，子作业的类型（数据获取、数据清洗以及数据转换），子作业的执行优先级等。为了便于调度器对子任务的有效调度以及监控，系统引入了子任务状态机的概念，子任务的状态标识着子任务在调度器中的不同阶段，目前，系统中的子任务的状态有七种，分别是阻塞状态、就绪状态、分发中状态、已分发状态、运行状态、完成状态以及失败状态[21]，下面对这些状态逐一描述：

- 阻塞状态

在子任务刚被生成的时候，它是阻塞状态的；若是它没有依赖上游任务的产出，则它会被转换为就绪态，否则一直维持阻塞状态，直到上游子作业的完成。

- 就绪状态：

就绪状态表示上游子任务已经运行完成；就绪状态表示子任务可以被分配资源并执行。

- 分发中状态

分发中状态是就绪状态的下一个状态，处于分发中状态的子任务，已经被发到一个执行机资源的队列上，等待该执行机空出运行资源，会从本执行机队列上根据一定算法抽出下一个要执行的子任务。

- 已分发状态

已分发状态表示子任务已经被分到了执行机资源，等待进入运行状态。

- 运行状态

运行状态表示该子任务正处在运行状态。

- 完成状态、失败状态

子任务运行的结果有以下几种，若是执行成功，则为完成状态；若是执行失败，则

为失败状态。

5.2.2 子作业调度流程设计

在上一章设计中讲到，系统的调度模块主要分为以下几个部分：任务触发器，资源分配器、依赖检查器、执行器；在上一小节也给出了子任务的状态机的设计方案。结合调度模块的结构和子任务状态机的设计，我们可以得出一个子任务完整的调度流程，即子任务在调度器中的生命周期；同时我们还能知道调度器的工作原理。

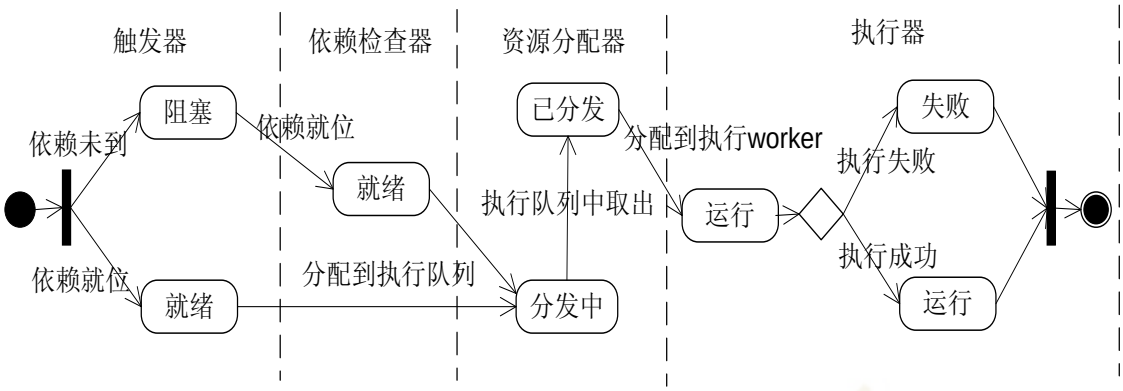

图 14 子任务运行时状态转换图

结合图 14，可以看出一个子任务完整的调度执行过程要依次经历任务触发器，资源分配器，执行器以及依赖检查器，它的具体流程如下：

- 任务触发器中流程：

1）首先，一个 ETL 作业被触发，任务触发器会在数据库中搜索得到若干个属于该 ETL 作业的子作业，然后分别这些子作业进行实例化；

2）根据子作业的配置，相应的子任务被触发出来，刚被出来触发出来的子任务，是阻塞状态的；

3）对于这个刚刚被触发出来的子任务，任务触发器会对其进行依赖检查，若是该子任务无上游依赖，或者上游依赖已经被解除，则该子任务的状态被置为就绪态；若是上游依赖仍然存在，则该子任务会被挂载数据库中的 depTree 表中，直至其依赖被解除；

- 资源分配器中流程

1）资源分配器会去不断的扫描数据库，状态为就绪的子任务会被资源分配器获取，做资源分配的处理；

2）资源分配器中存在多个队列，每个队列与一台执行机相对应，被获取到的就绪状态的子任务根据所属其子作业类型或者作业优先级等条件，被分配在一个执行机的队

列中；该子任务的状态变为分发中；

3）资源分配器会定期检查每个资源的队列，根据执行机的资源情况，用一定算法取出若干个队列中的子任务，把它们的状态置为已分发；

- 执行器中流程

1）在数据库中获取属于本执行机的，处于已分发状态的子任务；

2）执行器会把已经获取到的子任务分配各相应的 Worker 去执行，已经分配到 Worker 的子任务，状态改为运行；

3）根据子作业的运行情况，若子任务运行成功，则子任务的运行的状态被置为完成，否则被置为重试或者失败。

- 依赖检查器中流程

1）依赖检查器会定期扫描数据库中的 depTree 表，此表上挂着阻塞状态的子任务；

2）对于得到的子任务，依赖检查器会检查该子任务的依赖是否就位，若是没有就位，则不作任何处理，否则将其状态置为就绪态，等待资源分配器的调用。

5.2.3 任务触发器的设计

任务触发器是整个调度模块的重要组成部分，是整个调度流程的开端。它对 ETL 作业做了分析，或取出这个 ETL 作业的每一个从属子作业，然后逐个分析这些子作业的相互之间的依赖关系，把这些子作业实例化成逻辑上相互带有偏序关系的子任务，这些子任务便进入了调度系统开始接受调度。

- ETL 作业的触发原理

任务触发器是如何实现 ETL 作业的触发的呢？这里涉及到一个 ETL 作业周期的概念，ETL 作业的基本信息存储于数据库中，每个 ETL 作业表都有一个字段 Frequency，它代表了该 ETL 作业的周期，目前系统的所支持的周期有以下几种，刻钟级、小时级和天级，其中刻钟级表示该 ETL 作业每个一刻钟就需要被触发一次，宏观上说相关的 ETL 作业每十五分钟就要执行一次。ETl 作业的触发涉及到到两个重要的概念，调度记录日志表和 timeTree 哈希表。

- 调度记录日志表

调度记录日志表维持着一个平台自身的时钟，它的单位是分钟，在某一时刻，只有触发完这一分钟的所有 ETL 作业，平台的时钟才能进入下分钟。从提高存取效率方面的考虑，该表被存在数据库中。

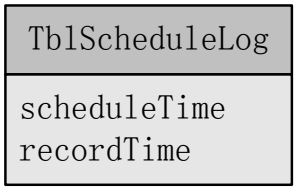

图 15 调度记录日志表图

该表有两个字段，其中 ScheduleTime 存储着当前的系统时间，而 recordTime 表示调度完成时间。

- timeTree 哈希表

timeTree 是个哈希表结构，它以 linux 时间戳做哈希，可以得到一个记载着 ETL 作业 id 的链表。每当平台自身的时钟走到当天的 23:59 时，timeTree 便被重置一次，第二天要运行的所有 ETL 作业装载到这个哈希表上。

通过调度记录日志表和 timeTree 哈希表，便可实现 ETL 作业的自动触发。每当平台的时钟超过上一次的调度时间一分钟时，系统便会用当前的整分钟时间的 linux 时间戳去到 timeTree 哈希表去做哈希，若能得到一个不为空的链表，则触发该链表上记载着的所有 ETL 作业。

- ETL 作业触发流程

把 ETL 作业触发成一系列的子任务，是任务触发器模块的一个主要功能，流程如下。首先从 timeTree 哈希表中获取要被触发的 ETL 作业的链表，然后对这些 ETL 作业做逐一的触发。根据其中一个 ETL 作业的 id，通过数据库获取其所有的子作业，然后检查逐一检查子作业当前时刻的任务是否被触发过（当前时间+子任务 Id 唯一），若是没被触发，则生成新的子任务。对新生的子任务，检查其是否有未就位的依赖，若没有，则把该子任务的状态改为就绪，若是有，则把该子任务挂载 depTree 上。图 16 是 ETL 作业触发流程图。

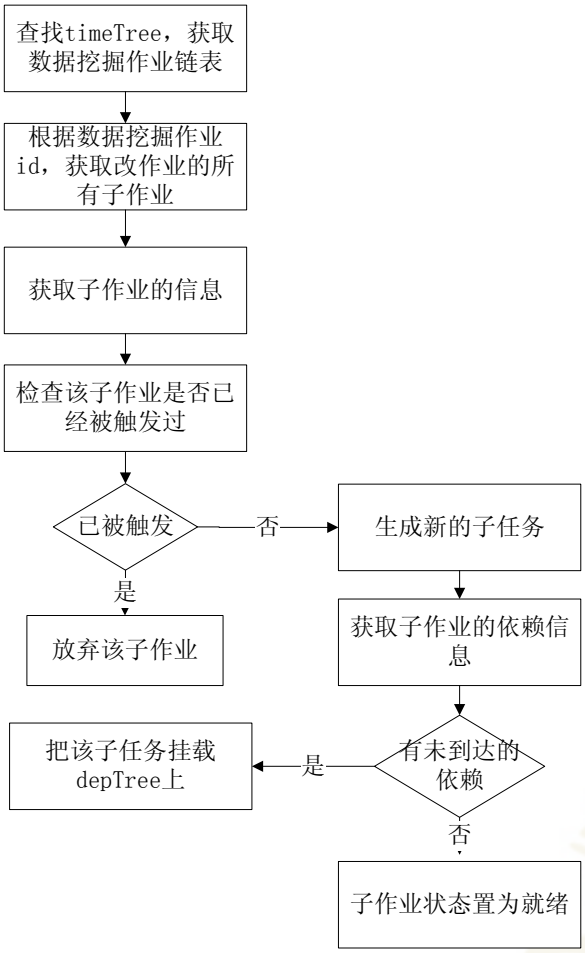

图 16 ETL 作业触发流程图

- 系统类图

图 17 是系统的类图，在图中罗列了几个主要的函数；

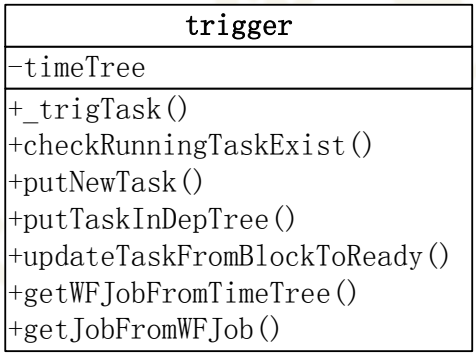

图 17 触发器类图

函数 getWFJobFromTimeTree()从 timeTree 中获取了 ETL 作业链表；函数 getJobFromWFJob()从 ETL 作业中获取了该作业的所有的子作业；函数 trigTask()实现了

子作业的实例化，把子作业触发成了子任务；函数 putNewTask()把新的子任务存入了数据库；函数 putTaskInDepTree()把依赖条件未到的子任务挂在了 depTree 上；函数 updateTaskFromBlockToReady()把子任务从阻塞状态变成了就绪状态。

5.2.4 资源分配器设计

资源分配器的主要功能是为子任务分配执行机资源。子任务在资源分配器中的状态主要有两种，分配中状态和已分配状态。分配中状态表示当前的子任务已被分到某个执行机的队列上，等待分配执行机资源；已分配状态表示子任务已经获得了执行机资源，等待在执行机的获取。

所以，资源分配器的设计主要分为两个方面：

1)）子任务选择执行机资源的策略，即当子任务有就绪变为分发中状态时，它要被插入到那个执行机资源队列；

2）执行机资源队列选择要执行的子任务的策略，即当子任务的状态由分发中状态变为已分发状态时所发生的改变；

- 执行机资源的获得

执行机资源状态的获得是一个重要的问题，它为子任务分配执行资源提供了重要的参考依据。由于整个平台是以数据库为核心的，所以执行机资源的状态应该有执行器写入到数据库中，然后资源分配器可以定时的读执行机的状态，然后更新资源分配器本身的资源分配策略，图 18 是执行机表图。

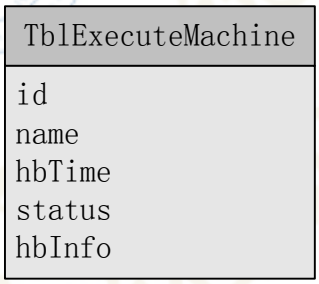

图 18 执行机表图

数据库表 TblExecuteMachine 是记载执行机资源信息的表，执行器负责对它的实时更新。表中有 5 个字段，id 表示唯一表示一台机器，name 表示机器名，hbTime 表示最新更新的时间，status 表示该机器是否有效，hbInfo 是个 json 串，这个 json 记载着该执行机资源的各种属性，比如该机器的 CPU 占用率，执行机的能够运行子任务的空余槽位数，和该机器的服务质量以及资源级别。

- 子任务选择执行机资源的策略

任务选择执行机的策略如下，主要有以下三个因素：

1）执行机资源的能够运行子任务的空余槽位数，子任务倾向于选择槽位数多的机器，这样有利于执行资源的负载均衡；

2）执行机资源的服务质量，子任务倾向于选择服务质量高的执行机资源，保证子任务健壮性执行；

3）执行机的服务效率，子任务倾向于选择服务效率高的执行机资源，保证子任务的执行效率；

- 执行机资源队列摘取子任务的策略

从执行机资源队列选择要执行的子任务，主要参考以下三个重要的因素：

1）子任务的紧急程度，紧急程度即 ETL 作业的 deadline，该属性由用户来配置，存在数据库中；

2）所属 ETL 作业的优先级，优先级高，则应优先执行；

3）子任务的等待时间，子任务等待的时间越长，则越应该及早得到执行；

- 子任务在资源分配器中的执行流程

首先，处于就绪状态的子任务被资源分发器模块获取，然后子任务的被分配到一个执行机资源队列上，子任务的状态变为分发中；资源分配器模快定时对所有执行机队列进行扫描，若是对应的执行机有空缺槽位，则若干个子任务会被分配到该执行机上，子任务的状态变为已分发。执行流程如图 19：

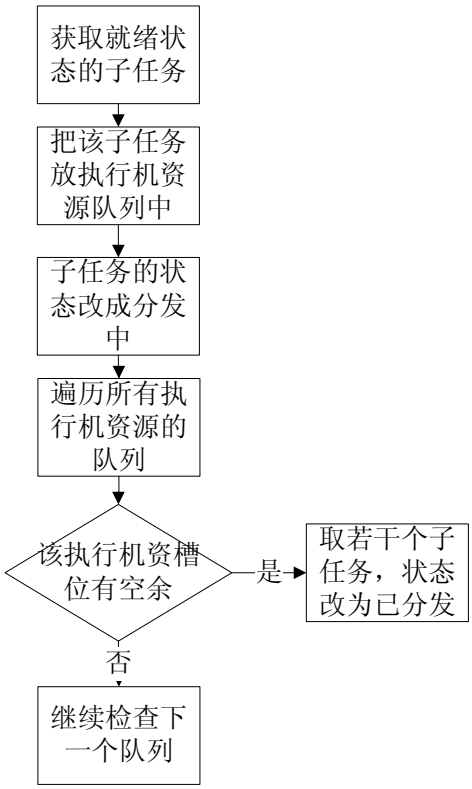

图 19 执行机资源分配流程图

- 资源分配模块类图

资源分配器中，主要包括三个大类，Dispatcher 类、ExecutorQueue 类和 WeightSort 类，其中 Dispatcher 类为资源分配器的主体，子任务的主要流程都在这个类中进行，ExecutorQueue 类为执行机队列，并封装了一些队列操作；WeightSort 类主要用来实现执行机资源队列选择要执行的子任务的策略，类图如下：

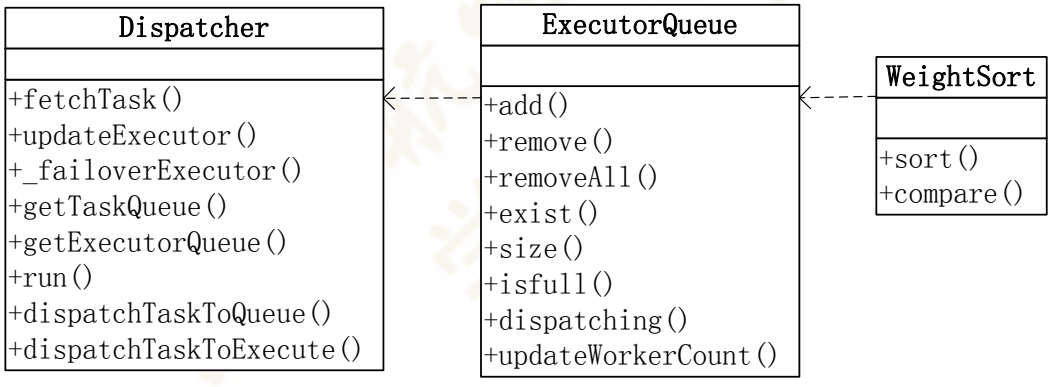

图 20 资源分配模块类图

Dispatcher 类中，函数 fetchTask()获取已经已经就绪的子任务，dispatchTaskToQueue()把子任务放入了执行机资源队列中，dispatchTaskToExecute()把子任务放在执行机中；

ExecutorQueue 类中，执行机资源队列的各种操作，比如 add、remove 操作等，其中 dispatching()函数负责调用 WeightSort 类，实现了执行机资源队列选择要执行的任务的策略。

5.2.5 依赖检查器的设计

依赖检查器的主要作用是检查系统目前正处于的阻塞状态的子任务，若是它们的上游依赖尚未就位，则不作任何操作，若是子任务此时依赖已经就位，则把它的状态从阻塞置为就绪状态，该子任务继续接受调度器的调度，进入分配资源的环节。

- depTree 表

阻塞子任务的信息应该被如何存储，依赖检查器应该如何去哪里查询当前系统中被阻塞的子任务呢？目前平台存储阻塞子任务信息的是靠数据库来完成的，阻塞子任务的都被存储在 depTree 表中，depTree 表如图 21 所示。

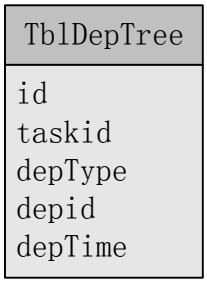

图 21 depTree 表图

depTree 表对应 taskid 记录着子任务的依赖信息，depType 表示该任务依赖的类型，或许是上游子任务的产出，比如一张 Hive 表数据，或许是上游子任务本身运行成功的标志，depid 和 depTime 表示了被依赖的资源 id 和资源的时间范围。

- 支持用户自定义依赖检查脚本的实现

某些时候，用户会对依赖检查的逻辑有一些更复杂的要求，因此，系统支持用户自定义自来脚本的检查。系统通过操作系统 Shell 命令调用脚本的方式，来启动用户自定义依赖检查脚本。每过一段时间，系统便会检查脚本运行状态以及返回值，判断依赖是否就位。

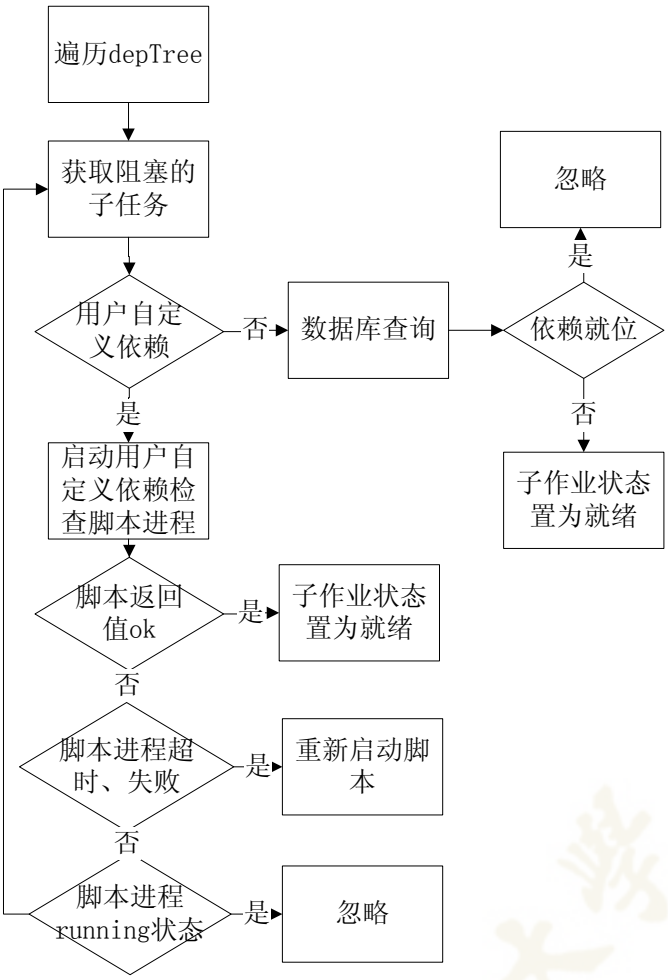

图 22 依赖检查流程图

首先，系统扫描 depTree，找到阻塞状态的子任务，分析器依赖的类型，若是普通依赖，则遍历数据库子任务产出表，查看依赖是否就绪；若是用户自定义依赖，则启动依赖检查脚本，并定时检查进程状态，若是进程返回值 ok，则依赖就位，进程失败、超时，重启进程，进程若是运行状态，则忽略。

- 依赖检查器类图

依赖检查器主要有三个类，依赖检查类（RelyChecker），进程池类（ProcessPool）和自定义依赖管理类（RelyManager），图 23 是依赖检查器的类图：

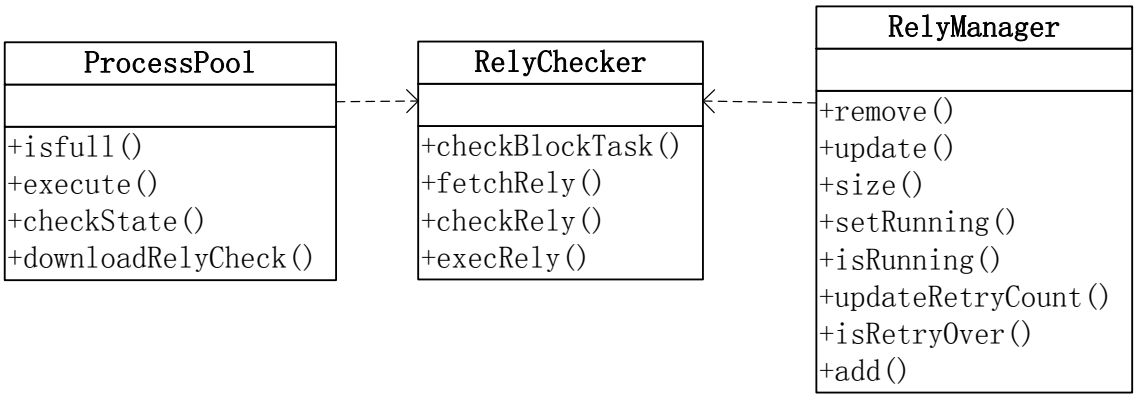

图 23 依赖检查器类图

依赖检查类中，checkBlockTask()函数主要负责阻塞子任务的依赖检查，余下三个函数负责自定义依赖的获取、检查和执行操作。

进程池 ProcessPool 类有几大功能，主要是自定依赖脚本的下载，自定义依赖脚本进程的执行以及进程状态的检查。

自定义依赖管理 RelyManager 类主要维护一个自定义依赖类型的子任务的队列，并带有该队列的增删改查操作。

5.2.6 执行器的设计

执行器运行在执行机上，它作用是接受资源分配器分配过来的子任务，并且为子任务分配一个合适的 Worker 加执行插件，启动这个 Worker 进程执行子任务。另外，执行器还有一个重要的功能，就是定期发送自身的心跳到资源分配器，让资源分配感知到它的存在，具体的实现方式不断更新 TblExecuteMachine 数据库表的 hbInfo，字段以及 hbTime 字段，其中 hbTime 字段是更新的时间，而 hbInfo 字段则表示执行机的最新状态，主要包括执行机的 CPU 占用率，剩余内存的空间，该执行机可以执行的子任务的类型，以及该执行机能够使用的执行子任务的槽位数，这个信息都记载在一个 json 串中，以下是该 json 串的实例：

```
{
    "resource":
    {
        "CPUInfo":{"core":"8","idle":"89.0"},
        "memInfo":{"free":13072},
        "netInfo":{"input":"0.00","output":"0.00"}},
```

```
"task":{"startType":0,"category":1},
"Worker":{"max":60}
}
```

在该串中，CPUInfo、memInfo、netInfo、task、Worker 分别代表了 CPU 占用率，内存空间，网络情况，可以执行子任务的类型以及剩余的资源槽位数。

- 执行器的工作流程

图 24 为执行器的流程图，该图中主要描述了执行器处理子任务、汇报心跳以及处理崩溃的 Worker 进程的相关流程。

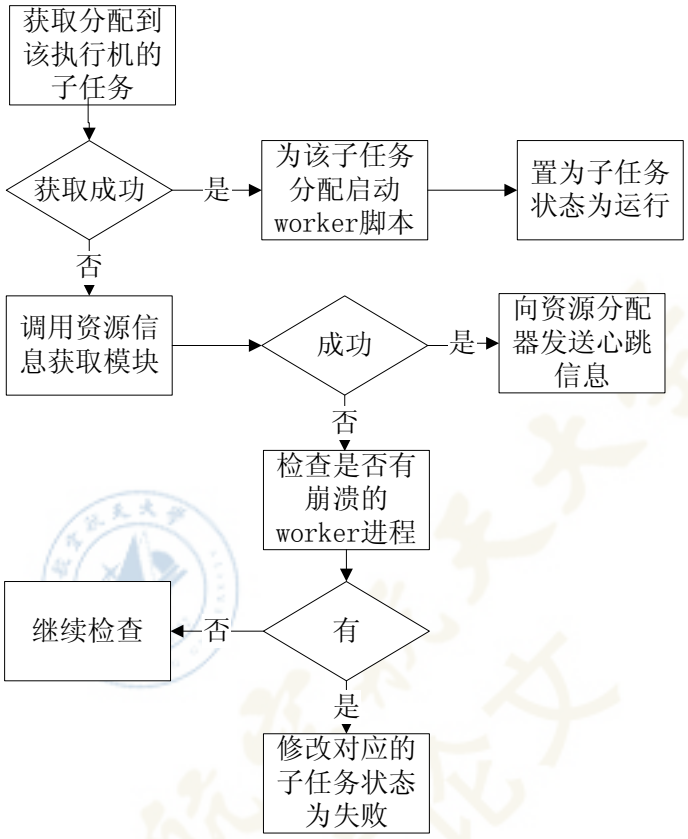

图 24 执行器工作流程图

执行器子扫描数据库，若是获取到分配到本台执行机的子任务，则为该子任务分配 Worker 执行脚本，并执行该脚本，执行脚本为直接调用 linux 系统命令的方式；然后执行器调用系统资源的获取模块，准备向资源分配器发送心跳信息，若是系统资源模块获取成功，则发送心跳，实现的方式就是更新数据库表中的 hbTime 字段和 hbInfo 字段，hbTime 字段的值为目前的时间，hbInfo 字段为系统资源信息统计情况。然后，执行遍历所有正在运行的子任务，检查正在本机上运行的 Worker 进程，通过 linux 系统命令“ps

-ef | grep”来实现，若是某个子任务的进程已经不存在，即崩溃，则把该子任务的状态置为失败。

- 执行器的类图

执行器部分主要有两个大类，ExecutorMachineResource 类以及 ExecutorD 类，其中 ExecutorMachineResource 类为获取资源状态所用，ExecutorD 类完成执行器的主要工作。

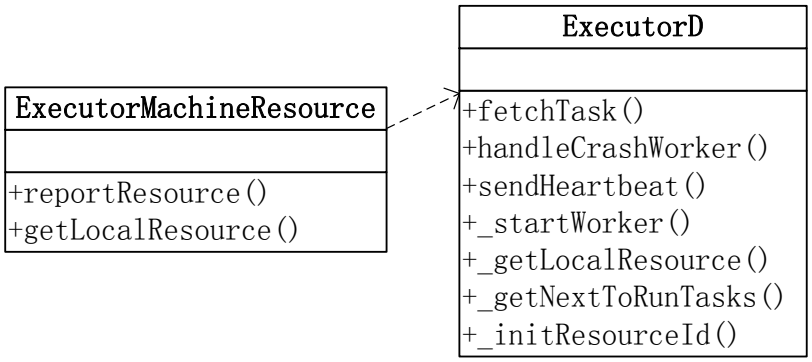

图 25 执行器类图

ExecutorMachineResource 类中方法 reportResource()的作用是向执行器报告本机资源情况，方法 getLocalResource 获取本机资源情况；

ExecutorD 类中方法 fetchTask()的作用是获取已分发的子任务，handleCrashWorker()处理崩溃的 Worker 进程，sendHeartbeat()发送本执行机的心跳信息，startWorker()可以开启一个 Worker 进程，getLocalResource()获得本机的资源状况。

5.3 ETL 模块的设计

ETL 模块，主要分为三个部分，数据获取插件、数据清洗插件以及数据转换插件，这三个被执行器模块的调用执行，利用 Hadoop/Hive 客户端来完成 ETL 过程的几个重要的步骤。

5.3.1 数据获取插件设计

数据获取模块的主要的作用是将线上服务器的日志数据或者是 HDFS 上的数据放在专门的指定集群的 HDFS 路径上。目前数据获取插件获取数据的方式主要有两种：

1）若日志存在用户的线上服务器，则先把数据 wget 到执行机指定的目录上，然后再上传到 HDFS 以供数据清洗；

2）若数据在同一集群上，则执行集群内拷贝；

3）若数据在不同的集群上，则执行集群间的数据传输；

- 数据获取插件执行流程图

首先对数据目标进行判断，若是线上服务器，则调用 linux 命令 wget，把文件下载到执行机之后，在上传到集群上去；若是同一个集群内的拷贝，则直接调用 Hadoop cp 命令来完成；若是集群间的文件拷贝，则调用 Hadoop distcp 命令来实现文件的集群间传输。流程图如图 26：

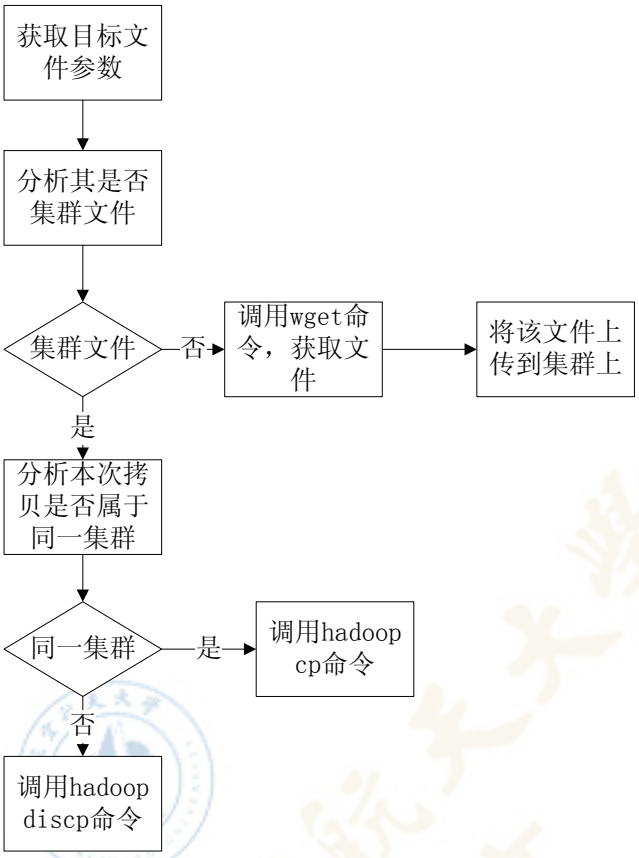

图 26 数据获取插件流程图

下面是对文件在线上服务器时的获取方式实现：

把文件下载到本执行机，用到的系统命令是：“wget --timeout=30 --tries=3 "'.$src_ftp_path."' -O ".$local_path”，其中根据系统需求，响应超时时间为 30 秒，则放弃下载，故参数“--timeout=30”；可以重试的次数为三次，设置参数为“--tries=3”；

将文件上传至集群，主要的步骤分为以下三个步骤：

1）$Hadoop_cli fs -ls $Hadoop_file_path 查看集群上该文件是否存在；

2）若文件存在，则执行命令“$Hadoop_cli fs -rm $Hadoop_file_path”将该文件删除；

3）上传文件至集群，用到的 Hadoop 命令是“$Hadoop_cli fs -D dfs.replication=25 -D

dfs.block.size=33554432 -put $local_path $Hadoop_file_path”，参数“dfs.replication n=25”表示相应时间不超级 25 秒，参数“dfs.block.size=33554432”表示文件大小不得超过33554432k；

而对于集群间的拷贝则直接利用 Hadoop 提供的 discp 接口来实现，命令如下：

hadoop distcp -D mapred.job.priority=\"HIGH\" -D mapred.job.map.capacity=50 -D mapred.job.reduce.capacity=50

该命令中，设置了下载作业优先级为高，Mapper 和 Reducer 的个数分别为 50，这样保证了数据传输的效率。

- 数据获取插件类图

数据获取插件主要有两个类，DownloadPlugin 类和 HadoopClient 类，

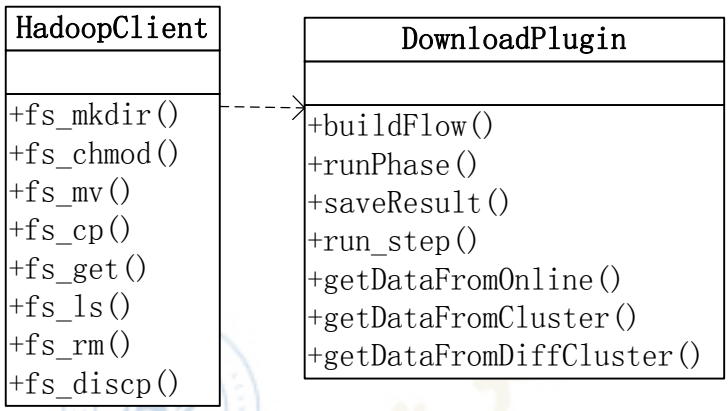

图 27 数据获取插件类图

DownloadPlugin 类是方法 getDataFromOnline()实现了从线上服务器获取数据，方法 getDataFromCluster()实现了从同一集群上获得数据，方法 getDataFromDiffCluster()实现了从不同的集群上获取数据；

HadoopClient 类封装了 Hadoop 的各种操作，包括 Hadoop ls 命令，Hadoop rm 命令，Hadoop cp 命令，Hadoop discp 命令等。

5.3.2 数据清洗插件设计

数据清洗插件完成对了数据的清洗以及抽取工作，数据清洗和抽取是 ETL 过程中的一个主要部分，它把原始的日志数据变成带有格式，即能够被 Hive 元数据所描述的数据，这种数据有个特点，数据都分割为几个固定的列，每个列之间都有约定的间隔符，只有这样的数据才会被 Hive 所识别；Hive 能够识别之后，利用类 SQL 语言 HQL，继

续对数据进行处理，例如 group by、join 等，最终得到一个 Hive 数据表，以供后续的数据转换之用。

- 数据清洗插件执行流程图

数据清洗插件主要分为两大部分，MapReduce 程序执行模块和 Hive 的执行模块。数据清洗与抽取的逻辑根据每份数据的不同而不同，都是有用户来实现的。对于一个数据清洗作业，可能分为几个步骤，每个不外乎两种类型，一种是 MapReduce 类型的任务，一般用于数据清洗；另一种是 Hive 类型任务，实现数据的抽取过程。数据清洗插件能够根据用户的配置，解析出分步执行信息，而后分别调用 MapReduce 程序执行模块和 Hive 的执行模块，串行执行每一个步骤。图 30 是数据清洗插件的执行流程图：

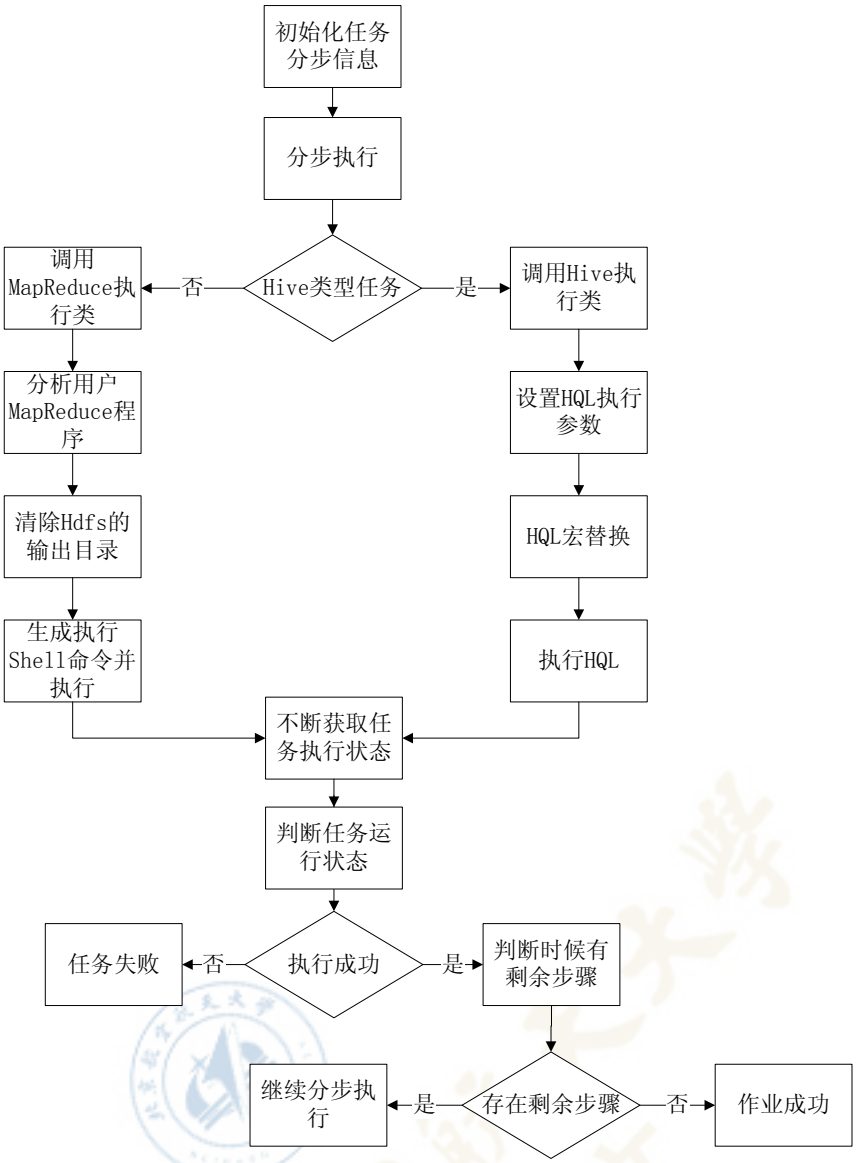

图 28 数据清洗插件执行流程图

图 28 是数据清洗插件的执行流程图，图中描绘了数据清洗插件的执行流程，该流程如下：

1）分析用户的任务配置信息，得出任务的每个步骤的类型，以及执行代码，建立任务分步执行信息；

2）开始分步执行任务，若该步骤是执行的 MapReduce 程序，则调用 MapReduce 执行类来执行；分析用户 MapReduce 程序，获取结果 HDFS 输出路径，若是该路径存在，则删除该路径，去掉干扰数据；生成 shell 命令，该命令用来调用 Hadoop 客户端执行 MapReduce 程序；首先，把 MapReduce 程序的输出定向到日志文件中，并对 MapReduce 的执行日志进行实时分析，同时，使用”hadoop job -status”作业状态检查接口来取得

Hadoop 作业执行状态。若是该 Hadoop 作业实行失败，则整个数据清洗任务置为失败，执行成功，则继续执行作业的下一个步骤；

3）若是该步骤执行的 Hive 程序，则调用 Hive 执行类来执行；设置 HQL 的执行参数以及日期宏的替换，参数主要分为以下几种，mapred.job.queueu.name，Hadoop.job.ugi，mapred.job.priority 以及 mapred.job.name 等，这些参数分别设置了 Hive 的执行队列，运行时账号，运行优先级的高低一个 Hadoop 作业名；执行 HQL，通过调用 Hive 客户端的来执行 HQL，用到的 shell 命令为“hive –f HQL”，其中 HQL 为已经设置执行参数和日期宏替换后的 HQL 文件。

4）在任务执行期间，系统不断对任务的状态进行获取并判断，若是任务运行成功，则进行数据清洗的下一步执行过程，若是执行完所有步骤，作业成功；否则，将数据清洗任务状态置为失败。

- 数据清洗插件类图

数据清洗插件中，主要有四个大类，分别是主类 Etl_plugin 类，Phase 类，MRPhase 类以及 HivePhase 类，其中 Phase 类是 MRPhase 类和 HivePhase 类的父类，父类中实现了例如 run()、kill()等基本方法。

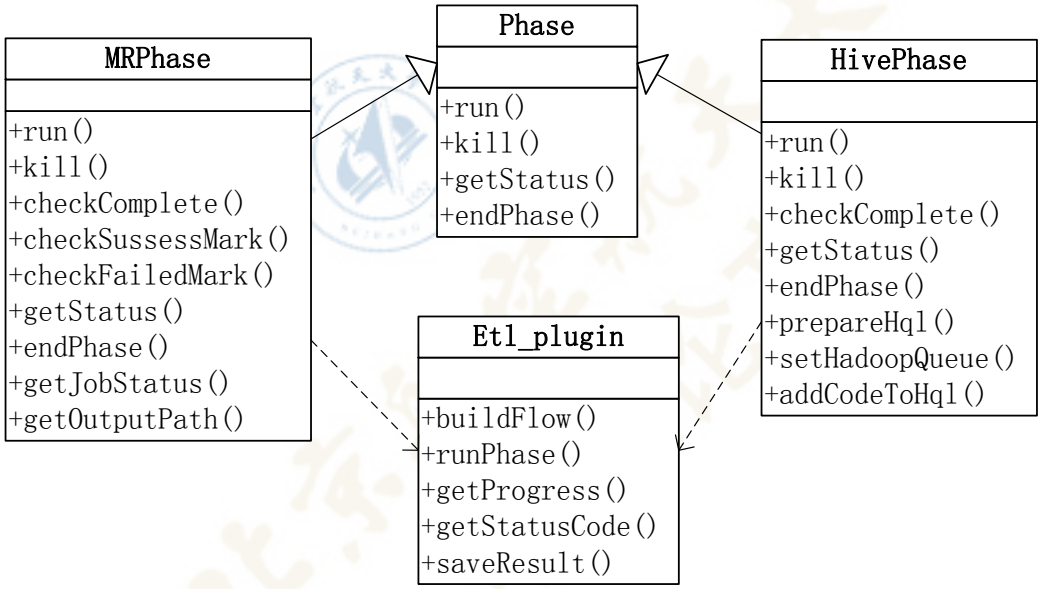

图 29 数据清洗插件类图

Etl_pluging 类是数据清洗插件的主类，buildFlow()方法实现了用户配置的分析以及分步执行的初始化，runPhase()方法实现了分步执行，getProgress()方法实现了获取任务运行时的状态，saveResult()方法保存任务运行结果。

Phase 类是 MRPhase 类和 HivePhase 类的父类，实现了两个类的共同方法，例如 run()方法和 kill()方法等。

MRPhase 类为执行 Hadoop 任务阶段所用，getOutputPath()方法获取 HDFS 输出路径，getJobStatus()方法获得 Hadoop 任务的执行状态，checkComplete()方法查看 Hadoop 任务是否完成，checkSuccessMark()方法查看 Hadoop 的任务是否成功，checkFailedMark()方法查看 Hadoop 的任务是否失败。

HivePhase 类为执行 Hive 任务阶段所用，prepareHQL()方法实现了 HQL 的预处理，setHadoopQueue() 方法、setHadoopUgi() 方法、setHadoopPriority() 方法以及 setHadoopJobName()方法实现了 HQL 中队列参数的设置。

5.3.3 数据转换插件的设计

数据转换插件完成了对数据的转换工作，数据转换是 ETL 过程的重要步骤，它负责把已经被清洗和抽取的数据，按照一定的业务逻辑，转化为相应的目标数据结构，以便装载到数据仓库，供后期的商业多为分析等。在系统中，数据转换插件实现了如下功能：

1）分析用户对维度表与事实表的映射配置，以及对物化视图的定义，自动生成 HQL 代码并执行，最终生成多个星型模型；

2）实现了生成数据转换的算法优化，当用户配置多个物化视图时，一轮计算就可以实现多个物化视图的星型模型的转换，极大的提高了海量数据转换的效率；

3）利用 Hive 的动态分区机制，在数据转换的同时，实现了载入数据仓库的操作；

- 物化视图概念：

对于一个 Hive 元数据表，物化视图是其中若干列的组合，通常，用户根据不同的关注点，会在同一个表上建立多个物化视图，以方便查询时的效率。也可以说，整张表是一个最大的物化视图。物化视图与索引有很多相似的地方，它们都是为了提高查询性能，而且，它们的增加和删除对整体数据不会有影响。在数据转换的过程中，每一个物化视图都要转换成一个星型模型[22]。

- 数据转换的算法描述以及优化

数据转换的过程都是由 Hive 来实现的，Hive 提供的 HQL 是一种类 SQL 的查询语言，它基本支持所有 SQL 类型的操作，比如 select、join 以及 group by 等，它操作的数据对象存储在 HDFS 上，在数据量级上是传统数据库无法比拟的。

由于 Hive 是被最终转化成若干轮 MapReduce 任务执行的，而 MapReduce 的轮数越少，数据转换执行的效率就越高，执行的时间就越少。由此分析可知，减少 MapReduce 的轮数是优化算法的关键所在[23]。下面我们分别介绍常规算法和优化后的算法（略去维度表的生成过程，前文已经讲到，维度表被所有物化视图所公用）：

- 常规算法

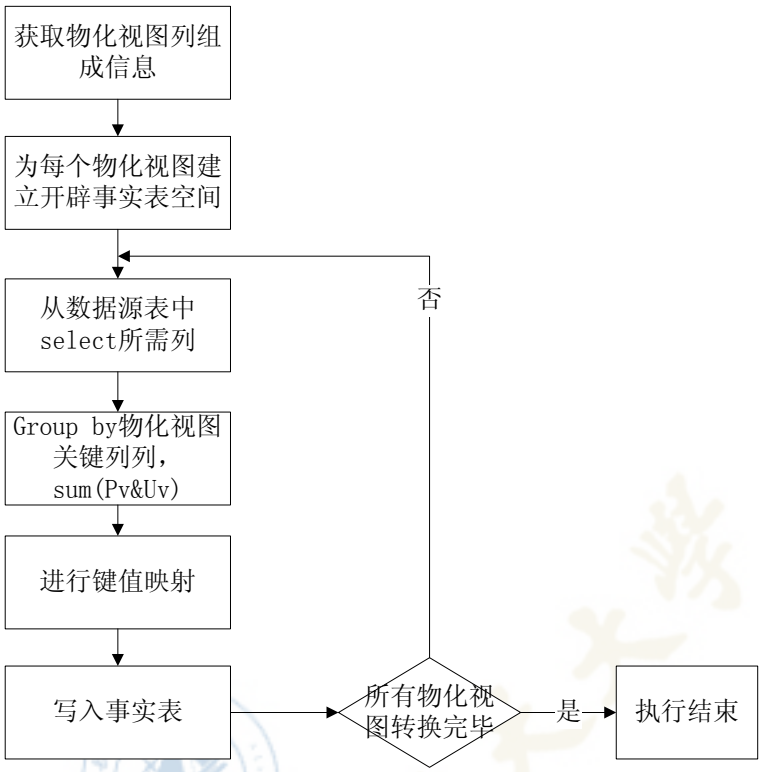

图 30 常规数据转换算法流程图

图 30 描述的是常规的数据转换算法，该算法描述如下：

a) 获取所有的事实表以及物化视图的配置信息；

b) 由于每个物化视图都对应生成一个星型模型，所有每个物化视图都应建立一个临时 Hive 表，该表用来存储事实表数据；

c) 对于每个物化视图，依次处理，从数据源表中 select 出该物化视图所需的列；

d) Group by 物化视图关键列，Sum 得到每个组的 pv 和 uv 等；

e) 根据维度表映射上述字段为对应的键值，写入 Hive 表；

f) 若所有物化视图都处理完，则运行结束。

- 优化后的算法

从上述描述可知，若是随着用户配置的物化视图逐渐增多，运算要进行的轮数也随之逐渐增多，由于 Hive 最终的运算最终转化为 MapReduce 任务，轮与轮之间的执行关

系为串行，更多的轮数，对应着更多的 MapReduce 执行轮数，若是用户配置的物化视图很多，那么该算法注定效率地下，执行时间偏长[24]。

所以，要提高数据转换任务的效率，减少 Hive 最终生成的 MapReduce 的轮数是最关键的点，毕竟根据上述分析，算法的效率低下的原因主要是每个物化视图的处理，都要经历一轮 MapReduce，所以新算法的核心就是致力于减少最终要执行的 MapReduce 轮数。如何减少最终要执行的 MapReduce 轮数呢？

可以从分析常规算法如法入手，对于每一个物化视图的处理，其中最主要的操作便是 group by 操作，而 Hive 会对每个 group by 操作生成一个 MapReduce 任务，所以减少 group by 的次数才能实现算法效率的大幅度提升。

根据以上的分析，可得出优化后的算法流程图如图 31 所示：

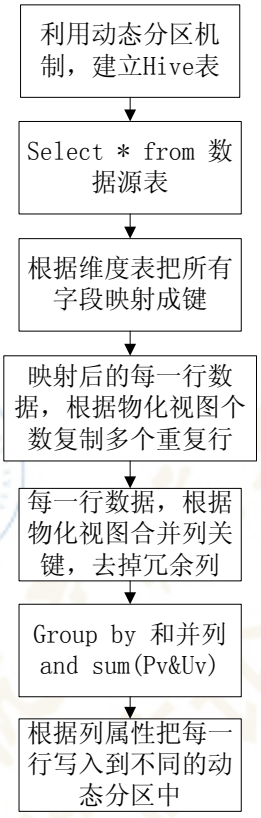

图 31 优化后的数据转换算法流程图

图 31 给出了优化的数据转换算法的流程图，在图中可以看出，整个数据转化过程中，只执行了一次 group by 操作，这就意味着整个 Hive 最终最终只转化为一个 MapReduce 任务，整个算法的效率得到了显著的提升。以下是算法的详细描述：

Step1：

建立一个 Hive 表，用于加载转换后的星型模型数据。Create Table…PARTITION BY

(table_type STRING, table_name STRING)，建表时，定义两个分区列，实现所有事实表以及维度表的分区存储；

Step2：

Select * from 数据源表。这一步的主要作用是把所有的数据从数据源表中取出，以便做出对数据的处理；

Step3：

根据维度表把所有字段映射成键。这一步数据转换成的星型模型的关键步骤，源数据的某一个字段或若干个字段，根据维度表被转化对应的键值，而真正的数据内容，被存储在维度表中（假设一个 Hive 数据表的列为年、月、日、国家、省、市、品牌、机器型号、浏览器、浏览器公司，PV，UV）。映射关系如表 9：

表 9 数据字段映射关系表

原字段	新字段
年、月、日	时间键
国家、省、市	地域键
品牌、机器型号	品牌机型键
浏览器、浏览器公司	浏览器键

转换的过程是利用 HQL 中提供的 Map oldColumns Using “shell” As newColumns 语句来完成的。其中 shell 的用来调用用户上传的执行脚本，是 Hive 的提供的特有功能，通过调用用户自定义脚本可以实现处理逻辑的多样化和复杂化。整个语句的功能是这样的，从源数据表 select 出来的所有数据，以标准输入流的形式，即 STDIN，输入到了用户自定义处理脚本当中，用户自定义脚本 DimenMatch.php 可直接从 STDIN 中读取到源数据表的所有数据，并以行的单位进行处理转换，把每一行的数据都转化新字段对应的值，输出到标准输出流（STDOUT）当中，生成新行。

Step4：

映射后的每一行数据，根据物化视图个数复制出多个重复行。对于每一个新行，根据物化视图合并列，去掉冗余列。这一步是整个新算法的核心所在，是最重要的步骤。在这一步中，首先，源数据行数发生了改变，在若是用户配置的物化视图为 N 个，则总的行数会变成原行数的（N+1）倍。其次，对于每一个新行，数据的列结构根据不同物化视图的又发生了一次变化。

表 10 生成新行实例表

物化视图名	物化视图列	旧列	新列
View0	地域、浏览器	地域键、时间键、品牌机型键、浏览器键，pv，uv	Concat_ws(‘\t’, 地域键、浏览器键)，‘View0’，pv，uv
View1	时间、浏览器	地域键、时间键、品牌机型键、浏览器键，pv，uv	Concat_ws(‘\t’, 时间键、浏览器键)，‘View1’，pv，uv

新行生成实例如表 10 所示，每行对于每一个不同的物化视图，都要生成一行新的数据，新的列结果都为 4 列：

第一个列为 Concat_ws（’\t’，column1，column2…），Concat_ws 为 HQL 自带函数，它的作用是用’\t’符号间隔，把每个物化视图的列都合成一列，这样一来，用这一列就可以区分所有不同的物化视图[25]；第二列为物化视图名，第三列和第四列分别为 pv 和 uv。

这一步的实现方式如下，Select Explode(Array(Struct(Concat_ws（’\t’，column1，column3），’View0’，pv，uv), Struct(Concat_ws（’\t’，column2，column3），’View1’，pv，uv))) as Line，其中 Explode 与 Array 配合，实现了一行变多行，Struct 和 Concat_ws 配合，完成了对新列的封装[26]。

Step5：

Sum(pv), Sum(uv) && Group by Line.col1，Line.col2，这一步最终实现了对不同的物化视图的 pv 以及 uv 的计算。其中 Line.col1 为 Concat_ws（’\t’，column1，column3），这一列所有物化视图所有列的和并列，它唯一标识某行属于哪个物化视图；Line.col2 为物化视图名称，从 Step4 可得知，它的值是物化视图名称，Line.col1，Line.col2 何在一起标识某一行数据的归属[27]。所以 Sum(pv), Sum(uv) && Group by Line.col1，Line.col2 操作合起来实现了不同物化视图的对于 pv 和 uv 的计算。

Step6：

根据物化视图名把每一行写入到不同的动态分区中。以上的几个步骤执行完毕之后，数据的行数已经扩展了若干倍，其中每一行的都归属于一个特定的物化视图。余下的工作就是把每一行分配其所属的物化视图对应的 Hive 分区中。

这一步的实现方式如下：Insert Overwrite Table eveTbl Partition (view_partition) Select Case when Line.col2 = ‘View0’ Then columns,‘View0’when Line.col2 = ‘View1’ Then columns, ‘View0’。不同星型模型的事实表信息被存储在 Hive 表的不同分区中[28]，通过

HQL 提供的 Case When Then 分支语句实现不同行的数据写入到不同的分区中，在数据转换的同时实现了数据仓库的载入。

通过上述分析可知，新的数据转换算法只用一轮的 group by 操作就实现了建立多个星型模型的操作，Hive 最终转换成的 MapReduce 任务也只有一轮，整个数据转换的效率得到了很多提高[29]。

- 数据转换插件的类图

数据转换插件共有五个类，分别是 transformPlugin 类、hiveExecutor 类、DimenConfParser 类、HQLPhrase 类和 HQLSentence 类。图 32 是数据转换插件的类图：

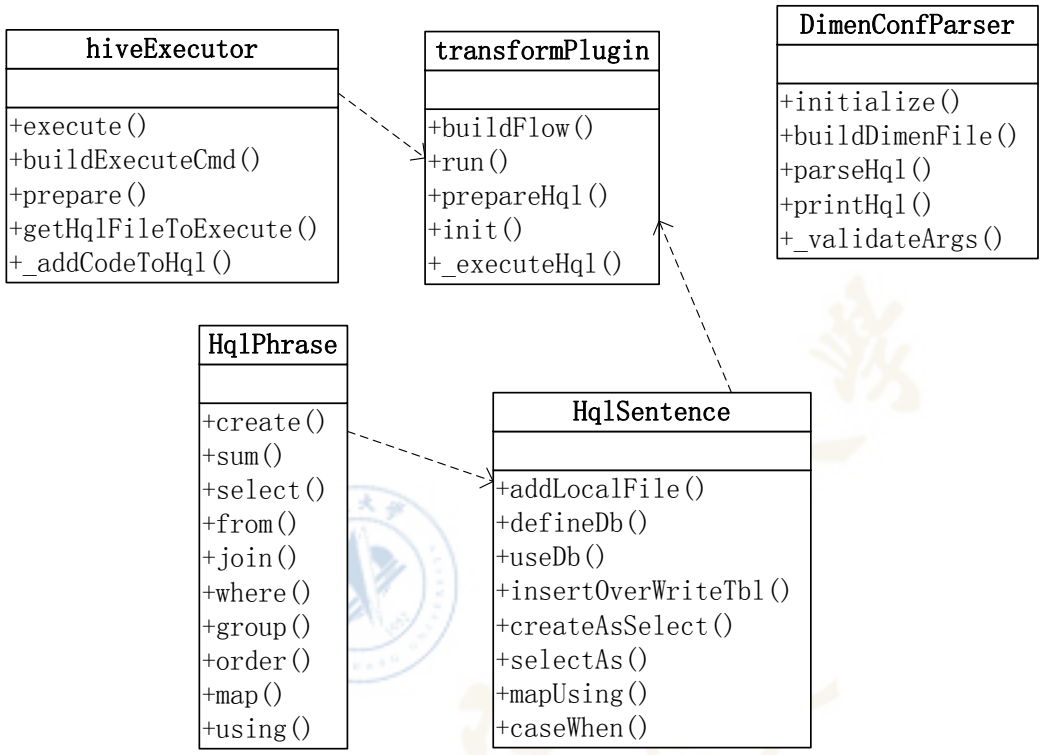

图 32 数据转换插件类图

transformPlugin 类是整个插件的入口类，其中 init()方法负责插件的初始化，run()发放启动数据转换任务的执行，prepareHQL()和 executeHQL()方法分别负责数据转换 HQL 的生成和执行；

hiveExecutor 类是主要负责 Hive 的 shell 执行命令的生成，以及 Hive 的执行，以及一些 Hive 参数的设置，类中主要有 execute()和 buildExecuteCmd()等方法；

DimenConfParser 类的作用是根据用户的配置，包括物化视图的配置等，自动生成用户数据转换的 HQL。类中 initialize()方法负责生成 HQL 前的初始化工作，buildDimenFile()方法负责维度表的生成，parseHQL()方法和 printHQL()方法分别负责

HQL 的生成和输出。

HQLSentence 类的作用是生成 HQL 语句的工具类，类中的每个方法都能生成一个 HQL 语句，类中的方法主要有 createAsSelect()，selectAs()，mapUsing()，caseWhen()等；

HQLPhrase 类的也是生成的 HQL 语句的工具类，区别于 HQLSentence 类，它的多个方法可以拼接出各种类型的 HQL 语句，类中的主要方法有 select()方法、sum()方法、where()方法、join()方法、group by()方法等；

5.4 本章小结

本章给出系统的三个重要模块的详细设计，包括用户交互模块、调度模块以及 ETL 模块。用户交互模块介绍了用户命令执行流程以及模块功能图；调度模块给出了子任务状态机的设计，以及子任务的在调度器的执行流程，然后分别讲解了任务触发器、资源分配器、依赖检查器以及执行器的实现；ETL 模块中，讲解了三个重要插件，数据获取插件、数据清洗插件和数据转化插件的设计和实现。

第六章 系统测试

测试是软件开发中很重要的一项，是对软件需求分析、设计规格说明和编码的最终复审，是软件质量保证的关键步骤。本章主要介绍整个系统的测试工具、方法及结果，最后给出测试结果的基本评估。

6.1 测试理论介绍

软件测试是利用测试工具按照测试方案和流程对产品进行功能性和非功能性测试。

整个测试过程是伴随着软件项目的整个开发过程，主要包括五个阶段：单元测试、集成测试、确认测试、系统测试和验收测试[30]。

单元测试是检验各单元模块能否正常工作。

集成测试是将已测试的模块组装起来进行测试，检验与软件设计相关的程序结构问题。

确认测试是对照软件规格说明，检验开发的软件能否满足所有功能和性能的需求，以及决定开发的软件是否合格，能否提交用户使用。

系统测试是检验开发的软件能否与系统的其它部分（如硬件、数据库、操作人员等）协调工作。

6.2 单元测试

整个系统主要依赖于开发过程中的单元测试来保证各个模块功能点的正确性与完备性。

6.2.1 触发器单元测试

触发器是整个调度系统中非常重要的部分，它的作用是触发 ETL 作业为一系列的子任务，触发方式可以分为系统自动触发和用户手动触发。触发 ETL 作业的功能是触发器的主要功能，因此，我们的单元测试主要围绕这个功能来实施。

我们主要测试的 trigger 类的 trigger()函数，trigger()函数的作用就是触发 ETL 作业，并且对触发出来的子任务进行依赖性检查：

表 11 触发 ETL 任务的测试用例表

<table>
<tr><td>测试编号</td><td colspan="3">测试用例 1</td></tr>
<tr><td>功能描述</td><td colspan="3">触发 ETL 作业</td></tr>
<tr><td>用例目的</td><td colspan="3">检查触发 ETL 作业功能的健壮性，以及功能的正确性</td></tr>
<tr><td>前提条件</td><td colspan="3">无</td></tr>
<tr><td colspan="2">输入</td><td>期望的输出</td><td>实际情况</td></tr>
<tr><td colspan="2">输入已经处在运行状态的 ETL 作业</td><td>函数返回错误码-2，表示该 ETL 作业已经处在运行状态，系统忽略用户的请求</td><td>与期望相符</td></tr>
<tr><td colspan="2">输入一个不存在的 ETL 作业</td><td>函数错误码-3，该错误码表示为“试图触发不存在的 ETL 作业”</td><td>与期望相符</td></tr>
<tr><td colspan="2">输入一个依赖关系有错误的 ETL 作业</td><td>函数错误码-4，该错误码表示表示 ETL 作业中有子任务的依赖不存在或者错误</td><td>与期望相符</td></tr>
<tr><td>评估准则</td><td colspan="3">检查函数返回错误码的值内容是否符合预期</td></tr>
<tr><td rowspan="2">测试规程</td><td colspan="3">分别将输入栏中的不同类型的 ETL 作业作为参数传入触发器中的 trigger()函数</td></tr>
<tr><td colspan="3">记录功能函数的返回结果</td></tr>
<tr><td>约束假设</td><td colspan="3">无</td></tr>
</table>

触发 ETL 作业的单元测试中提供了三种测试用例，分别用来测试三种正常或异常的输入的情况下，触发 ETL 作业的 trigger()函数是否能够正常运行：

1）第一组测试用例中输入了一个已经处在运行状态的 ETL 作业，这属于一种正常的输入，用户可能会做出这样的操作，该函数的输出结果为错误码-2，表示该作业已经被调度。经过测试，输出结果符合预期；

2）第二组测试用例中输入了一个不存在的 ETL 作业，这属于一个正常的输入，有些时候用户可能会手动提交一个不存在的作业，该函数的输出结果为错误码-3，表示该作业不存在。经过测试，输出的结果符合预期；

3）第三组输入了一个依赖关系有错误的 ETL 作业，这属于一个异常的输入，该函数返回的输出结果是错误码-4，表示作业依赖错误或依赖不存在。经过测试，输出的结果符合预期；

经过以上三组测试输入，覆盖了触发器触发 ETL 作业时可能遇见的几种异常情况，测试结果表明触发器能够正确处理这些异常；

6.2.2 资源分配器单元测试

资源分配器在调度系统中的作用是为子任务分配合适的执行资源，这种分配有两个阶段，一个是选择执行机资源队列，另一个是从执行机资源队列中取出子任务发送到执

行机。资源分配功能是资源分配器的是主要功能，我们主要围绕这个功能测试。

我们主要测试的dispatch类dispatchTaskToExecutor()函数，dispatchTaskToExecutor ()函数的作用就是分配执行机资源的队列上的子任务到执行机：

表 12 资源分配的测试用例表

测试编号	测试用例 2	
功能描述	为子任务分配执行机资源	
用例目的	检查资源分配器资源分配算法的健壮性，以及功能正确性	
前提条件	无	
输入	期望的输出	实际情况
输入三个子任务 A,B,C，它们的优先级函数增加，任务类型和截止时间一致	函数的返回子任务 A	与期望相符
输入三个子任务 A,B,C，它们的截止时间依次递增，任务类型和优先级一致	函数返回子任务 A	与期望相符
输入一个与该队列类型不符的子任务 A	返回错误码-1，该错误码表示该子任务类型与队列不符	与期望相符
评估准则	检查输出结果是否有值，值内容是否符合预期	
测试规程	分别将子任务输入到函数中	
	记录功能函数的返回结果	
约束假设	无	

资源分配单元测试中提供了三组测试用例，分别用来表示三种正常或异常的输入的情况下，资源分配算法是否有效：

1）第一组测试用例中输入三个子任务 A、B、C，它们的优先级一次增加，任务类型和截止时间一致，这属于一种正常的输入，函数输出的结果是子任务 A。经过测试，输出结果符合预期；

2）第二组测试用例中输入三个子任务 A、B、C，它们的截止时间依次递增，任务类型和优先级一致，这属于一种正常的输入，函数输出的结果是子任务 A。经过测试，输出结果符合预期；

3）第三组测试用例中输入一个与该队列类型不符的子任务 A，这属于一种异常的输入，返回错误码-1，该错误码表示该子任务类型与队列不符。经过测试，输出结果符

合预期；

经过以上三组测试输入，举出三个例子来测试资源分配的算法的正确性，测试结果表明资源分配器的资源分配算法正确；

6.2.3 依赖检查器单元测试

依赖检查器的主要作用是定时解除系统中处在阻塞状态的子任务的依赖，然后把子任务的置为就绪状态。解除子任务依赖是依赖检查器的主要功能，我们主要围绕这个功能测试。

我们主要测试的 RelyCheck 类 checkBlockTask()函数，checkBlockTask ()函数的作用就是解除阻塞子任务的依赖：

表 13 依赖检查器的测试用例表

测试编号	测试用例 3	
功能描述	解除子任务依赖	
用例目的	验证依赖检查器的健壮性，以及功能的正确性	
前提条件	无	
输入	期望的输出	实际情况
输入一个带有错误依赖的阻塞状态的子任务	返回错误码-1，该错误码表示系统异常，该子任务依赖不存在或者异常	与期望相符
输入一个不处于阻塞状态的子任务	返回错误码-2，该错误码表示系统异常，该子任务为处在阻塞状态	与期望相符
评估准则	检查输出结果是否有值，值内容是否符合预期	
测试规程	分别将子任务输入到函数中	
	记录功能函数的返回结果	
约束假设	无	

执行子任务的测试用了两组测试用例，分别用来表示二种正常或异常的输入的情况下，依赖检查功能的健壮性：

1）第一组测试用例中输入一个带有错误依赖的阻塞状态的子任务，这属于一种异常的输入，函数输出的结果错误码-1，该错误码表示系统异常，该子任务依赖不存在或者异常。经过测试，输出结果符合预期；

2）第二组测试用例中输入一个不处于阻塞状态的子任务，这属于一种异常的输入，函数输出的错误码-2，该错误码表示系统异常，该子任务未处在阻塞状态。经过测试，输出结果符合预期；

经过以上两组测试用例，测试依赖检查器的健壮性，以及功能的正确性，测试结果表明依赖检查器的健壮性，以及功能的正确性符合预期。

6.2.4 执行器单元测试

执行器在调度系统中的作用是定时捕捉已经被分发过来的子任务，然后为子任务选择合适的 Worker 脚本，来执行这个子任务。执行子任务是执行器的主要功能，我们主要围绕这个功能测试。

我们主要测试的 Executor 类 executor()函数，executor ()函数的作用就是执行已经分发过来的子任务：

表 14 执行子任务的测试用例表

<table>
<tr><td>测试编号</td><td colspan="3">测试用例 4</td></tr>
<tr><td>功能描述</td><td colspan="3">执行子任务</td></tr>
<tr><td>用例目的</td><td colspan="3">检查执行器的健壮性，以及功能的正确性</td></tr>
<tr><td>前提条件</td><td colspan="3">无</td></tr>
<tr><td colspan="2">输入</td><td>期望的输出</td><td>实际情况</td></tr>
<tr><td colspan="2">输入一个错误类型的子任务，它的类型系统中没有记载</td><td>返回错误码-1，该错误码表示系统异常，执行器收到错误类型的子任务</td><td>与期望相符</td></tr>
<tr><td colspan="2">输入一个状态不是已分发状态的子任务，</td><td>返回错误码-2，该错误码表示系统异常，执行器收到错误状态的子任务</td><td>与期望相符</td></tr>
<tr><td colspan="2">输入状态是已分发，任务类型正确，执行机 id 不是本机的子任务</td><td>返回错误码-3，该错误码表示系统异常，执行器收到不属于本执行机的子任务</td><td>与期望相符</td></tr>
<tr><td>评估准则</td><td colspan="3">检查输出结果是否有值，值内容是否符合预期</td></tr>
<tr><td rowspan="2">测试规程</td><td colspan="3">分别将子任务输入到函数中</td></tr>
<tr><td colspan="3">记录功能函数的返回结果</td></tr>
<tr><td>约束假设</td><td colspan="3">无</td></tr>
</table>

执行子任务的测试用了三组测试用例，分别用来测试三种异常的输入的情况下，执行子任务功能的健壮性：

1）第一组测试用例中输入一个错误类型的子任务，它的类型系统中没有记载，这属于一种异常的输入，函数输出的结果是错误码-1，该错误码表示系统异常，执行器到错误类型的子任务。经过测试，输出结果符合预期；

2）第二组测试用例中输入一个状态不是已分发状态的子任务，这属于一种异常的输入，函数输出的结果是错误码-2，该错误码表示系统异常，执行器收收到错误状态的

子任务。经过测试，输出结果符合预期；

3）第二组测试用例中输入状态是已分发，任务类型正确，执行机 id 不是本机的子任务，这属于一种异常的输入，返回错误码-3，该错误码表示系统异常，执行器收到不属于本执行机的子任务。经过测试，输出结果符合预期；

经过以上三组测试输入，利用三个测试用例来测试执行器的健壮性，以及功能的正确性，测试结果表明执行器的健壮性，以及功能的正确性符合预期。

6.3 系统的功能性测试

系统的主要功能是对海量的数据进行 ETL。所以，在系统的功能性测试中选用两个 ETL 作业作为测试的用例输入，并用这两个 ETL 作业的执行以及产出状况，作为测试评判的标准。

表 15 是两个 ETL 作业的相关属性的描述：

表 15 系统功能性测试用例输入表

作业描述项	ETL 作业一	ETL 作业二
作业名	insight_ps_channel_union	insight_lbs_channel_union
原始数据	ps_bz_log_udw_log，345GB ps_tn_dim_udw_log，545GB ps_click_access_log，234GB	lbs_vpui_freeapp_log，432GB lbs_vpui_appclient_log，441GB
数据获取子作业数	3 个	2 个
数据清洗子作业数	2 个	1 个
数据转换子作业数	1 个	1 个
作业周期	天级	天级
调度方式	自动	手动

两个 ETL 作业的原始数据都是若干个海量 web 文本日志，他们的子任务数各有不同，作业周期都是天级，启动方式一个是系统自动启动，一个是通过用户交互模块手动启动，表 16 是展示了两个 ETL 任务的执行状态，以及 ETL 过程中每个阶段的执行时间以及结果产出：

表 16 ETL 作业执行状态表

状态项	ETL 作业一	ETL 作业二
作业名	insight_ps_channel_union	insight_lbs_channel_union
数据获取执行时间	23 分钟	35 分钟
数据清洗时间	65 分钟	61 分钟
数据清洗产物	245GB	187GB
数据转换时间	35 分钟	29 分钟
生成星型模型大小	53GB	49GB

表 16 展示了两个 ETL 作业的执行状态以及数据产出状况，包括各个阶段的执行时间以及各个阶段的产出数据量的大小,最终两个 ETL 作业都成功完成了海量数据的 ETL。表 17 是系统功能性测试用例表：

表 17 系统功能性测试用例表

测试编号	测试用例 5	
功能描述	测试系统的海量数据 ETL 功能	
用例目的	验证系统的海量数据 ETL 功能	
前提条件	无	
输入	期望的输出	实际情况
ETL 作业一	ETL 作业被正常调度并执行，最终生成海量数据的星型模型	与期望相符
ETL 作业二	ETL 作业被正常调度并执行，最终生成海量数据的星型模型	与期望相符
评估准则	ETL 作业的调度执行状态是否正常，作业产出是否正常	
测试规程	系统自动触发 ETL 作业一，观察作业调度执行状态，验证产出	
	手动触发 ETL 作业二，观察作业调度执行状态，验证产出	
约束假设	无	

系统的功能测试用例，验证了平台具有调度执行海量数据 ETL 作业，并对海量数据进行 ETL 的功能。该测试用例证明了系统在功能性方面达到了初始的设计目的。

6.4 系统的非功能性测试

在系统的非功能性测试中，重点关注的是系统的系统的可靠性，对于每天数以万计

的 ETL 作业执行执行频率来说，系统的可靠性非常重要，可靠性测试主要验证系统在高并发、高负荷的状况下仍然能够保证正常运行而不至于发生系统崩溃现象。

表 18 是系统的可靠性测试用例：

表 18 系统的可靠性测试用例表

测试编号	测试用例 6	
功能描述	验证系统的可靠性	
用例目的	检查系统在高并发高负荷的环境下是否仍然能够保持运行	
前提条件	无	
输入	期望的输出	实际情况
运行 200 个 ETL 作业	系统正常运行	与期望相符
运行 600 个 ETL 作业	系统正常运行	与期望相符
运行 1000 个 ETL 作业	系统正常运行	与期望相符
评估准则	检查系统的中控机、执行机以及数据库机器的运行状态，CPU、内存、磁盘是否被打满	
测试规程	让系统同时运行多个 ETL 作业	
	观测系统的运行状态	
约束假设	无	

从测试用例中可以看出，系统在并发执行 200、600、1000 个 ETL 作业时，系统正常运行，各项指标均正常。该测试用例验证了系统在可靠性方面是有保障的。

6.5 测试结果评估

本章采用了单元测试、系统功能性测试与系统的非功能测试的结合的测试方法，对整个系统进行了测试。其中，单元测试验证了调度模块中，任务触发器、资源分配器、依赖检查器以及执行器在功能方面的正确性和健壮性；系统的功能性测试验证了系统对海量数据具有 ETL 的能力；系统的非功能性测试验证了系统在多个 ETL 作业并发执行的时候仍具有较高的可靠性。可以做出结论：系统已达到设计目标，在实现上不存在理解偏差，在功能性以及非功能性上均有良好的表现。

6.6 本章小结

在本章节里主要介绍了系统验证的过程和内容。本章首先简单叙述了软件测试的理

论基础。其次详细介绍了系统的单元测试，给出了各个模块功能点的测试用例以及测试结果。而后给出了整个系统的功能性测试用例，验证了整个平台的功能的正确性。然后给出了系统的非功能性测试，验证了系统的可靠性。最后针对整个测试的结果进行分析，得出系统已经达到设计目标的结论。

总结与展望

总结

本系统利用 Hadoop/Hive 实现了海量数据的 ETL，包括海量数据获取，海量数据清洗以及海量数据转换，经过这一系列的处理，海量数据被转换成星型模型，并装载到数据仓库中。

同时，系统还实现了一个工作流式的 ETL 作业调度模块，该模块完成了 ETL 过程中的三个步骤即数据获取，数据挖掘清洗以及数据转换的自动化调度并运行，极大方便了用户，用户不必再单独执行其中的每个步骤。而且，调度系统中独有的资源分配模块，实现了执行资源的有效利用，提高了执行资源的利用率。

展望

目前，系统已经实现了海量数据的 ETL，包括数据获取，数据清洗以及数据转换三个步骤，最终生成数据的星型模型，载入数据仓库。但是，对于数据仓库其它类型的目标数据结构支持的还不够好，相信在平台的下一个阶段，会对雪花模型以及混合模型提供支持。

另外，系统的人机交互方式还处在原始的命令行状态，用户对平台的使用还不是很方便，而且平台没有提供数据展示功能，数据的查看只能通过原始的 Hive 客户端形式；相信在平台的下一个阶段，系统会实现更友好的人机交互方式，并且引入良好的数据展示机制。

参考文献

[1]. 杨卓荦. 数据仓库分布式列存储技术研究与实现[D].昆明理工大学,2012.

[2]. 邰建华. Hadoop 平台下的海量数据存储技术研究[D].东北石油大学,2012.

[3]. J.Dean and S.Ghemawat.MaPReduee:SimPlified Data Processing on Large Clusters[Jl.In Communications of theACM, 2008

[4]. 龙泓. 分布式工作流调度框架的设计和实现[D].清华大学,2011.

[5]. 刘洋. 基于截止时间和自适应的动态网格工作流调度算法研究[D].华中科技大学,2012.

[6]. 陈新明. 淘宝网数据平台数据仓库建设[D].大连理工大学,2013.

[7]. Chen,Hsu,Dayal,A data warehouse/OLAP framework for scalable telecommunication tandem traffic analysis,InProc.200OInt.Conf.Data Engineering(ICDE.00),SanDiego,CA,2000

[8]. 崔杰,李陶深,兰红星. 基于 Hadoop 的海量数据存储平台设计与开发[J]. 计算机研究与发展,2012,S1:12-18.

[9]. 韩鹏. ETL 工具的设计实现与应用[D].吉林大学,2006.

[10].廉博. 数据仓库中 ETL 技术的研究与实现[D].沈阳工业大学,2006.

[11].Thanakorn Naenna,Data Mining Application for elf-OrganizingMaps,Rensselaer Polytechnic Institute,New York,2003

[12].于立. ETL 关键技术研究[D].东南大学,2005.

[13].罗小称. 基于元数据的 ETL 工具设计和实现[D].华东师范大学,2007.

[14].周华炜. 新型工作流资源调度器的研究[D].中南大学,2007.

[15].陈兰. 高性能计算中工作流调度引擎的设计与实现[D].解放军信息工程大学,2007.

[16].Vokan Akcelik, Jacobo Bielak, George Biros, loannis Epanomeritakis. High resolution forward and inverse earthquake modeling on ascale computers.IEEE Computer Society. 2003:5

[17].朱珠. 基于 Hadoop 的海量数据处理模型研究和应用[D].北京邮电大学,2008.

[18].孙如祥,阳琼芳,夏曼. 基于遗传算法的网格工作流调度综述[J]. 轻工科技,2013,06:101-103.

[19].贾自艳,黄友平,罗平,李嘉佑,秦亮曦,史忠植. 面向数据质量的 ETL 过程建模与实现[J]. 系统仿真学报,2004,05:907-911+914.

[20].何晨钢. ETL 系统的设计和实现技术研究[J]. 计算机应用与软件,2009,04:198-201.

[21].陈璐. 基于云计算的海量数据存储技术的研究及应用[D].武汉科技大学,2011.

[22].T. White. Hadoop: The Definitive Guide[M]. 2009. O'Reilly Media, Inc. June 2009.

[23].吴金虎. 基于 Hadoop 的大型网站海量数据的统计与应用[D].南京大学,2012.

[24].Massive Data Covert Transmission Scheme Based on Shamir Threshold[J]. Wuhan University Journal of Natural Sciences,2010,03:227-231.

[25].周敦陬. 基于星型模型的 NBA 球队数据挖掘应用研究[D].华中科技大学,2012.

[26].李晓君. 基于星型模型的单目视频序列人体行为识别[D].西安电子科技大学,2013.

[27].周建芳,徐海银,卢正鼎. 语义信息集成中基于星型模型的上下文转换[J]. 小型微型计算机系统,2009,06:1038-1042.

[28].徐 骥,陶树平. 基于星型模型的数据仓库中维变技术的研究[J]. 计算机工程,2002,04:91-93+208.

[29].张延松,焦敏,王占伟,王珊,周烜. 海量数据分析的 One-size-fits-all OLAP 技术[J]. 计算机学报,2011,10:1936-1946.

[30].张宁,贾自艳,史忠植. 数据仓库中 ETL 技术的研究[J]. 计算机工程与应用,2002,24:213-216.

致谢

本文是在导师康一梅的悉心指导下完成的，论文的完成包含了康老师辛勤的汗水，每一次导师都是在百忙中抽出宝贵的时间对我进行指导，无论从专业知识，写作水平，书写标准等多个方面，对我都有很大的提高和帮助。在此谨向康一梅导师致以诚挚的问候和衷心的感谢。

在论文完成过程中，得到了百度在线网络技术（北京）有限公司的大力支持，在此也感谢公司全体人员对我的悉心指导和无私的帮助。

我的成长离不开每一位关心和帮助过我的人，我也向它们表示由衷的谢意。我愿在未来的工作与学习过程中，不懈努力，以更加丰厚的成果来答谢曾经关心、帮助、支持过我的所有老师、家人、同学和朋友。

最后，也衷心地感谢在百忙之中评阅论文和参加答辩的各位专家、教授。